

Using Zeroization in SmartFusion2 and IGLOO2 Devices - Libero SoC v2024.2

Table of Contents

Purpose	1
Introduction	1
References	1
Design Requirements	2
SmartFusion2 and IGLOO2 Zeroization Features	3
SmartFusion2 and IGLOO2 Zeroization Procedures	4
SmartFusion2 and IGLOO2 Zeroization Options	4
Using Zeroization in SmartFusion2 and IGLOO2 Devices	7
Using Tamper Macro to Set Zeroization Option	7
Zeroization Request from FPGA Fabric through Tamper Macro	10
Zeroization Request through COMM_BLK using Cortex-M3 processor or FPGA Fabric	11
Design Description	14
Design Example 1: Zeroize_M2GL090 Design	14
Design Example 2: Zeroize_M2S090 Design	22
Conclusion	25
Appendix A: Design Files	26
Appendix B: Running SmartDebug	27
List of Changes	29

Purpose

This application note describes how to use the built-in zeroization features and options of SmartFusion[®]2 system-on-chip (SoC) field programmable gate array (FPGA) and IGLOO[®]2 FPGA devices.

Introduction

Zeroization is a method of erasing electronically stored data by altering or deleting the contents of the data storage to prevent the recovery of the data. It is a key feature in several security applications where high-value assets are at risk.

Zeroization plays a key role in both financial and military applications, it also plays a key role in several consumer, commercial, and industrial applications, where it is desired that the intellectual property comprising the design itself is to be kept confidential.

The zeroization capability provides the designer with the ability to destroy the data in circumstances where a tamper event is detected. The SmartFusion2 and IGLOO2 devices have built-in, easy to use tamper detection and response capability.

The user can monitor the SmartFusion2 and IGLOO2 devices tamper detection flags and send zeroization request to the system controller and erase the content inside the FPGA. The zeroization requests to the system controller can be sent directly from the user logic in the FPGA fabric through the user services interface (USI) or as system service from FPGA fabric or ARM[®] Cortex[®]-M3 processor (SmartFusion2 only) through the COMM_BLK. The zeroization feature is supported in both S and non-S versions of the SmartFusion2 and IGLOO2 devices.

References

The following documents are referenced in this document:

- [UG0331: SmartFusion2 Microcontroller Subsystem User Guide](#)
- [UG0443: SmartFusion2 and IGLOO2 FPGA Security and Reliability User Guide](#)
- [UG0541: SmartFusion2 SoC FPGA Evaluation Kit User Guide](#)
- [UG0448: IGLOO2 FPGA High Performance Memory Subsystem User Guide](#)
- [UG0478: IGLOO2 FPGA Evaluation Kit User Guide](#)
- [Microsemi EnforcIT Security Monitor Product Overview](#)

Note: A zeroization technical report is available under NDA and on a need-to-know basis.

Design Requirements

[Table 1](#) shows the SmartFusion2 reference design requirements and details.

Table 1 • SmartFusion2 Reference Design Requirements and Details

Reference Design Requirements and Details	Description
Hardware Requirements	
SmartFusion2 Security Evaluation Kit (M2S090TS-EVAL-KIT) • 12 V adapter (provided along with the kit) • FlashPro4 programmer (provided along with the kit) • M2S090TS-1FGG484 ¹	Rev D or later
Host PC or Laptop	Any 64-bit Windows Operating System
Software Requirements	
Libero® System-on-Chip (SoC)	v11.6
SoftConsole	v3.4 SP1

[Table 2](#) shows the IGLOO2 reference design requirements and details.

Table 2 • IGLOO2 Reference Design Requirements and Details

Reference Design Requirements and Details	Description
Hardware Requirements	
IGLOO2 Evaluation Kit (M2GL-EVAL-KIT) • 12 V adapter (provided along with the kit) • FlashPro4 programmer (provided along with the kit) • M2GL090TS-1FGG484 ^{1, 2}	Rev C or later
Host PC or Laptop	Any 64-bit Windows Operating System
Software Requirements	
Libero SoC	v11.6

Notes:

1. For more information about "Zeroization Support", refer to the device errata document.
2. The IGLOO2 design uses M2GL090TS-1FGG484 device in the IGLOO2 Evaluation Kit. However, the official IGLOO2 Evaluation Kit uses M2GL010T-1FGG484 device. If you want to run the application note design in M2GL010T-1FGG484, refer to the [KB5659](#) for migrating M2GL090TS-1FGG484 to M2GL010T-1FGG484.

SmartFusion2 and IGLOO2 Zeroization Features

The SmartFusion2 and IGLOO2 devices support two types of zeroization options: Like New and Unrecoverable. For more information about these two zeroization options, refer to "[SmartFusion2 and IGLOO2 Zeroization Options](#)" section on page 4. The user can monitor the tamper detection flag and then decide to trigger one of the two types of built-in zeroization request to zeroize the device or activate one of the other predefined or user customized tamper responses.

Zeroization is a high priority system service to the system controller in the SmartFusion2 and IGLOO2 devices. When the system controller receives the high priority Zeroization command, it stops the execution of a low priority command and service the Zeroization command.

The low priority command is aborted.

Figure 1 shows a block diagram of the SmartFusion2 and IGLOO2 devices. The user should enable zeroization and set zeroization option (using Tamper macro) and then send a zeroization request from FPGA fabric to system controller through the USI. Zeroization request can also be sent through the COMM_BLK from FPGA fabric (for both the SmartFusion2 and IGLOO2 devices) or Cortex-M3 processor (for SmartFusion2 only).

