

**DG0798**  
**Demo Guide**  
**PolarFire FPGA System Services**



---

a  **MICROCHIP** company



a  **MICROCHIP** company

**Microsemi Headquarters**

One Enterprise, Aliso Viejo,  
CA 92656 USA

Within the USA: +1 (800) 713-4113

Outside the USA: +1 (949) 380-6100

Sales: +1 (949) 380-6136

Fax: +1 (949) 215-4996

Email: [sales.support@microsemi.com](mailto:sales.support@microsemi.com)

[www.microsemi.com](http://www.microsemi.com)

©2021 Microsemi, a wholly owned subsidiary of Microchip Technology Inc. All rights reserved. Microsemi and the Microsemi logo are registered trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

### About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at [www.microsemi.com](http://www.microsemi.com).

# Contents

---

<b>1</b>	<b>Revision History</b>	<b>1</b>
1.1	Revision 8.0	1
1.2	Revision 7.0	1
1.3	Revision 6.0	1
1.4	Revision 5.0	1
1.5	Revision 4.0	1
1.6	Revision 3.0	1
1.7	Revision 2.0	1
1.8	Revision 1.0	1
<b>2</b>	<b>PolarFire FPGA System Services</b>	<b>2</b>
2.1	PF_SYSTEM_SERVICES Core Overview	2
2.2	Design Requirements	4
2.3	Prerequisites	5
2.4	Demo Design	5
2.4.1	Design Implementation	6
2.5	Clocking Structure	11
<b>3</b>	<b>Libero Design Flow</b>	<b>12</b>
3.1	Synthesize	12
3.2	Place and Route	13
3.2.1	Resource Utilization	13
3.3	Verify Timing	13
3.4	Generate FPGA Array Data	14
3.5	Configure Design Initialization Data and Memories	14
3.6	Configure Programming Options	17
3.7	Generate Bitstream	17
3.8	Run PROGRAM Action	18
3.9	Export FlashPro Express Job	20
<b>4</b>	<b>Setting up the Serial Terminal Program - PuTTY</b>	<b>21</b>
<b>5</b>	<b>Running the Demo</b>	<b>23</b>
<b>6</b>	<b>Appendix 1: Programming the Device Using FlashPro Express</b>	<b>29</b>
<b>7</b>	<b>Appendix 2: Device Certificate Information</b>	<b>32</b>
<b>8</b>	<b>Appendix 3: Query Security</b>	<b>34</b>
<b>9</b>	<b>Appendix 4: Debug Information</b>	<b>35</b>
<b>10</b>	<b>Appendix 5: Digest Information</b>	<b>36</b>
<b>11</b>	<b>Appendix 6: Running the TCL Script</b>	<b>37</b>
<b>12</b>	<b>Appendix 7: References</b>	<b>38</b>

# Figures

Figure 1	Core System Services IP Interfacing with Fabric User Logic	3
Figure 2	Firmware Catalog	4
Figure 3	System Services Design Block Diagram	5
Figure 4	Top Level Libero Design	6
Figure 5	PF_CCC_0 Input Clock Configuration	7
Figure 6	PF_CCC_0 Output Clock Configuration	7
Figure 7	Mi-V RV32 Configuration	8
Figure 8	Mi-V RV32 Configuration1	9
Figure 9	CoreGPIO_0 Configuration	10
Figure 10	Memory Map	10
Figure 11	Clocking Structure	11
Figure 12	Libero Design Flow Options	12
Figure 13	Design and Memory Initialization	14
Figure 14	Fabric RAMs Tab	15
Figure 15	Edit Fabric RAM Initialization Client	15
Figure 16	Apply Fabric RAM Content	16
Figure 17	Add PlainText NonAuthenticated Client Option	16
Figure 18	Edit PlainText NonAuthenticated Client	17
Figure 19	Configure Programming Options	17
Figure 20	Board Setup—Evaluation Kit	19
Figure 21	Board Setup—Splash Kit	20
Figure 22	Finding the COM Port	21
Figure 23	Select Serial as the Connection Type	21
Figure 24	PuTTY Configuration	22
Figure 25	System Services Options	23
Figure 26	Device Serial Number	23
Figure 27	Device User-code	23
Figure 28	Device Design Information	24
Figure 29	Device Certificate	25
Figure 30	Digest	26
Figure 31	Security Locks Information	26
Figure 32	Debug Information	26
Figure 33	Digital Signature	27
Figure 34	Secure NVM Services	28
Figure 35	PUF Emulation Service	28
Figure 36	Generated Nonce	28
Figure 37	FlashPro Express Job Project	29
Figure 38	New Job Project from FlashPro Express Job	30
Figure 39	Programming the Device	30
Figure 40	FlashPro Express—RUN PASSED	31
Figure 41	Copy Device Certificate	32
Figure 42	Certificate Decoding Using Java Script	33
Figure 43	Decoded Certificate	33

# Tables

---

Table 1	System Services in the Demo .....	2
Table 2	System Services Descriptor .....	3
Table 3	Design Requirements .....	4
Table 4	I/O Signals .....	6
Table 5	Resource Utilization—Evaluation Kit .....	13
Table 6	Resource Utilization—Splash Kit .....	13
Table 7	Jumper Settings for PolarFire Device Programming—Evaluation Kit .....	18
Table 8	Jumper Settings for PolarFire Device Programming—Splash Kit .....	18
Table 9	Device Certificate Fields (1024 bytes) .....	32
Table 10	Security Locks Fields .....	34
Table 11	Debug Info Fields .....	35
Table 12	Digest Information Bit Fields .....	36

# 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 8.0

The following is a summary of the changes made in this revision.

- Replaced text CoreSysServices\_PF with PF\_SYSTEM\_SERVICES.
- Updated [Figure 1](#), page 3, [Figure 3](#), page 5, and [Figure 11](#), page 11.
- Replaced [Figure 4](#), page 6, [Figure 10](#), page 10, and [Figure 25](#), page 23 through [Figure 36](#), page 28.

## 1.2 Revision 7.0

Added [Appendix 6: Running the TCL Script](#), page 37.

## 1.3 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.

## 1.4 Revision 5.0

Updated the document for Libero SoC v12.0.

## 1.5 Revision 4.0

The following is a summary of the changes made in this revision.

- Updated for Libero® SoC PolarFire v2.3.
- Merged SPLASH kit related content.

## 1.6 Revision 3.0

Updated the document for Libero SoC PolarFire v2.2.

## 1.7 Revision 2.0

Updated the document for Libero SoC PolarFire v2.1.

## 1.8 Revision 1.0

The first publication of this document.

## 2 PolarFire FPGA System Services

System services are System Controller actions initiated from the FPGA design using the PF\_SYSTEM\_SERVICES core. The System Controller hard block in PolarFire® FPGAs provide various system services. The PF\_SYSTEM\_SERVICES core issues service requests to the System Controller and fetches the relevant data.

This document describes how to run the system services listed in the following table using the demo design.

**Table 1 • System Services in the Demo**

Service Category	Services
Device and Data Services	Read Device Serial Number Read Device User-code Read Device Design-info
Design and Data Security Services	Read Device Certificate Read Digest Query security Read Debug Information Digital signature Secure NVM services PUF Emulation Nonce service

The demo design includes the Mi-V soft processor, which initiates the system service requests and enables the PF\_SYSTEM\_SERVICES core to access the System Controller. For more information about the system services design implementation, and the necessary blocks and IP cores instantiated in Libero SoC, see, [Demo Design](#), page 5.

The demo design can be programmed using any of the following options:

- **Using the pre-generated .job file:** To program the device using the .job file provided along with the demo design, see [Appendix 1: Programming the Device Using FlashPro Express](#), page 29.
- **Using Libero SoC:** To program the device using Libero SoC, see [Libero Design Flow](#), page 12.

The demo design can be used as a reference to build a fabric design with the system services feature.

### 2.1 PF\_SYSTEM\_SERVICES Core Overview

System controller actions are initiated by the fabric logic through the System Service Interface (SSI) of the System Controller. The fabric logic requires the PF\_SYSTEM\_SERVICES core for initiating the system services. A service request interrupt to the system controller is triggered when the fabric user logic writes a 16-bit system service descriptor to the SSI. The lower seven bits of the descriptor specify the service to be performed. The upper nine bits specify the address offset (0–511) in the 2 KB mailbox RAM. The mailbox address specifies the service-specific data structure used for any additional inputs or outputs for the service. The fabric logic must write additional parameters to the mailbox before requesting a system service.

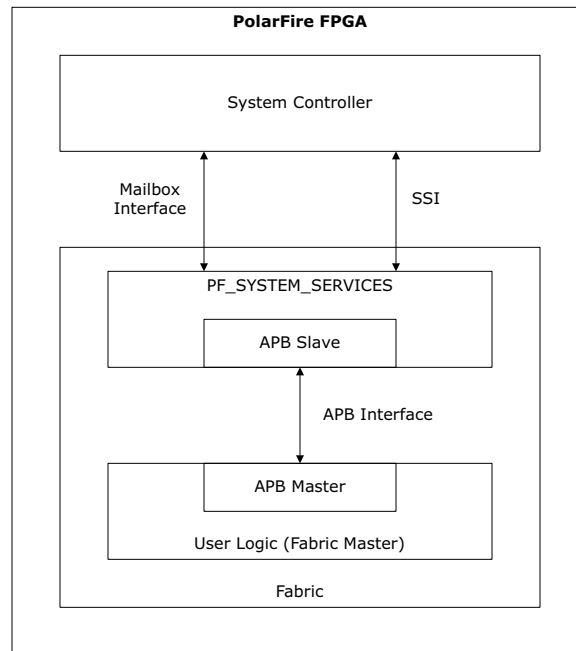
The following table lists the system service descriptor bits.

**Table 2 • System Services Descriptor**

Descriptor Bit	Value
15:7	MBOXADDR
6:0	SERVICEID

SSI consists of an asynchronous command-response interface that transfers a system service command from the fabric master to the system controller and the status from the system controller to the fabric master. The following figure shows how the PF\_SYSTEM\_SERVICES Interfaces with the fabric logic.

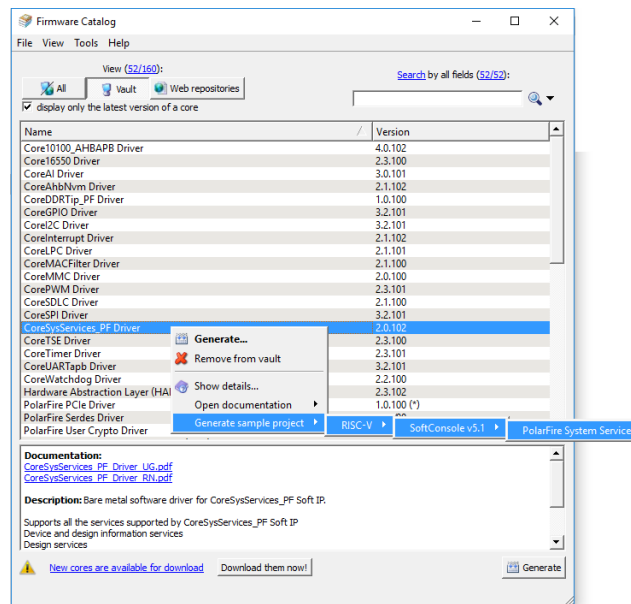
**Figure 1 • Core System Services IP Interfacing with Fabric User Logic**



The system services driver and the sample SoftConsole project are generated from Firmware Catalog as shown [Figure 2](#), page 4.

In this demo, the sample SoftConsole project is migrated to SoftConsole and the application file `main.c` is modified to provide the user options.



**Figure 2 • Firmware Catalog**

## 2.2 Design Requirements

The following table lists the resources required to run the demo.

**Table 3 • Design Requirements**

Requirement	Version
Operating System	Windows 7, 8.1, or 10
<b>Hardware</b>	
PolarFire Evaluation Kit (MPF300T-IFCG1152E) or	Rev D or later
PolarFire Splash Kit (MPF300T-1FCG484E)	Rev 2 or later
<b>Software</b>	
FlashPro Express	<b>Note:</b> Refer to the <code>readme.txt</code> file provided in the design files for the software versions used with this reference design.
Libero SoC Design Suite	
SoftConsole	
Serial Terminal Emulation Program	PutTY or HyperTerminal <a href="http://www.putty.org">www.putty.org</a>

**Note:** Any serial terminal emulation program can be used. PutTY is used in this demo.

**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.

## 2.3 Prerequisites

Before you begin:

1. 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>
2. For demo design files download link:
  - For Evaluation Kit  
[http://soc.microsemi.com/download/rsc/?f=mpf\\_dg0798\\_eval\\_df](http://soc.microsemi.com/download/rsc/?f=mpf_dg0798_eval_df)
  - For Splash Kit  
[http://soc.microsemi.com/download/rsc/?f=mpf\\_dg0798\\_splash\\_df](http://soc.microsemi.com/download/rsc/?f=mpf_dg0798_splash_df)

The latest versions of ModelSim and Synplify Pro are included in the Libero SoC installation package.

