AC464
Application Note
PolarFire FPGA: Implementing Data Security Using User Cryptoprocessor

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
www.microsemi.com

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.

Contents

Figures

Tables

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 10.0

Added Appendix 3: Running the TCL Script, page 34.

1.2 Revision 9.0

In this revision, information about Prerequisites was updated.

1.3 Revision 8.0

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.
- Added information about enabling protection for SPI Flash client. For more information, refer to TCM Initialization from SPI Flash, page 15.

1.4 Revision 7.0

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

- Information about how to unlock the sNVM pages was added. See Adding a Dummy Client to Unlock the sNVM Pages, page 17.
- Updated the document for Libero SoC v12.1.

1.5 Revision 6.0

Updated the document for Libero SoC v12.0.

1.6 Revision 5.0

The document was updated for Libero SoC PolarFire v2.3 release.

1.7 Revision 4.0

The document was updated for Libero SoC PolarFire v2.2 release.

1.8 Revision 3.0

The document was updated for Libero SoC PolarFire v2.1 release.

1.9 Revision 2.0

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

- Information about initializing the SRAM component with the HEX file was added, see TCM Initialization from SPI Flash, page 15.
- The document was updated to include features and enhancements introduced in the Libero SoC PolarFire v2.0 release.

1.10 Revision 1.0

The first publication of this document.

2 Implementing Data Security Using User Cryptoprocessor

PolarFire® FPGAs represent the industry's most advanced security programmable FPGAs.

Data security protects application data—stored, communicated, or computed at run-time—from being copied, altered, or corrupted. PolarFire devices have a dedicated crypto processor, referred as User Cryptoprocessor, for data security applications.

This application note describes User Cryptoprocessor features, how to build a Libero and SoftConsole project to perform cryptographic operations using the User Cryptoprocessor as a coprocessor to a host general purpose processor.

2.1 User Cryptoprocessor

The User Cryptoprocessor is an Athena TeraFire® EXP-F5200B cryptography microprocessor. It provides complete support for the Commercial National Security Algorithm (CNSA) Suite and beyond, and includes Side-Channel Analysis (SCA) resistant cryptography using patented leakage reduction countermeasures. These countermeasures provide strong resistance against SCA attacks such as Differential Power Analysis (DPA) and Simple Power Analysis (SPA). The User Cryptoprocessor is available in PolarFire "S" grade devices.

The User Cryptoprocessor supports numerous cryptographic algorithms, including the following:

- AES with 128-, 192-, and 256-bit key sizes in ECB, CBC, CFB, OFB, CTR, and GCM modes
- AES key wrap and unwrap
- SHA1, SHA2-224, SHA2-256, SHA2-384, and SHA2-512
- AES-CMAC and AES-GMAC
- HMAC-SHA
- True random number generation (non-deterministic random bit generator plus NIST SP800-90A deterministic random bit generator)
- RSA, DSA, and modular exponentiation (Diffie-Hellman) with key sizes up to 4096-bits
- EC key pair generation, point validation, point multiplication (EC Diffie-Hellman), and ECDSA for
  - NIST P-curves: P-192, P-224, P-256, P-384, and P-521
  - Brainpool curves: P-256, P-384, and P-512
- Key-tree function

The User Cryptoprocessor is a hard block in PolarFire FPGAs and its maximum operating frequency is 189 MHz. It is accessible to a soft processor in the FPGA fabric through the AHB-Lite slave interface for control and primary data input and output. The User Cryptoprocessor has built-in DMA to offload the main processor from doing data transfers between the User Cryptoprocessor and the user memory. The DMA functionality is accessible through an AMBA AHB-Lite master interface. The Libero SoC design suite provides a PF_CRYPTO macro in Catalog, which must be integrated with the user design to use the User Cryptoprocessor. The following figure shows the input and output ports of the PF_CRYPTO macro.

Figure 1 • **PF_CRYPTO Macro**

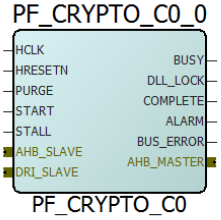
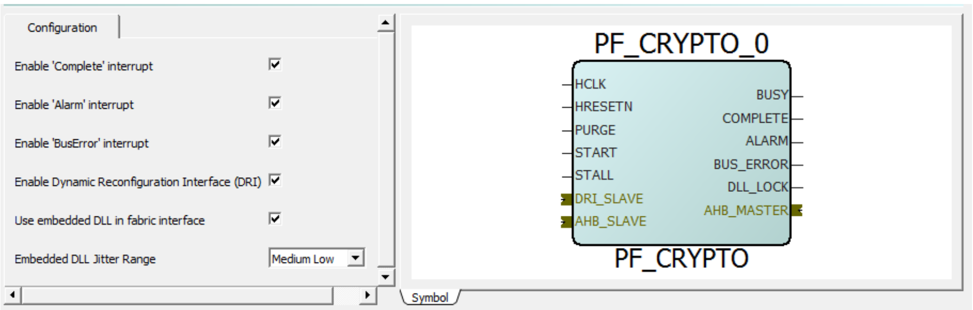


Figure 2, page 3 shows the crypto configurator used to enable Complete interrupt, Alarm interrupt, BusError interrupt, and DRI. If the frequency of the crypto block is greater than or equal to 125 MHz, select the Use embedded DLL in the fabric interface option for removing clock insertion delay. The Embedded DLL Jitter Range can be set to Low, Medium_Low, Medium_High, or High according to the crypto clock jitter specification. For more information, see *DS0141: PolarFire FPGA Datasheet* for embedded DLL jitter tolerance ranges.

Figure 2 • **Crypto Configurator**



Enable 'Complete' interrupt ☑

Enable 'Alarm' interrupt ☑

Enable 'BusError' interrupt ☑

Enable Dynamic Reconfiguration Interface (DRI) ☑

Use embedded DLL in fabric interface ☑

Embedded DLL Jitter Range: Medium Low

The following table lists and describes the PF_CRYPTO ports. The control and status signals initiate action and obtain status. Corresponding control and status signals are also accessible through the dynamic reconfiguration interface (DRI). Contact Microsemi tech-support for information on how to use DRI.

Table 1 • **PF_CRYPTO Port List**

Port Name	Direction	Description
AHB_SLAVE		AHB-Lite slave interface.
AHB_MASTER		AHB-Lite master interface.
DRI_SLAVE		Control and status signals are accessible through the DRI.
HCLK	Input	AHB bus clock.
HRESETN	Input	AHB bus reset. Asserts the functional reset of the User Cryptoprocessor block and zeroizes all the internal RAM and registers as PURGE signal.
PURGE	Input	When the signal is set to '1', it initializes the Zeroization of User Cryptoprocessor internal RAM and registers. For normal operation, this signal must be tied low. The PURGE input is level sensitive, and if the PURGE pin is still asserted when a purge operation completes, another purge operation is initiated.
START	Input	External execution initiation input when the User Cryptoprocessor operates in the standalone configuration without a host processor connected to the bus interface. Asserting the START signal causes the User Cryptoprocessor to initiate execution. During execution, the status of the User Cryptoprocessor is reflected by the BUSY and DLL_LOCK ports. This signal must be tied low when the User Cryptoprocessor is used as a coprocessor.

Table 1 • **PF_CRYPTO Port List *(continued)***

Port Name	Direction	Description
STALL	Input	Stalls the User Cryptoprocessor for a clock cycle, to introduce variance in the external signatures. The STALL input is expected to be generated by a LFSR circuit in the fabric and asserted randomly for a single cycle to achieve the required stall rates. The STALL input must not be asserted until at least three clock cycles after the HRESETN is de-asserted and the DLL has indicated LOCK for three cycles.
BUSY	Output	Execution status signal.
COMPLETE	Output	Asserted to indicate that the User Cryptoprocessor has completed an operation. This signal can be connected to the host microprocessor as an interrupt request signal, enabling the User Cryptoprocessor to interrupt the processor when it completes an operation.
ALARM	Output	Asserted to indicate an uncorrectable memory error condition. An uncorrectable memory error causes the crypto core to perform a reset and purge. This reset terminates any in-progress operation. For most CAL operations, the CALPKTrfRes() function is used to complete the operation and generates a hardware fault code in the event of an alarm.
BUS_ERROR	Output	Asserted when a HRESP response error is detected by the User Cryptoprocessor AHB master. Once set, a reset is required to clear.
DLL_LOCK	Output	DLL lock status.

Microsemi provides an Athena TeraFire Cryptographic Applications Library (CAL) to access User Cryptoprocessor functions. TeraFire CAL is a C language library that provides functions to access symmetric key, elliptic curve, public key, hash, random number generation, and message authentication code algorithms. It is available for download in the Microsemi Firmware Catalog software. The user application running on the main processor must include CAL APIs to perform the cryptographic operations on the User Cryptoprocessor.

For information about User Cryptoprocessor, supported cryptographic functions and their CAL API descriptions, email *FPGA_marketing@microchip.com* to request Athena TeraFire Cryptographic Algorithm Library (CAL) Users Guide.

2.2 Design Requirements

The following table lists the hardware and software required to perform cryptographic operations using the User Cryptoprocessor.

Table 2 • **Resource Requirements**

Requirement	Version
Host PC Operating system	Windows 7, 8.1, or 10
Hardware	
PolarFire Evaluation Kit (MPF300TS-1FCG1152)	Rev D or later
Software	
Libero SoC[1]	See the readme.txt file provided in the design files for the software versions used with this reference design.
SoftConsole	
Tera Term	

[1] Libero Evaluation, Gold, Platinum, or Standalone license is required to use PolarFire ‘S’ grade devices.

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

Email *FPGA_marketing@microchip.com* to request the project design files. This design is targeted for PolarFire Evaluation kit only.

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

Note: The latest versions of ModelSim, Synplify Pro, and FTDI drivers are included in the Libero SoC installation package.

2.4 Design Description

Microsemi offers a freely available RISC-V processor IP core called Mi-V and software tool chain to support Mi-V processor-based designs. In this reference design, a Mi-V soft processor core is used as the main processor and the User Cryptoprocessor as the coprocessor.

RISC-V, a standard open Instruction Set Architecture (ISA) under the governance of the RISC-V Foundation, offers numerous benefits, including enabling the open source community to test and improve cores at a faster pace than closed ISAs.

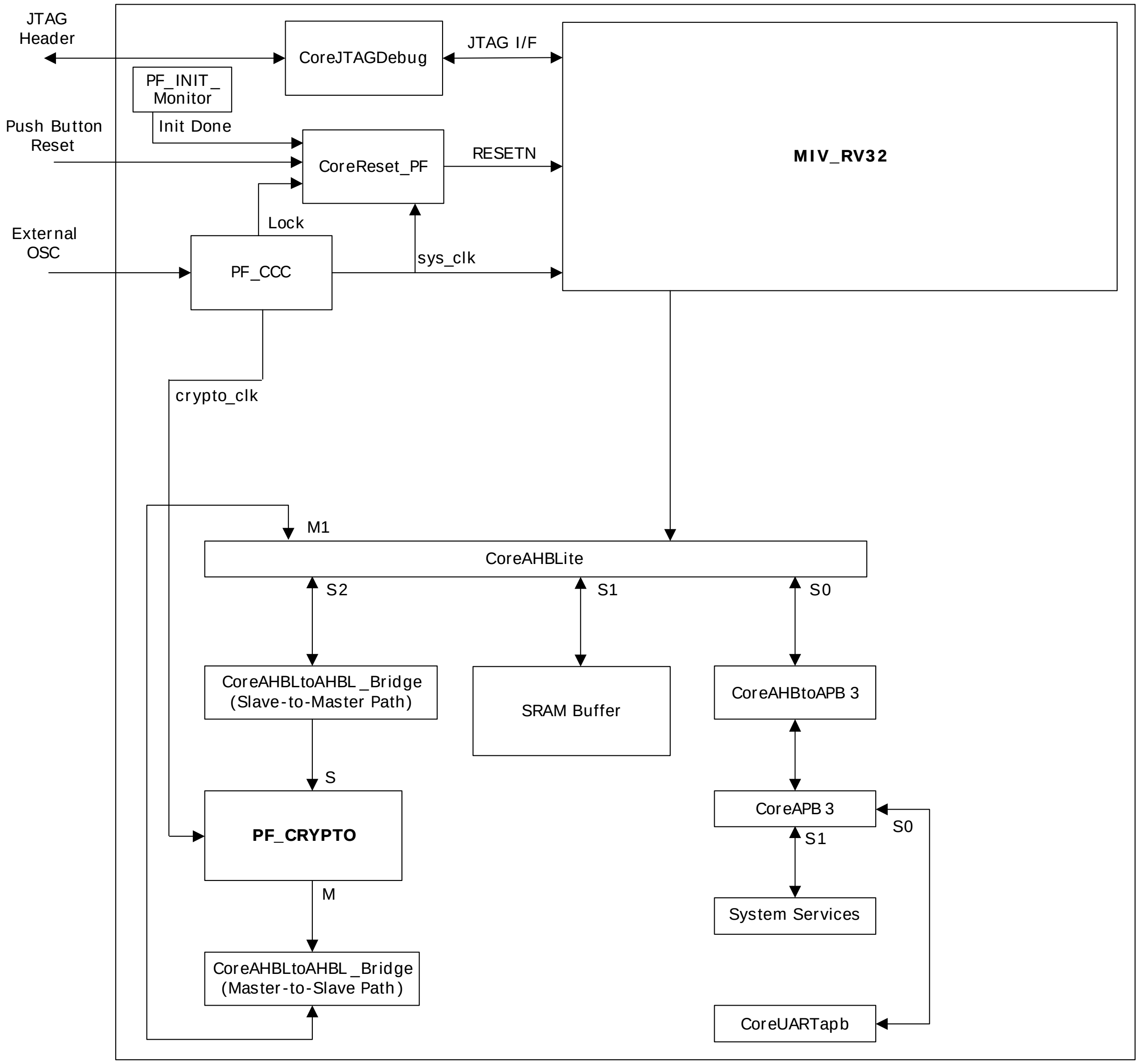
The Libero design provided with this application note shows how to integrate the User Cryptoprocessor in a Mi-V processor subsystem. The SoftConsole project shows how to integrate and build a TeraFire CAL driver into a Mi-V processor application project. A similar process can be used to integrate the User Cryptoprocessor and its CAL driver into other general purpose processor subsystem environments.

The Mi-V application provided with the reference design demonstrates the Advanced Encryption Standard (AES) algorithm features of the User Cryptoprocessor. It provides a user interface on the host PC using the UART. The user can download and run the other User Crypto sample projects available in the Firmware catalog to explore using the User Cryptoprocessor cryptographic algorithms.

Each PolarFire FPGA has 56 KB of secure non-volatile memory (sNVM), which can be used for storing cryptographic keys. The sNVM pages are accessible through system services for read/write. The reference design integrates CoreSysServices_PF IP for sNVM read/write.

The following figure shows a block diagram of the reference design.

Figure 3 • **Reference Design Block Diagram**



2.4.1 Clocking Structure

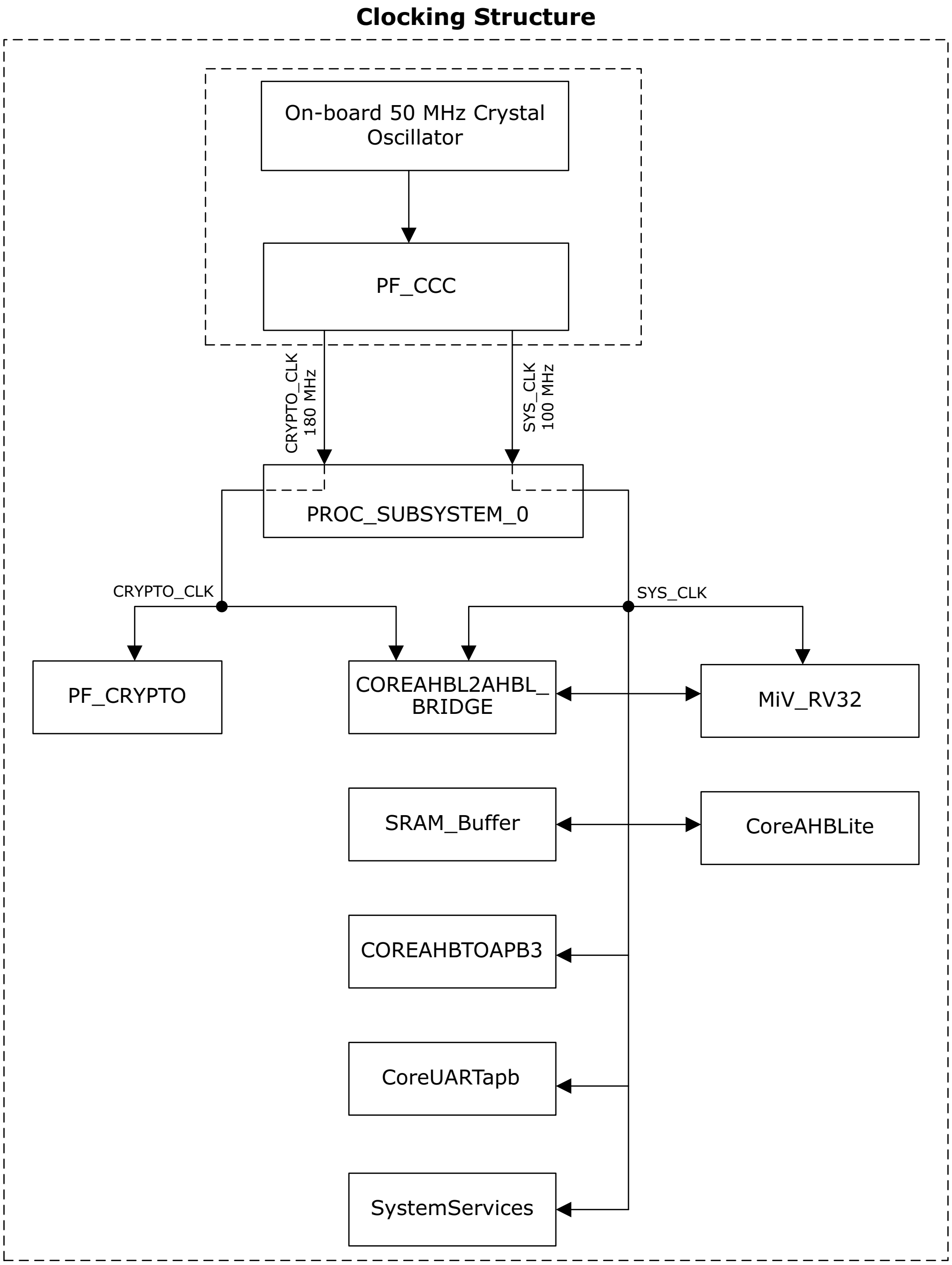
In the reference design, there is two clock domain. The on-board 50 MHz crystal oscillator is connected to the PF_CCC block which generates 100 MHz and 180 MHz clocks.