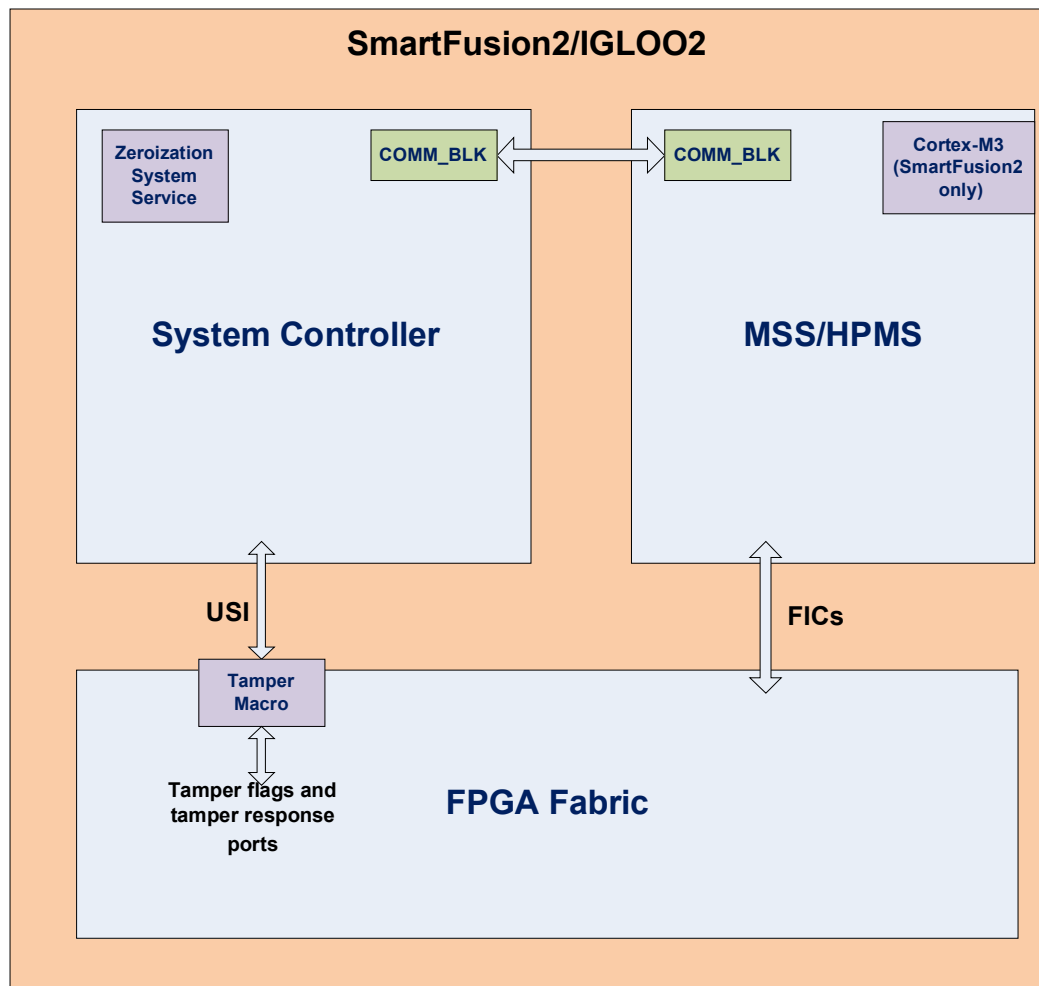


Figure 1 • SmartFusion2 and IGLOO2 Block Diagram

SmartFusion2 and IGLOO2 Zeroization Procedures

In the SmartFusion2 and IGLOO2 devices, zeroization includes several erase and programming operations to reduce any data remnants in the flash array to undetectable levels (that is, scrubbing). This section describes zeroization procedure in the SmartFusion2 and IGLOO2 devices.

After receiving zeroization request, the system controller programs a set of non-volatile bits that act as status flags during zeroization process. These status bits get updated throughout zeroization process to maintain the current status.

These status bits are internal to FPGA, and are not available to the user. This procedure cannot be bypassed once invoked. It is designed to complete even in the event of intentional tampering.

If zeroization cycle is interrupted prior to completion, the system controller resumes zeroization on the next power-up of the device based on the state of the status bits. Along with programming of status bits, the system controller initiates zeroization of the volatile memories of the FPGA first (that is, ECC memory, PUF and PUF key management memory, Key buffers, Frame buffers, AES and SHA256 registers, JTAG I/O buffer, MSS eSRAMs, and Fabric SRAMs), then the system controller performs verification by reading back these memories, computing, and validating a digest. After the volatile memories of the FPGA are zeroized, the non-volatile memory segments are destroyed starting with eNVM.

The FPGA flash configuration array, user security segments, and factory security segments (if applicable, depending on zeroization option selected) are erased. After all the relevant memories are cleared and scrubbed, the system controller performs a read back operation on the memory segments and calculates a digest for volatile memories, eNVM, security segments, and the fabric configuration flash. After both the erase and verify portions of zeroization are complete, a secure protocol exists to confirm if zeroization is completed successfully.

SmartFusion2 and IGLOO2 Zeroization Options

The SmartFusion2 and IGLOO2 devices have several types of zeroization options to meet the user application requirements. [Table 3](#) shows the various zeroization options available in the SmartFusion2 and IGLOO2 devices. These options zeroize various FPGA components and security segments based on user settings. This application note assumes user is familiar with the SmartFusion2 and IGLOO2 devices security segments and programming model, which are described in the [UG0443: SmartFusion2 and IGLOO2 FPGA Security and Reliability User Guide](#).

Table 3 • SmartFusion2 SoC and IGLOO2 FPGAs Zeroization Options

Zeroization Option	What Remains Intact	Comments
Like New	Factory keys and factory configuration segments.	The FPGA configuration is destroyed and the part behaves similar to a new part from the factory.
Unrecoverable	Nothing stays.	All the configurations are destroyed and the device is permanently disabled and unusable.

Like New Zeroization Mode

The first zeroization option is **Like New**, which destroys all the user programmed configuration data (that is, it erases the user design information). This mode erases the device into a new device state. In other words, the device behaves as it was when it was shipped from Microsemi originally, with no user design or key information stored and is immediately ready for programming by the user.

There is no logistical impact beyond reprogramming the device to get back into operation. Selecting the **Like New** zeroization mode may not be advisable for some high security applications, like a new part, once zeroized the part can be reprogrammed by whoever has access to it.

The **Like New** zeroization mode is useful when devices go through repeated zeroization, such as when an information assurance device is being intentionally erased after a mission, for routine re-keying.

During zeroization, a number of erase and write cycles are applied to the configuration NVM. Also, due to the scrubbing process, each zeroization and subsequent reprogramming uses several erase and write cycles. So, it is recommended not to use frequent zeroization.

Refer to the datasheet for the number of allowable zeroization cycle.

Unrecoverable Zeroization Mode

When **Unrecoverable** zeroization system services are activated, all the data in all NVMs in the device is destroyed (excluding metal-mask ROMs).

The following factory keys are erased:

- Factory key (FK)
- Default key-loading key (KLK)
- Factory ECC private key (SKFE, in larger devices)
- Pseudo-PUF secret (FPP)

Note: Also, one-half of the device serial number (DSN), namely the 64-bit serial number modifier (SNM), is erased.

The eNVM data erased includes the factory segment, which is retained in the Like New Zeroization Mode. The calibration data used to program devices is erased, and then the device intentionally puts itself into an unrecoverable state where it does not respond to any programming commands. The zeroization certification protocol is still active.

Table 4 provides a summary of each block within the FPGA that is erased by zeroization, along with, which zeroization option initiates the erase and how it is cleared.

Table 4 • FPGA Components or Blocks Erased by Zeroization

FPGA Component or Block	Erase Protocol	Applicable Zeroization Options
System Controller Memory (including memories associated with ECC + PUF)	Erased by zeroization command	All Active Zeroization options
Programming Frame, Key, and JTAG I/O Buffers	Actively Cleared	All Active Zeroization options
AES/SHA Accelerators – Registers and RAMs	Actively Cleared	All Active Zeroization options
MSS SRAM	Actively Cleared	All Active Zeroization options
Fabric SRAM	Actively Cleared	All Active Zeroization options
Fabric Registers	Removal of Power	All Active Zeroization options
Cortex-M3 processor Cache	N/A	N/A
Fabric Configuration Flash Cells	Erased + Scrubbed by zeroization command	All Active Zeroization options

Note: The non-volatile memory segments are erased during zeroization. The eNVM segments are erased and scrubbed. As certain eNVM segments contain factory provisioned data (such as, device certificates and public keys), full erasure of eNVM sectors are executed according to the zeroization options set by the user.

Table 5 provides a summary for the FPGA security segments.

Table 5 • Security Segments during Zeroization

FPGA Component	Erase Method	Applicable Zeroization Options
Security Segment – User Lock bits	Actively Cleared + Scrubbed	All Active Zeroization options
Security Segment – User Keys (UEK1&2, Passkeys)	Actively Cleared + Scrubbed	All Active Zeroization options
Security Segment – Factory Keys	Actively Cleared + Scrubbed	Unrecoverable Zeroization Options
Security Segment – Device unique factory provisioned data	Actively Cleared + Scrubbed	Unrecoverable Zeroization Option
eNVM – Factory Data	Actively Cleared	Unrecoverable Zeroization Options

Using Zeroization in SmartFusion2 and IGLOO2 Devices

This section describes the procedure to configure and run zeroization system service in the SmartFusion2 and IGLOO2 devices. The first step is to enable zeroization and set the zeroization option using the **Tamper** macro.

The **Tamper** macro exposes the built-in tamper detection and response signals and also allows the designer to enable zeroization and to set the zeroization option. Add the **Tamper** macro in the design in order to use zeroization feature.

Using Tamper Macro to Set Zeroization Option

To configure zeroization option, drag and configure the **tamper** macro from the IP catalog window, as shown in [Figure 2](#).