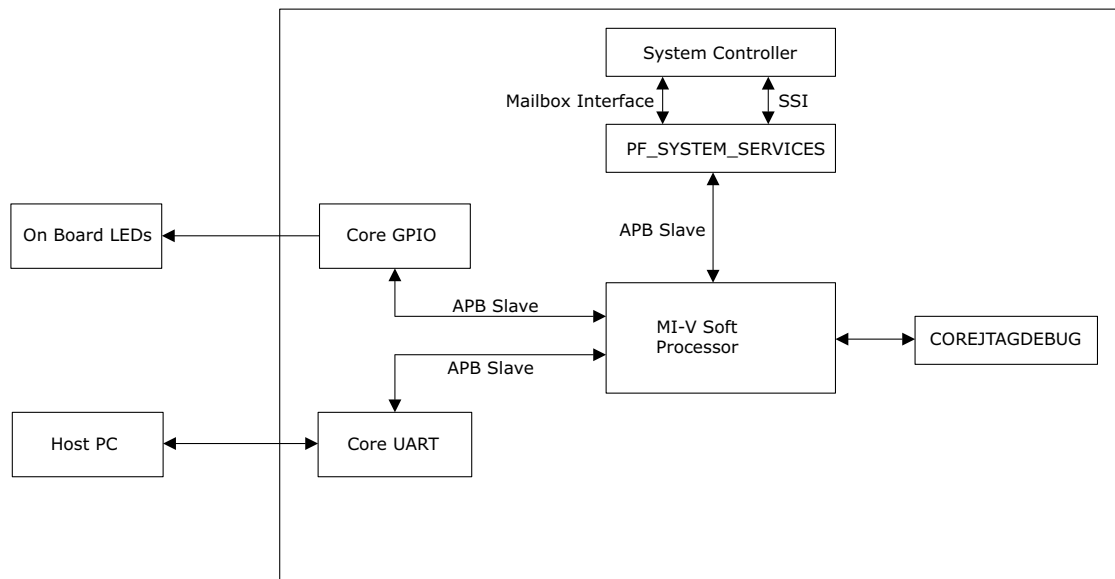
## 2.4 Demo Design

The following steps describe the data flow in the demo design:

1. The host PC sends the system service requests to CoreUARTapb block through the UART Interface.
2. The Mi-V soft processor initializes the system controller using the PF\_SYSTEM\_SERVICES and sends the requested system service command to the system controller.
3. The system controller executes the system service command and sends the relevant response to the PF\_SYSTEM\_SERVICES over the mailbox interface.
4. The Mi-V processor receives the service response and forwards the data to the UART interface.

The following figure shows the block diagram of the system services design.

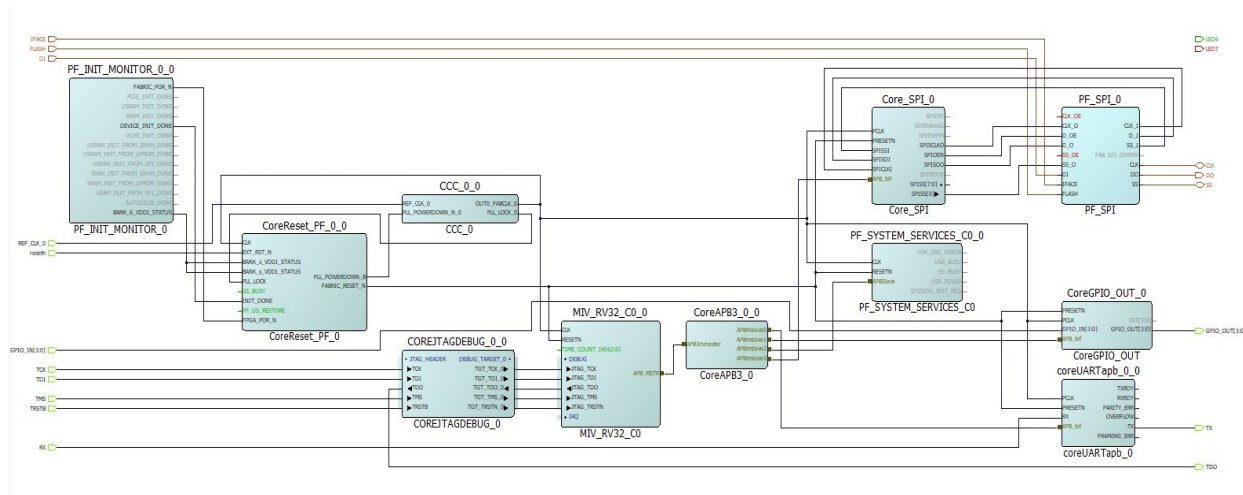
**Figure 3 • System Services Design Block Diagram**



## 2.4.1 Design Implementation

The following figure shows the top-level Libero design of the PolarFire system services design.

**Figure 4 • Top Level Libero Design**



The following table lists the important I/O signals of the design.

**Table 4 • I/O Signals**

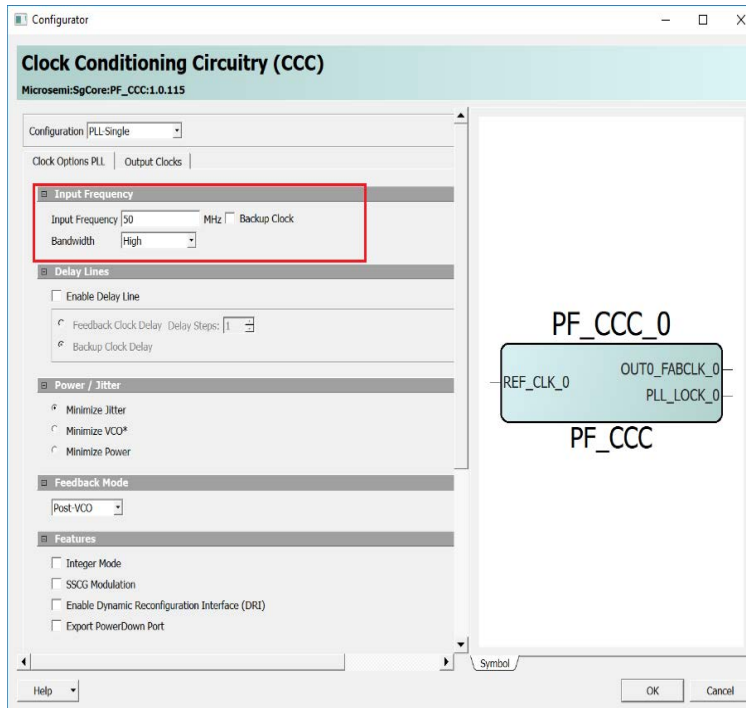
Signal	Description
REF_CLK_0	Input 50 MHz clock from the onboard 50 MHz oscillator
resetn	Onboard reset push-button for the PolarFire device
RX	Input signals received from the serial UART terminal
TX	Output signals transmitted to the serial UART terminal
GPIO_OUT[3:0]	Onboard LED outputs

### 2.4.1.1 PF\_CCC\_0 Configuration

The PolarFire Clock Conditioning Circuitry (CCC) block takes an input clock of 50 MHz from the onboard oscillator and generates a 100 MHz fabric clock to the Mi-V processor subsystem and other peripherals.

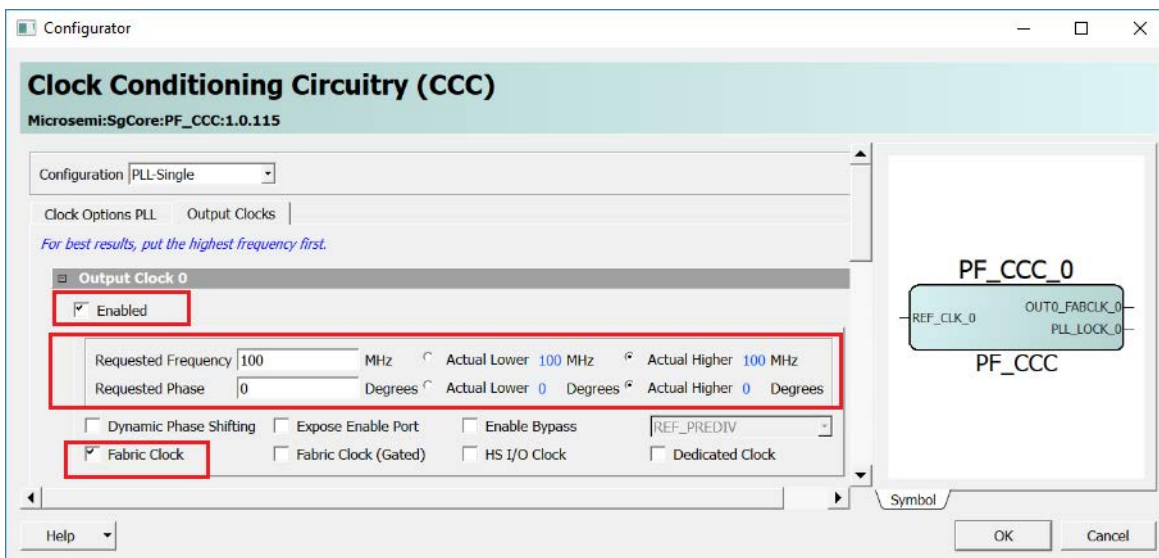
The following figures show the input and output clock configurations.

**Figure 5 • PF\_CCC\_0 Input Clock Configuration**



The following figure shows the PF\_CCC\_0 output clock configuration. The Mi-V processor supports up to 120 MHz. This design uses a 100 MHz system clock for configuring the APB peripherals.

**Figure 6 • PF\_CCC\_0 Output Clock Configuration**



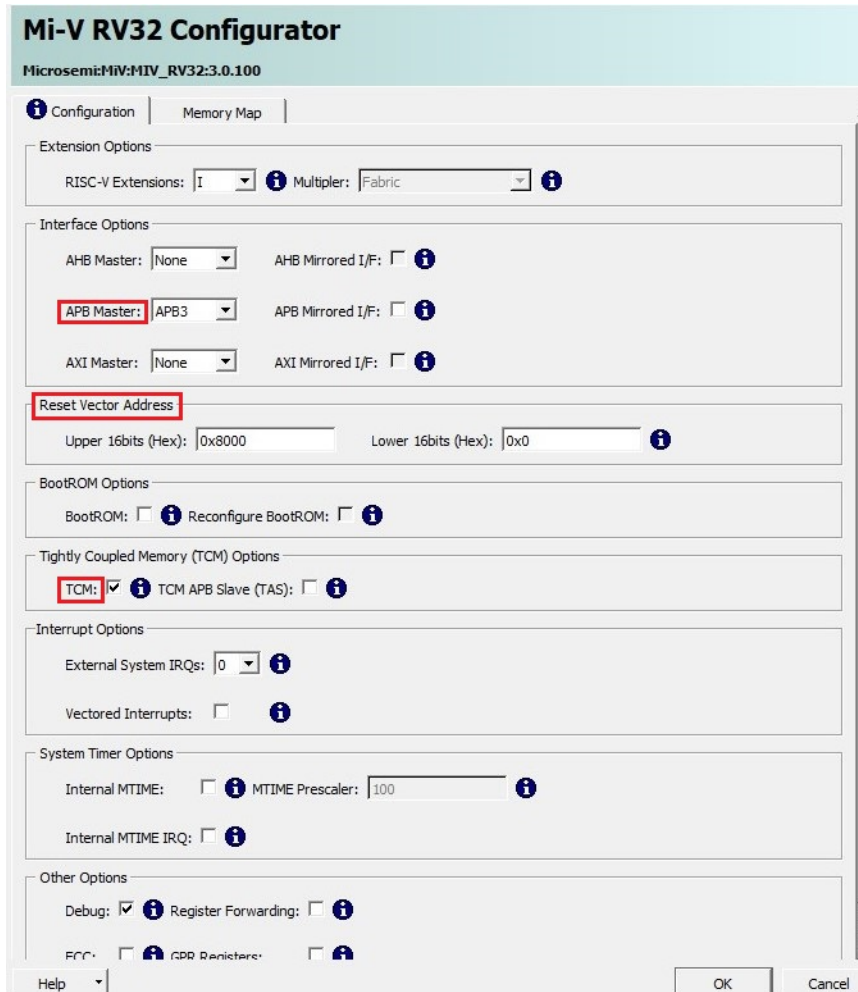
### 2.4.1.2 Mi-V Soft Processor Configuration

The Mi-V soft processor default Reset Vector Address is 0x8000\_0000. After the device reset, the processor executes the application from TCM, which is mapped to 0x80000000, hence the Reset Vector Address is set to 0x80000000 as shown in Figure 7, page 8.