The 180 MHz crypto clock is connected to PF_CRYPTO and COREAHBL2AHBL_BRIDGE blocks.

The 100 MHz system clock is connected to COREAHBL2AHBL_BRIDGE, Mi_V_RV32_AHB, SRAM_Buffer, CoreAHBLite, COREAHBTOAPB3, System Services, and CoreUARTapb blocks.

The following figure shows the clocking structure.

Figure 4 • **Clocking Structure**



2.4.2 Hardware Implementation

To build a Mi-V subsystem for PolarFire FPGAs, use the Libero SoC design suite to create an FPGA design using the Mi-V processor core and peripheral cores. The following table lists the IP cores used in the reference design.

Table 3 • **IP Cores Used in the Reference Design**

IP Core	Description
MIV_RV32	Mi-V soft processor
CoreJTAGDEBUG	Facilitates the connection of Joint Test Action Group (JTAG) compatible soft core processors to the JTAG header for debugging. It provides fabric access to the JTAG interface using UJTAG macro.
PF_INIT_monitor	System Controller uses this macro to check the status of device initialization. The device initialization includes SRAM initialization from µPROM/sNVM/SPI Flash. The DEVICE_INIT_DONE signal is used as a reset.
CoreAHBLite	Multi-master AHB-Lite bus
COREAHBL2AHBL_BRIDGE	AHB to AHB bridge connects two AHB-Lite buses clocked by asynchronous clocks. This is required because the User Cryptoprocessor clock is different than the rest of the Mi-V processor subsystem clock.
PF_CRYPTO	Macro to access hard User Cryptoprocessor
PF_SRAM_AHBL_AXI	PolarFire LSRAM. Used as system memory for Mi-V processor.
CoreAHBtoAPB3	Bridge between AHB master and APB slave.
CoreUARTapb	UART Controller with APB interface
PF_CCC	Macro to access PolarFire CCC block. It is used to synthesize 100 MHz and 180 MHz clock frequencies from the CCC with an on-board 50 MHz reference clock.
CoreSysServices_PF	The CoreSysServices_PF provides access to the System Services supported by the PolarFire device. These are System Controller actions initiated from the user design using the CoreSysServices.

Configure and connect the IP cores listed in Table 3, page 8, then run the Libero design flow to create a programming file. 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*. This Mi-V processor subsystem can be extended to integrate a User Cryptoprocessor into the system.

The MIV_RV32 processor core comprises of an instruction fetch unit, an execution pipeline, and a data memory system. In the Mi-V processor memory map, the 0x8000_0000 to 0x8000_FFFF range is defined for the TCM memory interface, and the 0x6000_0000 to 0x6FFF_FFFF range is defined for the AHBLite I/O interface. The MIV_RV32 processor's reset vector address is set to 0x80000000. The processor’s reset vector address is configurable. The MIV_RV32's reset is an active-low signal, which must be de-asserted in sync with the system clock through a reset synchronizer. For more information about the MIV_RV32 core, see *MIV_RV32 Handbook* from Libero Catalog.

The MIV_RV32 processor accesses the application execution memory using the TCM options. The MiV_RV32 processor is configured to provide a 64 KB memory slot beginning at address 0x8000_0000.

The MIV_RV32 processor directs the data transactions between addresses 0x60000000 and 0x6FFF_FFFF to the AHBL interface. The AHBL interface is connected to the CoreAHBLite_0 bus to communicate with peripherals connected to its slave slots. The CoreAHBLite_0 bus instance is configured to provide 16 slave slots, each of size 64 KB. The sixteen 64 KB slots consume a total address space of 16*64*1024 = 2^20 bytes and can be addressed using a 20-bit address bus. The CoreAHBLite_0 bus, using only the lower 20-bits of the MMIO bus address and maps the connected peripherals within the address range. The User Cryptoprocessor, UART peripheral, System Services, and LSRAM memory are connected to the CoreAHBLite_0 bus.

Note: To initialize the TCM in PolarFire using the system controller, a local parameter **l_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 table summarizes the memory map of the reference design.

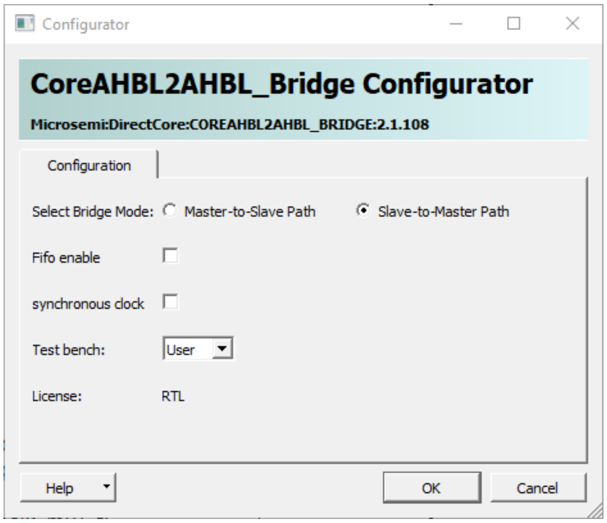
Table 4 • **System Memory Map**

Component	Description	Memory Map
PF_CRYPTO_0	User Cryptoprocessor	0x62000000 to 0x6200FFFF
SystemServices_0	System Services	0x60001000 to 0x60001FFF
CoreUARTapb_0	UART peripheral	0x60000000 to 0x60000FFF
RAM_Buffer_0	Memory buffer for User Cryptoprocessor	0x61000000 to 0x61FFFFFF

In this reference design, the User Cryptoprocessor clock (crypto_clk) is configured to operate at 180 MHz and the clock for the rest of the Mi-V subsystem (sys_clk) operates at 100 MHz. This reference design uses the CoreAHBL2AHBL_Bridge IP to provide clock domain crossing between sys_clk and crypto_clk.

The CoreAHBL2AHBL_Bridge IP functions as a bridge between the AHB master and AHB slave where master and slave operate in two clock domains that are asynchronous in nature. The following figure shows the CoreAHBL2AHBL_Bridge IP Configurator. This IP can be configured by setting the **Select bridge Mode** option to either **Master-to-Slave Path** or **Slave-to-Master Path**.

Figure 5 • **CoreAHBL2AHBL_Bridge Configurator**



Configurator

CoreAHBL2AHBL_Bridge Configurator

Microsemi:DirectCore:COREAHBL2AHBL_BRIDGE:2.1.108

Configuration

Select Bridge Mode: ○ Master-to-Slave Path ● Slave-to-Master Path

Fifo enable □

synchronous clock □

Test bench: User

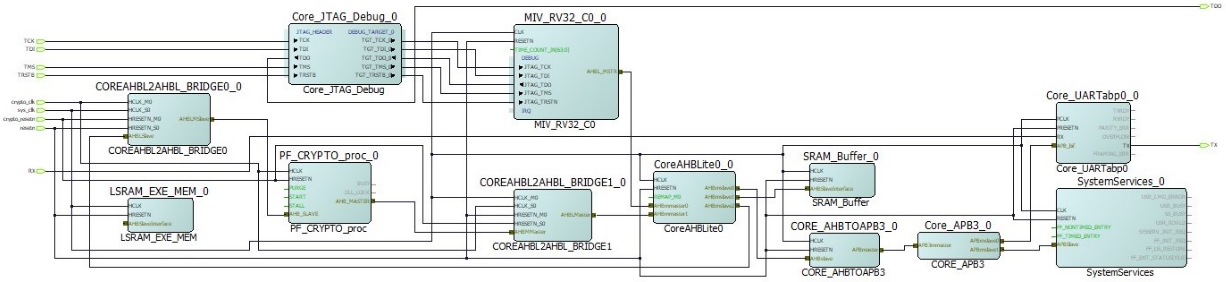
License: RTL

The CoreAHBL2AHBL_Bridge_0 is configured in the Slave-to-Master path to connect the User Cryptoprocessor to the Mi-V processors peripheral MMIO bus for control and primary data input and output. In this configuration, the sys_clk must be connected to the bridges slave interface clock (hclk_s0) and the crypto_clk must be connected to the bridge's master interface clock (hclk_m0).

The CoreAHBL2AHBL_Bridge_1 is configured in the Master-to-Slave path to connect the User Cryptoprocessors AHB master port to the Mi-V processors peripheral bus for DMA transactions. In this configuration, the sys_clk must be connected to the bridges master interface clock (hclk_m0) and the crypto_clk must be connected to the bridges slave interface clock (hclk_s0).

The following figure shows the SmartDesign view of the Mi-V processor subsystem with a User Cryptoprocessor.

Figure 6 • **Mi-V Processor Subsystem with User Cryptoprocessor**

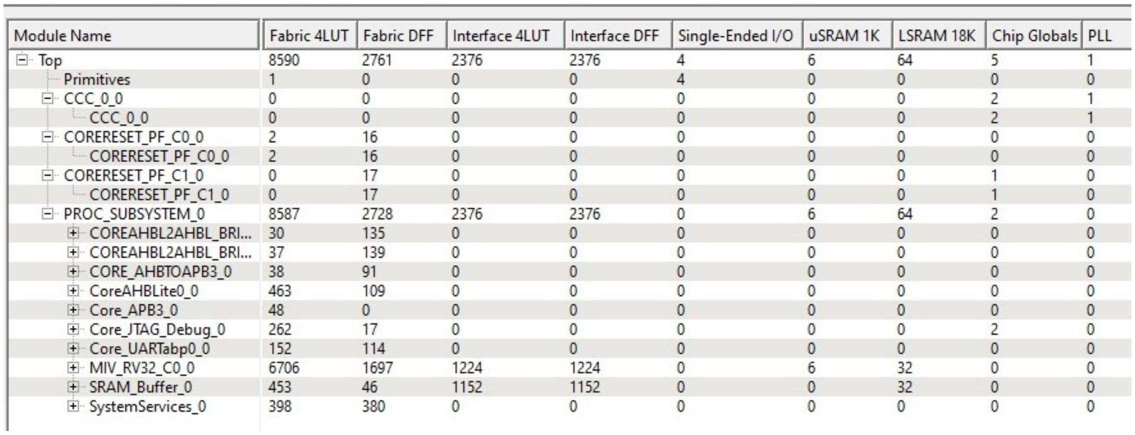


For more details on component configurations and connections, see the provided Libero project.

2.4.2.1 FPGA Resource Utilization

The following figures show the reference design resource utilization report under Synthesize in Design Flow window.

Figure 7 • **Design Resource Utilization—Evaluation Kit**



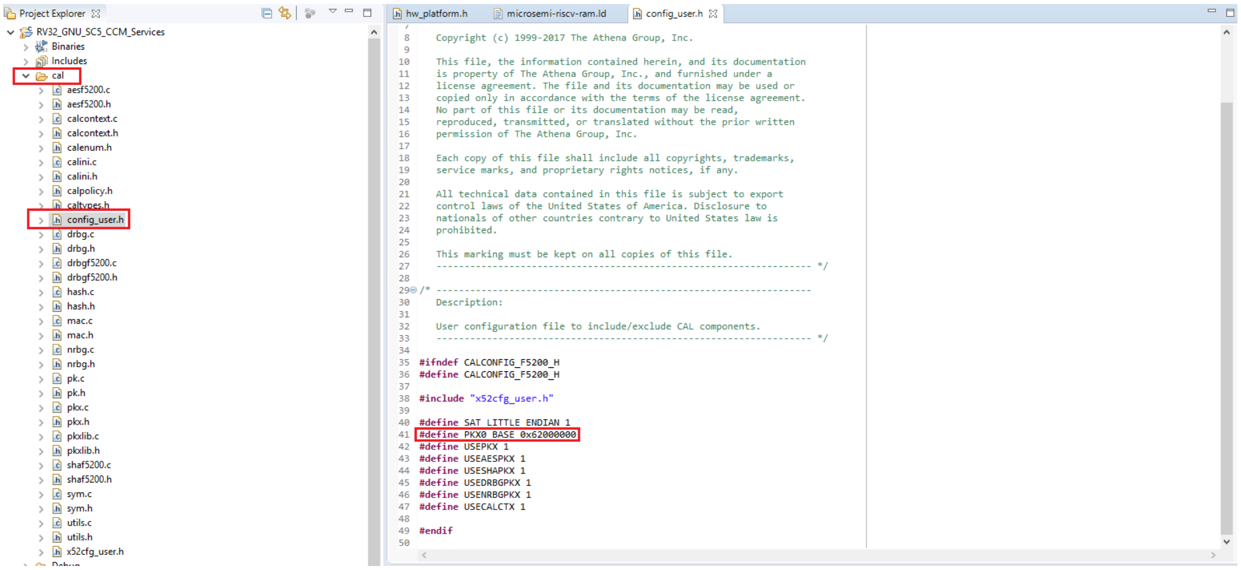
Module Name	Fabric 4LUT	Fabric DFF	Interface 4LUT	Interface DFF	Single-Ended I/O	uSRAM 1K	LSRAM 18K	Chip Globals	PLL
⊟ Top	8590	2761	2376	2376	4	6	64	5	1
Primitives	1	0	0	0	4	0	0	0	0
⊟ CCC_0_0	0	0	0	0	0	0	0	2	1
CCC_0_0	0	0	0	0	0	0	0	2	1
⊟ CORERESET_PF_C0_0	2	16	0	0	0	0	0	0	0
CORERESET_PF_C0_0	2	16	0	0	0	0	0	0	0
⊟ CORERESET_PF_C1_0	0	17	0	0	0	0	0	1	0
CORERESET_PF_C1_0	0	17	0	0	0	0	0	1	0
⊟ PROC_SUBSYSTEM_0	8587	2728	2376	2376	0	6	64	2	0
⊞ COREAHBL2AHBL_BRI...	30	135	0	0	0	0	0	0	0
⊞ COREAHBL2AHBL_BRI...	37	139	0	0	0	0	0	0	0
⊞ CORE_AHBTOAPB3_0	38	91	0	0	0	0	0	0	0
⊞ CoreAHBLite0_0	463	109	0	0	0	0	0	0	0
⊞ Core_APB3_0	48	0	0	0	0	0	0	0	0
⊞ Core_JTAG_Debug_0	262	17	0	0	0	0	0	2	0
⊞ Core_UARTabp0_0	152	114	0	0	0	0	0	0	0
⊞ MIV_RV32_C0_0	6706	1697	1224	1224	0	6	32	0	0
⊞ SRAM_Buffer_0	453	46	1152	1152	0	0	32	0	0
⊞ SystemServices_0	398	380	0	0	0	0	0	0	0

2.4.3 Software Implementation

For information about how to create and build a SoftConsole project for the Mi-V processor subsystem, see *TU0775: PolarFire FPGA: Building a Mi-V Processor Subsystem Tutorial*.

To use User Cryptoprocessor services in the application, download the PolarFire User Crypto CAL driver from the firmware catalog and add the driver to the SoftConsole project. The User Crypto CAL driver contains `config_user.h` file for driver configuration. In the `config_user.h` file, define the PKX0_BASE macro as the base address of the User Cryptoprocessor according to the Libero design.

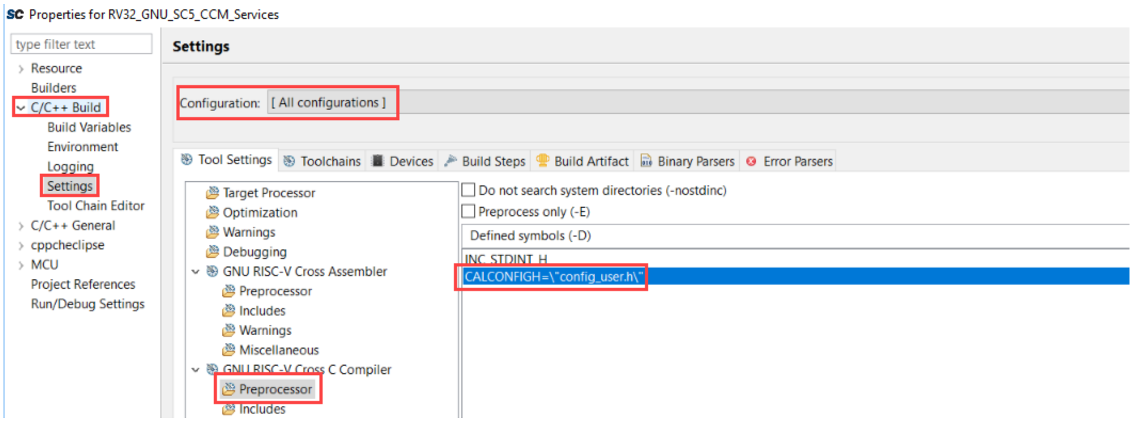
Figure 8 • User Crypto CAL Driver



```
   Copyright (c) 1999-2017 The Athena Group, Inc.

   This file, the information contained herein, and its documentation
   is property of The Athena Group, Inc., and furnished under a
   license agreement. The file and its documentation may be used or
   copied only in accordance with the terms of the license agreement.
   No part of this file or its documentation may be read,
   reproduced, transmitted, or translated without the prior written
   permission of The Athena Group, Inc.

   Each copy of this file shall include all copyrights, trademarks,
   service marks, and proprietary rights notices, if any.

   All technical data contained in this file is subject to export
   control laws of the United States of America. Disclosure to
   nationals of other countries contrary to United States law is
   prohibited.

   This marking must be kept on all copies of this file.
   ------------------------------------------------------------------ */

/* ------------------------------------------------------------------
   Description:

   User configuration file to include/exclude CAL components.
   ------------------------------------------------------------------ */

#ifndef CALCONFIG_F5200_H
#define CALCONFIG_F5200_H

#include "x52cfg_user.h"

#define SAT_LITTLE_ENDIAN 1
#define PKX0_BASE 0x62000000
#define USEPKX 1
#define USEAESPKX 1
#define USESHAPKX 1
#define USEDRBGPKX 1
#define USENRBGPKX 1
#define USECALCTX 1

#endif
```

Browse and add the `config_user.h` file to GNU RISC-V Cross C Compiler -> Preprocessor for both debug and release configurations as shown in the following figure.