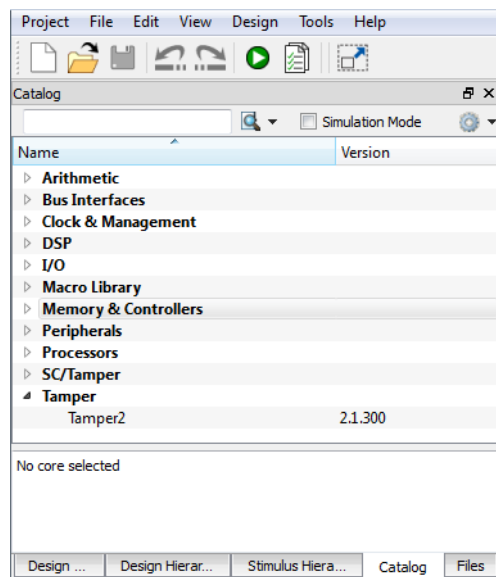


Figure 2 • Tamper Macro in IP Catalog Window

Figure 3 shows the option to enable and set zeroization option in **Tamper** macro configurator for the SmartFusion2 and IGLOO2 (M2S050 and M2GL050) and smaller devices.

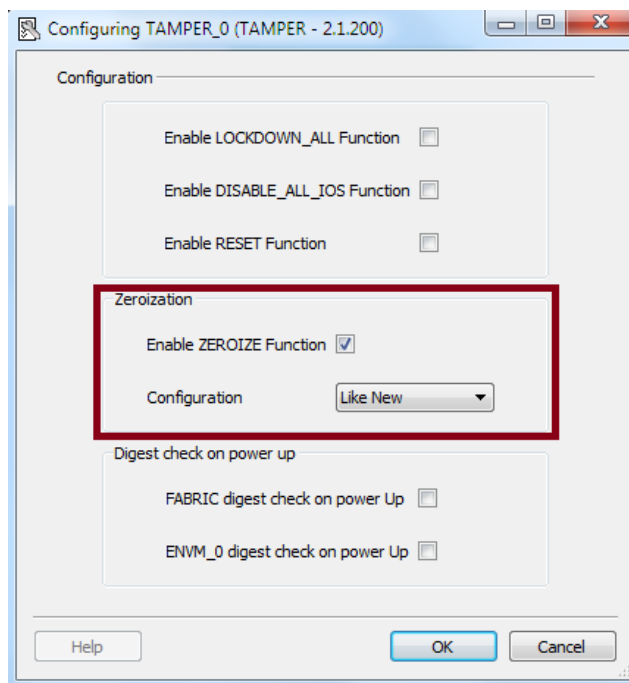


Figure 3 • Tamper Macro Configurator for SmartFusion2 and IGLOO2 (M2S050 and M2GL050) and Smaller Devices

Figure 4 shows the option to enable and set zeroization option in **Tamper** macro configurator for M2S090, M2S150, M2GL090, and M2GL150 devices.

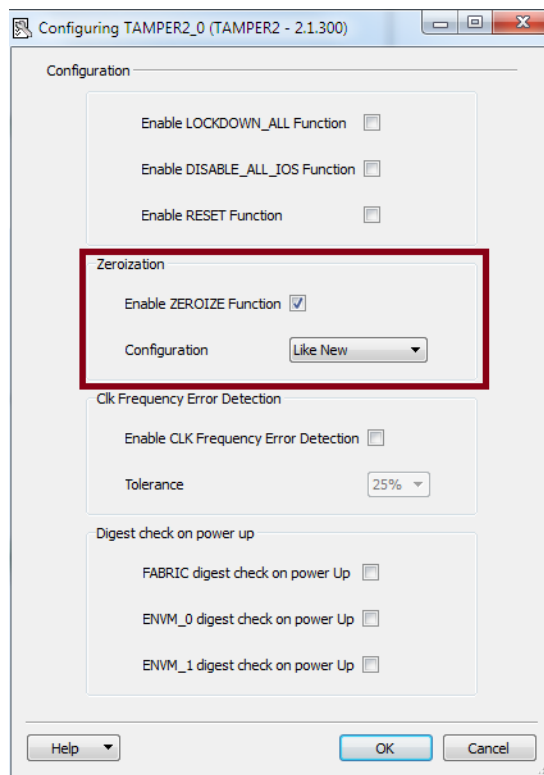


Figure 4 • Tamper Macro Configurator for M2S090, M2S150, M2GL090, and M2GL150 Devices

When zeroization is enabled and the zeroization option is set, user can send zeroization request in the following two ways:

- Send zeroization request from the FPGA fabric through the **tamper** macro.
- Send zeroization request through the COMM_BLK using Cortex-M3 processor or FPGA fabric.

Zeroization Request from FPGA Fabric through Tamper Macro

In the SmartFusion2 and IGLOO2 devices, zeroization request is triggered using the **Tamper** macro. The **Tamper** macro uses the USI bus to send zeroization request from the FPGA fabric to system controller, as shown in Figure 5.

The system controller block performs all the functions required for setting status flags and executing zeroization process.

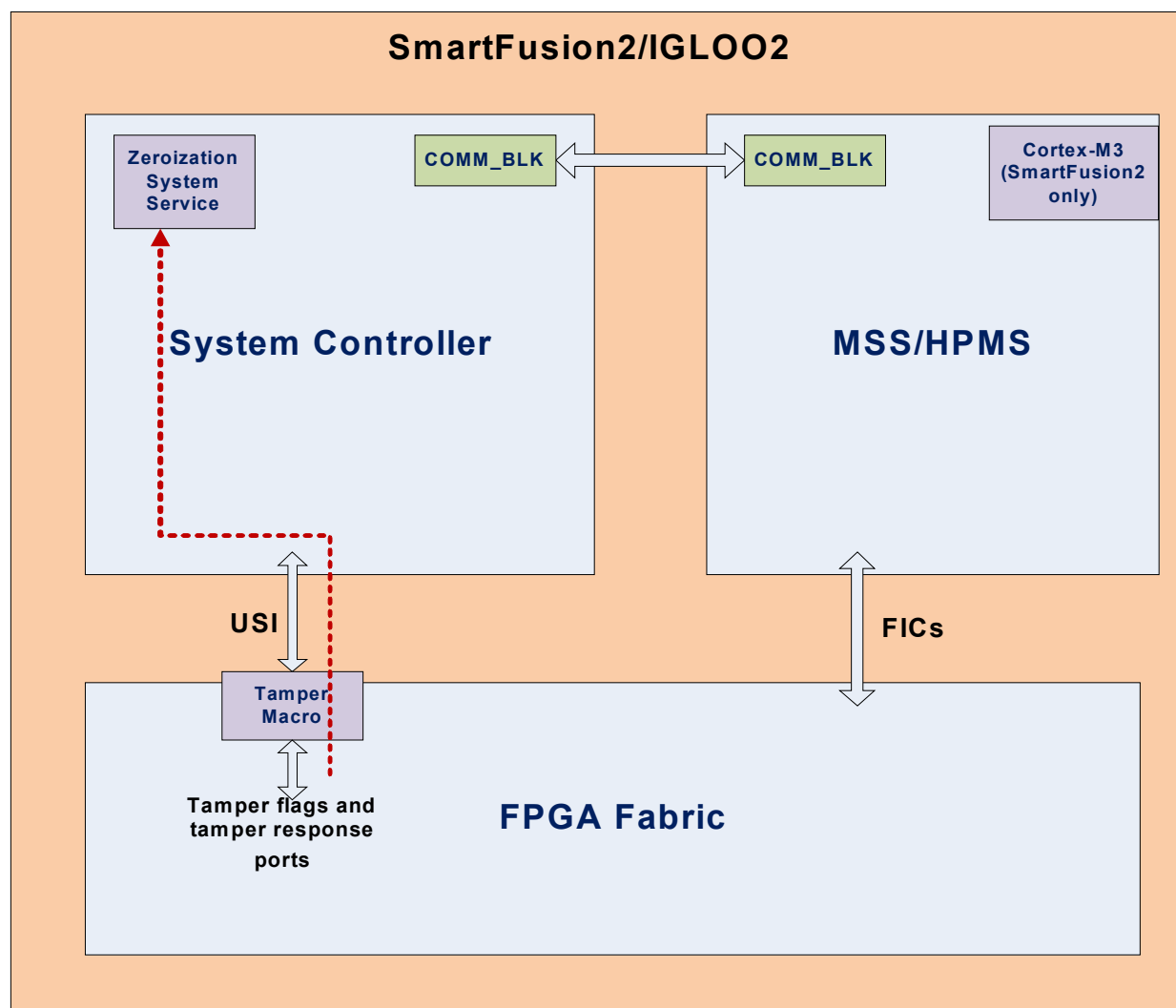


Figure 5 • Zeroization Request through USI Bus

The **Tamper** macro exposes the built-in tamper detection and responses and also allows enabling zeroization and setting the zeroization option. It also provides one control pin (ZEROIZE_N) that initiates zeroization process.