TCM is the main memory of the Mi-V processor. It gets initialized with the user application from µPROM.

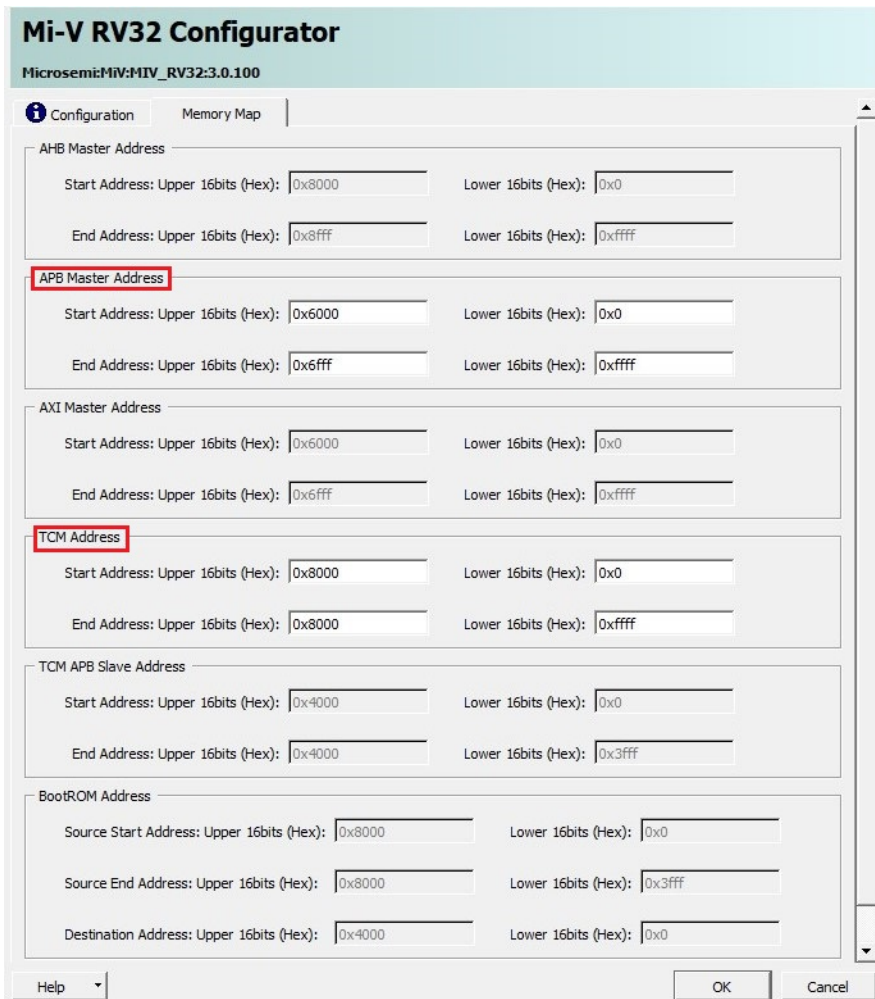
In the Mi-V processor memory map, the 0x8000\_0000 to 0x8000\_FFFF range is defined for TCM memory interface and the 0x6000\_0000 to 0x6FFF\_FFFF range is defined for APB3 I/O interface.

**Figure 7 • Mi-V RV32 Configuration**



The screenshot shows the 'Mi-V RV32 Configurator' window with the 'Configuration' tab selected. The version is 'MicrosemiMiV:MiV\_RV32:3.0.100'. The 'Reset Vector Address' field is highlighted with a red box, showing 'Upper 16bits (Hex): 0x8000' and 'Lower 16bits (Hex): 0x0'. Other fields include 'RISC-V Extensions' set to 'I', 'Multiplier' set to 'Fabric', 'APB Master' set to 'APB3', 'TCM' checked, and 'Debug' checked. The 'Memory Map' tab is also visible at the top.

- **Memory depth:** This field is set to 16384 words to accommodate an application of up to 64 KB into TCM. The present application is less than 50 KB so this can fit into either sNVM or µPROM. In this demo, µPROM is selected as a data storage client as shown in the following figure.

**Figure 8 • Mi-V RV32 Configuration1**


**Mi-V RV32 Configurator**  
 MicrosemiMiV:MiV\_RV32:3.0.100

**Configuration** | Memory Map

**AHB Master Address**

Start Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x8fff Lower 16bits (Hex): 0xffff

**APB Master Address**

Start Address: Upper 16bits (Hex): 0x6000 Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x6fff Lower 16bits (Hex): 0xffff

**AXI Master Address**

Start Address: Upper 16bits (Hex): 0x6000 Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x6fff Lower 16bits (Hex): 0xffff

**TCM Address**

Start Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0xffff

**TCM APB Slave Address**

Start Address: Upper 16bits (Hex): 0x4000 Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x4000 Lower 16bits (Hex): 0x3fff

**BootROM Address**

Source Start Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x0

Source End Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x3fff

Destination Address: Upper 16bits (Hex): 0x4000 Lower 16bits (Hex): 0x0

Help OK Cancel

### 2.4.1.3 CoreGPIO\_0 Configuration

The CoreGPIO IP controls the on-board LEDs using GPIOs. It is connected to Mi-V soft processor as an APB slave.

The configuration settings of the COREGPIO\_0 IP are as follows:

In the **Global Configurations** pane:

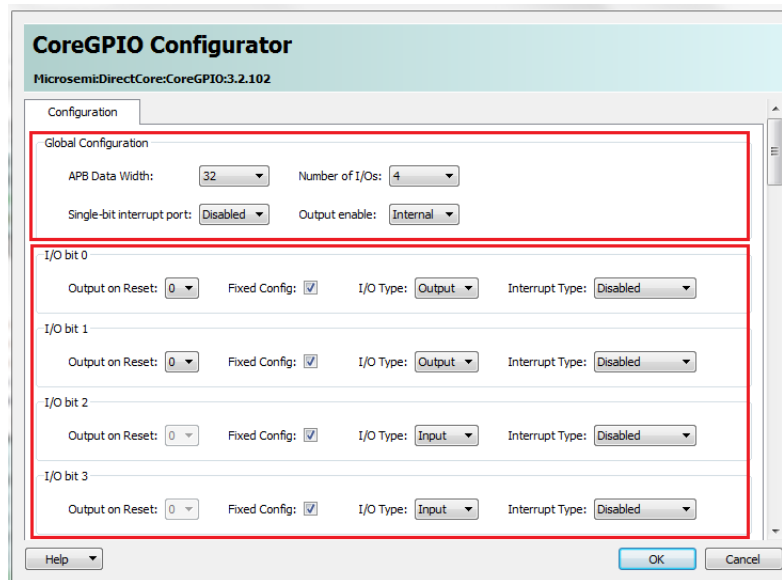
- **APB Data width** is set to 32  
The design uses a 32-bit data width for APB read and write data.
- **Number of I/Os** is set to 4  
The design controls 2 onboard LEDs for output and 2 DIP Switches for input.
- **I/O Bit:** The following list shows the sub-options under I/O Bit option.
  - **Output on reset:** Set to 0
  - **Fixed Config:** Yes
  - **I/O type:** As shown in the following figure, the first two I/Os are configured as an output and the last two I/Os are configured as an input.

**Note:** The first two I/Os configured as output are used by the design and the last two I/Os are not used. The I/Os are interfaced with on-board LEDs to control the LED states.

- **Interrupt Type:** Disabled  
When I/O states change, no interrupt is required for the application as these are used for only LEDs.

The following figure shows the CoreGPIO\_0 configuration.

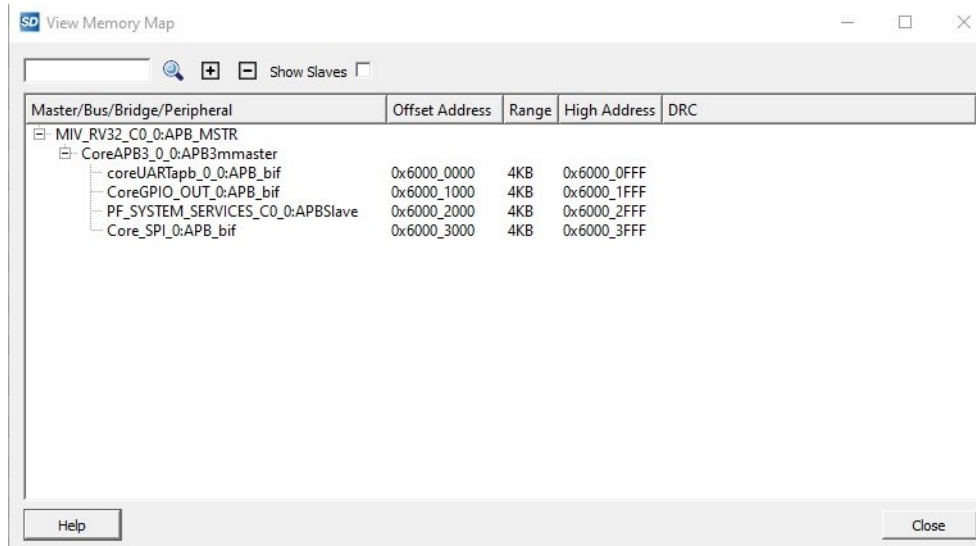
**Figure 9 • CoreGPIO\_0 Configuration**



#### 2.4.1.4 Design Memory Map

The Mi-V processor bus interface memory map is shown in the following figure.

**Figure 10 • Memory Map**



#### 2.4.1.5 CoreAPB3 Configuration

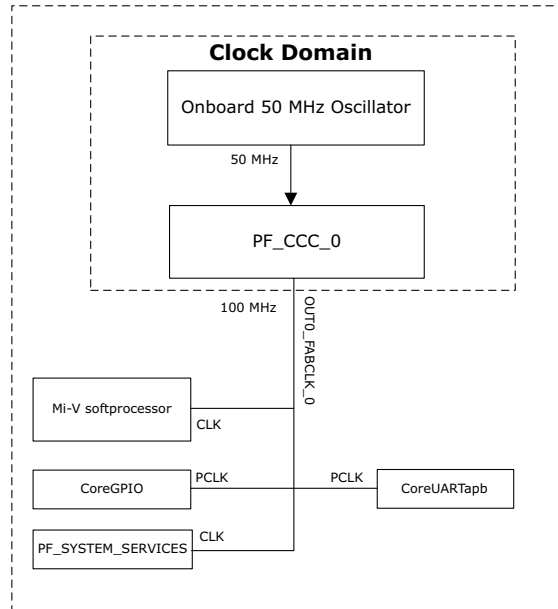
The CoreAPB3 IP connects the peripherals, PF\_SYSTEM\_SERVICES, CoreSPI, CoreGPIO and CoreUARTapb as slaves. The configuration settings of **COREAPB3** are as follows:

- **APB Master Data bus width:** 32-bit  
The design uses a 32-bit data width for APB read and write data.
- **Number of address bits driven by the master:** 16  
The Mi-V processor accesses the slaves using the 16-bit. The final addresses for these slaves are translated into 0x6000\_0000, 0x6000\_1000, 0x6000\_2000, and 0x6000\_3000.
- **Enabled APB slave slots:** Slot 0 for CoreUARTapb, Slot 1 for CoreGPIO, Slot 2 for PF\_SYSTEM\_SERVICES, and Slot 3 for CoreSPI.

## 2.5 Clocking Structure

The following figure shows the clocking structure of the demo design. The Mi-V processor supports clock up to 120 MHz. This design uses a 100 MHz system clock.