Figure 9 • CALCONFIGH



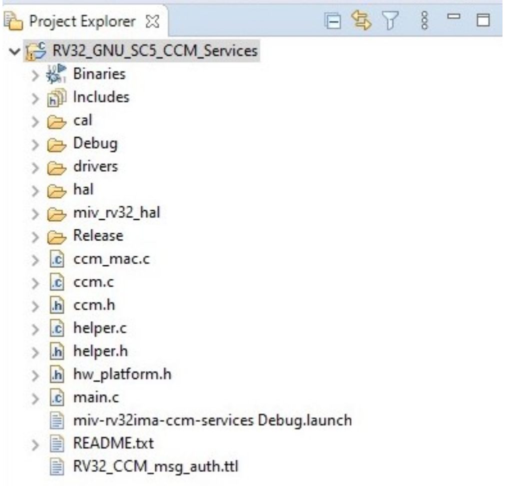
Configuration: [All configurations]

☐ Do not search system directories (-nostdinc)
☐ Preprocess only (-E)

Defined symbols (-D)
INC_STDINT_H
CALCONFIGH=\"config_user.h\"

The following figure shows the intended directory structure for a project based on MIV_RV32 using SoftConsole. This project uses the User Crypto Library (CAL) and CoreUARTapb drivers. CoreUARTapb drivers rely on the MIV_RV32 HAL to access the hardware. The content of the drivers directory results from generating the driver source files for each peripheral in the project using Firmware Catalog. The contents of the HAL and miv_rv32_hal directories result from generating the source files for the MIV_RV32 HAL from the Firmware Catalog.

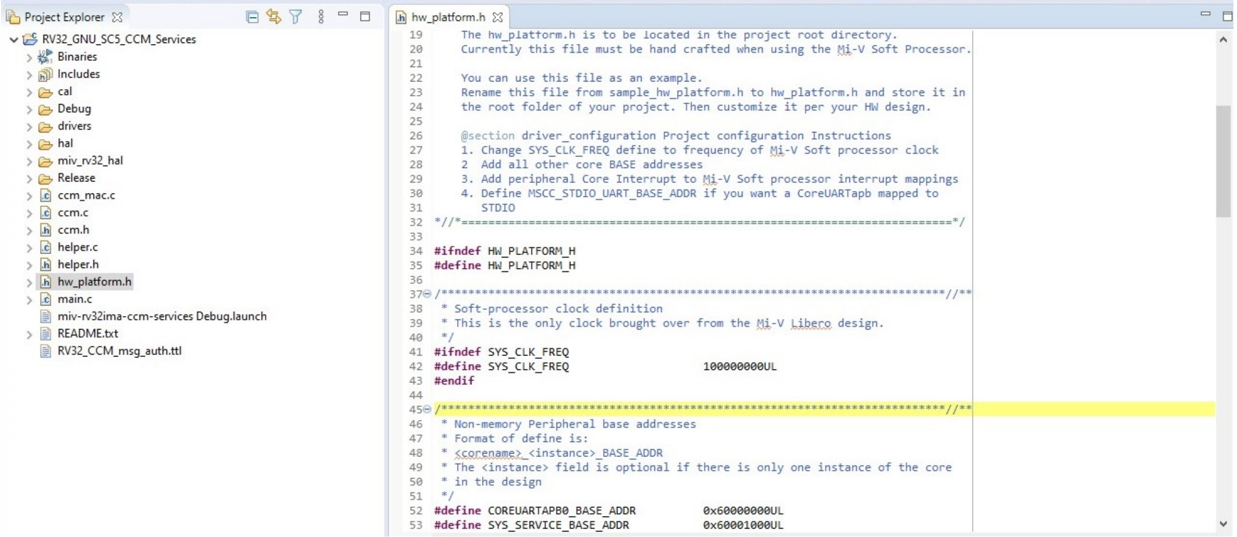
Figure 10 • **SoftConsole Project Directory Structure**



```
RV32_GNU_SC5_CCM_Services
    Binaries
    Includes
    cal
    Debug
    drivers
    hal
    miv_rv32_hal
    Release
    ccm_mac.c
    ccm.c
    ccm.h
    helper.c
    helper.h
    hw_platform.h
    main.c
    miv-rv32ima-ccm-services Debug.launch
    README.txt
    RV32_CCM_msg_auth.ttl
```

The UART peripheral base address, System Services base address and system clock frequency are provided in the `hw_platform.h` file.

Figure 11 • **Peripheral Base Address**



```
RV32_GNU_SC5_CCM_Services
    Binaries
    Includes
    cal
    Debug
    drivers
    hal
    miv_rv32_hal
    Release
    ccm_mac.c
    ccm.c
    ccm.h
    helper.c
    helper.h
    hw_platform.h
    main.c
    miv-rv32ima-ccm-services Debug.launch
    README.txt
    RV32_CCM_msg_auth.ttl
```

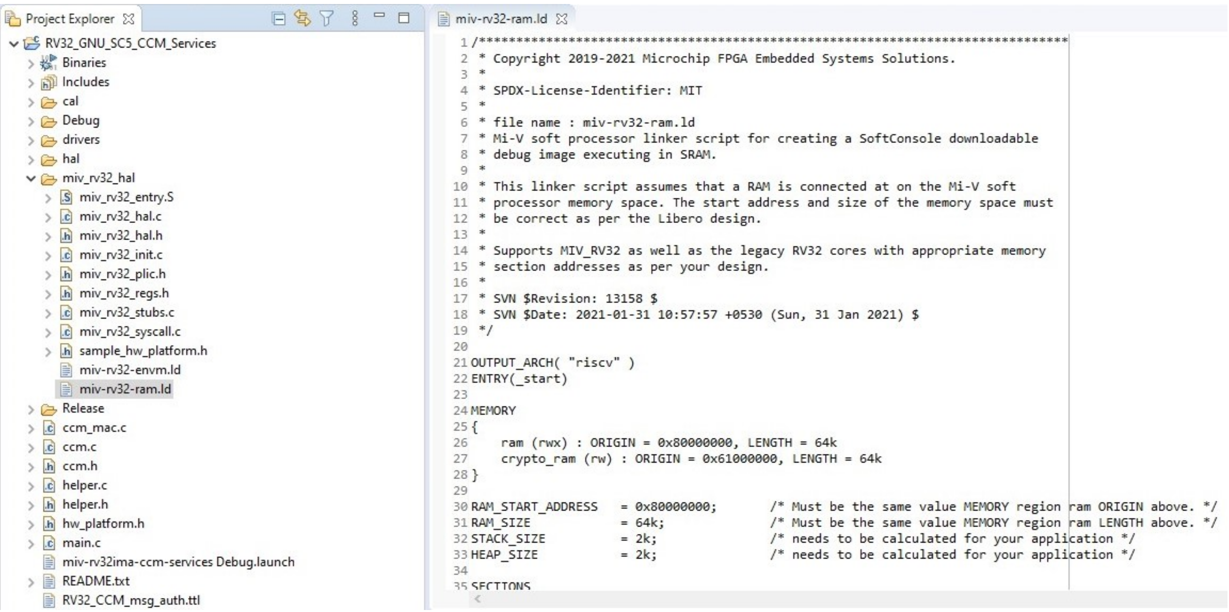
```
    The hw_platform.h is to be located in the project root directory.
    Currently this file must be hand crafted when using the Mi-V Soft Processor.

    You can use this file as an example.
    Rename this file from sample_hw_platform.h to hw_platform.h and store it in
    the root folder of your project. Then customize it per your HW design.

    @section driver_configuration Project configuration Instructions
    1. Change SYS_CLK_FREQ define to frequency of Mi-V Soft processor clock
    2  Add all other core BASE addresses
    3. Add peripheral Core Interrupt to Mi-V Soft processor interrupt mappings
    4. Define MSCC_STDIO_UART_BASE_ADDR if you want a CoreUARTapb mapped to
       STDIO
*//*==========================================================================*/

#ifndef HW_PLATFORM_H
#define HW_PLATFORM_H

/***************************************************************************//**
 * Soft-processor clock definition
 * This is the only clock brought over from the Mi-V Libero design.
 */
#ifndef SYS_CLK_FREQ
#define SYS_CLK_FREQ                        100000000UL
#endif

/***************************************************************************//**
 * Non-memory Peripheral base addresses
 * Format of define is:
 * <corename>_<instance>_BASE_ADDR
 * The <instance> field is optional if there is only one instance of the core
 * in the design
 */
#define COREUARTAPB0_BASE_ADDR              0x60000000UL
#define SYS_SERVICE_BASE_ADDR               0x60001000UL
```

2.4.3.1 Linker Script Update

A sample linker script file (`miv-rv32-ram.ld`) is provided along with the MIV_RV32 HAL files. This linker script assumes that the SRAM is connected to the Mi-V processor memory space. The start address and size of the memory space must correspond with the Libero design.

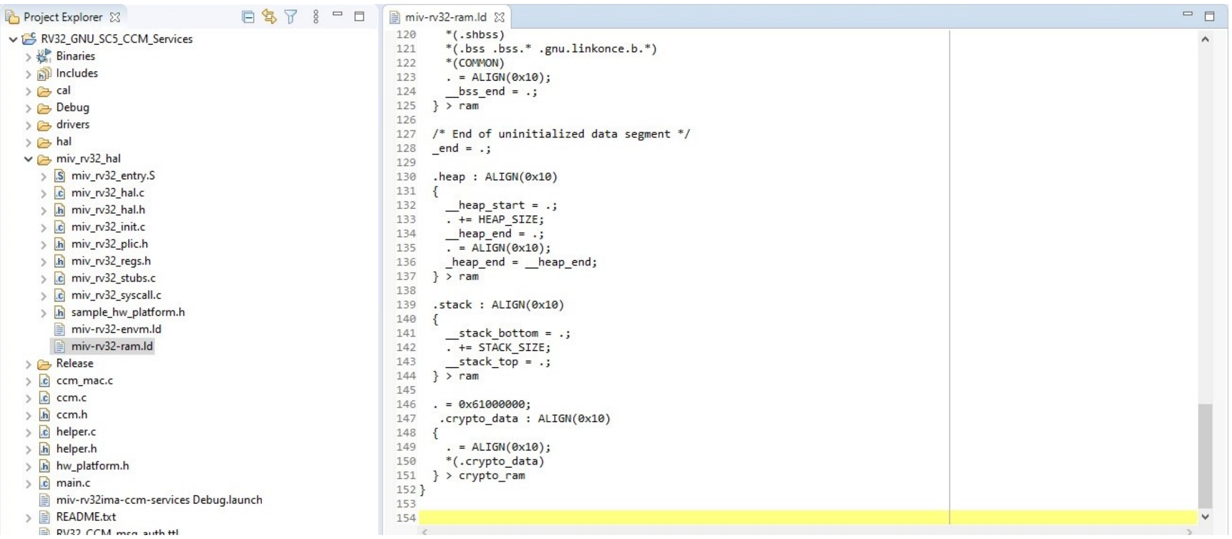
A *crypto_ram* user section is defined in the linker script (see the following figure) to map the User Crypto input and output buffers to a common memory located on the Mi-V processor AHBL interface. The common memory is located at address 0x61000000.

Figure 12 • **Linker Script**



```
/*******************************************************************************
 * Copyright 2019-2021 Microchip FPGA Embedded Systems Solutions.
 *
 * SPDX-License-Identifier: MIT
 *
 * file name : miv-rv32-ram.ld
 * Mi-V soft processor linker script for creating a SoftConsole downloadable
 * debug image executing in SRAM.
 *
 * This linker script assumes that a RAM is connected at on the Mi-V soft
 * processor memory space. The start address and size of the memory space must
 * be correct as per the Libero design.
 *
 * Supports MIV_RV32 as well as the legacy RV32 cores with appropriate memory
 * section addresses as per your design.
 *
 * SVN $Revision: 13158 $
 * SVN $Date: 2021-01-31 10:57:57 +0530 (Sun, 31 Jan 2021) $
 */

OUTPUT_ARCH( "riscv" )
ENTRY(_start)

MEMORY
{
    ram (rwx) : ORIGIN = 0x80000000, LENGTH = 64k
    crypto_ram (rw) : ORIGIN = 0x61000000, LENGTH = 64k
}

RAM_START_ADDRESS   = 0x80000000;       /* Must be the same value MEMORY region ram ORIGIN above. */
RAM_SIZE            = 64k;              /* Must be the same value MEMORY region ram LENGTH above. */
STACK_SIZE          = 2k;               /* needs to be calculated for your application */
HEAP_SIZE           = 2k;               /* needs to be calculated for your application */

SECTIONS
```

Figure 13 • **Linker Script (Continued)**

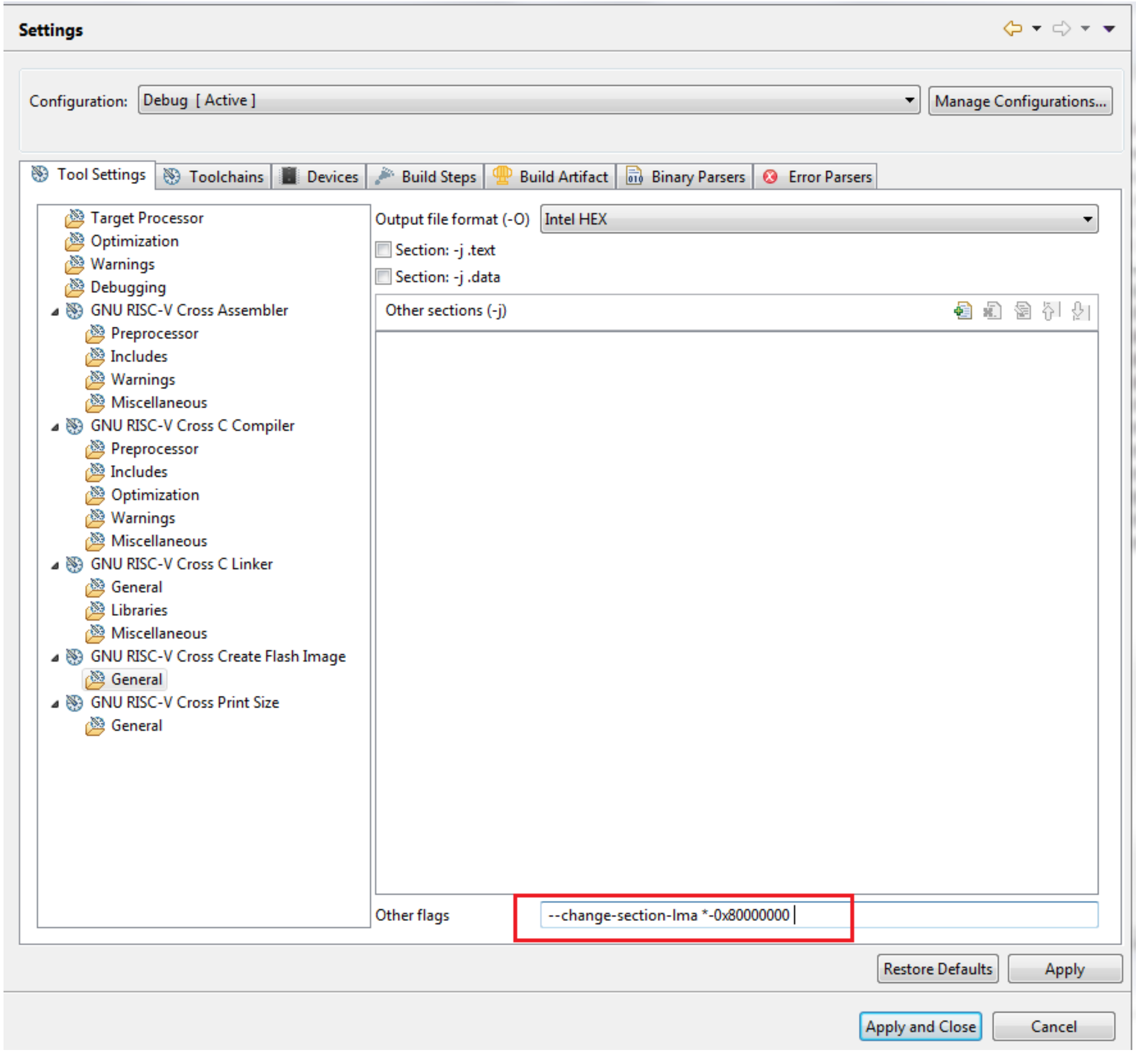


```
    *(.shbss)
    *(.bss .bss.* .gnu.linkonce.b.*)
    *(COMMON)
    . = ALIGN(0x10);
    __bss_end = .;
  } > ram

  /* End of uninitialized data segment */
  _end = .;

  .heap : ALIGN(0x10)
  {
    __heap_start = .;
    . += HEAP_SIZE;
    __heap_end = .;
    . = ALIGN(0x10);
    _heap_end = __heap_end;
  } > ram

  .stack : ALIGN(0x10)
  {
    __stack_bottom = .;
    . += STACK_SIZE;
    __stack_top = .;
  } > ram

  . = 0x61000000;
   .crypto_data : ALIGN(0x10)
  {
    . = ALIGN(0x10);
    *(.crypto_data)
  } > crypto_ram
}

```

2.4.3.2 RISC-V Flash Image Setting

As shown in Figure 14, page 14, add the `--change-section-lma *-0x80000000` command to remove the non-linear address (the first line) of the HEX file.

Figure 14 • **RISC-V Flash Image Setting**



Configuration: Debug [Active]

Output file format (-O) Intel HEX

□ Section: -j .text

□ Section: -j .data

Other flags --change-section-lma *-0x80000000

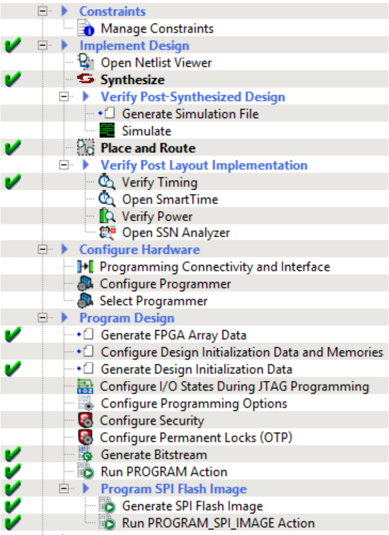
2.4.4 TCM Initialization from SPI Flash

This section describes how to load the user application hex image file into the TCM from SPI flash using System Controller.

To configure Design Initialization Data and Memories, perform the following steps:

1. Copy the `RV32_GNU_SC5_CCM_Services.hex` file from *<Download_Directory>\mpf_ac464_eval\SoftConsole\RV32_GNU_SC5_CCM_Services\Debug folder* and place it in *<Download_Directory>\mpf_ac464_eval\Libero_Project*
2. Double-click **Configure Design Initialization Data and Memories** from the **Design Flow** window as shown in the following figure.