Figure 6 shows the **Tamper** macro with zeroization input in Libero SoC software canvas. The user can apply an active low signal to ZEROIZE_N input and trigger zeroization.

Do not assert this pin. Microsemi recommends that the user use test modes and debug their logic connected to ZEROIZE_N to ensure proper functionality prior to connecting it to the **Tamper** macro ZEROIZE_N pin.

Note: During debug, the part requires re-programming after zeroization.

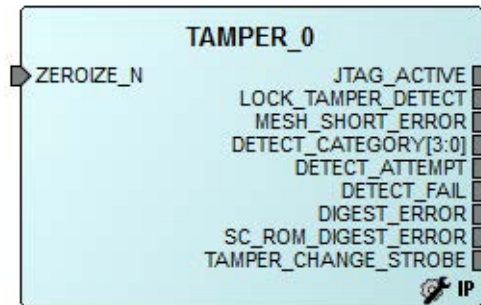


Figure 6 • Tamper Macro with Zeroization Input in Libero SoC Software Canvas

Zeroization Request through COMM_BLK using Cortex-M3 processor or FPGA Fabric

The SmartFusion2 zeroization is initiated through the communication block (COMM_BLK) in the microcontroller subsystem (MSS). There are two COMM_BLK instances one in the MSS and the other in the system controller, which communicate with each other.

The COMM_BLK consists of an APB interface, an eight byte transmit FIFO, and an eight byte receive FIFO. Refer to COMM_BLK chapter in [UG0331: SmartFusion2 Microcontroller Subsystem User Guide](#).

The COMM_BLK provides a bi-directional message passing facility between the MSS and the system controller. To initiate zeroization, set up the MSS COMM_BLK register in byte mode and send the zeroization command (0xF0). The system controller reads the command from its COMM_BLK interface port and use the zeroization option set by the **Tamper** macro and start zeroization. The user can send zeroization system service request either from Cortex-M3 processor or through a FPGA fabric master.

Figure 7 shows the SmartFusion2 zeroization initiation flow and the blocks used for initiating zeroization in this method.

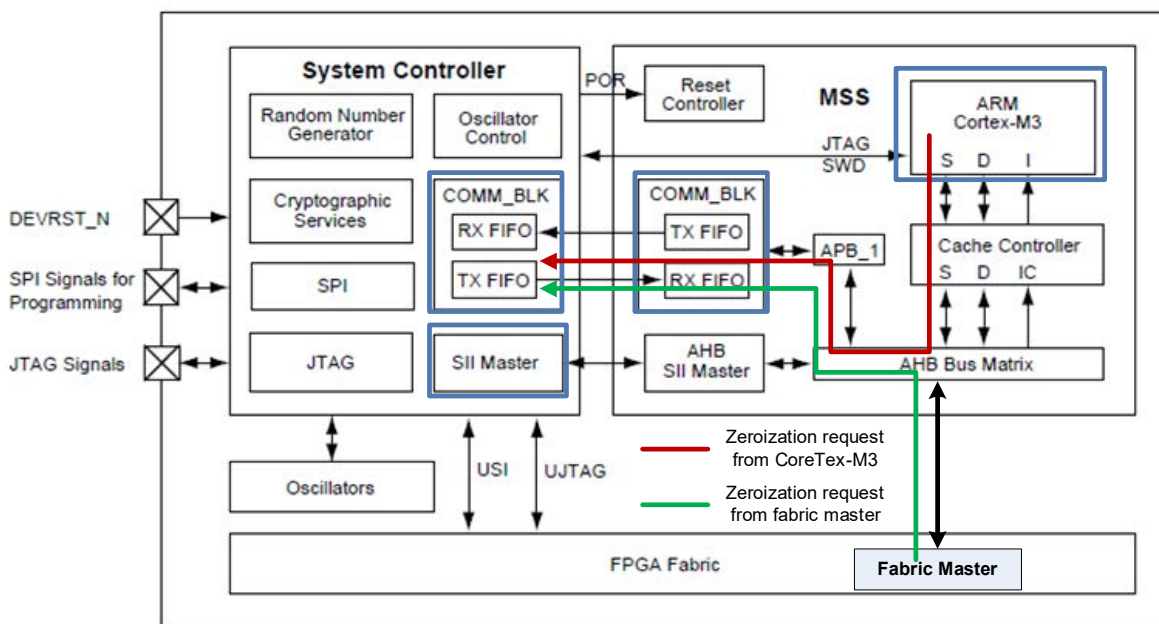


Figure 7 • System Controller Block in SmartFusion2 Device

Microsemi provides system service driver and CoreSysServices soft IP to run zeroization operation in SmartFusion2. As a result, the user need not create software code from scratch in M3 or create a complex master in fabric to run zeroization system services.

CoreSysServices provides a simple user interface in one side and an AHB-Lite master interface on the FIC side to use any system services through the COMM_BLK.

For more information about CoreSysServices, refer to the CoreSysServices IP Handbook, which can be accessed through Libero SoC software.

In IGLOO2, the COMM_BLK in the system controller communicates with COMM_BLK in high-performance memory subsystem (HPMS). To initiate zeroization through COMM_BLK in IGLOO2, use a fabric master.

Microsemi provides the CoreSysServices Directcore IP that acts as fabric master to use zeroization service. The CoreSysServices soft IP communicates with the COMM_BLK through one of the FICs, and sends zeroization service request.

Figure 8 shows the IGLOO2 zeroization request flow and the blocks used for initiating zeroization system service in this method.