**Figure 11 • Clocking Structure**





## 3 Libero Design Flow

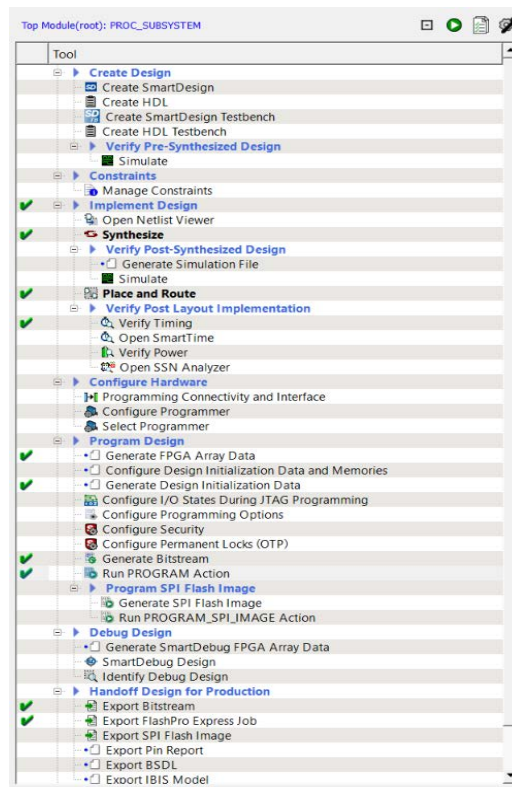
The Libero design flow involves running the following processes in the Libero SoC PolarFire:

- [Synthesize](#), page 12
- [Place and Route](#), page 13
- [Verify Timing](#), page 13
- [Generate FPGA Array Data](#), page 14
- [Configure Design Initialization Data and Memories](#), page 14
- [Configure Programming Options](#), page 17
- [Generate Bitstream](#), page 17
- [Run PROGRAM Action](#), page 18
- [Export FlashPro Express Job](#), page 20

**Note:** To initialize the TCM in PolarFire using the system controller, a local parameter `I_cfg_hard_tcm0_en`, in the `miv_rv32_opsrv_cfg_pkg.v` file should be changed to `1'b1` prior to synthesis. See the 2.7 TCM section in the [MIV\\_RV32 Handbook](#).

The following figure shows these options in the **Design Flow** tab.

**Figure 12 • Libero Design Flow Options**



### 3.1 Synthesize

To synthesize the design, perform the following steps:

1. On the **Design Flow** window, double-click **Synthesize**.  
When the synthesis is successful, a green tick mark appears as shown in [Figure 12](#), page 12.
2. Right-click **Synthesize** and select **View Report** to view the synthesis report and log files in the **Reports** tab.

**Note:** The `PROC_SUBSYSTEM.srr` and `PROC_SUBSYSTEM_compile_netlist.log` files are recommended to be viewed for debugging synthesis and compile errors.

## 3.2 Place and Route

The Place and Route process requires the I/O, timing, and floor planner constraints. The demo design includes following constraint files in the **Constraint Manager** window:

- The `io.pdc` and the `user.pdc` file for the I/O assignments
- The `PROC_SUBSYSTEM_derived_constraints.sdc` file for timing constraints

To Place and Route, on the **Design Flow** window, double-click **Place and Route**.

When place and route is successful, a green tick mark appears next to Place and Route.

**Note:** The file, `PROC_SUBSYSTEM_place_and_route_constraint_coverage.xml` is recommended to be viewed for place and route constraint coverage.

### 3.2.1 Resource Utilization

The resource utilization report is written to the `PROC_SUBSYSTEM_layout_log.log` file in the Reports tab -> `PROC_SUBSYSTEM` reports -> Place and Route. It lists the resource utilization of the design after place and route. These values may vary slightly for different Libero runs, settings, and seed values. The following table lists the resource utilization of the evaluation kit.

**Table 5 • Resource Utilization—Evaluation Kit**

Type	Used	Total	Percentage
4LUT	14207	299544	4.74
DFF	8066	299544	2.69
I/O Register	0	1536	0.00
Logic Element	15187	299544	5.07

The following table lists the resource utilization splash kit.

**Table 6 • Resource Utilization—Splash Kit**

Type	Used	Total	Percentage
4LUT	14597	299544	4.87
DFF	8069	299544	2.69
I/O Register	0	732	0.00
Logic Element	15556	299544	5.19

## 3.3 Verify Timing

To verify timing, perform the following steps:

1. On the **Design Flow window**, double-click **Verify Timing**.  
When the design successfully meets the timing requirements, a green tick mark appears as shown in [Figure 12](#), page 12.
2. Right-click **Verify Timing** and select **View Report**, to view the verify timing report and log files in the **Reports** tab.

### 3.4 Generate FPGA Array Data

To generate the FPGA array data, perform the following steps:

1. On the **Design Flow** window, double-click **Generate FPGA Array Data**.
2. A green tick mark is displayed after the successful generation of the FPGA array data as shown in [Figure 12](#), page 12.

### 3.5 Configure Design Initialization Data and Memories

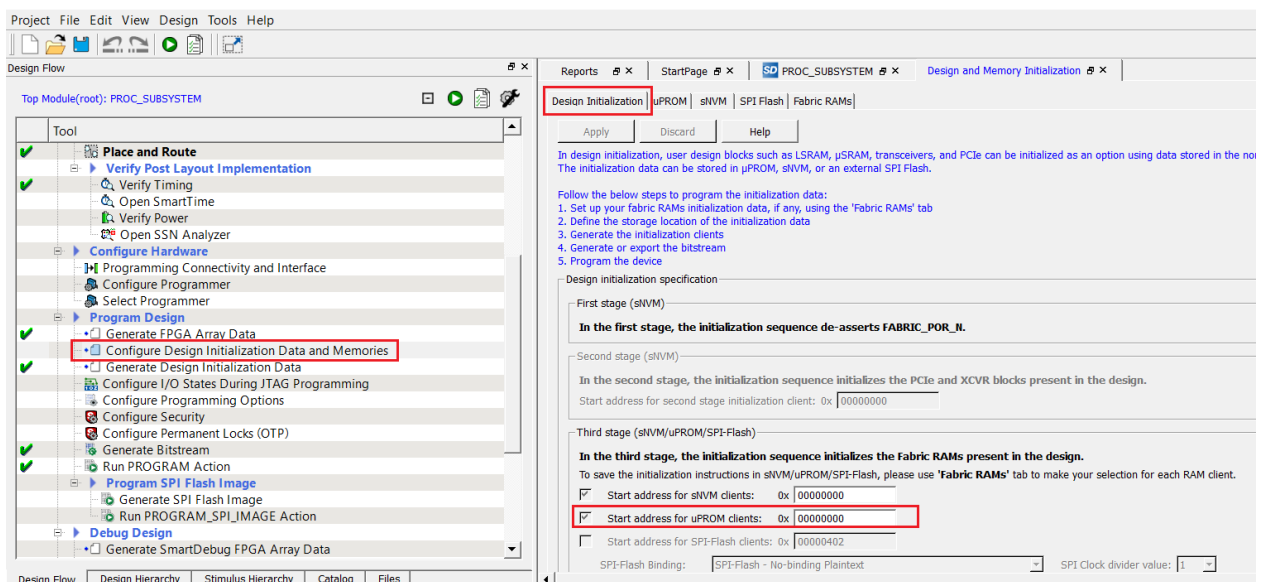
The **Configure Design Initialization Data and Memories** step generates the TCM initialization client and adds it to sNVM, uPROM, or an external SPI flash, based on the type of non-volatile memory selected. In the demo, the TCM initialization client is stored in the uPROM.

This process requires the user application executable file (hex file) to initialize the TCM blocks on device power-up. The hex file (application.hex) is available in the `DesignFiles_Directory\Libero_Project` folder. When the hex file is imported, a memory initialization client is generated for TCM blocks.

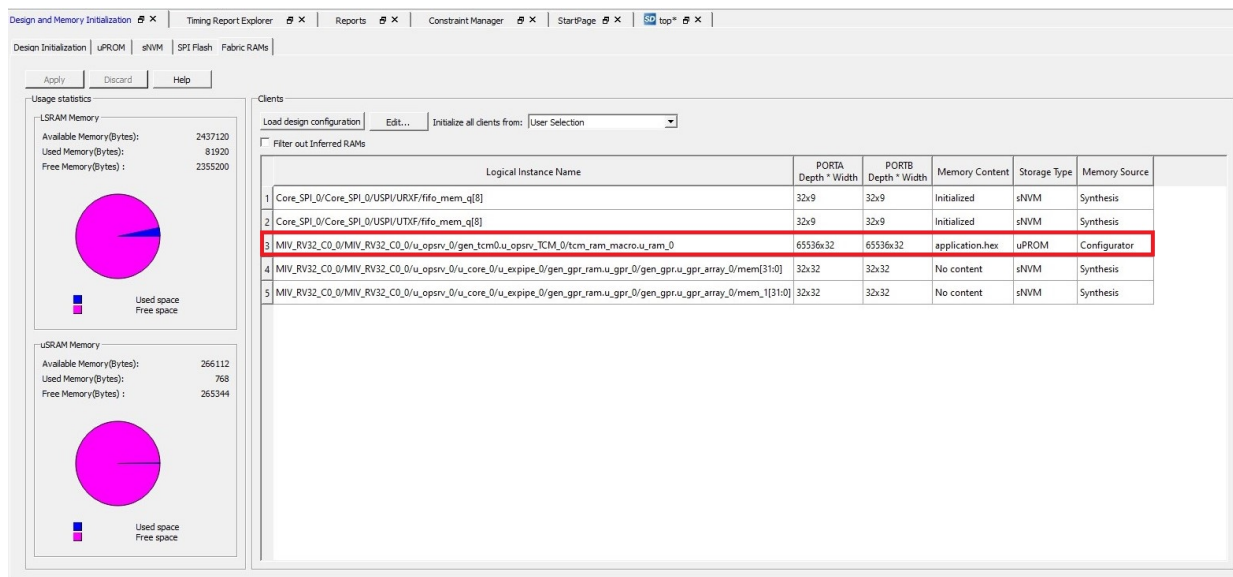
Follow these steps:

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.

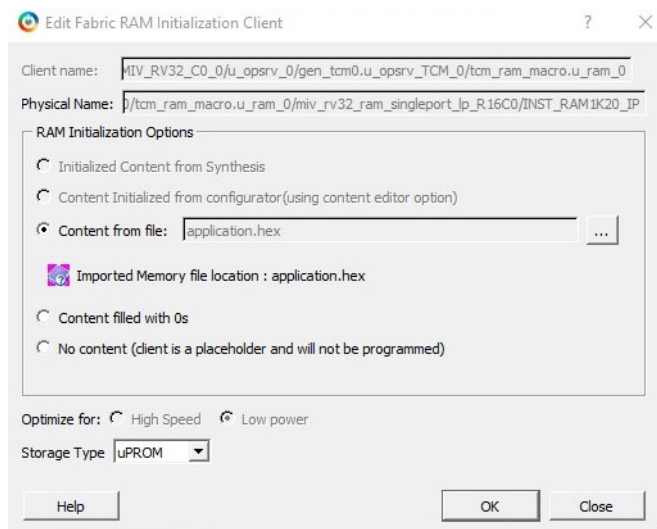
**Figure 13 • Design and Memory Initialization**



2. Select the **Fabric RAMs** tab, select the **tcm\_ram** client from the list, and click **Edit** as shown in the following figure.

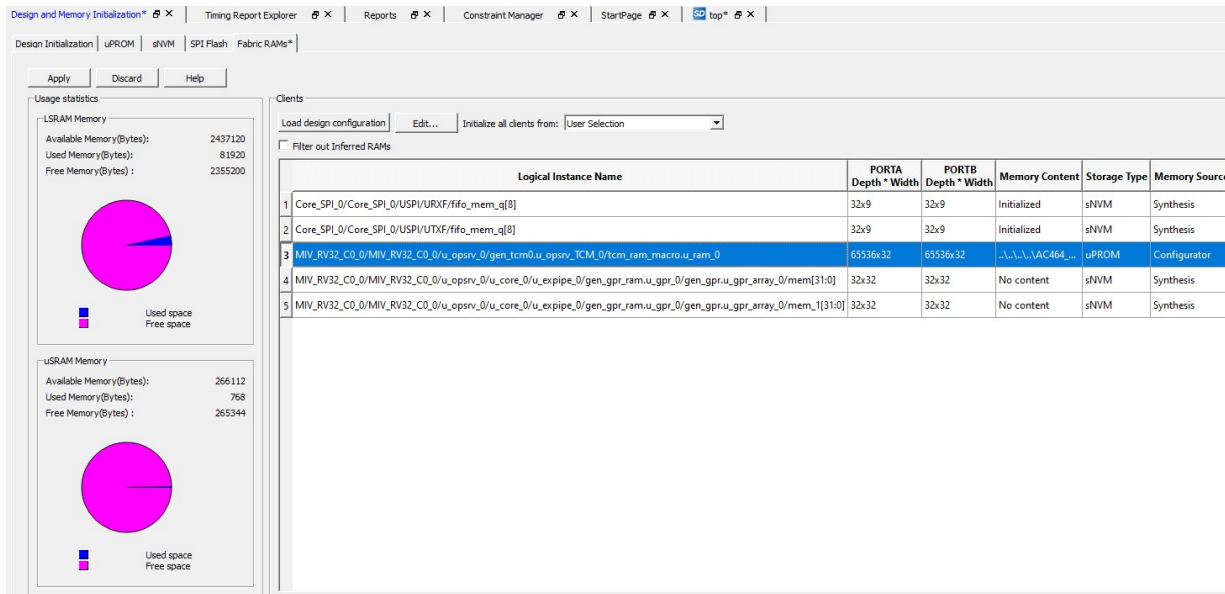
**Figure 14 • Fabric RAMs Tab**

3. In the **Edit Fabric RAM Initialization Client** dialog box, select the **Content from file** option, and locate the `application.hex` file from `DesignFiles_directory\Libero_Project` folder and Click **OK** as shown in the following figure.