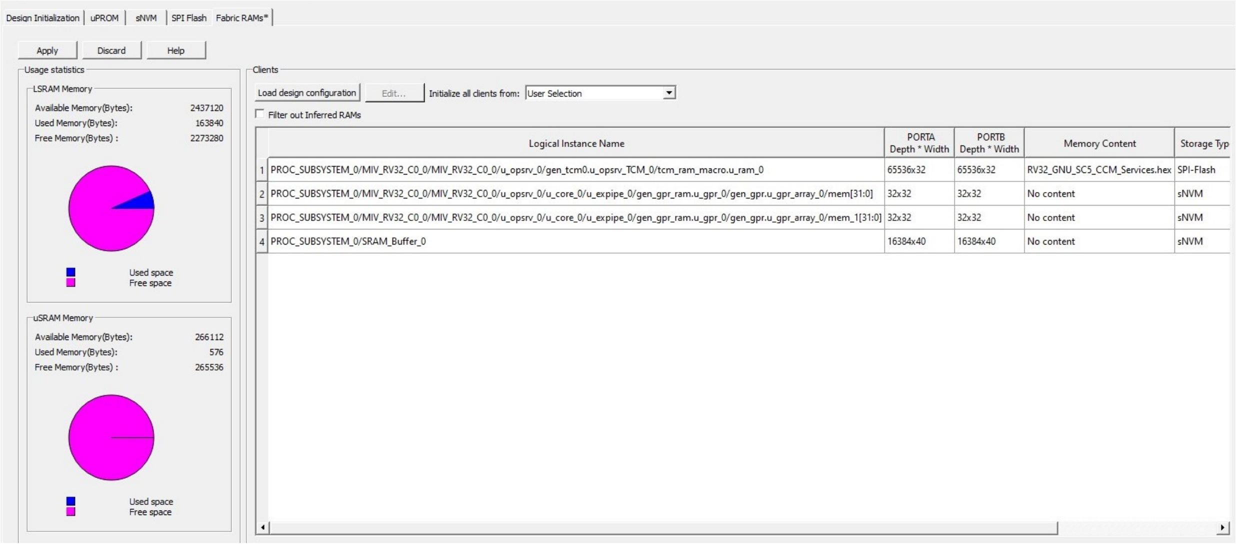
Figure 15 • **Configure Design Initialization Data and Memories**



- Constraints
  - Manage Constraints
- ✓ Implement Design
  - Open Netlist Viewer
  - ✓ Synthesize
  - Verify Post-Synthesized Design
    - Generate Simulation File
    - Simulate
  - ✓ Place and Route
  - Verify Post Layout Implementation
    - ✓ Verify Timing
    - Open SmartTime
    - Verify Power
    - Open SSN Analyzer
- Configure Hardware
  - Programming Connectivity and Interface
  - Configure Programmer
  - Select Programmer
- Program Design
  - ✓ Generate FPGA Array Data
  - Configure Design Initialization Data and Memories
  - ✓ Generate Design Initialization Data
  - Configure I/O States During JTAG Programming
  - Configure Programming Options
  - Configure Security
  - Configure Permanent Locks (OTP)
  - ✓ Generate Bitstream
  - ✓ Run PROGRAM Action
  - ✓ Program SPI Flash Image
    - ✓ Generate SPI Flash Image
    - ✓ Run PROGRAM_SPI_IMAGE Action

3. Click the **Fabric RAMs** tab and select the TCM instance that needs to be initialized and click **Edit,** as shown in the following figure. In this design, TCM instance must be initialized with the user application.

Figure 16 • **Fabric RAMs**



Usage statistics

LSRAM Memory

Available Memory(Bytes):	2437120
Used Memory(Bytes):	163840
Free Memory(Bytes) :	2273280

uSRAM Memory

Available Memory(Bytes):	266112
Used Memory(Bytes):	576
Free Memory(Bytes) :	265536

Clients

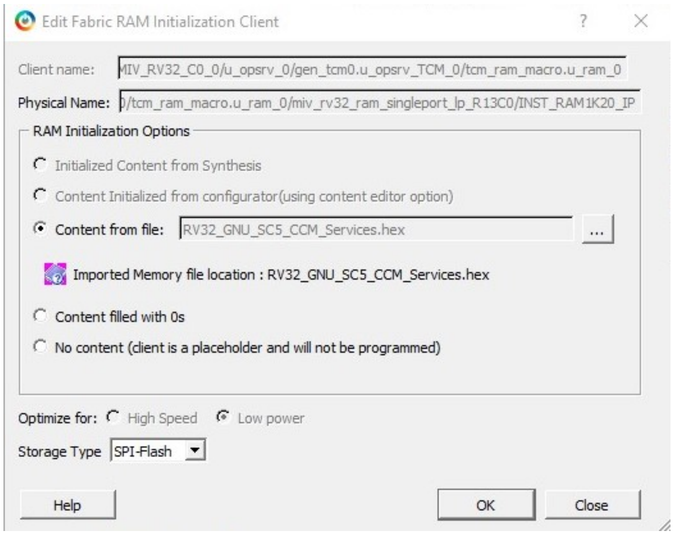
Initialize all clients from: User Selection

☐ Filter out Inferred RAMs

	Logical Instance Name	PORTA Depth * Width	PORTB Depth * Width	Memory Content	Storage Typ
1	PROC_SUBSYSTEM_0/MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsrv_0/gen_tcm0.u_opsrv_TCM_0/tcm_ram_macro.u_ram_0	65536x32	65536x32	RV32_GNU_SC5_CCM_Services.hex	SPI-Flash
2	PROC_SUBSYSTEM_0/MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsrv_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
3	PROC_SUBSYSTEM_0/MIV_RV32_C0_0/MIV_RV32_C0_0/u_opsrv_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
4	PROC_SUBSYSTEM_0/SRAM_Buffer_0	16384x40	16384x40	No content	sNVM

4. Double-click the **TCM** instance to add initialization client and storage location. In the **Edit Fabric RAM Initialization Client** dialog, select the **Content from file** option, locate the `RV32_GNU_SC5_CCM_Services.hex` file from the Libero Project folder, select **Storage Type** as **SPI-Flash**, and then click **OK,** as shown in the following figure.

Figure 17 • Edit Fabric RAM Initialization Client



Edit Fabric RAM Initialization Client

Client name: MIV_RV32_C0_0/u_opsrv_0/gen_tcm0.u_opsrv_TCM_0/tcm_ram_macro.u_ram_0

Physical Name: 0/tcm_ram_macro.u_ram_0/miv_rv32_ram_singleport_lp_R13C0/INST_RAM1K20_IP

RAM Initialization Options

○ Initialized Content from Synthesis

○ Content Initialized from configurator(using content editor option)

● Content from file: RV32_GNU_SC5_CCM_Services.hex

Imported Memory file location : RV32_GNU_SC5_CCM_Services.hex

○ Content filled with 0s

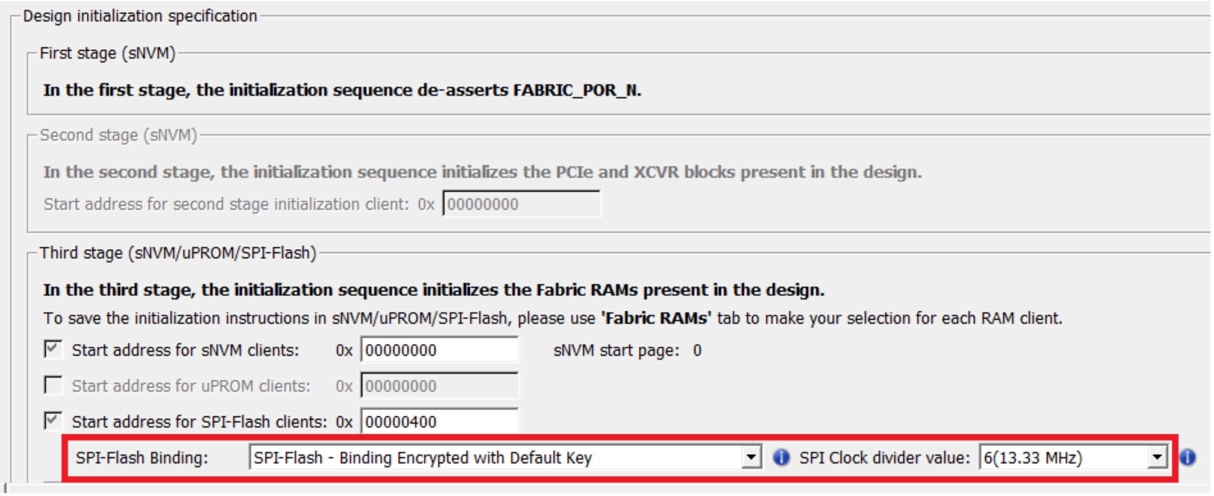
○ No content (client is a placeholder and will not be programmed)

Optimize for: ○ High Speed ● Low power

Storage Type: SPI-Flash

5. Click **Design Initialization** tab, ensure **SPI-Flash - Binding Encrypted with Default Key** is selected for **SPI-Flash Binding** type as shown in the following figure. **SPI clock divider value** must be set to 6. Refer to note given below for more information. In this reference design, the initialization client is stored in the SPI flash in encrypted format with authentication. When this option is selected, the design initialization client file (<root>_uic.bin) is encrypted with the default encryption key. When Default key is selected, the user does not need to specify any other details. Enable user security protection using UEK1/UEK2 for the SPI-Flash client if required.

Figure 18 • Design Initialization



Design initialization specification

First stage (sNVM)

In the first stage, the initialization sequence de-asserts FABRIC_POR_N.

Second stage (sNVM)

In the second stage, the initialization sequence initializes the PCIe and XCVR blocks present in the design.

Start address for second stage initialization client: 0x 00000000

Third stage (sNVM/uPROM/SPI-Flash)

In the third stage, the initialization sequence initializes the Fabric RAMs present in the design.

To save the initialization instructions in sNVM/uPROM/SPI-Flash, please use **'Fabric RAMs'** tab to make your selection for each RAM client.

☑ Start address for sNVM clients: 0x 00000000 sNVM start page: 0

☐ Start address for uPROM clients: 0x 00000000

☑ Start address for SPI-Flash clients: 0x 00000400

SPI-Flash Binding: SPI-Flash - Binding Encrypted with Default Key SPI Clock divider value: 6(13.33 MHz)

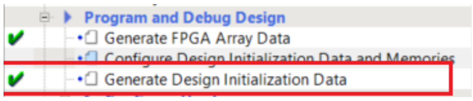
Note: The SPI clock divider value specifies the required SPI SCK frequency to read the initialization data from SPI Flash. The SPI Clock divider value must be selected based on the external SPI Flash operating frequency range.

Table 5 • SPI Clock Divider Value

SPI Clock Divider Value	SCK Frequency
1	80 MHz
2	40 MHz
4	20 MHz
6	13.3 MHz

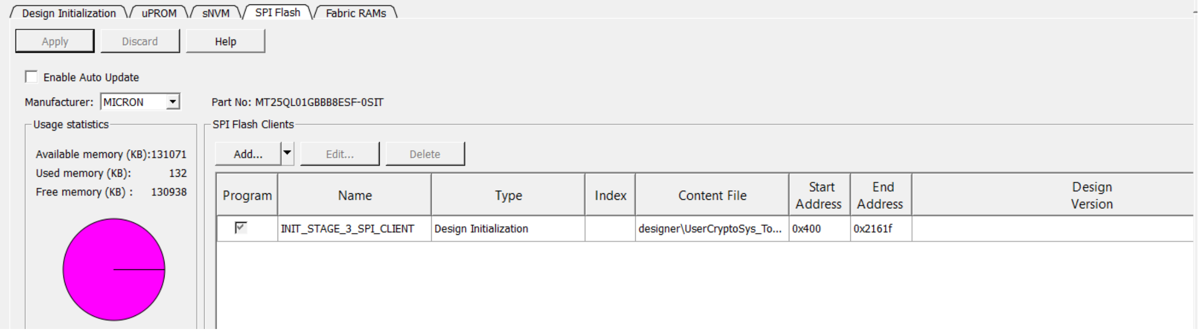
6. Click Generate Initialization clients under the **Design Initialization** tab to generate the External SPI-Flash (Non-authenticated), client.
7. When the initialization clients are generated, the Generate Initialization clients status window is displayed as shown in the following figure.

Figure 19 • **Generate Design Initialization Data**


Program and Debug Design
Generate FPGA Array Data
Configure Design Initialization Data and Memories
Generate Design Initialization Data

8. Select the **SPI Flash** tab to verify that the SPI flash client is generated as shown in the following figure.

Figure 20 • **sNVM Client Verification**



☐ Enable Auto Update

Manufacturer: MICRON

Part No: MT25QL01GBBB8ESF-0SIT

Usage statistics

Available memory (KB):131071
Used memory (KB): 132
Free memory (KB) : 130938

SPI Flash Clients

Program	Name	Type	Index	Content File	Start Address	End Address	Design Version
☑	INIT_STAGE_3_SPI_CLIENT	Design Initialization		designer\UserCryptoSys_To...	0x400	0x2161f	

Configuration of Design Initialization Data and Memories is successfully completed.

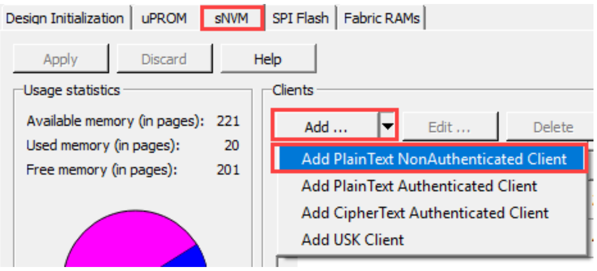
Note: In the scenario where a PolarFire device is first programmed with a design with an sNVM client, and then reprogrammed with a (different) design without an sNVM client, upon completion of programming with the second design, the sNVM client will not be erased. In such a case, if there are sNVM pages that are locked, writes to those pages using a system service call fails. There is no programming action to erase sNVM completely. To work around this issue, create a dummy sNVM client (filled with 0's) in the second design.

2.4.4.1 Adding a Dummy Client to Unlock the sNVM Pages

In this design, the AES key provided by the user is stored in the sNVM at page#4 using system service call. Hence, a dummy client to clear the sNVM page needs to be added to sNVM configurator. To add a dummy client, perform the following steps:

1. In the Libero **Design Flow** window, double-click **Configure Design Initialization Data and Memories**.
2. In the Design and Memory Initialization page, select the **sNVM** tab.
3. In sNVM Clients pane, click **Add...** > **Add PlainText NonAuthenticated Client**. as shown in the following figure.

Figure 21 • **Adding Dummy Client**

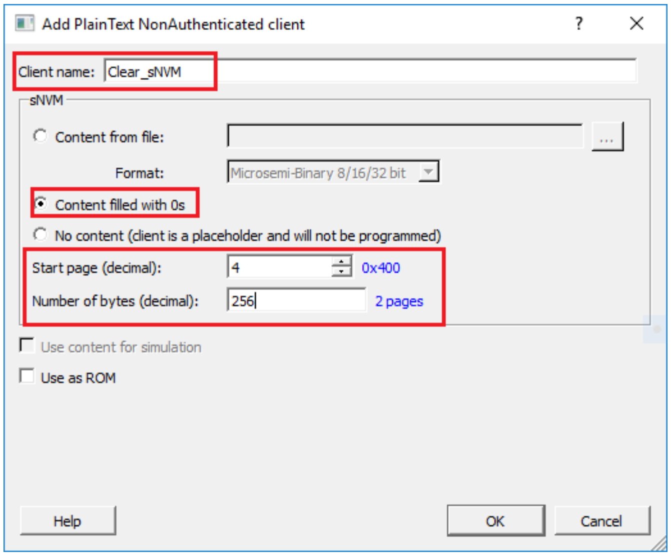


Usage statistics

Available memory (in pages): 221
Used memory (in pages): 20
Free memory (in pages): 201

4. In the **Add PlainText NonAuthenticated client** window, select **Content filled with 0s** radio button and enter:
   - **Client name**: Clear_sNVM
   - **Start page (decimal)**: 4
   - **Number of bytes (decimal)**: 256
5. Click **OK** to create a dummy client.

Figure 22 • **Dummy Client Parameters**



Add PlainText NonAuthenticated client

Client name: Clear_sNVM

sNVM

○ Content from file:

Format: Microsemi-Binary 8/16/32 bit

● Content filled with 0s

○ No content (client is a placeholder and will not be programmed)

Start page (decimal): 4 0x400

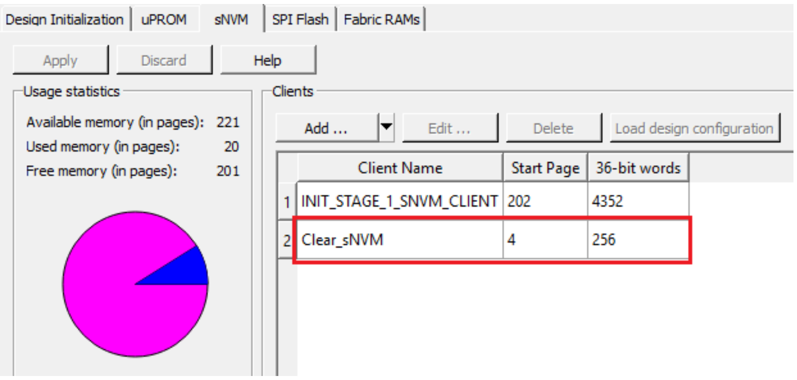
Number of bytes (decimal): 256 2 pages

□ Use content for simulation

□ Use as ROM

The following figure shows the dummy client added to sNVM Clients pane.

Figure 23 • **Dummy Client**



Usage statistics

Available memory (in pages):	221
Used memory (in pages):	20
Free memory (in pages):	201

Clients

	Client Name	Start Page	36-bit words
1	INIT_STAGE_1_SNVM_CLIENT	202	4352
2	Clear_sNVM	4	256

2.5 Programming the PolarFire Device and SPI Flash

This section describes how to program the PolarFire device and SPI Flash.