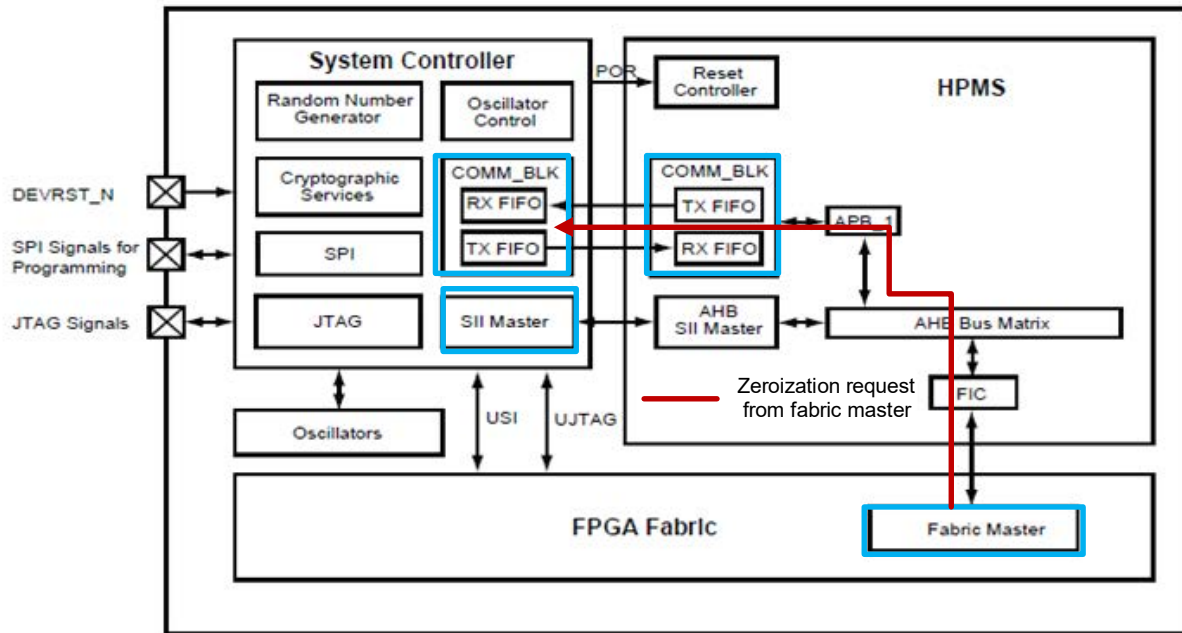


Figure 8 • System Controller Block in the IGLOO2 Device

The following steps describe how to run zeroization:

Step1: Use **Tamper** macro and turn on zeroization and set the zeroization option.

Step2: Send zeroization request using the following method:

- SmartFusion2 has three options to send zeroization request:
 - Send through the USI from FPGA fabric
 - Send through the COMM_BLK using Cortex-M3 as a master
 - Send through the COMM_BLK using a fabric master or CoreSysServices IP
- IGLOO2 has two options to send zeroization request:
 - Send through the USI from FPGA fabric
 - Send through the COMM_BLK using a fabric master or CoreSysServices IP

Use **Tamper** macro to send zeroization request, as this provides a simple and easy to use interface to the user.

Design Description

This application note includes two design examples for demonstrating zeroization:

- Zeroize_M2GL090 design example. Demonstrates running zeroization system service in the IGLOO2 device using the **tamper** macro.
- Zeroize_M2S090 design example. Demonstrates running zeroization system service in the SmartFusion2 device using the system driver firmware code supplied by Microsemi and also using the **tamper** macro.

The SmartFusion2 device design is implemented on the SmartFusion2 Security Evaluation Kit Board using an M2S090TS-1FGG484 device and the IGLOO2 device design is implemented on the IGLOO2 Evaluation Kit Board using an M2GL090TS-1FGG484 device.

Design Example 1: Zeroize_M2GL090 Design

This design allows zeroizing IGLOO2 device using ZEROIZE_N input in the **tamper** macro. The design example uses eNVM, SRAM, and FPGA fabric. The ZEROIZE_N input is triggered through the USER_TAMPER_FLAG_0 input (connected to SW3) or using a tamper detect event.

The FlashPro4/5 read_idcode operation generates a tamper detection flag and initiates zeroization service through the ZEROIZE_N input. The FPGA components are zeroized using **Like New** option.

The design example allows viewing eNVM content before and after zeroization.

Hardware Implementation

Figure 9 shows the top level block diagram of the IGLOO2 design example. It includes the following components:

- **TAMPER_0**: This is the **tamper** macro, configured with **Like New** option. The **tamper** macro exposes the tamper response input, called ZEROIZE_N. It also exposes the various tamper detection flags. In the design example, the JTAG_ACTIVE output is used as one of the input to initialize zeroization.
- **Zeroize_interface_0**: This block uses the USER_TAMPER_FLAG_0 and JTAG_ACTIVE signals and generate ZEROIZE_N signal. The other tamper detection flags are connected to this block, but not used in this design example. In addition, the JTAG_FLAG_GATING_SIGNAL is used to block the JTAG_ACTIVE signal from accidentally zeroizing the device.
- **RAM_init_top_0**: This block includes HPMS block and clock input. It also includes RAM_init_top_0 block to store the eNVM data after power up, as shown in Figure 9. The AHBMASTER_FIC_0 inside RAM_init_top_0 block acts as an AHB master and moves data from eNVM to SRAM after power up. The RAM_init_top_0 uses 50 MHz RC oscillator as clock input and a fabric PLL is used to generate a 100 MHz clock from the 50 MHz RC oscillator clock. This 100 MHz clock is used as the base clock for the HPMS and also for the fabric.
- **Counter28_0**: The counter block is used to show that the device is up and running.

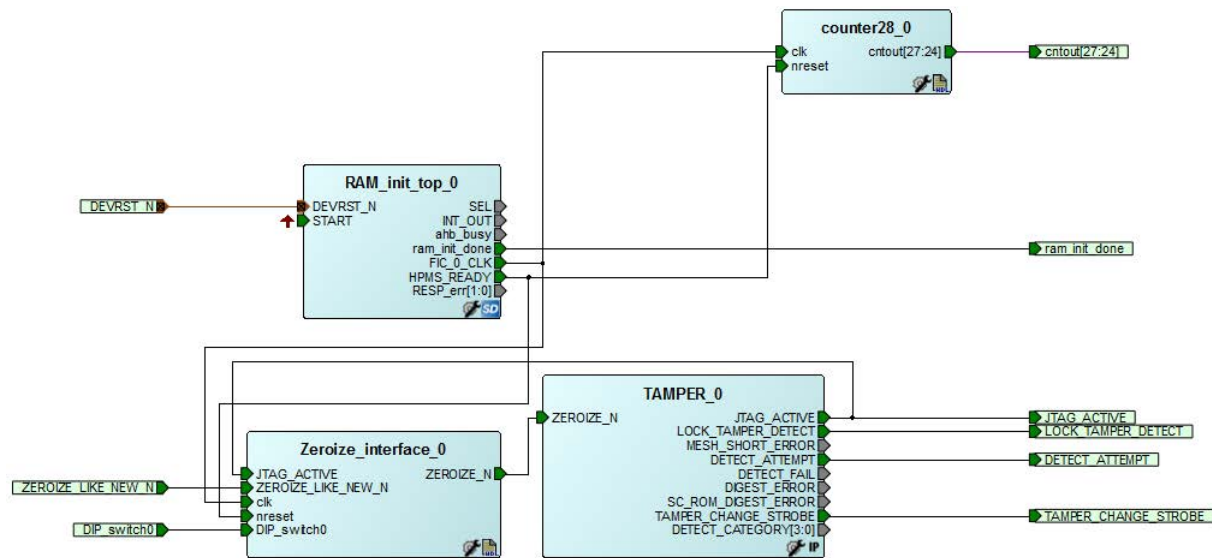


Figure 9 • Zeroize_M2GL090 Top Level Block Diagram

Figure 10 shows the RAM_init_top_0 sub-block. It includes HPMS, 50 MHz RC oscillator, fabric PLL and some fabric logic.

The AHBMASTER_FIC_0 inside RAM_init_top_0 block acts as an AHB master and moves data from the eNVM to SRAM after power up. The RAM_init_top_0 uses 50 MHz RC oscillator as clock input and a fabric PLL is used to generate a 100 MHz clock from the 50 MHz RC oscillator clock. This 100 MHz clock is used as the base clock for the HPMS (my_hpms_top_0) and also for the fabric.