**Figure 15 • Edit Fabric RAM Initialization Client**

4. Click **Apply** as shown in the following figure.

**Figure 16 • Apply Fabric RAM Content**

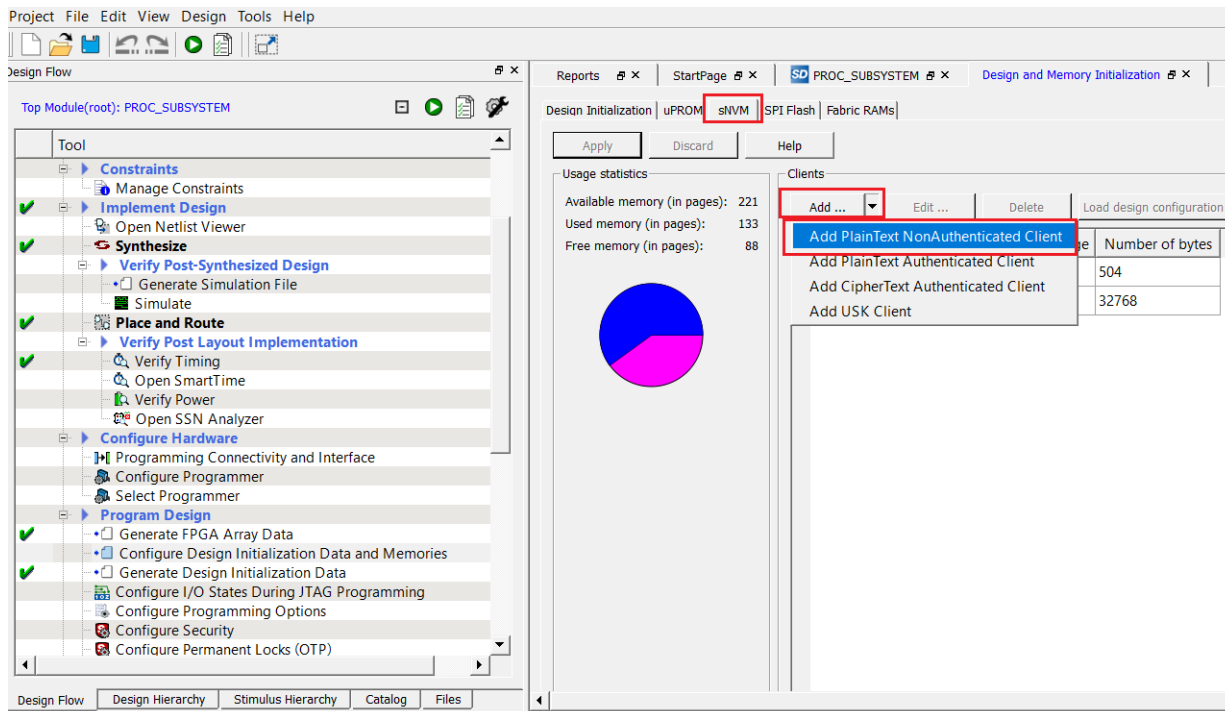


The screenshot shows the 'Design and Memory Initialization' window with the 'Apply' button highlighted. The 'Usage statistics' section displays memory usage for LSRAM and uSRAM. The 'Clients' table lists various memory clients with their logical instance names, depths, widths, memory content, storage types, and memory sources.

	Logical Instance Name	PORTA Depth * Width	PORTB Depth * Width	Memory Content	Storage Type	Memory Source
1	Core_SPI_0/Core_SPI_0/USPI/URXF/ffifo_mem_q[8]	32x9	32x9	Initialized	sNVM	Synthesis
2	Core_SPI_0/Core_SPI_0/USPI/UTXF/ffifo_mem_q[8]	32x9	32x9	Initialized	sNVM	Synthesis
3	MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsvr_0/gen_icm0_u_opsvr_TCM_0/tcm_ram_macro_u_ram_0	65536x32	65536x32	...	uPROM	Configurator
4	MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsvr_0/u_core_0/u_expipe_0/gen_gpr_ram_u_gpr_0/gen_gpr_u_gpr_array_0/mem[31:0]	32x32	32x32	No content	sNVM	Synthesis
5	MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsvr_0/u_core_0/u_expipe_0/gen_gpr_ram_u_gpr_0/gen_gpr_u_gpr_array_0/mem_1[31:0]	32x32	32x32	No content	sNVM	Synthesis

5. Select **sNVM** tab -> Select **Add** from the list -> click **Add PlainText NonAuthenticated Client** as shown in the following figure:

**Figure 17 • Add PlainText NonAuthenticated Client Option**



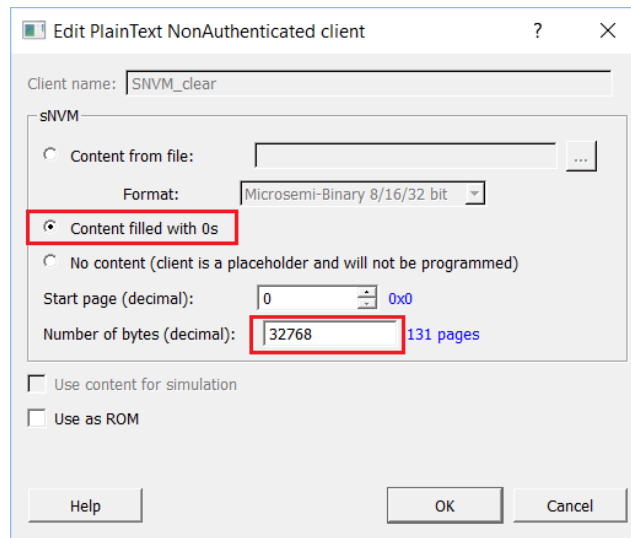
The screenshot shows the 'Design and Memory Initialization' window with the 'sNVM' tab selected. The 'Add' button is highlighted, and the 'Add PlainText NonAuthenticated Client' option is selected from the dropdown menu. The 'Usage statistics' section displays memory usage for sNVM. The 'Clients' table lists various memory clients with their logical instance names, depths, widths, memory content, storage types, and memory sources.

	Logical Instance Name	PORTA Depth * Width	PORTB Depth * Width	Memory Content	Storage Type	Memory Source
1	Core_SPI_0/Core_SPI_0/USPI/URXF/ffifo_mem_q[8]	32x9	32x9	Initialized	sNVM	Synthesis
2	Core_SPI_0/Core_SPI_0/USPI/UTXF/ffifo_mem_q[8]	32x9	32x9	Initialized	sNVM	Synthesis
3	MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsvr_0/gen_icm0_u_opsvr_TCM_0/tcm_ram_macro_u_ram_0	65536x32	65536x32	...	uPROM	Configurator
4	MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsvr_0/u_core_0/u_expipe_0/gen_gpr_ram_u_gpr_0/gen_gpr_u_gpr_array_0/mem[31:0]	32x32	32x32	No content	sNVM	Synthesis
5	MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsvr_0/u_core_0/u_expipe_0/gen_gpr_ram_u_gpr_0/gen_gpr_u_gpr_array_0/mem_1[31:0]	32x32	32x32	No content	sNVM	Synthesis

6. In the preceding step select a client and click **Edit**.

- In the **Edit PlainText NonAuthenticated client** dialog box, select **Content filled with 0's** option and provide the **Number of bytes**. Click **OK** as shown in the following figure.

**Figure 18 • Edit PlainText NonAuthenticated Client**



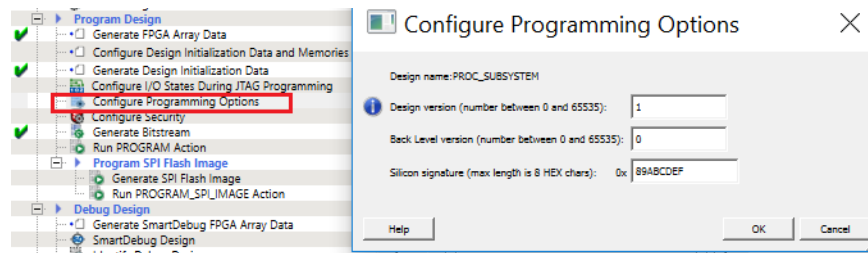
- In the **Design Initialization** tab, click **Apply**.
- On the **Design Flow** window, click **Generate Initialization Data** to generate design initialization data.

After successful generation of the Initialization data, a green tick mark appears next to **Generate Initialization Data** option as shown in the Figure 12, page 12.

## 3.6 Configure Programming Options

The Design version and user code (Silicon signature) are configured in this step. Double click **Configure Programming Options** to give values as shown in the following figure.

**Figure 19 • Configure Programming Options**



## 3.7 Generate Bitstream

To generate the bitstream, perform the following steps:

- On the **Design Flow window**, double-click **Generate Bitstream**. When the bitstream is successfully generated, a green tick mark appears as shown in Figure 12, page 12
- Right-click **Generate Bitstream** and select **View Report** to view the corresponding log file in the **Reports** tab.

## 3.8 Run PROGRAM Action

After generating the bitstream, the PolarFire device must be programmed with the system services design.

To program the PolarFire device, perform the following steps:

1. Ensure that the following jumper settings are set on the evaluation board.

**Table 7 • Jumper Settings for PolarFire Device Programming—Evaluation Kit**

Jumper	Description
J18, J19, J20, J21, and J22	Close pin 2 and 3 for programming the PolarFire FPGA through FTDI
J28	Close pin 1 and 2 for programming through the on-boardFlashPro5
J4	Close pin 1 and 2 for manual power switching using SW3
J12	Close pin 3 and 4 for 2.5 V

Ensure that the following jumper settings are set on the splash board.

**Table 8 • Jumper Settings for PolarFire Device Programming—Splash Kit**

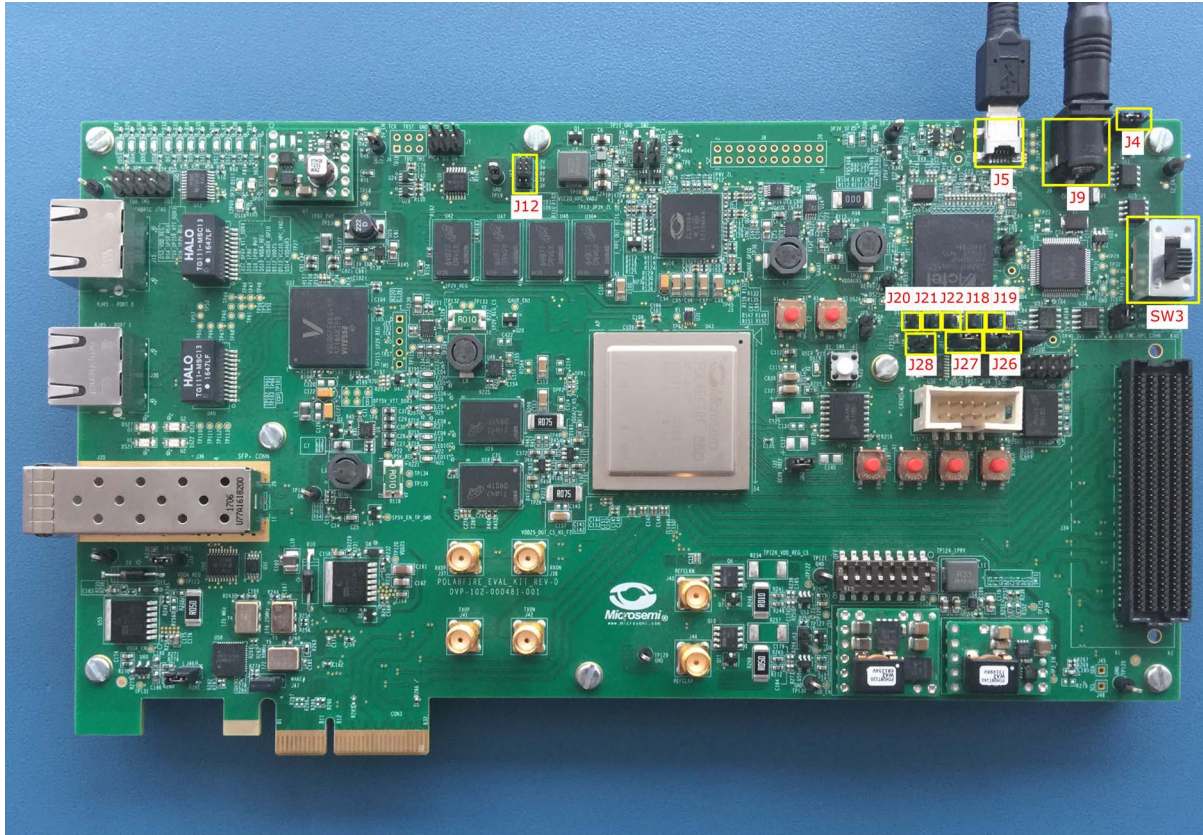
Jumper	Description
J5, J6, J7, J8, and J9	Close pin 2 and 3 for programming the PolarFire FPGA through FTDI
J11	Close pin 1 and 2 for programming through FTDI chip
J10	Close pin 1 and 2 for programming through FTDI SPI
J4	Close pin 1 and 2 for manual power switching using SW1
J3	Open pin 1 and 2 for 1.0 V

2. Connect the power supply cable to the connector **J9** on Evaluation board or **J2** on Splash board.
3. Connect the USB cable from the host PC to the **J5** on Evaluation board or **J1** on Splash board (FTDI port).
4. Power on the board using the slide switch **SW3** on Evaluation board or **SW1** on Splash board.