To program the PolarFire device, perform the following steps:

1. Ensure that the jumpers on the evaluation board are set as specified in the following tables.

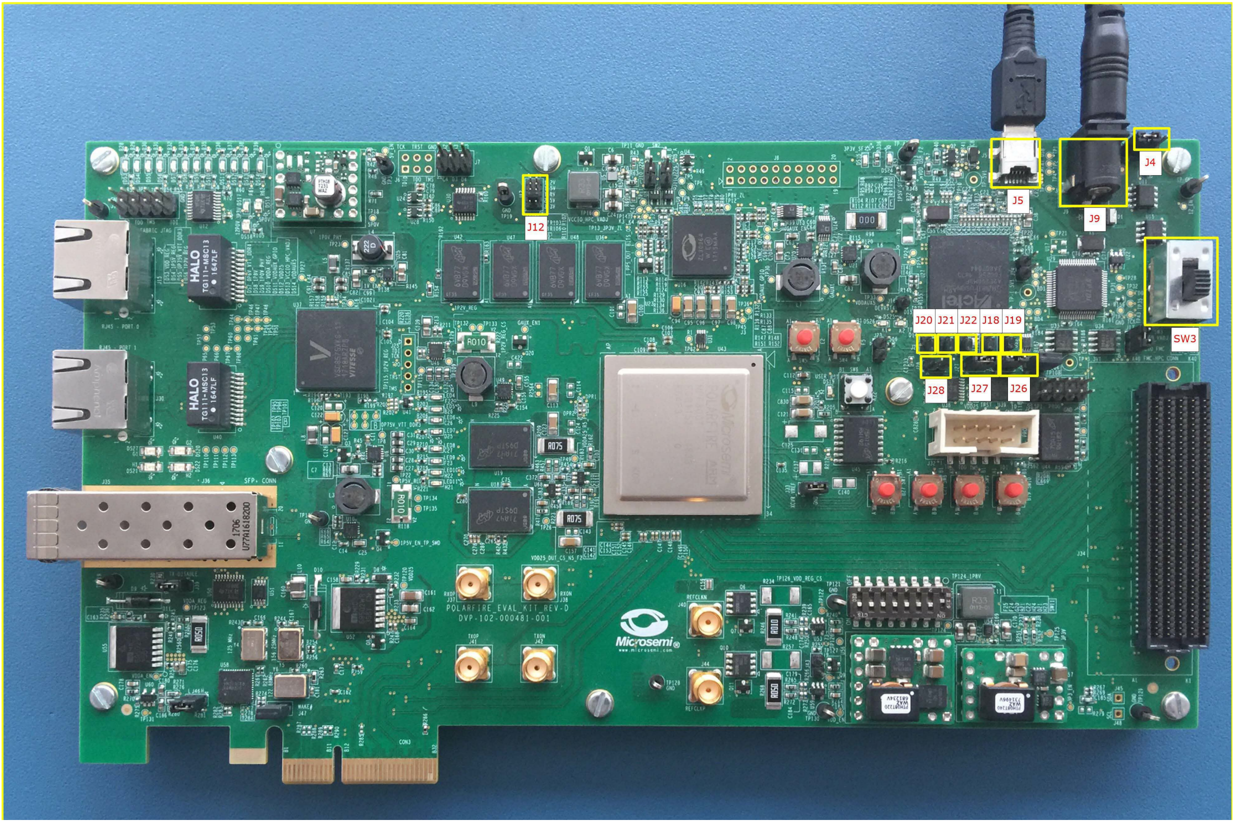
Table 6 • **Jumper Settings—Evaluation Board**

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-board FlashPro5
J26	Close pin 1 and 2 for programming through the FTDI SPI
J27	Close pin 1 and 2 for programming through the FTDI SPI
J23	Open pin 1 and 2 for programming SPI Flash
J4	Close pin 1 and 2 for manual power switching using SW3
J12	Close pin 3 and 4 for 2.5 V

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

The following figure shows the PolarFire Evaluation Kit set up to be programmed.

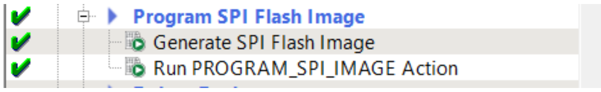
Figure 24 • **Board Setup—Evaluation**



5. Open the Libero project.
6. Click **Run PROGRAM Action** to program the device.

7. Double-click **Run Program_SPI_IMAGE Action** to program the SPI flash. A green tick mark is displayed after the successful generation as shown in the following figure.

Figure 25 • **Program SPI Flash Image**



✓ Program SPI Flash Image
✓ Generate SPI Flash Image
✓ Run PROGRAM_SPI_IMAGE Action

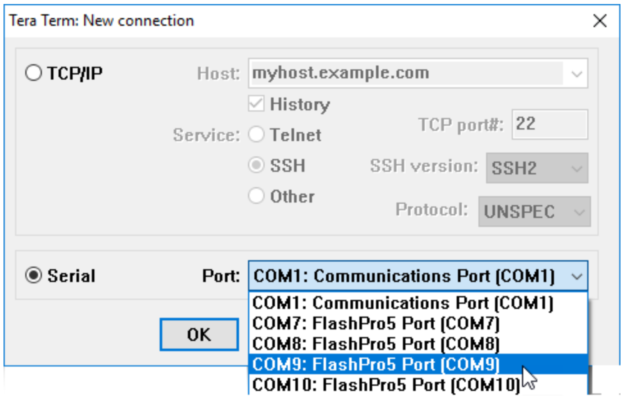
2.5.1 Tera Term Setup

The user application provides a user interface on the Tera Term terminal through the UART interface.

To set up the Tera Term program, perform the following steps:

1. Ensure that the USB cable connects the host PC to the **J5** (USB) port on the PolarFire Evaluation board.
2. Start Tera Term.
3. Select **Serial** as the Connection type.
4. Set the Serial **Port** to the second highest COM port number from the drop-down list as shown in the following figure. For example, **COM9: FlashPro5 Port [COM9]** in this instance.

Figure 26 • **Select Serial as the Connection Type**



Tera Term: New connection

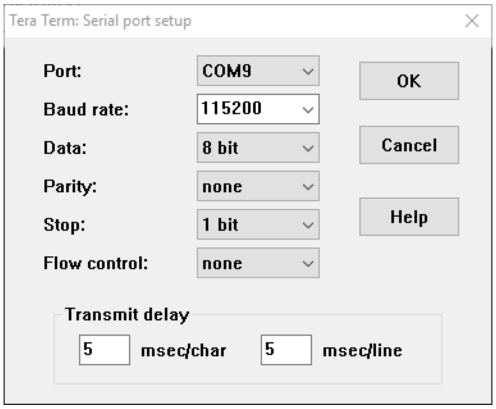
○ TCP/IP Host: myhost.example.com
☑ History
Service: ○ Telnet TCP port#: 22
● SSH SSH version: SSH2
○ Other Protocol: UNSPEC

● Serial Port: COM1: Communications Port (COM1)
COM1: Communications Port (COM1)
COM7: FlashPro5 Port (COM7)
COM8: FlashPro5 Port (COM8)
COM9: FlashPro5 Port (COM9)
COM10: FlashPro5 Port (COM10)

5. In the **Tera Term** window, go to **Setup > Serial port...**, set;
   - **Baud rate**: 115200
   - **Transmit delay**: 5 msec/char and 5 msec/line

   Rest of the serial port settings must be at default state as shown in the following figure and click **OK**.

Figure 27 • **Tera Term Configuration**



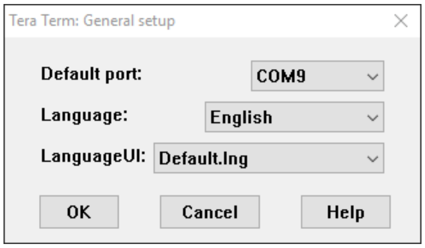
Tera Term: Serial port setup

Port: COM9
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none

Transmit delay
5 msec/char 5 msec/line

6. In the **Tera Term** window, go to **Setup > General...**, set the **Language** to **English** and click **OK**, as shown in the following figure. This setup is required for running the Tera Term macro script.

Figure 28 • **Tera Term General Setup**



Tera Term: General setup

Default port: COM9

Language: English

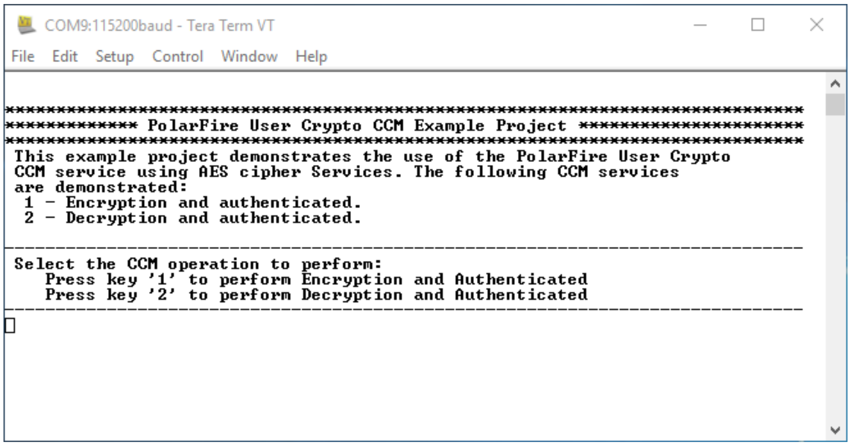
LanguageUI: Default.lng

This completes the Tera Term program setup.

2.6 Running the Demo

After the device is programmed, power cycle the board. The application prints the menu on the Tera Term program through the UART interface, as shown in following figure.

Figure 29 • **Application Menu**



COM9:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
******************************************************************************
************* PolarFire User Crypto CCM Example Project **********************
******************************************************************************
 This example project demonstrates the use of the PolarFire User Crypto
 CCM service using AES cipher Services. The following CCM services
 are demonstrated:
  1 - Encryption and authenticated.
  2 - Decryption and authenticated.

------------------------------------------------------------------------------
 Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------
```

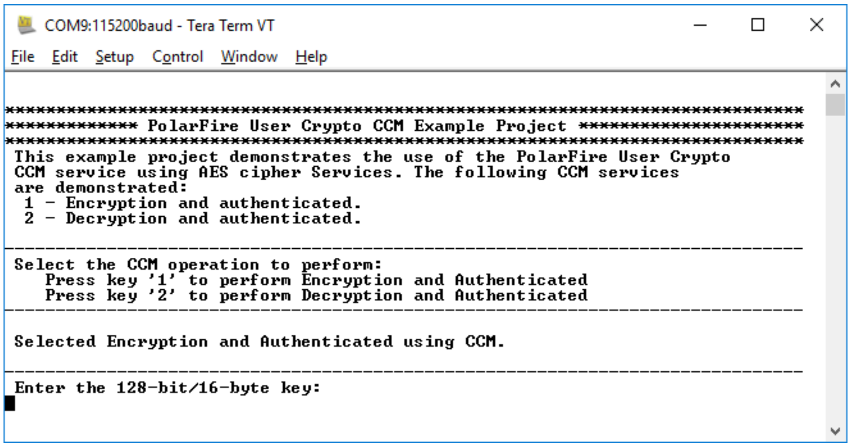
Use the following test vector to verify the AES CCM encryption and decryption operation:

- AES Key = 404142434445464748494A4B4C4D4E4F
- Nonce = 101112131415161718191A1B
- Additional authentication data (AAD) = 000102030405060708090A0B0C0D0E0F10111213
- Input data to Encrypt and authenticate = 202122232425262728292A2B2C2D2E2F3031323334353637

To run the demo, perform the following steps:

1. Press '1' to perform CCM AES-128 encryption using the User Cryptoprocessor. The application prompts to enter a 128-bit key, as shown in the following figure. The application securely stores the AES key into sNVM using System Service function call.

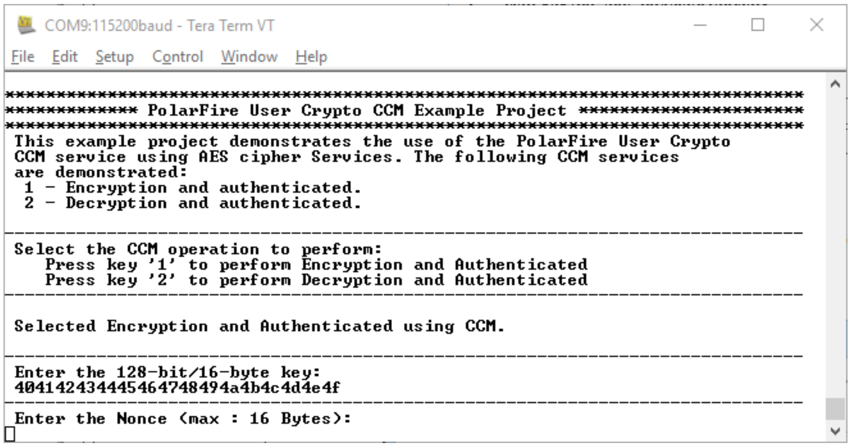
Figure 30 • **Application Menu—Enter 128-bit Key**



```
******************************************************************************
************* PolarFire User Crypto CCM Example Project **********************
******************************************************************************
 This example project demonstrates the use of the PolarFire User Crypto
 CCM service using AES cipher Services. The following CCM services
 are demonstrated:
  1 - Encryption and authenticated.
  2 - Decryption and authenticated.

------------------------------------------------------------------------------
 Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

 Selected Encryption and Authenticated using CCM.

------------------------------------------------------------------------------
 Enter the 128-bit/16-byte key:
```

2. Enter the AES key provided in the test vector and press Enter. The application prompts for Nonce, as shown in the following figure.

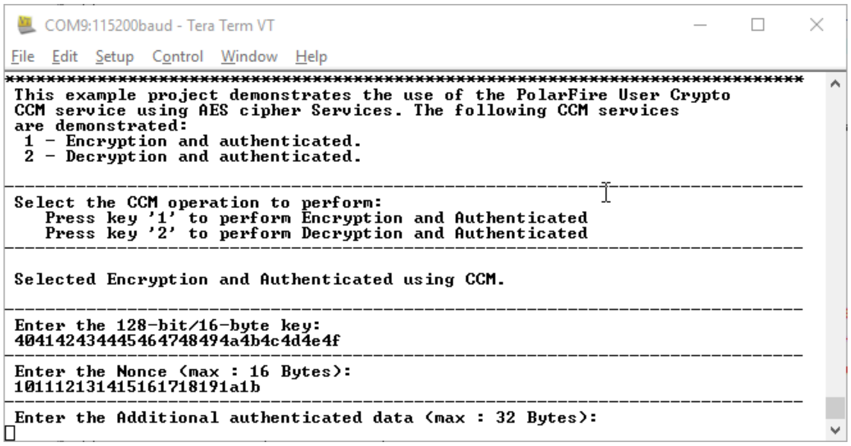
Figure 31 • **Application Menu—Nonce**



```
******************************************************************************
************* PolarFire User Crypto CCM Example Project **********************
******************************************************************************
 This example project demonstrates the use of the PolarFire User Crypto
 CCM service using AES cipher Services. The following CCM services
 are demonstrated:
  1 - Encryption and authenticated.
  2 - Decryption and authenticated.

------------------------------------------------------------------------------
 Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

 Selected Encryption and Authenticated using CCM.

------------------------------------------------------------------------------
 Enter the 128-bit/16-byte key:
 404142434445464748494a4b4c4d4e4f
------------------------------------------------------------------------------
 Enter the Nonce (max : 16 Bytes):
```

3. Enter the Nonce provided in the test vector and press Enter. The application prompts to enter AAD, as shown in the following figure.

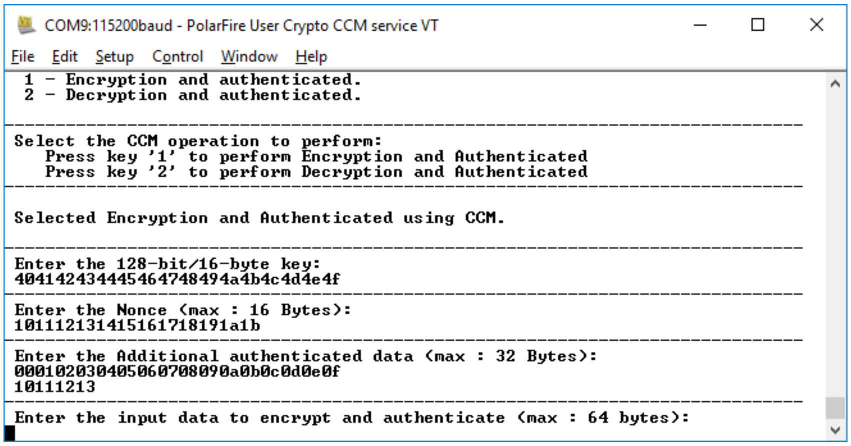
Figure 32 • **Application Menu—Additional Authentication Data**



```
******************************************************************************
 This example project demonstrates the use of the PolarFire User Crypto
 CCM service using AES cipher Services. The following CCM services
 are demonstrated:
  1 - Encryption and authenticated.
  2 - Decryption and authenticated.

------------------------------------------------------------------------------
 Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

 Selected Encryption and Authenticated using CCM.

------------------------------------------------------------------------------
 Enter the 128-bit/16-byte key:
 404142434445464748494a4b4c4d4e4f
------------------------------------------------------------------------------
 Enter the Nonce (max : 16 Bytes):
 101112131415161718191a1b
------------------------------------------------------------------------------
 Enter the Additional authenticated data (max : 32 Bytes):
```

4. Enter the AAD provided in the test vector and press Enter. The application prompts to enter input data to encrypt and authenticate, as shown in the following figure.

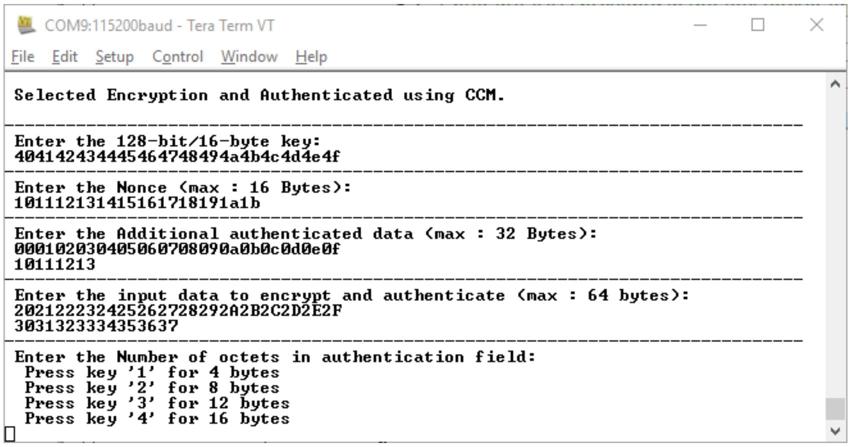
Figure 33 • Application Menu—Input Data to Encrypt and Authenticate


COM9:115200baud - PolarFire User Crypto CCM service VT
File Edit Setup Control Window Help

```
 1 - Encryption and authenticated.
 2 - Decryption and authenticated.
------------------------------------------------------------------------------
Select the CCM operation to perform:
   Press key '1' to perform Encryption and Authenticated
   Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

Selected Encryption and Authenticated using CCM.