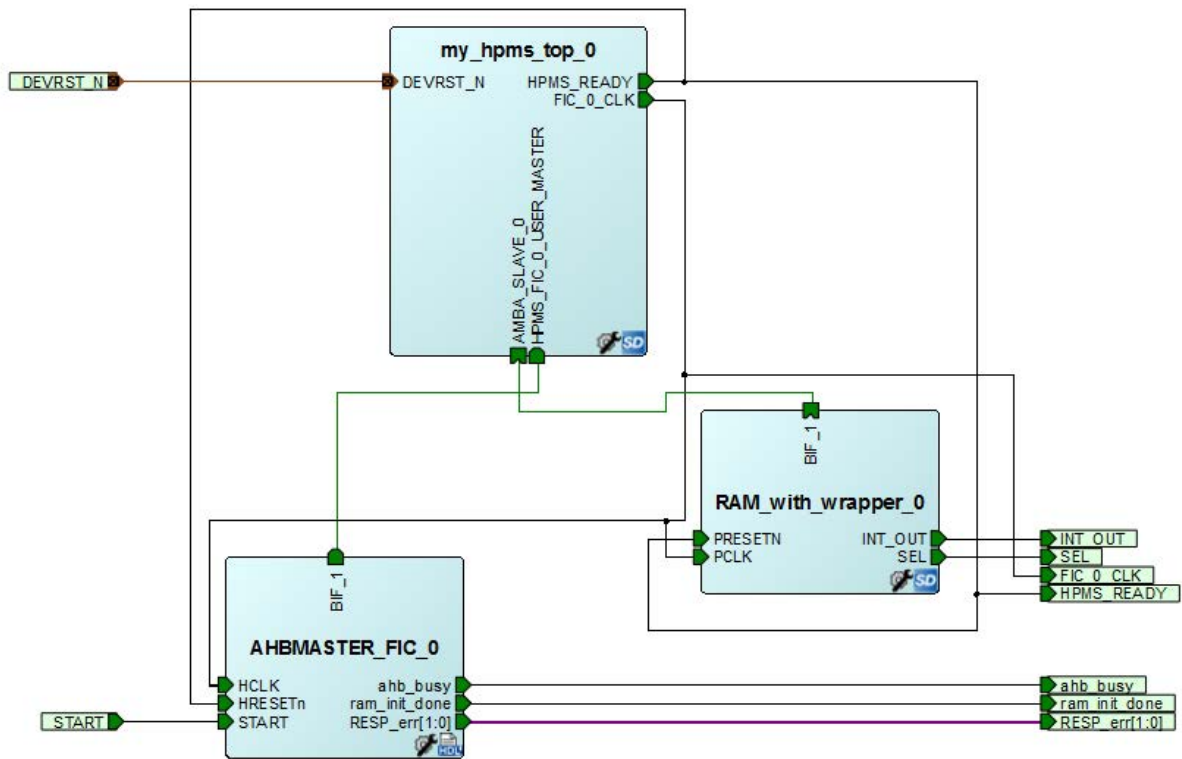


Figure 10 • RAM_init_top Block

Running the Design

The following steps describe how to run the design example on the M2GL_M2S-EVAL Kit board using the M2GL090TS-1FGG484 device:

1. Connect the power supply to the IGLOO2 Evaluation Kit with M2GL090TS-1FGG484 Device.
2. Plug the FlashPro4 ribbon cable into JTAG Programming Header on the SmartFusion2 Security Evaluation Kit board.
3. Program the SmartFusion2 Evaluation board with the provided STAPL file and power cycle the device. For more information about STAPL file, refer to "[Appendix A: Design Files](#)" section on [page 26](#).

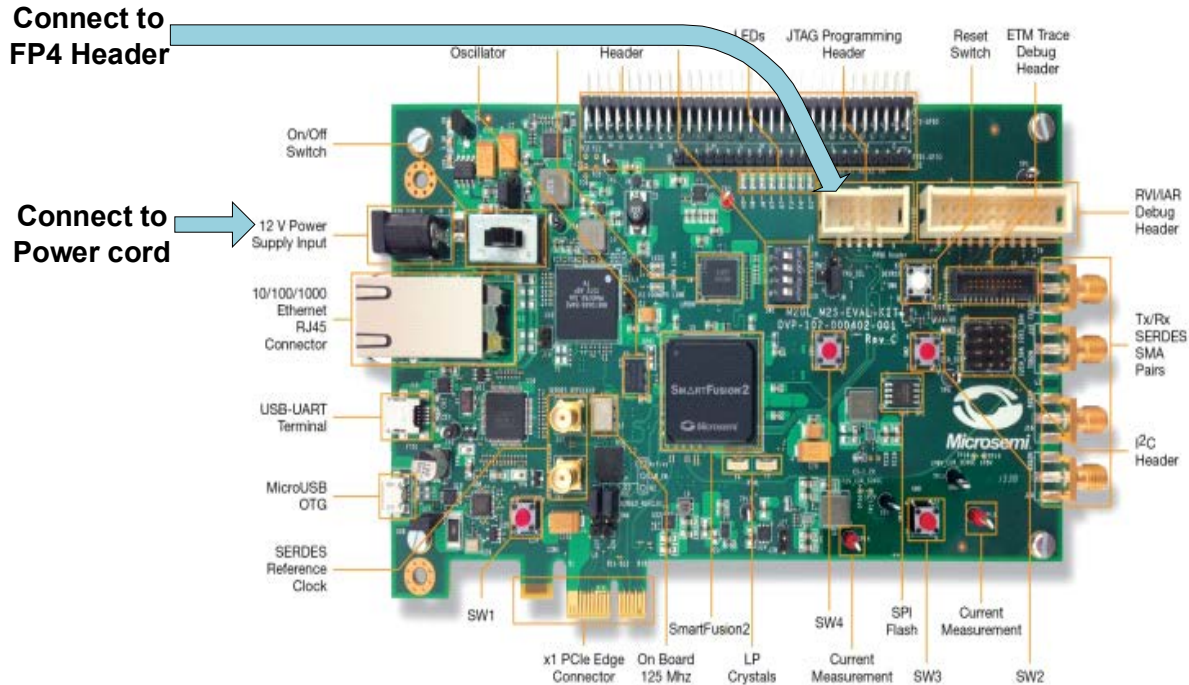


Figure 11 • IGLOO2 Evaluation Kit with M2GL090TS-1FGG484 Device

4. Read the eNVM or SRAM content using the steps shown in "[Appendix B: Running SmartDebug](#)" section on [page 27](#). Power cycle the device after you exit from SmartDebug tool, so that JTAG port gets de-activated.