The following figure shows the board setup of evaluation kit.

**Figure 20 • Board Setup—Evaluation Kit**



The device is successfully programmed and the onboard LEDs 7, 8, 9, 10, and 11 glow. A green tick mark appears next to Run PROGRAM Action as shown in [Figure 12](#), page 12.

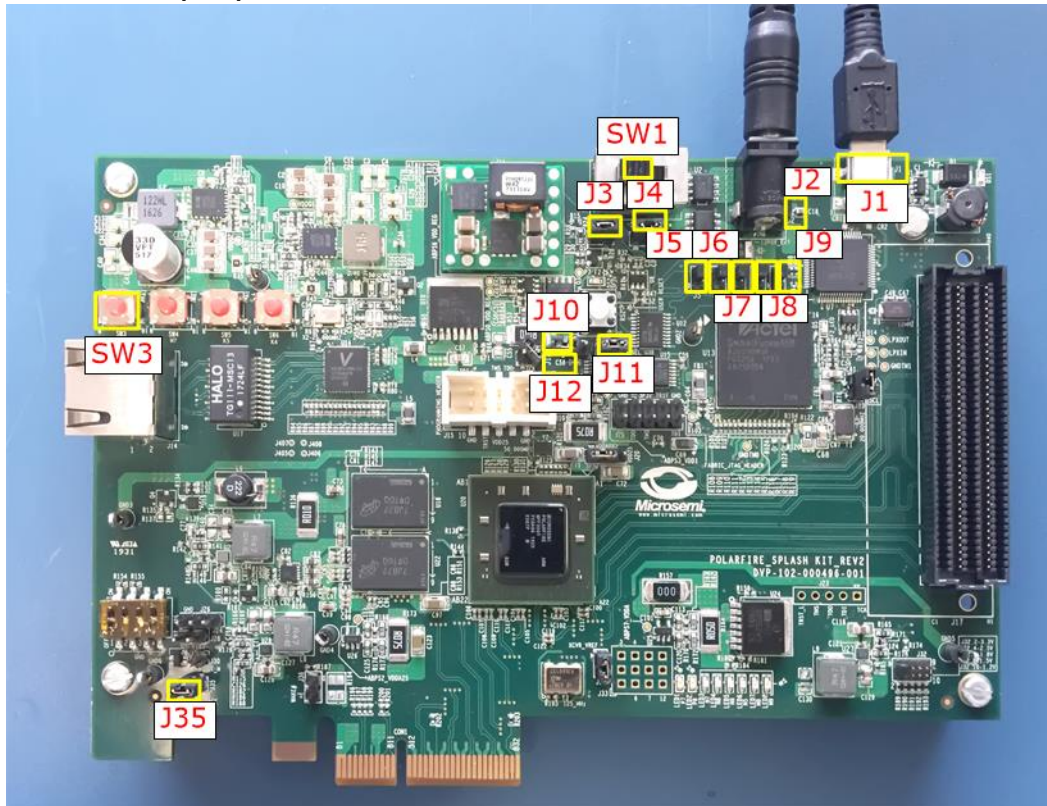
[Figure 21](#), page 20 shows the board setup of splash kit.



### 3.9 Export FlashPro Express Job

On the **Design Flow window**, double-click Export FlashPro Express Job. When the job file is successfully generated, a green tick mark appears as shown in [Figure 12](#), page 12

**Figure 21 • Board Setup—Splash Kit**



5. On the **Design Flow window**, double-click **Run PROGRAM Action**.

The device is successfully programmed and the onboard LEDs 4, 5, 6, 7, and 8 glow. A green tick mark appears next to **Run PROGRAM Action** as shown in [Figure 12](#), page 12.

To program the device using the .job file provided along with the demo design, see [Appendix 1: Programming the Device Using FlashPro Express](#), page 29.

## 4 Setting up the Serial Terminal Program - PuTTY

The user application receives system service commands on the serial terminal through the UART interface. This chapter describes how to set up the serial terminal program.

To Setup PuTTY, perform the following steps:

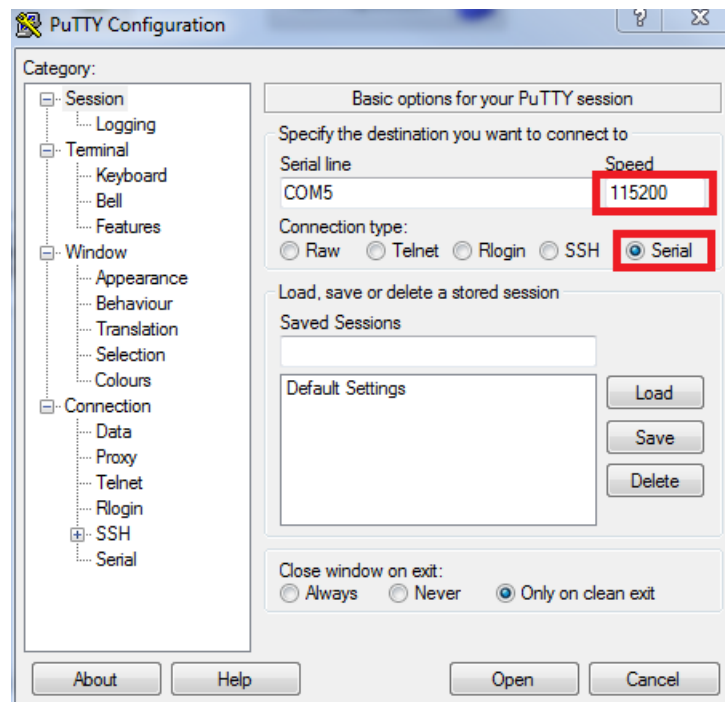
1. Connect the USB cable from the host PC to the **J5** (USB) port on the Evaluation board or **J1** port on the Splash board.
2. Connect the power supply cable to the **J9** connector on the Evaluation board or **J2** on Splash board.
3. Power on the board using the **SW3** slide switch for Evaluation board or **SW1** slide switch for the Splash board.
4. From the host PC, click **Start** and open **Device Manager** to note the second highest COM Port number and use that in the PuTTY configuration. In this example, COM Port 5 (**COM5**) is selected as shown in the following figure. COM Port-numbers may vary.

**Figure 22 • Finding the COM Port**



5. On the host PC, click **Start**, find and select PuTTY program.
6. Select **Serial** as the **Connection type** as shown in the following figure.

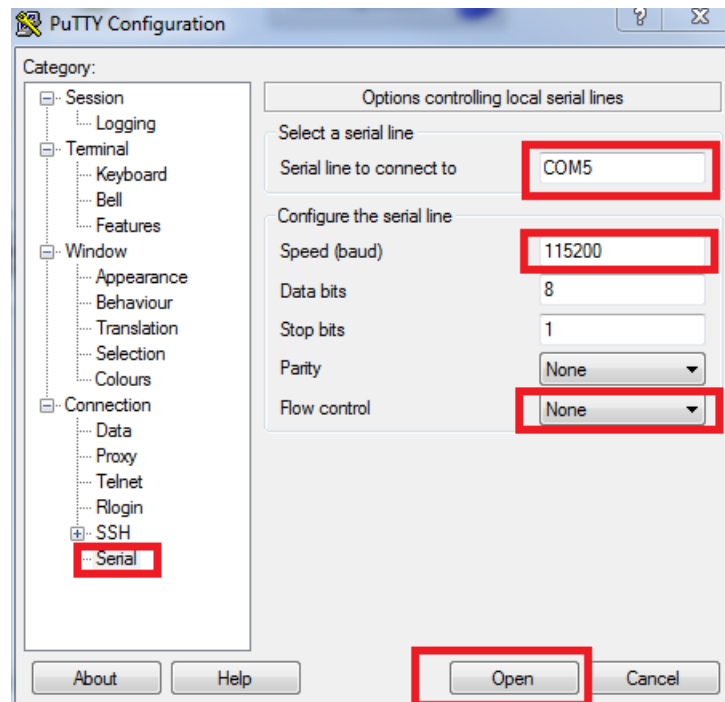
**Figure 23 • Select Serial as the Connection Type**



7. Set the **Serial line to connect to** COM port number noted in step 4.

8. Set the **Speed (baud)** to 115200 as shown in the following figure.

**Figure 24 • PuTTY Configuration**



9. Set the **Flow control** to **None** as shown in the preceding figure and click **Open**.

PuTTY opens successfully, and this completes the serial terminal emulation program setup. See [Running the Demo](#), page 23.

## 5 Running the Demo

This chapter describes how to run the system services demo using the serial terminal program (PuTTY). The prerequisite for the following procedure is to program the device and to set up the serial terminal. For more information on setting up the serial terminal, see [Setting up the Serial Terminal Program - PuTTY](#), page 21.

To run the demo, perform the following steps:

1. Power on the board using the **SW3** slide switch.

System services options are displayed on the PuTTY as shown in the following figure.

**Figure 25 • System Services Options**

```

**** PolarFire Device and Design system services Example ****
Notes: Return data from System controller is displayed byte-wise with LSB first
Input data is provided LSB first. Each ASCII character is one Nibble of data

Select Service:
1. Read Device Serial number
2. Read Device User-code
3. Read Device Design-info
4. Read Device certificate
5. Read Digest
6. Query security
7. Read debug information
8. Digital signature
9. Secure NUM services
a. PUF Emulation
b. Nonce

```

2. Enter 1 to select **Read Device Serial number**.

The 128-bit Device Serial Number (DSN) is displayed as shown in the following figure.

**Figure 26 • Device Serial Number**

```

-----
Device serial number:  278398138C7F3CDBC1A4AB3059CECAB5
-----

```

Each PolarFire FPGA device has a unique, publicly readable, 128-bit DSN. The DSN can be used in cryptographic protocols to uniquely identify the device. The DSN comprises two 64-bit fields: FSN and SNM.

**Factory Serial Number (FSN)**—The first (most significant) field is the FSN. It is a pseudo-random per-device unique value assigned during Microsemi's manufacturing test and persists for the lifetime of the device.

**Serial Number Modifier (SNM)**—The second component is the SNM. It is initialized during the factory test and is destroyed during the recoverable zeroization action. If the device is subsequently recovered, a new SNM is assigned such that each SNM generated for a given FSN, is unique.

**Note:** The system services main menu is displayed after the execution of any of the options.

3. Enter 2 to select **Read Device User-code**.

The 32-bit device USERCODE/Silicon signature is displayed as shown in the following figure.

**Figure 27 • Device User-code**

```

-----
32bit USERCODE/Silicon signature <MSB first>:  89ABCDEF
-----

```

This can be configured from Design flow->Program and Debug Design->Configure Programming Options.

4. Enter 3 to select **Read Device Design-info**.

As shown in [Figure 28](#), page 24, the device design information consists of:

- 256-bit user-defined Design ID
- 16-bit design version  
This can be configured from Design flow->Program and Debug Design->Configure Programming Options. In auto update programming, the current design version is compared with the available images in external SPI flash to initiate the auto update on power up.
- 16-bit design back-level  
This can be configured from Design flow->Program and Debug Design->Configure Security. When back level protection is enabled, the device can only be programmed if the target design version is more than the back level value.

**Figure 28 • Device Design Information**

```
-----  
Design ID:  
6F44746F700000000000000000000000  
00000000000000000000000000000000  
Design Version: 0100  
Design Back-Level: 0000  
-----
```

5. Enter 4 to select **Read Device certificate**.

The device supply chain assurance certificate is displayed as shown in [Figure 29](#), page 25.

For more information about decoding the device certificate, see [Appendix 2: Device Certificate Information](#), page 32.