------------------------------------------------------------------------------
Enter the 128-bit/16-byte key:
404142434445464748494a4b4c4d4e4f
------------------------------------------------------------------------------
Enter the Nonce (max : 16 Bytes):
101112131415161718191a1b
------------------------------------------------------------------------------
Enter the Additional authenticated data (max : 32 Bytes):
000102030405060708090a0b0c0d0e0f
10111213
------------------------------------------------------------------------------
Enter the input data to encrypt and authenticate (max : 64 bytes):
```

5. Enter the input data to encrypt and authenticate provided in the test vector and press Enter. The application prompts to enter the number of octets in the authentication field, as shown in the following figure.

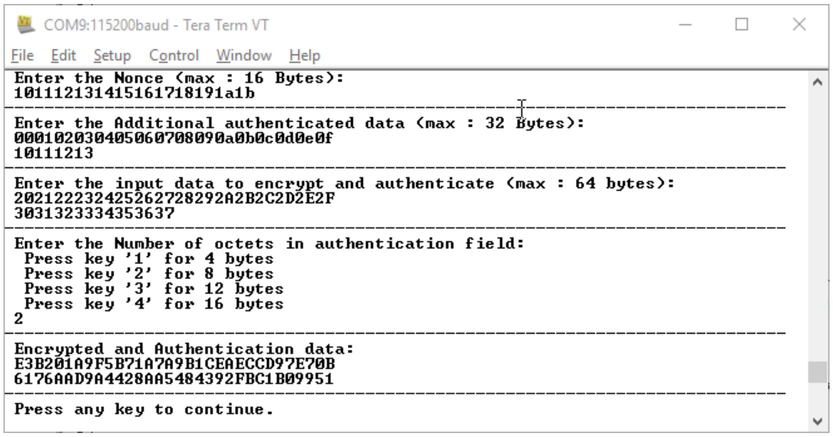
Figure 34 • Application Menu—Number of Octets in authentication Field


COM9:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
Selected Encryption and Authenticated using CCM.

------------------------------------------------------------------------------
Enter the 128-bit/16-byte key:
404142434445464748494a4b4c4d4e4f
------------------------------------------------------------------------------
Enter the Nonce (max : 16 Bytes):
101112131415161718191a1b
------------------------------------------------------------------------------
Enter the Additional authenticated data (max : 32 Bytes):
000102030405060708090a0b0c0d0e0f
10111213
------------------------------------------------------------------------------
Enter the input data to encrypt and authenticate (max : 64 bytes):
202122232425262728292A2B2C2D2E2F
3031323334353637
------------------------------------------------------------------------------
Enter the Number of octets in authentication field:
 Press key '1' for 4 bytes
 Press key '2' for 8 bytes
 Press key '3' for 12 bytes
 Press key '4' for 16 bytes
```

6. Press '2' for Encrypted and Authentication data. The result of the CCM AES encryption is printed on the Tera Term program, as shown in the following figure. Observe that the result is matched with the test vector, as shown in the following figure.

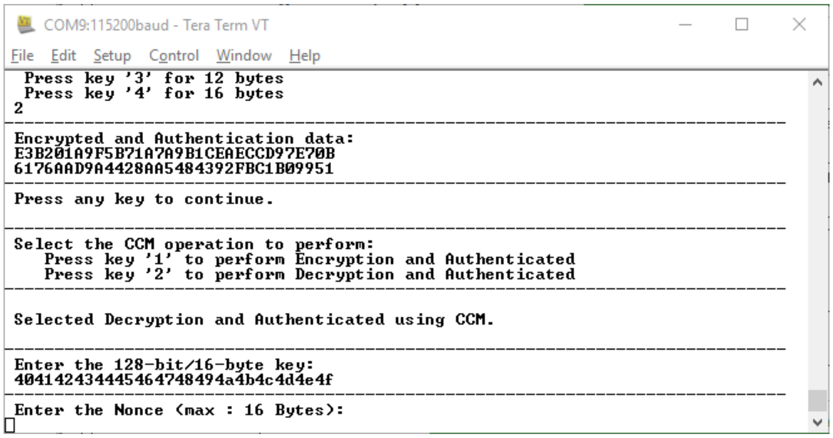
Figure 35 • Application Menu—Encrypted and Authentication Data


COM9:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
Enter the Nonce (max : 16 Bytes):
101112131415161718191a1b
------------------------------------------------------------------------------
Enter the Additional authenticated data (max : 32 Bytes):
000102030405060708090a0b0c0d0e0f
10111213
------------------------------------------------------------------------------
Enter the input data to encrypt and authenticate (max : 64 bytes):
202122232425262728292A2B2C2D2E2F
3031323334353637
------------------------------------------------------------------------------
Enter the Number of octets in authentication field:
 Press key '1' for 4 bytes
 Press key '2' for 8 bytes
 Press key '3' for 12 bytes
 Press key '4' for 16 bytes
2
------------------------------------------------------------------------------
Encrypted and Authentication data:
E3B201A9F5B71A7A9B1CEAECCD97E70B
6176AAD9A4428AA5484392FBC1B09951
------------------------------------------------------------------------------
Press any key to continue.
```

7. Press any key to continue to evaluate the AES operation. The application prints the AES encryption/decryption menu again on Tera Term program.
8. Press '2' to perform AES CCM decryption using User Cryptoprocessor. The application prompts to enter 128-bit key, as shown in the following figure.

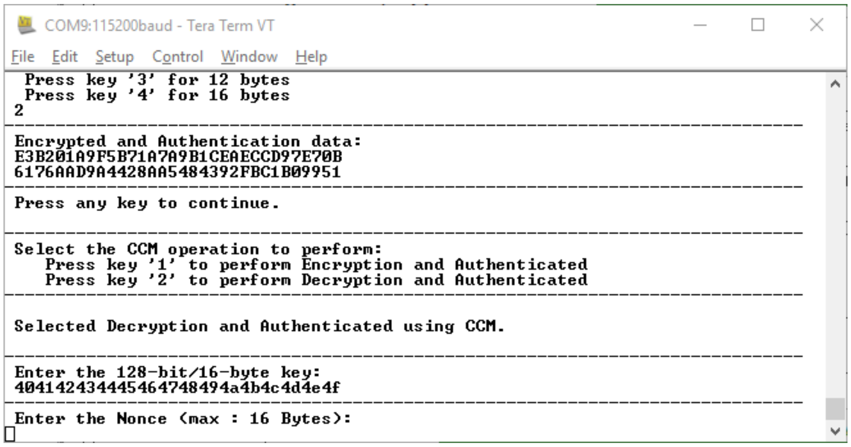
Figure 36 • **Application Menu Decryption—Enter 128-bit Key for AES Decryption**



```
 Press key '3' for 12 bytes
 Press key '4' for 16 bytes
2
------------------------------------------------------------------------------
Encrypted and Authentication data:
E3B201A9F5B71A7A9B1CEAECCD97E70B
6176AAD9A4428AA5484392FBC1B09951
------------------------------------------------------------------------------
Press any key to continue.

------------------------------------------------------------------------------
Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

Selected Decryption and Authenticated using CCM.

------------------------------------------------------------------------------
Enter the 128-bit/16-byte key:
404142434445464748494a4b4c4d4e4f
------------------------------------------------------------------------------
Enter the Nonce (max : 16 Bytes):
```

9. Enter the key provided in the preceding test vector and press Enter. The application prompts for Nonce.

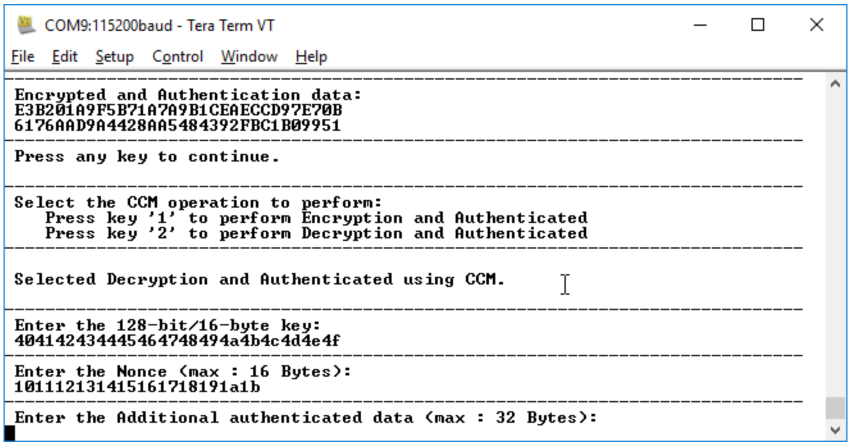
Figure 37 • **Application Menu Decryption—Nonce**



```
 Press key '3' for 12 bytes
 Press key '4' for 16 bytes
2
------------------------------------------------------------------------------
Encrypted and Authentication data:
E3B201A9F5B71A7A9B1CEAECCD97E70B
6176AAD9A4428AA5484392FBC1B09951
------------------------------------------------------------------------------
Press any key to continue.

------------------------------------------------------------------------------
Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

Selected Decryption and Authenticated using CCM.

------------------------------------------------------------------------------
Enter the 128-bit/16-byte key:
404142434445464748494a4b4c4d4e4f
------------------------------------------------------------------------------
Enter the Nonce (max : 16 Bytes):
```

10. Enter the Nonce provided in the test vector and press Enter. The application prompts to enter AAD, as shown in the following figure.

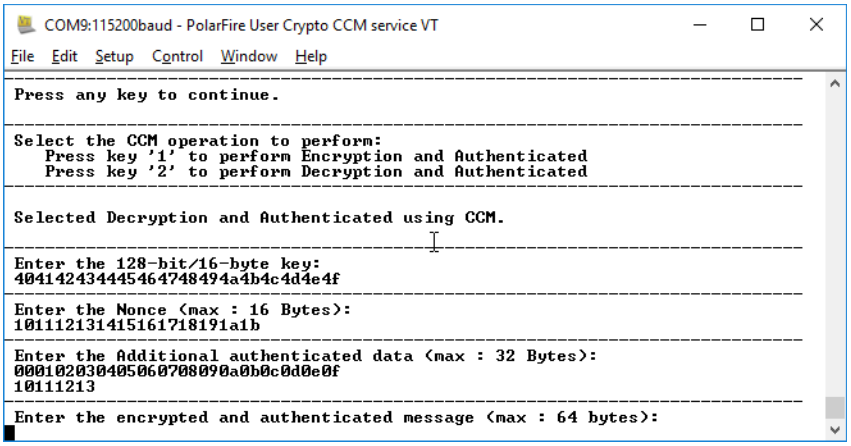
Figure 38 • **Application Menu—Additional Authentication Data**



```
------------------------------------------------------------------------------
Encrypted and Authentication data:
E3B201A9F5B71A7A9B1CEAECCD97E70B
6176AAD9A4428AA5484392FBC1B09951
------------------------------------------------------------------------------
Press any key to continue.

------------------------------------------------------------------------------
Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

Selected Decryption and Authenticated using CCM.

------------------------------------------------------------------------------
Enter the 128-bit/16-byte key:
404142434445464748494a4b4c4d4e4f
------------------------------------------------------------------------------
Enter the Nonce (max : 16 Bytes):
101112131415161718191a1b
------------------------------------------------------------------------------
Enter the Additional authenticated data (max : 32 Bytes):
```

11. Enter the AAD provided in the test vector and press Enter.The application prompts to enter input data to encrypt and authenticate, as shown in the following figure.

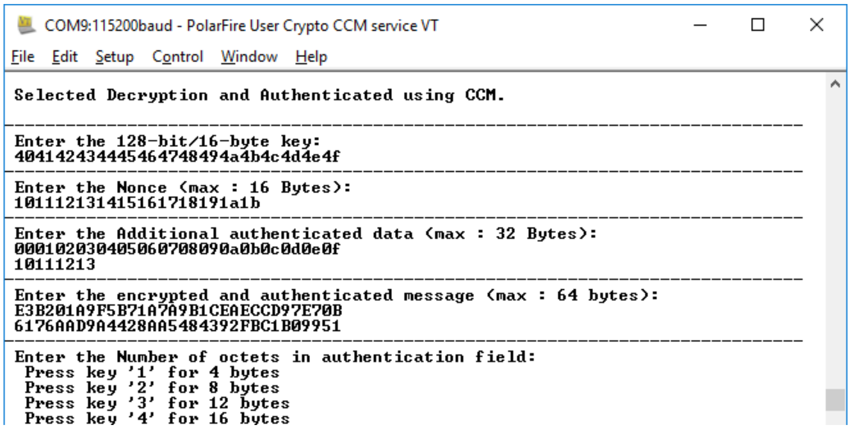
Figure 39 • Application Menu—Input Data to Encrypt and Authenticate


COM9:115200baud - PolarFire User Crypto CCM service VT
File Edit Setup Control Window Help

```
--------------------------------------------------------------------------------
Press any key to continue.
--------------------------------------------------------------------------------
Select the CCM operation to perform:
   Press key '1' to perform Encryption and Authenticated
   Press key '2' to perform Decryption and Authenticated
--------------------------------------------------------------------------------

Selected Decryption and Authenticated using CCM.

--------------------------------------------------------------------------------
Enter the 128-bit/16-byte key:
404142434445464748494a4b4c4d4e4f
--------------------------------------------------------------------------------
Enter the Nonce (max : 16 Bytes):
101112131415161718191a1b
--------------------------------------------------------------------------------
Enter the Additional authenticated data (max : 32 Bytes):
000102030405060708090a0b0c0d0e0f
10111213
--------------------------------------------------------------------------------
Enter the encrypted and authenticated message (max : 64 bytes):
```

12. Enter the input data to encrypt and authenticate provided in the test vector and press Enter. The application prompts to enter the number of octets in authentication field, as shown in the following figure.

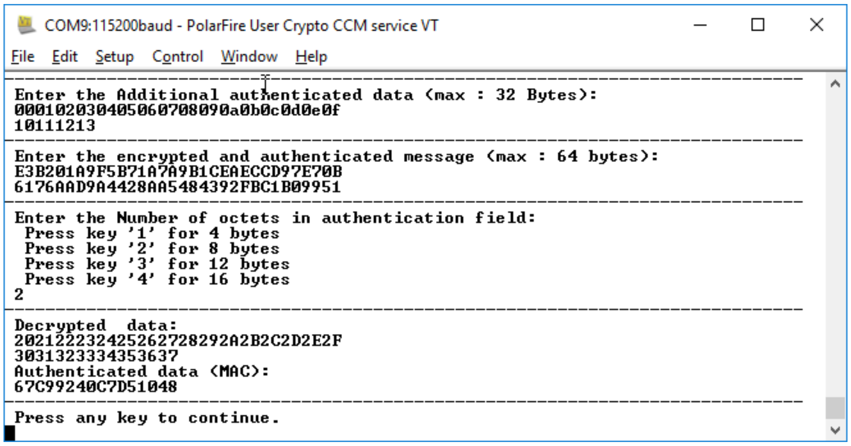
Figure 40 • Application Menu—Encrypted and Authentication Data


COM9:115200baud - PolarFire User Crypto CCM service VT
File Edit Setup Control Window Help

```
Selected Decryption and Authenticated using CCM.

--------------------------------------------------------------------------------
Enter the 128-bit/16-byte key:
404142434445464748494a4b4c4d4e4f
--------------------------------------------------------------------------------
Enter the Nonce (max : 16 Bytes):
101112131415161718191a1b
--------------------------------------------------------------------------------
Enter the Additional authenticated data (max : 32 Bytes):
000102030405060708090a0b0c0d0e0f
10111213
--------------------------------------------------------------------------------
Enter the encrypted and authenticated message (max : 64 bytes):
E3B201A9F5B71A7A9B1CEAECCD97E70B
6176AAD9A4428AA5484392FBC1B09951
--------------------------------------------------------------------------------
Enter the Number of octets in authentication field:
 Press key '1' for 4 bytes
 Press key '2' for 8 bytes
 Press key '3' for 12 bytes
 Press key '4' for 16 bytes
```

13. Enter '2' to proceed. The result of the AES CCM decryption is printed on the Tera Term program, as shown in the following figure. Observe that the result is matched with the test vector.

Figure 41 • Application Menu—AES CCM Decryption Result


COM9:115200baud - PolarFire User Crypto CCM service VT
File Edit Setup Control Window Help

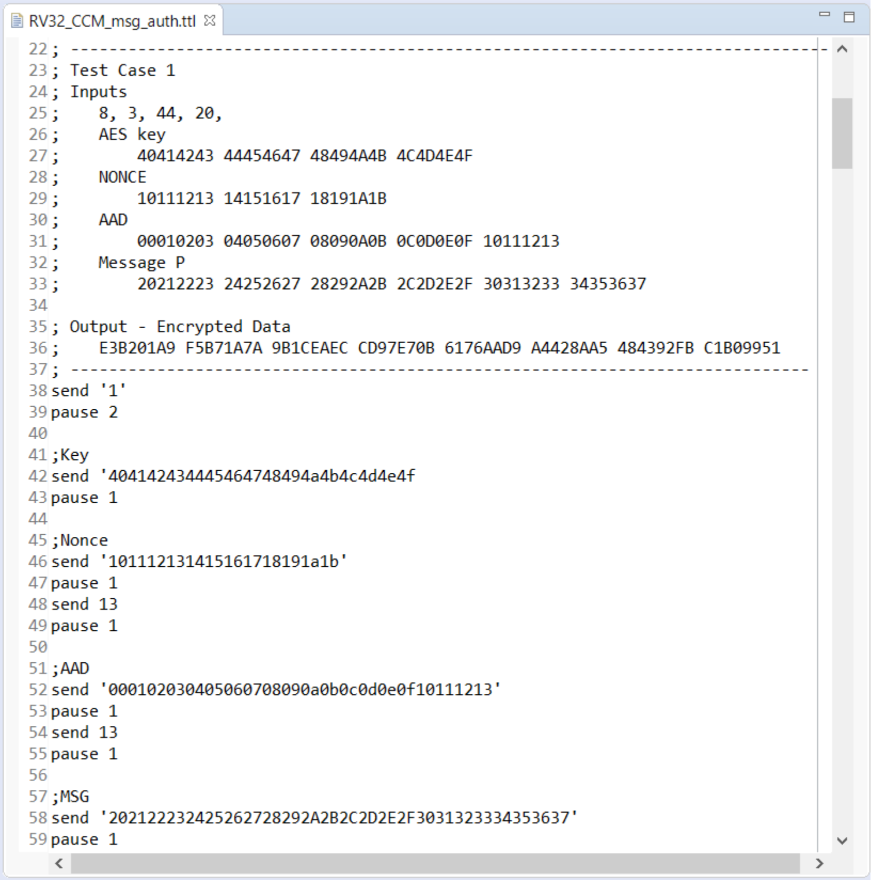
```
--------------------------------------------------------------------------------
Enter the Additional authenticated data (max : 32 Bytes):
000102030405060708090a0b0c0d0e0f
10111213
--------------------------------------------------------------------------------
Enter the encrypted and authenticated message (max : 64 bytes):
E3B201A9F5B71A7A9B1CEAECCD97E70B
6176AAD9A4428AA5484392FBC1B09951
--------------------------------------------------------------------------------
Enter the Number of octets in authentication field:
 Press key '1' for 4 bytes
 Press key '2' for 8 bytes
 Press key '3' for 12 bytes
 Press key '4' for 16 bytes
2
--------------------------------------------------------------------------------
Decrypted  data:
202122232425262728292A2B2C2D2E2F
3031323334353637
Authenticated data (MAC):
67C99240C7D51048
--------------------------------------------------------------------------------
Press any key to continue.
```

2.6.1 Running Tera Term Macro Script

Each sample project contains Tera Term macro script (.ttl) file to test the cryptographic algorithms used in the sample project. The Tera Term macro script contain commands to pass required input test vectors. It also contains the expected output for each input test vector for user verification. The following figure shows the Tera Term macro script available in the sample projects provided in the design file.

Note: Tera Term Macros in this design do not work with Windows 10 build 14393.0. You must update to Windows 10 build 14393.0.105 or later.

Figure 42 • **Tera Term Macro Script to Test AES Service**

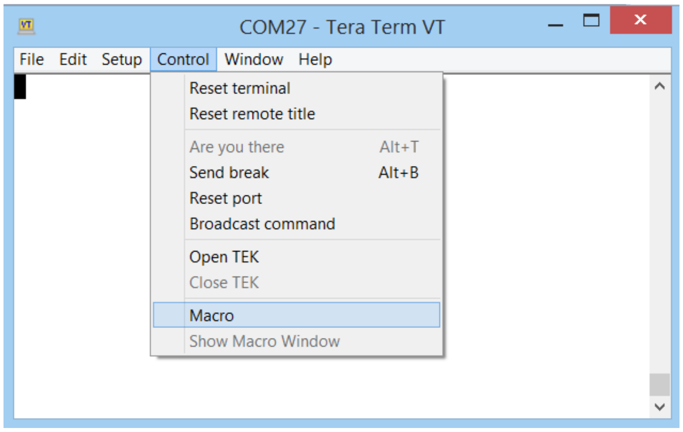