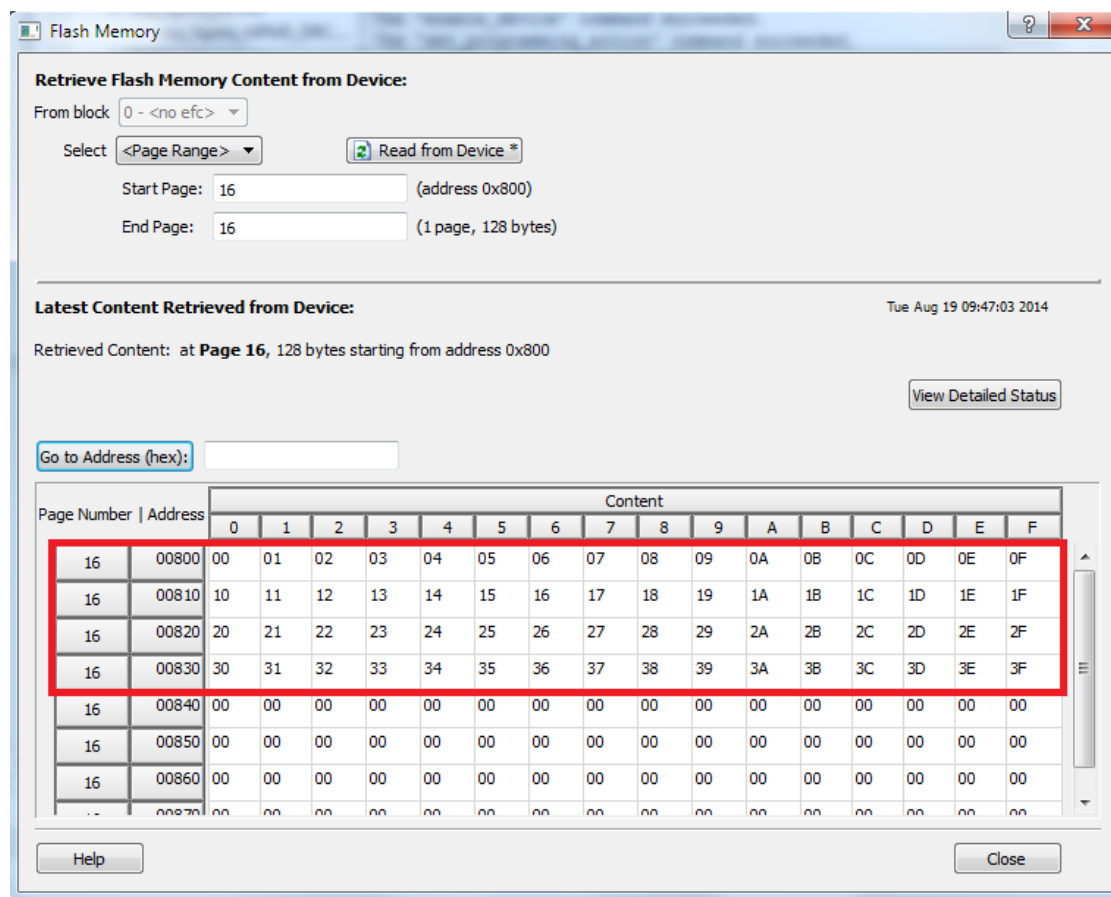


Figure 12 • SmartDebug Showing eNVM Content before Zeroization

5. Change the DIP switch 1 to ON to allow JTAG_ACTIVE to initiate zeroization. This DIP switch 1 signal is used as gating logic for JTAG_ACTIVE signal.

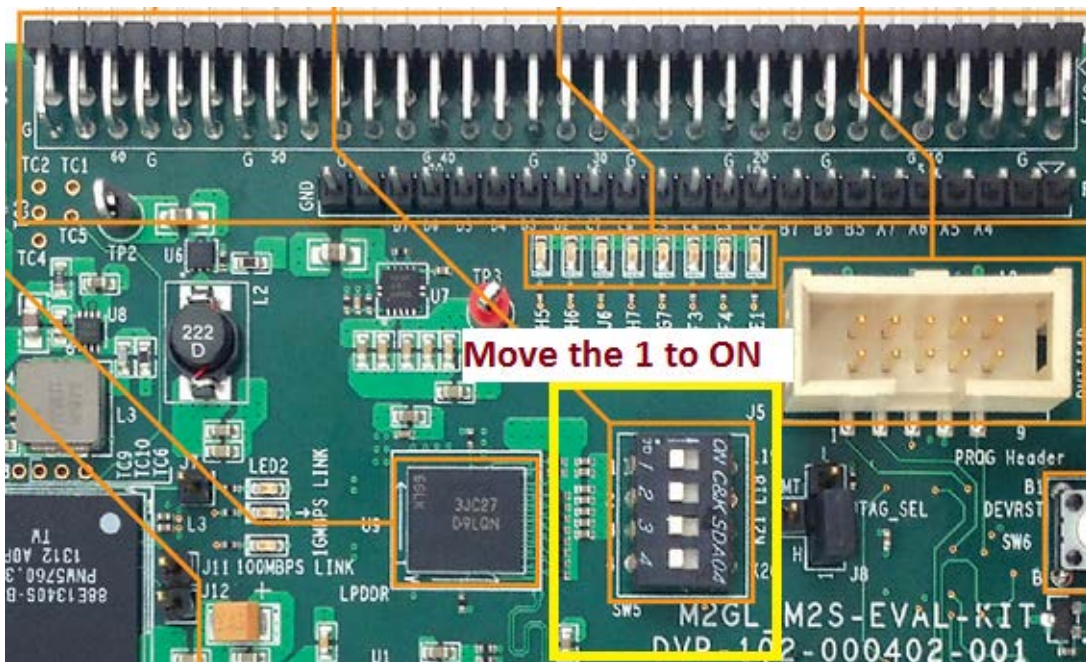


Figure 13 • DIP Switch in M2GL_M2S-EVAL Kit Board

6. Right-click on **Run PROGRAM Action** and select **Configure Action/Procedures**. Figure 15 shows **Select Action and Procedures** Window.

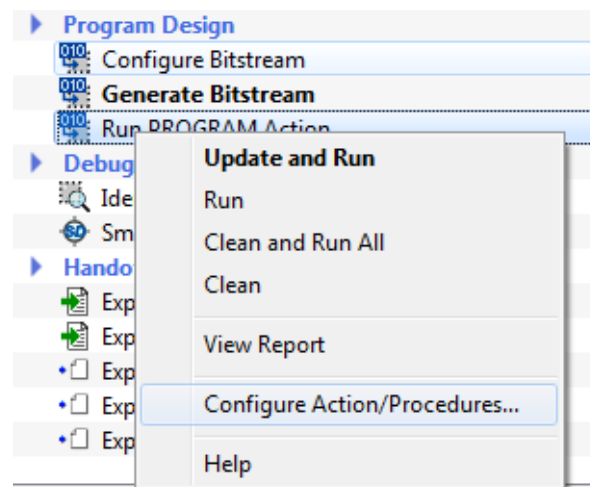


Figure 14 • Configuring Program Design Action

7. Select **READ_IDCODE** under **Action** drop-down list and click **OK**.

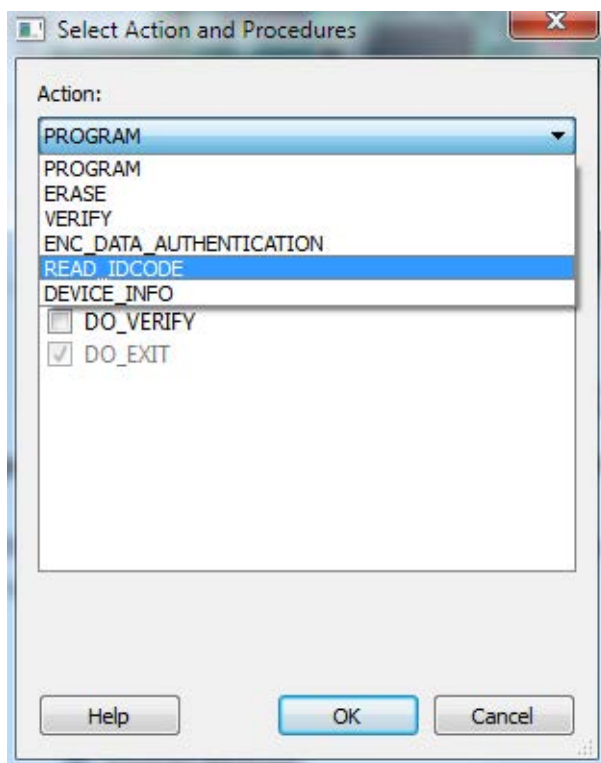


Figure 15 • Select Action and Procedures Window

The Program action changes to **Run READ_IDCODE Action**.

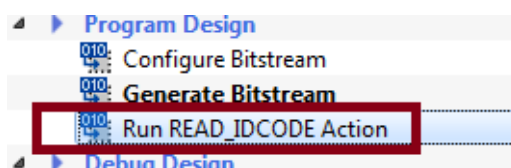


Figure 16 • Run READ_IDCODE Action

8. Double-click **Run READ_IDCODE Action** and system controller detects this action as tamper event and it activates the JTAG_ACTIVE signal. The JTAG_ACTIVE and DIP switch 1 signal asserts a low signal to **ZEROIZE_N** input and initiate zeroization.

The counter stops counting as the device gets zeroized. Power cycle the device and you can then read the eNVM content using SmartDebug, similar to step 4 and verify that the actual content is zeroized.