**Figure 29 • Device Certificate**

[illegible]



6. Enter 5 to select **Read Digest**.

The 416 byte Digest contains the fabric digest, sNVM digest, and user key digests. The Digest protects data integrity. The following figure shows the 416 byte digest displayed. For more information about decoding the digest output, see [Appendix 5: Digest Information](#), page 36

**Figure 30 • Digest**

```
-----
Read Digest:
4AF18C0D5DE7AEB8C2063F93CD0FD40B
290340AC4B3F91844B9AD283EE57B5C2
DF42E8184C7CC2F71415E77FBB507984
56FA2FA5B26A2F61DDFE70B2F8F7AF90
CCDE013695FAB60B68F6999D3A5B5FBA
538D226C3B0E820E840CE673E5FBF063
223186F913839429B186AC19F222D712
2212B305ACAAE0725D96B9FB918F34BC
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
2EA9AB9198D1638007400CD2C3BEF1CC
745B864B76011A0E1BC52180AC6452D4
F5A5FD42D16A20302798EF6ED309979B
43003D2320D9F0E8EA9831A92759FB4B
00000000000000000000000000000000
00000000000000000000000000000000
2EA9AB9198D1638007400CD2C3BEF1CC
745B864B76011A0E1BC52180AC6452D4
F5A5FD42D16A20302798EF6ED309979B
43003D2320D9F0E8EA9831A92759FB4B
93BB6A72F664DE76EBC6E0E8AF8991CA
9D7B5DDFED02F2DF6195FF4F8C81A549
6BFAEAE10355BBD76FB7358971A803CF
2598CD4EEBB8A08533049D3DE4C04F32
-----
```

7. Enter 6 to select **Query Security**.

The non-volatile states of user security locks are displayed as shown in the following figure.

**Figure 31 • Security Locks Information**

```
-----
Security locks: 00000000000000000000
-----
```

The design does not include any security settings for device safety. Security locks can be configured from Design flow->Program and Debug Design->Configure Security. For more information about security locks, see [Appendix 3: Query Security](#), page 34.

**Note:** If security locks are not handled properly, the device can go into the locked state.

8. Enter 7 to select **Read debug information**.

The debug information is displayed as shown in the following figure.

**Figure 32 • Debug Information**

```
-----
Debug info:
000000A2280022002300CB09BC09C309
15074002CC1900000000000000000000
0102D42D01944FC11400E77F00000000
00012FA500000000722000001E020000
00DE013695FAB60B000000000
-----
```

In [Figure 32](#), page 26, the highlighted 4 bytes E7000000 (LSB first) indicates the number of times (in this example, 0x000000E7), the device was programmed (programming cycles). For more information about the Debug Info fields, see [Appendix 4: Debug Information](#), page 35.

9. Enter 8 to select **Digital Signature**.

The digital signature in both Raw and DER formats are displayed in the following figure.

**Figure 33 • Digital Signature**

```

-----
Digital Signature service:
48 byte hash value:
911601143D01D7E2676B0434FF45D7BB
5634E5CBF8BB6F685E5A2D7D038AB2F9
911601143D01D7E2676B0434FF45D7BB
Raw format:
Digital Signature service successful.
Output Digital Signature - Raw format:
2B04CA57A62DE3F6F9E910EBBE4BBD03
89F5DB43244AC2BBC51A4475ED95B007
73F5BB94828064851B7A6F1E08A33E31
AC3D3985F185FEC392CA68552F2199E
8F74C00B198ECF2ACD2378637F94F458
51FF5AF30983757CDC1F6123DE4699AD
DER format:
Digital Signature service successful.
Output Digital Signature - DER format:
3066023100A8ED918D910DB652980E4D
EB43CFD8BE29EF631B0D7421D57C72BA
F7E66AFF29D2F939D6C621BCE069AFCB
BAF31A1DF20231009F6A900B7511A4AE
84C72C61D11E3F0002F7EEF6C3BA7C82
7AF86FD3F83389C1005AD627AD99326C
21A8A7A589F1D978
-----

```

The digital signature service takes a user-supplied SHA384 hash and signs it with the device's private key. The application randomly generates the SHA384 hash value. The Digital Signature service sends the hash value to the System Controller. The Athena core runs the Elliptic Curve Digital Signature Algorithm (ECDSA) using the hash and the device private key to generate the signature.



10. Enter 9 to select **Secure NVM**.

When the sNVM page/ module address is entered (in this case, 0), the randomly generated 60 byte data is written to the specified sNVM page and read back, as shown in [Figure 34](#), page 28.

**Figure 34 • Secure NVM Services**

```
Secure NUM (sNUM) Functions
sNUM write format
Non-authenticated plain-text

Enter the sNUM page/module address <1 HEX byte. Page value range is from 0x01 to 0x83> and Press Enter key:
1 1

Input Data(60 Byte):
554D9036446F48C97E9A95C1124B9747
E55E699A554D9036446F48C97E9A95C1
124B9747E55E699A554D9036446F48C9
7E9A95C1124B9747E55E699A
Secure NUM write successful.

Data read from sNUM region:
554D9036446F48C97E9A95C1124B9747
E55E699A554D9036446F48C97E9A95C1
124B9747E55E699A554D9036446F48C9
7E9A95C1124B9747E55E699A00000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
001000600200000000000000000000
000000000000000000000000000000
000000000000000000000000000000
56501820802E2528AF397294554D9036
446F48C97E9A95C1124B9747E55E699A
0102D42D01944FC11400E77F0000000
00012FA5000000072200001E020000
00DE013695FAB60B00000000
```

11. Enter 'a' (without quotes) to select **PUF Emulation\_service**

The PUF emulation service provides a mechanism for authenticating a device, or for generating a pseudo random bit strings that can be used for different purposes. When this service is selected, the service by default accepts a 128-bit challenge and an 8-bit optype, and returns a 256-bit response unique to the challenge and the optype as shown in the following figure.

**Figure 35 • PUF Emulation Service**

```
The challenge OPTYPE range(0x0 to 0xFF): 85
16 byte challenge:
85AAFCC1E77A041F7F009C31B6F1CD10
PUF emulation service successful.Generated Response:
7120EB27A136CCCE385BA62E0A747E4A
D0EEDA6A74A1A9D4575B734499662DA1
```

12. Enter 'b' to select **Nonce service**.

The 32 byte nonce value is displayed as shown in the following figure. The nonce service provides the ability to strengthen the Deterministic Random Bit Generator (DRBG) of the Athena by providing an alternate entropy source to use as additional seed data in its DRBG functions.

**Figure 36 • Generated Nonce**

```
Generated Nonce:
FF5249B99A41CFB23CAF8E3A97E7C37
E1660B46AE820A8A8916CE964BA96C43
```

## 6 Appendix 1: Programming the Device Using FlashPro Express

This chapter describes how to program the PolarFire device with the Job programming file using a FlashPro programmer. The default location of the Job file are located at following location:

`mpf_dg0798_eval_df\Programming_Job\top.job`

and

`mpf_dg0798_splash_df\Programming_Job\top.job`

To program the PolarFire device using FlashPro Express, perform the following steps:

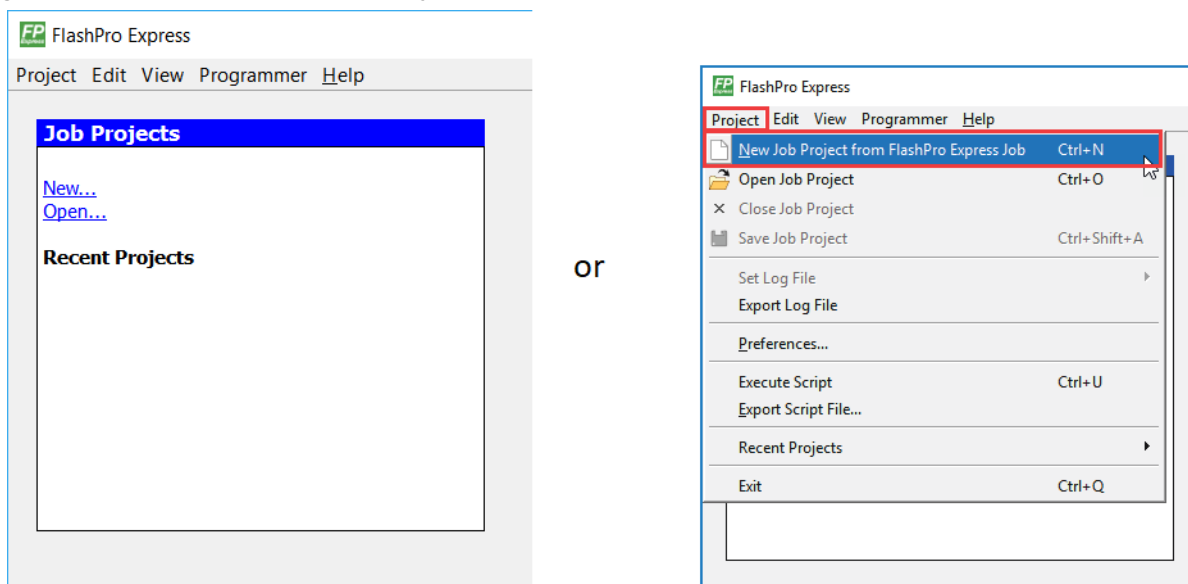
1. Ensure that the jumper settings on the board are the same as listed in [Table 8](#), page 18 and [Table 7](#), page 18.

**Note:** The power supply switch must be switched off while making the jumper connections.

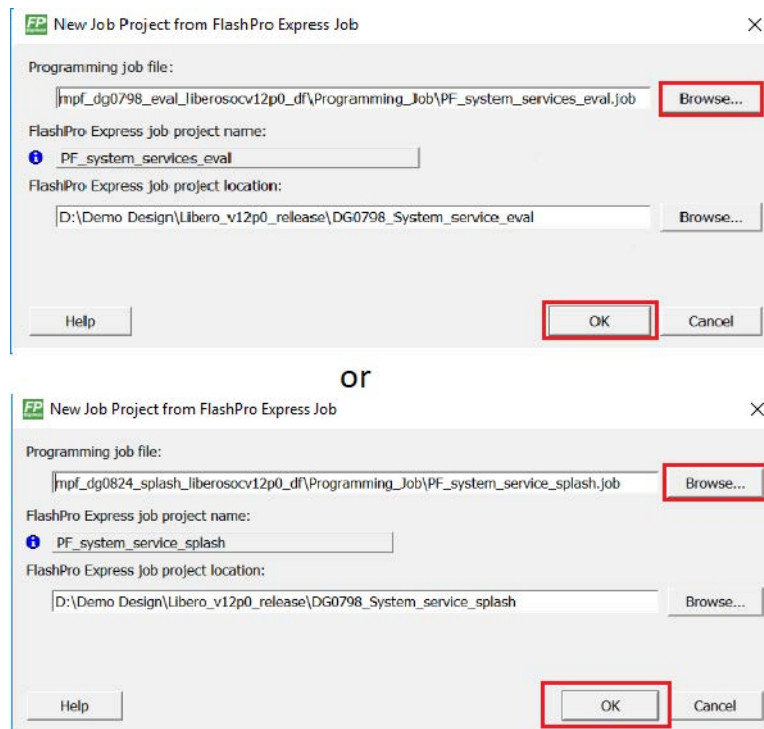
2. Connect the power supply cable to the **J9** connector on the board.
3. Connect the USB cable from the Host PC to the **J5** (FTDI port) on the board.
4. Power on the board using the **SW3** slide switch.
5. On the host PC, launch the FlashPro Express software.
6. To create a new job project, click **New** or

In the **Project** menu, select **New Job Project from FlashPro Express Job**, as shown in the following figure.

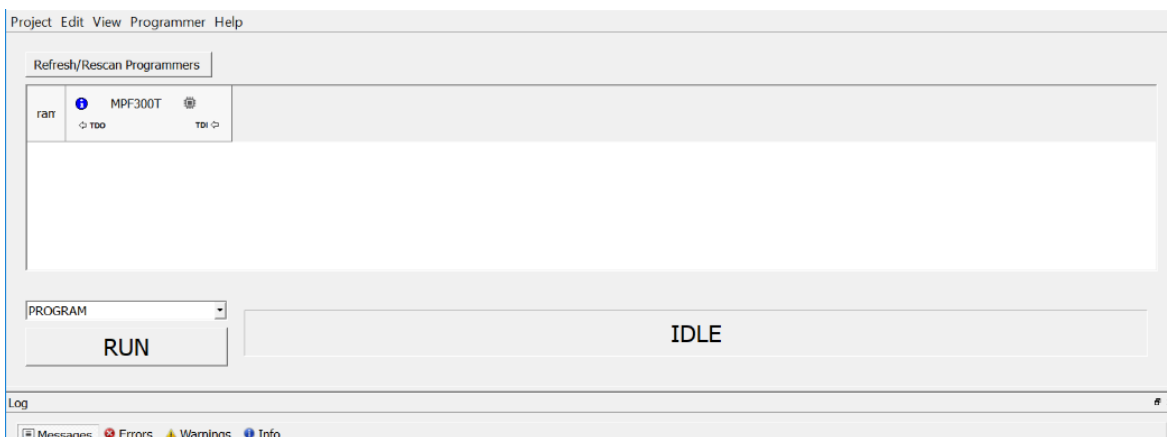
**Figure 37 • FlashPro Express Job Project**



7. Enter the following in the **New Job Project from FlashPro Express Job** dialog box:
  - Programming job file: Click **Browse**, and navigate to the location where the .job file is located and select the file. The default location is: `<download_folder>mpf_dg0798_eval_df\Programming_Job`. and `<download_folder>mpf_dg0798_splash_df\Programming_Job`
  - FlashPro Express job project location: Click **Browse** and navigate to the location where you want to save the project.