```
; ------------------------------------------------------------------------------
; Test Case 1
; Inputs
;    8, 3, 44, 20,
;    AES key
;        40414243 44454647 48494A4B 4C4D4E4F
;    NONCE
;        10111213 14151617 18191A1B
;    AAD
;        00010203 04050607 08090A0B 0C0D0E0F 10111213
;    Message P
;        20212223 24252627 28292A2B 2C2D2E2F 30313233 34353637

; Output - Encrypted Data
;    E3B201A9 F5B71A7A 9B1CEAEC CD97E70B 6176AAD9 A4428AA5 484392FB C1B09951
; ----------------------------------------------------------------------------
send '1'
pause 2

;Key
send '404142434445464748494a4b4c4d4e4f
pause 1

;Nonce
send '101112131415161718191a1b'
pause 1
send 13
pause 1

;AAD
send '000102030405060708090a0b0c0d0e0f10111213'
pause 1
send 13
pause 1

;MSG
send '202122232425262728292A2B2C2D2E2F3031323334353637'
pause 1
```

To run the Tera Term script, perform the following steps:

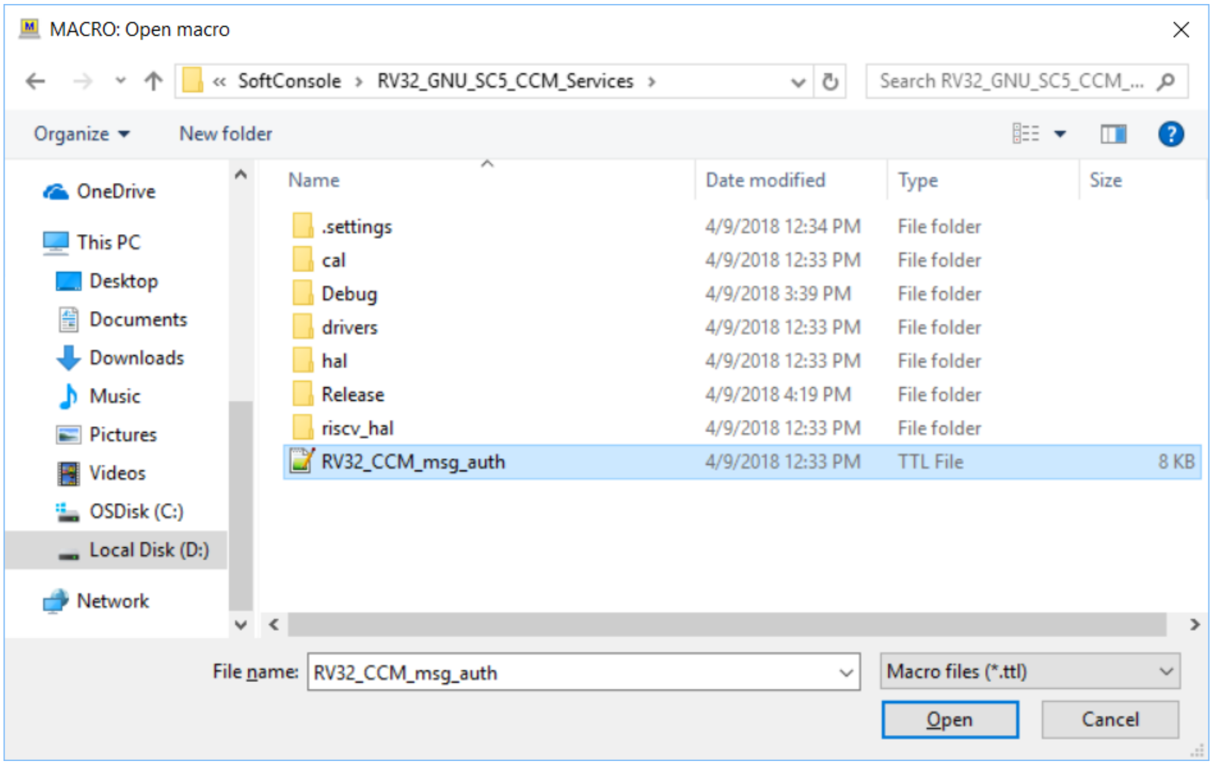
1. On the **Control** menu, click **Macro**, as shown in the following figure.

Figure 43 • **Executing Macro Script**



2. In the **Open macro** window, select the `RV32_CCM_msg_auth.ttl` file available in the sample project provided with the design files and click **Open** as shown in the following figure.

Figure 44 • **Selecting Macro Script File**

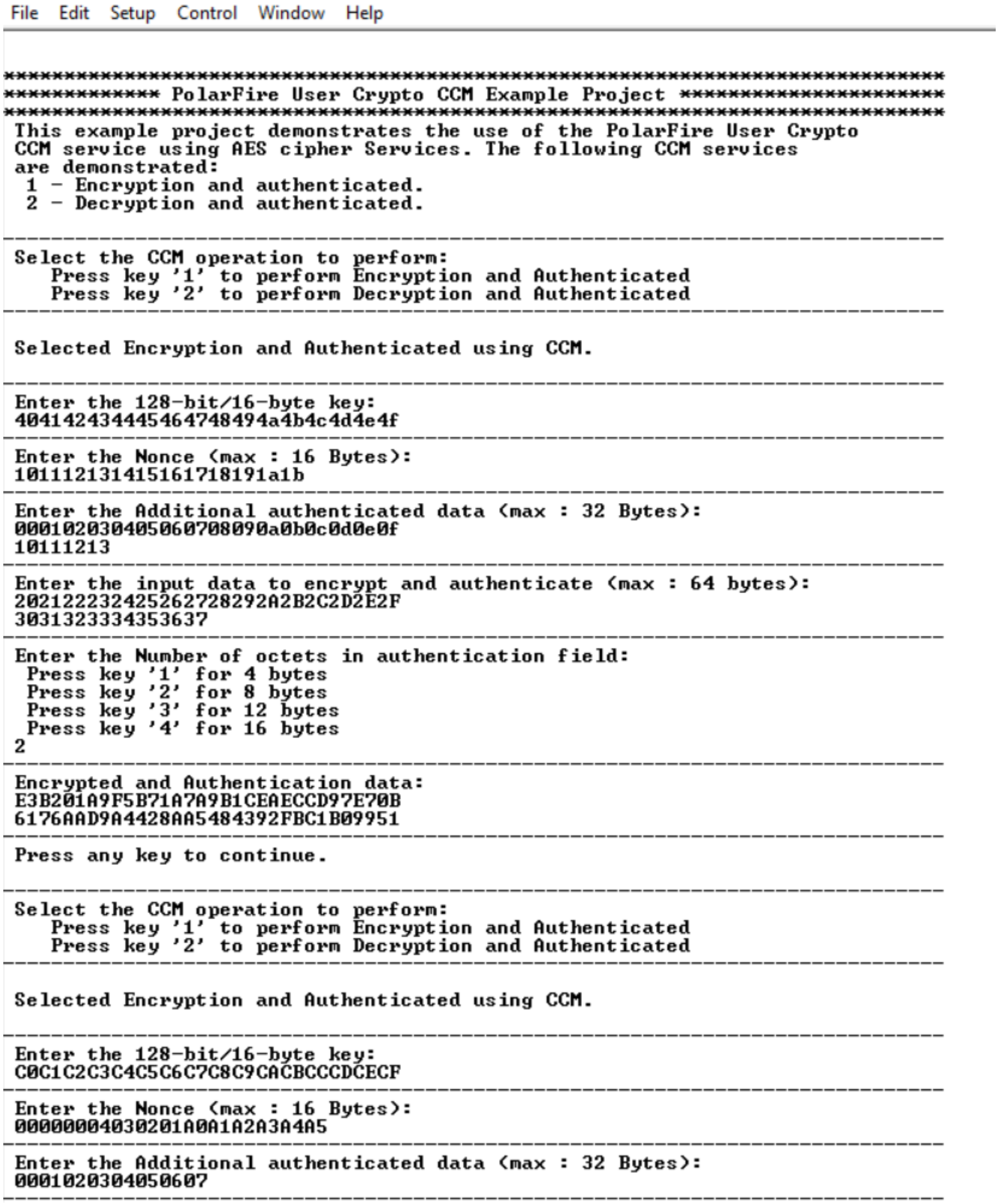


Name	Date modified	Type	Size
.settings	4/9/2018 12:34 PM	File folder	
cal	4/9/2018 12:33 PM	File folder	
Debug	4/9/2018 3:39 PM	File folder	
drivers	4/9/2018 12:33 PM	File folder	
hal	4/9/2018 12:33 PM	File folder	
Release	4/9/2018 4:19 PM	File folder	
riscv_hal	4/9/2018 12:33 PM	File folder	
RV32_CCM_msg_auth	4/9/2018 12:33 PM	TTL File	8 KB

File name: RV32_CCM_msg_auth

Tera Term executes the script and the results are printed on the Tera Term. Compare the output of the AES encryption and decryption service with the test vectors provided in the macro script. The following figure shows the result of the macro script execution.

Figure 45 • **Macro Script Execution Result**



```
*****************************************************************************
************* PolarFire User Crypto CCM Example Project ********************
*****************************************************************************
 This example project demonstrates the use of the PolarFire User Crypto
 CCM service using AES cipher Services. The following CCM services
 are demonstrated:
  1 - Encryption and authenticated.
  2 - Decryption and authenticated.

------------------------------------------------------------------------------
 Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

 Selected Encryption and Authenticated using CCM.

------------------------------------------------------------------------------
 Enter the 128-bit/16-byte key:
 404142434445464748494a4b4c4d4e4f
------------------------------------------------------------------------------
 Enter the Nonce (max : 16 Bytes):
 101112131415161718191a1b
------------------------------------------------------------------------------
 Enter the Additional authenticated data (max : 32 Bytes):
 000102030405060708090a0b0c0d0e0f
 10111213
------------------------------------------------------------------------------
 Enter the input data to encrypt and authenticate (max : 64 bytes):
 202122232425262728292A2B2C2D2E2F
 3031323334353637
------------------------------------------------------------------------------
 Enter the Number of octets in authentication field:
  Press key '1' for 4 bytes
  Press key '2' for 8 bytes
  Press key '3' for 12 bytes
  Press key '4' for 16 bytes
 2
------------------------------------------------------------------------------
 Encrypted and Authentication data:
 E3B201A9F5B71A7A9B1CEAECCD97E70B
 6176AAD9A4428AA5484392FBC1B09951
------------------------------------------------------------------------------
 Press any key to continue.

------------------------------------------------------------------------------
 Select the CCM operation to perform:
    Press key '1' to perform Encryption and Authenticated
    Press key '2' to perform Decryption and Authenticated
------------------------------------------------------------------------------

 Selected Encryption and Authenticated using CCM.