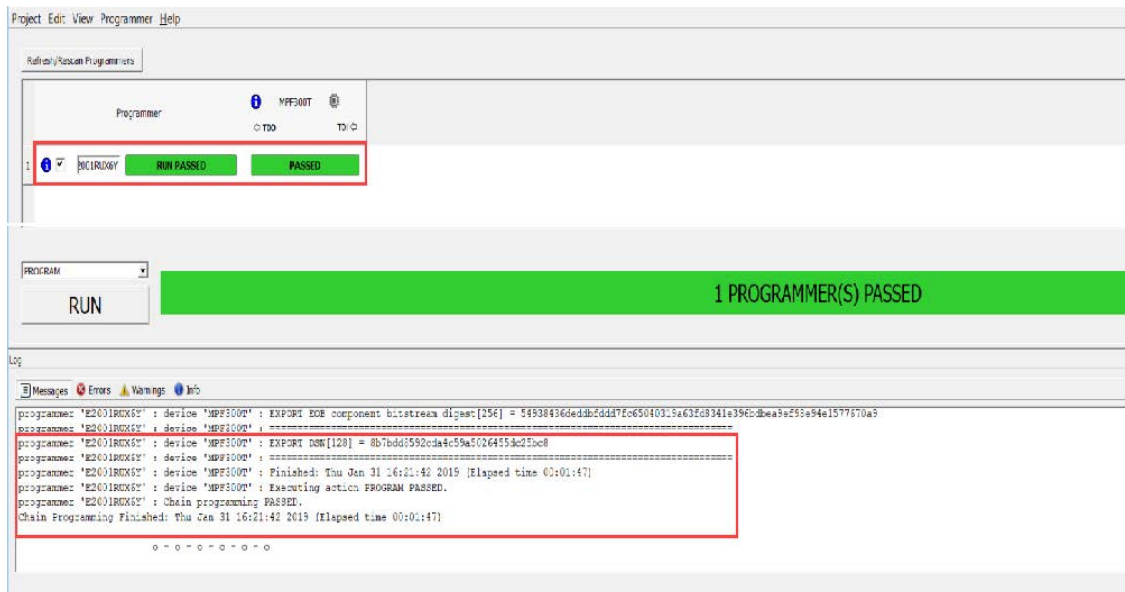
**Figure 38 • New Job Project from FlashPro Express Job**

8. Click **OK**. The required programming file is selected and ready to be programmed in the device.
9. The FlashPro Express window appears as shown in the following figure. Confirm that a programmer number appears in the Programmer field. If it does not, confirm the board connections and click **Refresh/Rescan** Programmers.

**Figure 39 • Programming the Device**

10. Click **RUN**. When the device is programmed successfully, a **RUN PASSED** status is displayed as shown in the following figure.

**Figure 40 • FlashPro Express—RUN PASSED**



11. Close **FlashPro Express** or in the **Project** tab, click **Exit**.

## 7 Appendix 2: Device Certificate Information

The Device Certificate is a 1024 byte Microsemi-signed X-509 certificate programmed during manufacturing. The certificate is used to guarantee the authenticity of a device and its characteristics.

The following table lists the main fields of the device certificate.

**Table 9 • Device Certificate Fields (1024 bytes)**

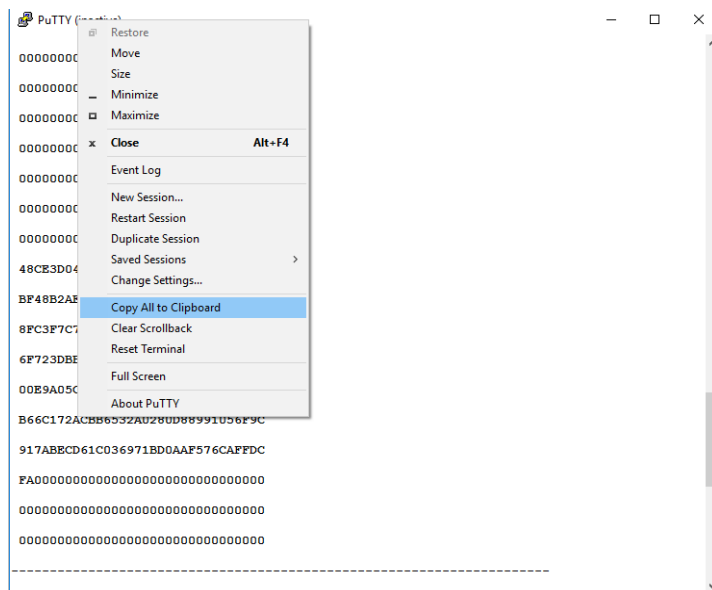
Offset (Byte)	Length (Bytes)	Data
4	854	Signed region of certificate
234	120	Device Public Key
368	16	DSN

The device certificate is encoded in the ASN.1 format. To view the content, the certificate must be decoded to a user readable format using the online JAVA tool: <http://lapo.it/asn1js/#>.

For decoding a certificate, perform the following steps:

1. Right click PuTTY, select **Copy All to Clipboard**, and paste the same to notepad as shown in the following figure.

**Figure 41 • Copy Device Certificate**



2. Copy the 1024 bytes of device certificate from notepad to ASN.1 decoder as shown in the following figure and click **decode** button.

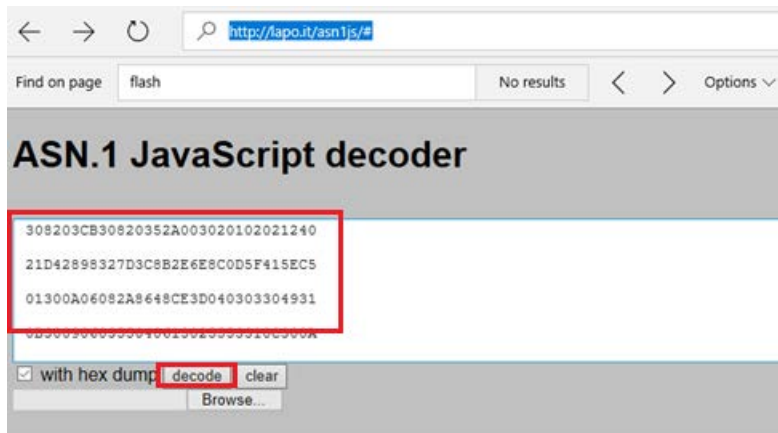
**Note:** The sample device certificate (1024 bytes) is provided at:

For Evaluation kit `mpf_dg0798_eval_df\Device_Certificate\sample.txt`

or

For Splash kit

`mpf_dg0798_splash_df\Device_Certificate\sample.txt`

**Figure 42 • Certificate Decoding Using Java Script**

3. The web page displays all the fields in certificate as shown in the following figure.

**Figure 43 • Decoded Certificate**

## 8 Appendix 3: Query Security

The following table lists each security lock bit and its features.

**Table 10 • Security Locks Fields**

Byte	Bit	Lock	Description
0			
	0	UL_DEBUG	Debug instructions disabled
	1	UL_SNVM_DEBUG	sNVM debug disabled
	2	UL_LIVEPROBE	Live probes disabled
	3	UJTAG_DISABLE	User JTAG interface disabled
	4	JTAG_BS_DISABLE	JTAG boundary scan disabled
	5	UL_TVS_MONITOR	External access to system Temperature and Voltage Sensor (TVS) disabled
	6	JTAG_MONITOR	JTAG fabric monitor enabled
	7	JTAG_TAP	JTAG TAP disabled
1			
	0	UL_PLAINTEXT	Plain text passcode unlock disabled
	1	UL_FAB_PROTECT	Fabric erase/write disabled
	2	UL_EXT_DIGEST	External digest check disable
	3	UL_VERSION	Replay protection enabled
	4	UL_FACT_UNLOCK	Factory test disabled
	5	UL_IAP	IAP disabled
	6	UL_EXT_ZEROIZE	External zeroization disabled
	7	SPI_SLAVE_DISABLE	SPI port disabled
2-8		Reserved	Reserved

## 9 Appendix 4: Debug Information

The following table lists the debug information bit fields.

**Table 11 • Debug Info Fields**

Byte Offset	Size (Bytes)	Parameter	Description
0	32	RESERVED	
32	4	TOOL_INFO	<ul style="list-style-type: none"> <li>Reflects the TOOL_INFO passed in during ISC_ENABLE prior to programming</li> <li>IAP sets this to 0</li> </ul>
36	1	TOOL_TYPE	Tool type used to program device 1 = JTAG, 2 = IAP, and 3 = SPI_SLAVE
37	4	RESERVED	
41	7	RESERVED	
48	1	UIC_STATUS	Design initialization status
49	1	UIC_SOURCE_TYPE	Design initialization Data source type when execution finished or halted
50	2	RESERVED	
52	4	UIC_START_ADDRESS	Design initialization Data source address when execution finished or halted
56	4	UIC_INSTR_ADDRESS	Design initialization Data instruction count from the start of Design initialization execution
60	4	CYCLECOUNT	Programming cycle count
64	1	IAP_ERROR_CODE	IAP error information
65	7	RESERVED	
72	4	IAP_LOCATION	External SPI flash memory address that was used during IAP



## 10 Appendix 5: Digest Information

The following table lists the digest information bit fields.

**Table 12 • Digest Information Bit Fields**

Offset (byte)	Size (bytes)	Value	Note
0	32	CFD	Fabric digest
32	32	CCDIGEST	Fabric Configuration segment digest
64	32	SNVMDIGEST	sNVM Digest
96	32	ULDIGEST	User lock segment
128	32	UKDIGEST0	User Key Digest 0 in User Key segment (includes SRAM PUF activation code and device Integrity bit)
160	32	UKDIGEST1	User Key Digest 1 in User Key segment
192	32	UKDIGEST2	User Key Digest 2 in User Key segment (UPK1)
224	32	UKDIGEST3	User Key Digest 3 in User Key segment (UEK1)
256	32	UKDIGEST4	User Key Digest 4 in User Key segment (DPK)
288	32	UKDIGEST5	User Key Digest 5 in User Key segment (UPK2)
320	32	UKDIGEST6	User Key Digest 6 in User Key segment (UEK2)
352	32	UPDIGEST	User Permanent lock (UPERM) segment
384	32	FDIGEST	Digest for Factory Key Segments
Total	416 bytes		

# 11 Appendix 6: 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_dg0798_eval_df/TCL_Scripts/readme.txt` or `mpf_dg0798_splash_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.

## 12 Appendix 7: References

---

This section lists documents that provide more information about system services and other IP cores used to build the system services.

- For more information on Design and Data Security Services, see [UG0753: PolarFire FPGA Security User Guide](#).
- For more information about the CoreJTAGDEBUG IP core, see [CoreJTAGDebug\\_HB.pdf](#) from **Libero->Catalog**.
- For more information about the MIV\_RV32 IP core, see [MIV\\_RV32 Handbook](#) from the Libero SoC Catalog.
- For more information about the CoreUARTapb IP core, see [CoreUARTapb\\_HB.pdf](#).
- For more information about the CoreAPB3 IP core, see [CoreAPB3\\_HB.pdf](#).
- For more information about the CoreGPIO IP core, see [CoreGPIO\\_HB.pdf](#).
- For more information about the PolarFire initialization monitor, see [UG0725: PolarFire FPGA Device Power-Up and Resets User Guide](#).
- For more information about how to build a Mi-V processor subsystem for PolarFire devices, see [TU0775: PolarFire FPGA: Building a Mi-V Processor Subsystem Tutorial](#).
- For more information about the PF\_CCC IP core, see [UG0684: PolarFire FPGA Clocking Resources User Guide](#).
- For more information about Libero, ModelSim, and Synplify, see [Microsemi Libero SoC PolarFire webpage](#).