------------------------------------------------------------------------------
 Enter the 128-bit/16-byte key:
 C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF
------------------------------------------------------------------------------
 Enter the Nonce (max : 16 Bytes):
 00000004030201A0A1A2A3A4A5
------------------------------------------------------------------------------
 Enter the Additional authenticated data (max : 32 Bytes):
 0001020304050607
------------------------------------------------------------------------------
```

This concludes the demo of User Crypto AES service. Terminate the SoftConsole debugger session.

To run other User Crypto sample projects, see Appendix 1: Running UserCrypto Sample Projects, page 29.

3 Appendix 1: Running UserCrypto Sample Projects

The Firmware Catalog contains sample projects to show how to use User Crypto (CAL) APIs in the user application. You can use the sample projects to evaluate various cryptographic features supported by the User Cryptoprocessor on PolarFire devices. This section provides instructions on how to run UserCrypto sample projects with the provided Libero design.

The following table list the UserCrypto algorithms demonstrated in the sample projects. For more information about the sample projects, refer README.txt, file available in the sample projects.

Table 7 • **Algorithm**

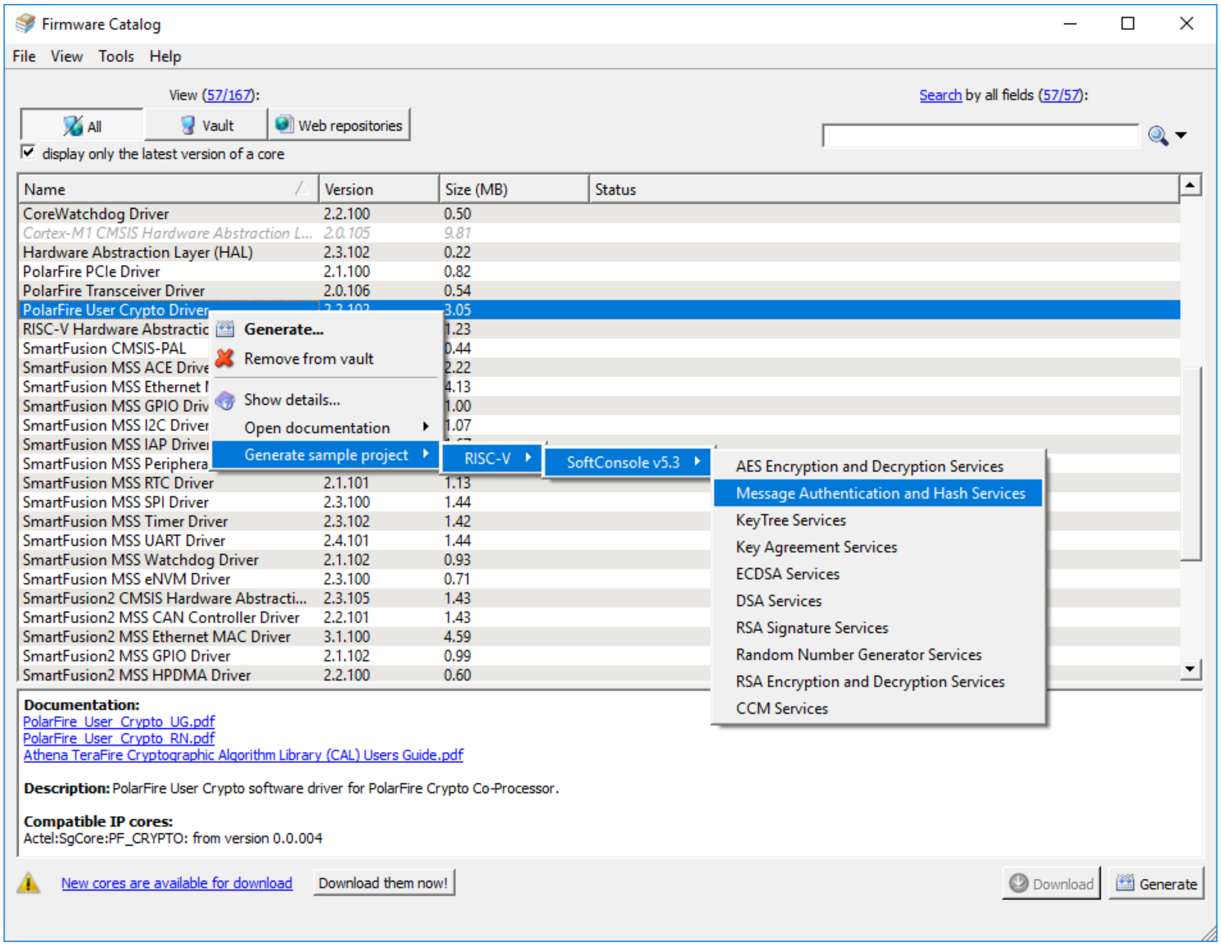
Algorithm	Parameters and Modes
AES	AES-ECB-256 encrypt
	AES-ECB-256 encrypt
	AES-CCM-128
GMAC	AES-GCM-256, 128-bit tag
HMAC	HMAC-SHA-256, 256-bit key
CMAC	AES-CMAC-256
KEY TREE	128-bit nonce + 8-bit optype
SHA	SHA-256
ECC	ECDSA SigGen, P-384/SHA-384, DPA
IFC (RSA)	Encrypt, RSA-3072, e=65537
	Decrypt, RSA-3072, CRT, DPA
FFC (DH)	SigGen, DSA-3072/SHA-384, DPA
	Key Agreement (KAS), DH-3072
NRBG	Instantiate: strength, s = 256, 384-bit nonce, 384-bit personalization string
	Generate: (no add input, no prediction resistance) s = 256

Note: CCM is used to provide assurance of the confidentiality and the authenticity of computer data by combining the techniques of the Counter (CTR) mode and the Cipher Block Chaining-Message Authentication Code (CBC-MAC) algorithm.

To run the sample projects, perform the following steps:

1. Download the sample project from Firmware Catalog by right-clicking on **PolarFire UserCrypto Driver** and select a sample project available through **Generate Sample Project > RISC-V > SoftConsole 5.3 > project name**, as shown in the following figure.

Figure 46 • **Firmware Catalog—Sample Projects**



☑ display only the latest version of a core

Name	Version	Size (MB)	Status
CoreWatchdog Driver	2.2.100	0.50	
Cortex-M1 CMSIS Hardware Abstraction L...	2.0.105	9.81	
Hardware Abstraction Layer (HAL)	2.3.102	0.22	
PolarFire PCIe Driver	2.1.100	0.82	
PolarFire Transceiver Driver	2.0.106	0.54	
PolarFire User Crypto Driver	2.2.102	3.05	
RISC-V Hardware Abstractic		1.23	
SmartFusion CMSIS-PAL		0.44	
SmartFusion MSS ACE Drive		2.22	
SmartFusion MSS Ethernet I		4.13	
SmartFusion MSS GPIO Driv		1.00	
SmartFusion MSS I2C Driver		1.07	
SmartFusion MSS IAP Driver			
SmartFusion MSS Periphera			
SmartFusion MSS RTC Driver	2.1.101	1.13	
SmartFusion MSS SPI Driver	2.3.100	1.44	
SmartFusion MSS Timer Driver	2.3.102	1.42	
SmartFusion MSS UART Driver	2.4.101	1.44	
SmartFusion MSS Watchdog Driver	2.1.102	0.93	
SmartFusion MSS eNVM Driver	2.3.100	0.71	
SmartFusion2 CMSIS Hardware Abstracti...	2.3.105	1.43	
SmartFusion2 MSS CAN Controller Driver	2.2.101	1.43	
SmartFusion2 MSS Ethernet MAC Driver	3.1.100	4.59	
SmartFusion2 MSS GPIO Driver	2.1.102	0.99	
SmartFusion2 MSS HPDMA Driver	2.2.100	0.60	

Documentation:
PolarFire_User_Crypto_UG.pdf
PolarFire_User_Crypto_RN.pdf
Athena TeraFire Cryptographic Algorithm Library (CAL) Users Guide.pdf

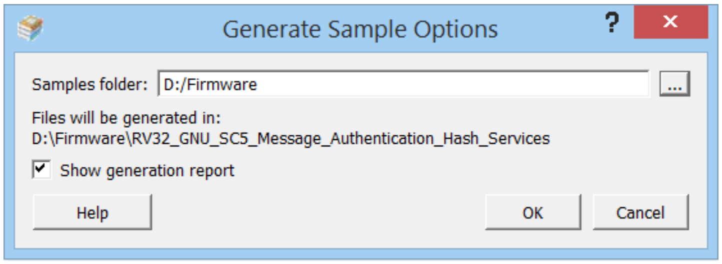
Description: PolarFire User Crypto software driver for PolarFire Crypto Co-Processor.

Compatible IP cores:
Actel:SgCore:PF_CRYPTO: from version 0.0.004

Note: CCM Services sample project is provided with the design files. Generate other sample projects and import into the SoftConsole workspace.

2. Click **OK** to generate the selected sample project to a local folder on the host PC.

Figure 47 • **Generate Sample Project**



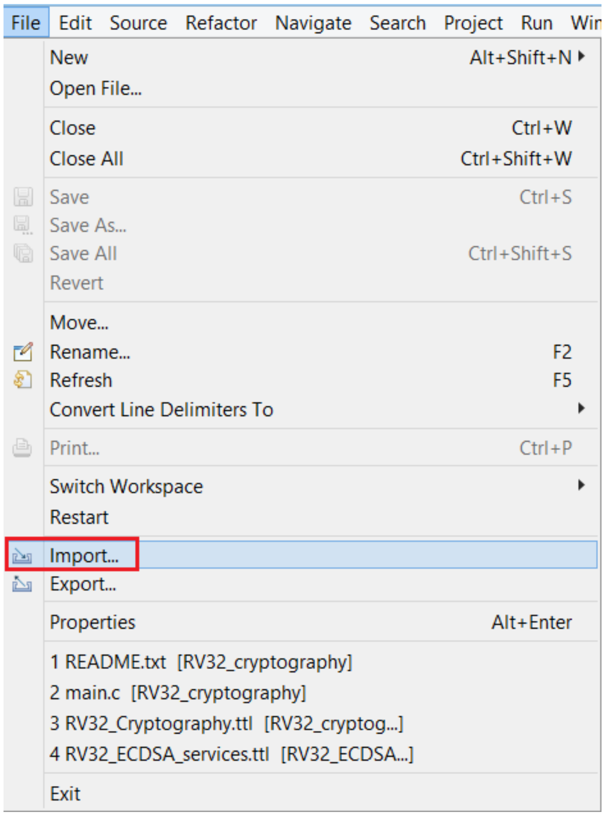
Samples folder: D:/Firmware

Files will be generated in:
D:\Firmware\RV32_GNU_SC5_Message_Authentication_Hash_Services

☑ Show generation report

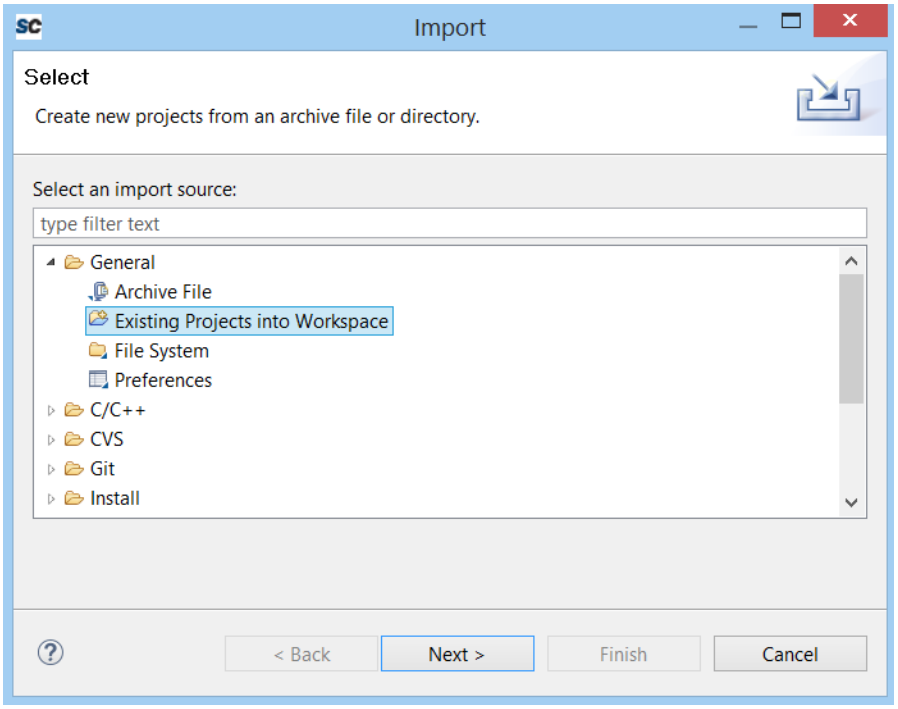
3. In the SoftConsole, go to **File** and select **Import** as shown in the following figure.

Figure 48 • **Import Options**



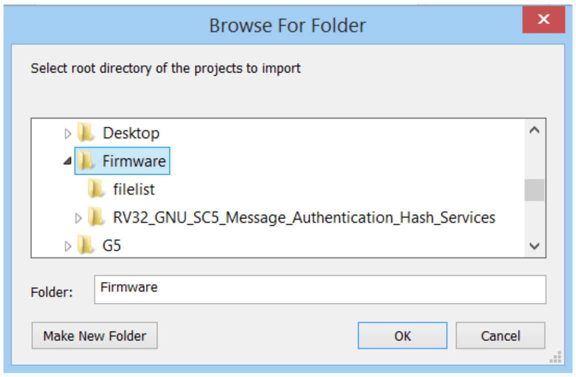
4. Select import source as **Existing Projects into Workspace** and click **Next**, as shown in the following figure.

Figure 49 • **Select An Import Source**



5. In the Import dialog, click **Browse..** to locate the generated sample project in the local PC folder and click **OK**, as shown in the following figure.

Figure 50 • **Import Dialog**



Browse For Folder

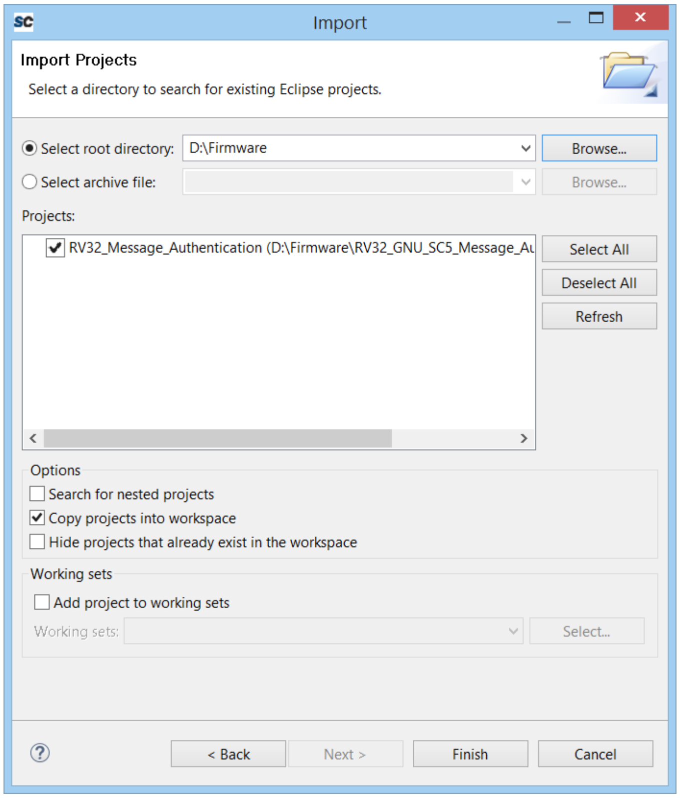
Select root directory of the projects to import

- Desktop
- Firmware
  - filelist
  - RV32_GNU_SC5_Message_Authentication_Hash_Services
- G5

Folder: Firmware

6. Ensure that the generated project is selected and click **Finish** to import the generated sample project in a SoftConsole workspace, as shown in the following figure.

Figure 51 • **Importing RV32_Message_Authentication Project**



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: D:\Firmware

○ Select archive file:

Projects:

☑ RV32_Message_Authentication (D:\Firmware\RV32_GNU_SC5_Message_Au

Options

□ Search for nested projects

☑ Copy projects into workspace

□ Hide projects that already exist in the workspace

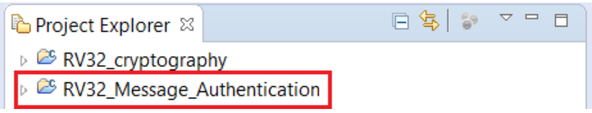
Working sets

□ Add project to working sets

Working sets:

7. The new sample project is imported in the SoftConsole workspace, as shown in the following figure.

Figure 52 • **SoftConsole Workspace—Sample Projects**



Project Explorer

- RV32_cryptography
- RV32_Message_Authentication

8. See Software Implementation, page 11 to make necessary changes to the imported sample project.
9. After making necessary changes, right-click on the imported sample project and click **Build Project** to build the project.
10. Start the SoftConsole debugger to run the project.See Running Tera Term Macro Script, page 26 for running the macro script provided in the sample project.

4 Appendix 2: User Cryptoprocessor Simulation

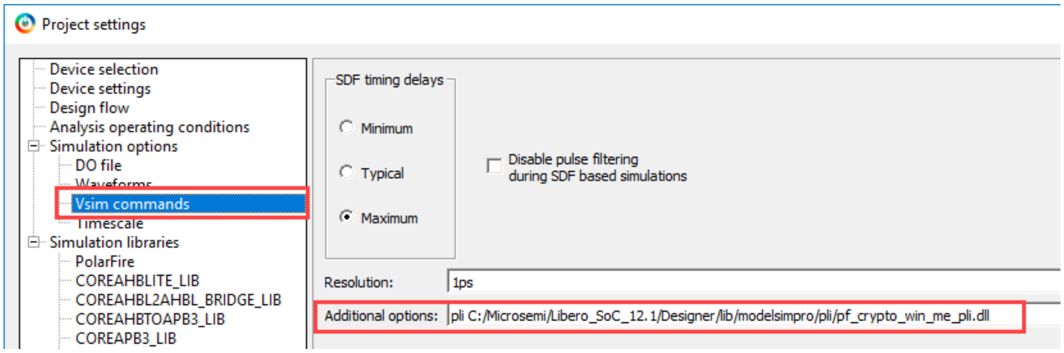
Microsemi Libero SoC provides a PLI simulation library for the User Cryptoprocessor to show the functional behavior of the User Cryptoprocessor. The PLI library for User Cryptoprocessor is available at *<Libero_Installation_Directory>/Designer/lib/modelsimpro/pli*. The PLI library must be passed to the ModelSim using VSIM command for User Cryptoprocessor simulation. The VSIM command can be set in Libero Project settings under Simulation options.

- VSIM command for Windows:
  -pli <$Libero_Installation_Directory>/Designer/lib/modelsimpro/pli/pf_crypto_win_me_pli.dll
- VSIM command for Linux:
  -pli <$Libero_Installation_Directory>/Designer/lib/modelsimpro/pli/pf_crypto_lin_me_pli.so

Edit <*Libero_Installation_Directory*> to match the location of Libero SoC on the host PC.

In the following figure, the Libero installation folder is C:/Microsemi/Libero_SoC.

Figure 53 • **Libero Project settings—VSIM Command for User Cryptoprocessor Simulation**



Project settings

- Device selection
- Device settings
- Design flow
- Analysis operating conditions
- Simulation options
  - DO file
  - Waveforms
  - Vsim commands
  - Timescale
- Simulation libraries
  - PolarFire
  - COREAHBLITE_LIB
  - COREAHBL2AHBL_BRIDGE_LIB
  - COREAHBTOAPB3_LIB
  - COREAPB3_LIB

SDF timing delays

○ Minimum
○ Typical
● Maximum

□ Disable pulse filtering during SDF based simulations

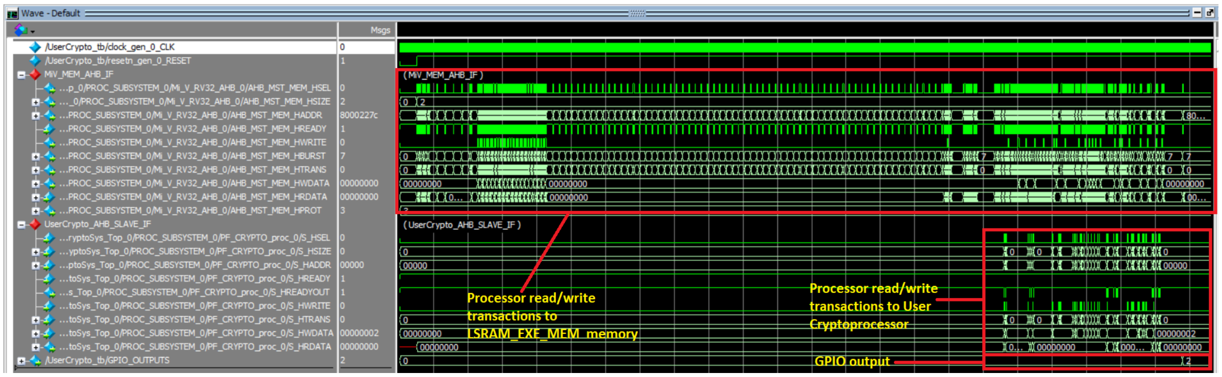
Resolution: 1ps

Additional options: pli C:/Microsemi/Libero_SoC_12.1/Designer/lib/modelsimpro/pli/pf_crypto_win_me_pli.dll

The simulation steps include:

1. Generating the top-level component, which includes the Mi-V processor system with PF_CRYPTO core in it.
2. Build a Mi-V application with required User Cryptoprocessor functions. User Cryptoprocessor functions are accessible through Athena TeraFire CAL driver.
3. Import the Mi-V application hex file into the designated TCM for execution.
4. Create a testbench for the complete processor system.
5. Simulate the complete processor system to execute the imported application image. You can observe that the Mi-V processor sends the commands and data to the User Cryptoprocessor, and the User Cryptoprocessor responds with the result, as shown in the following figure.

Figure 54 • **User Cryptoprocessor Simulation**



Wave - Default

	Msgs
/UserCrypto_tb/clock_gen_0_CLK	0
/UserCrypto_tb/resetn_gen_0_RESET	1
MiV_MEM_AHB_IF	
...p_0/PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HSEL	0
..._0/PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HSIZE	2
...PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HADDR	8000227c
...PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HREADY	1
...PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HWRITE	0
...PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HBURST	7
...PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HTRANS	0
...PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HWDATA	00000000
...PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HRDATA	00000000
...PROC_SUBSYSTEM_0/Mi_V_RV32_AHB_0/AHB_MST_MEM_HPROT	3
UserCrypto_AHB_SLAVE_IF	
...ryptoSys_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HSEL	0
...yptoSys_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HSIZE	0
...ptoSys_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HADDR	00000
...toSys_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HREADY	1
...s_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HREADYOUT	1
...toSys_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HWRITE	0
...toSys_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HTRANS	0
...toSys_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HWDATA	00000002
...toSys_Top_0/PROC_SUBSYSTEM_0/PF_CRYPTO_proc_0/S_HRDATA	00000000
/UserCrypto_tb/GPIO_OUTPUTS	2

(MiV_MEM_AHB_IF)
(UserCrypto_AHB_SLAVE_IF)
Processor read/write transactions to LSRAM_EXE_MEM_memory
Processor read/write transactions to User Cryptoprocessor
GPIO output

5 Appendix 3: 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_ac464_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.