Figure 17 shows eNVM content after zeroization. Press the SW3 (connected to USER_TAMPER_FALG_0 signal) on the M2GL_M2S-EVAL Kit board to initiate zeroization instead of reading the IDCODE operation.

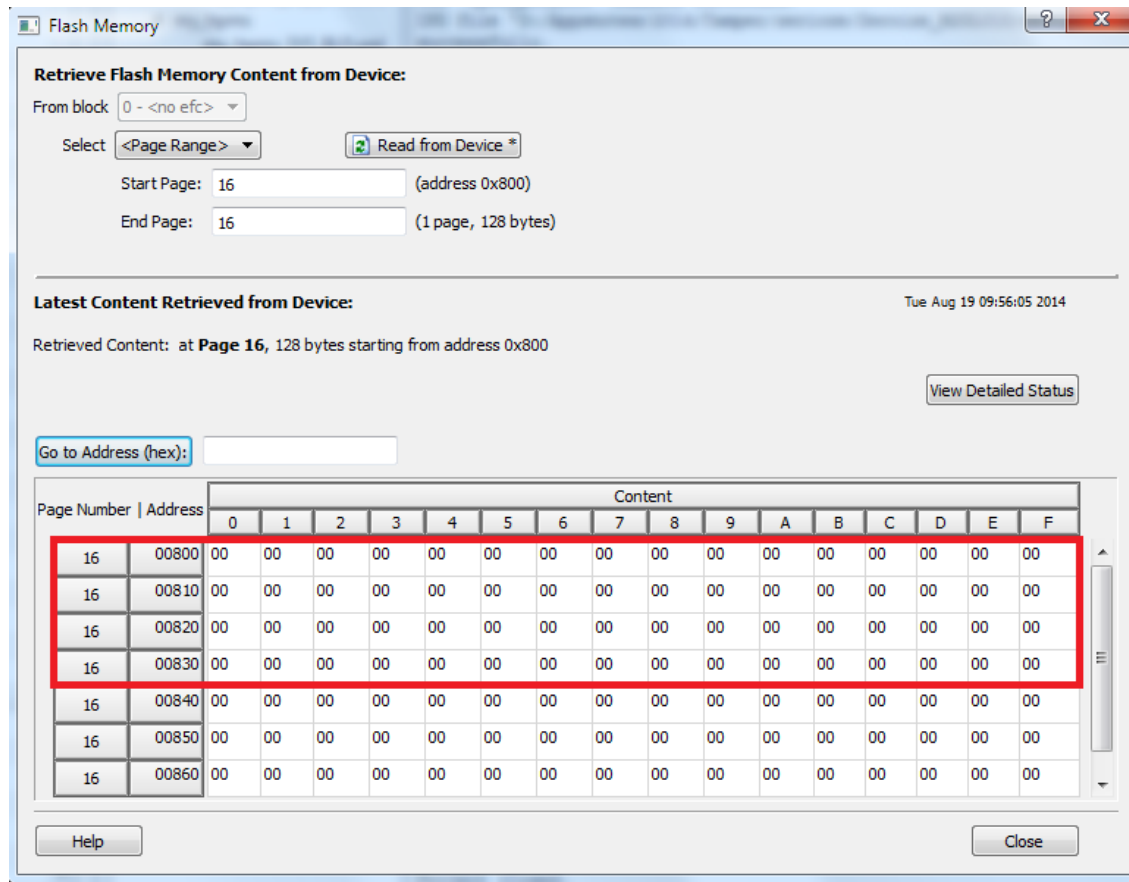


Figure 17 • SmartDebug Showing eNVM Content after Zeroization

Design Example 2: Zeroize_M2S090 Design

This design allows initiating zeroization in the SmartFusion2 device in two ways:

- Using C routine in the MSS
- Using ZEROIZE_N signal in tamper macro

A universal asynchronous receiver/transmitter (MMUART_0) in the MSS is used to display the initiation of zeroization system service when using C routine. When using ZEROIZE_N signal, the Flash pro READ_IDCODE action initiates zeroization service, similar to IGLOO2 design.

The design example uses eNVM, SRAM, and FPGA fabric similar to IGLOO2 design. These components zeroized using **Like New** option.

Hardware Implementation

Figure 18 shows the top level block diagram of the design example. It includes three main components:

- **TAMPER_0:** This is the **tamper** macro, configured with **Like New** option. The **tamper** macro exposes the tamper response input, called ZEROIZE_N. It also exposes the various tamper detection flags. In the design example, the JTAG_ACTIVE output is used as one of the input to initialize zeroization.
- **Zeroize_interface_0:** This block uses the USER_TAMPER_FALG_0 and JTAG_ACTIVE signals and generate ZEROIZE_N signal. The other tamper detection flags are connected to this block, but not used in this design example. In addition, the JTAG_FLAG_GATING_SIGNAL is used to block the JTAG_ACTIVE signal from accidentally zeroizing the device.
- **Top_M3_Master_0:** This block includes MSS, RC Oscillator as clock input, and various fabric resources. The 50 MHz RC oscillator as clock input and a fabric PLL is used to generate a 100 MHz clock from the 50 MHz RC oscillator clock. This 100 MHz clock is used as the base clock for the MSS and also for the fabric. The Top_M3_Master_0 includes a RAM_init_top_0 block that is used to store the eNVM data after power-up.

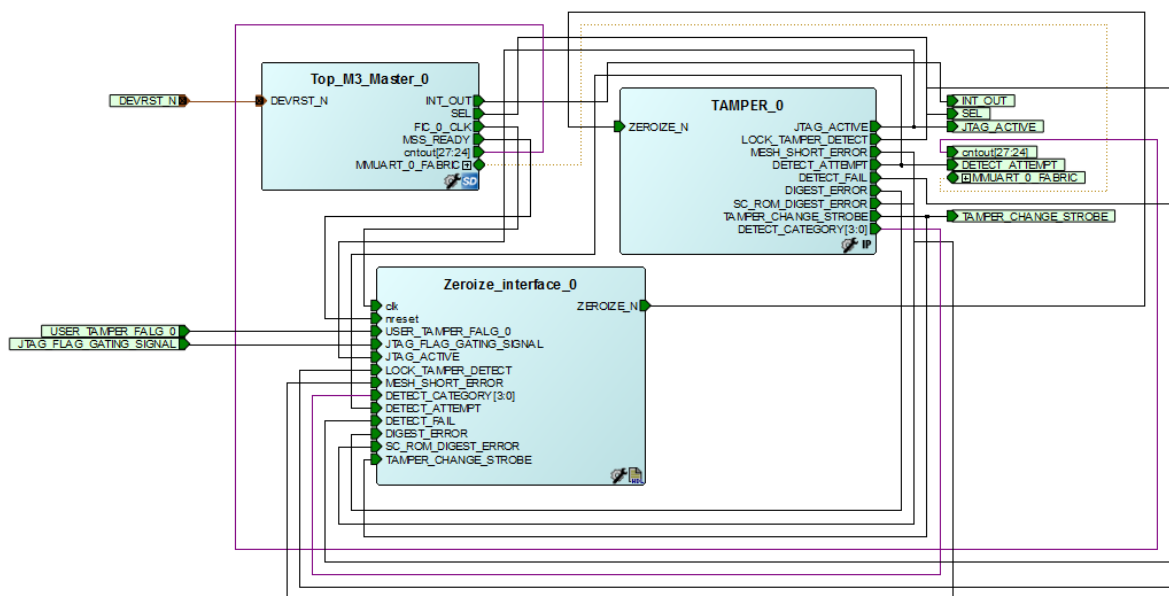


Figure 18 • Zeroize M2S090 Top Level Block Diagram

Figure 19 shows the block diagram of Top_M3_Master_0. The counter block is used to show that the device is up and running. After zeroization, the LED does not blink.

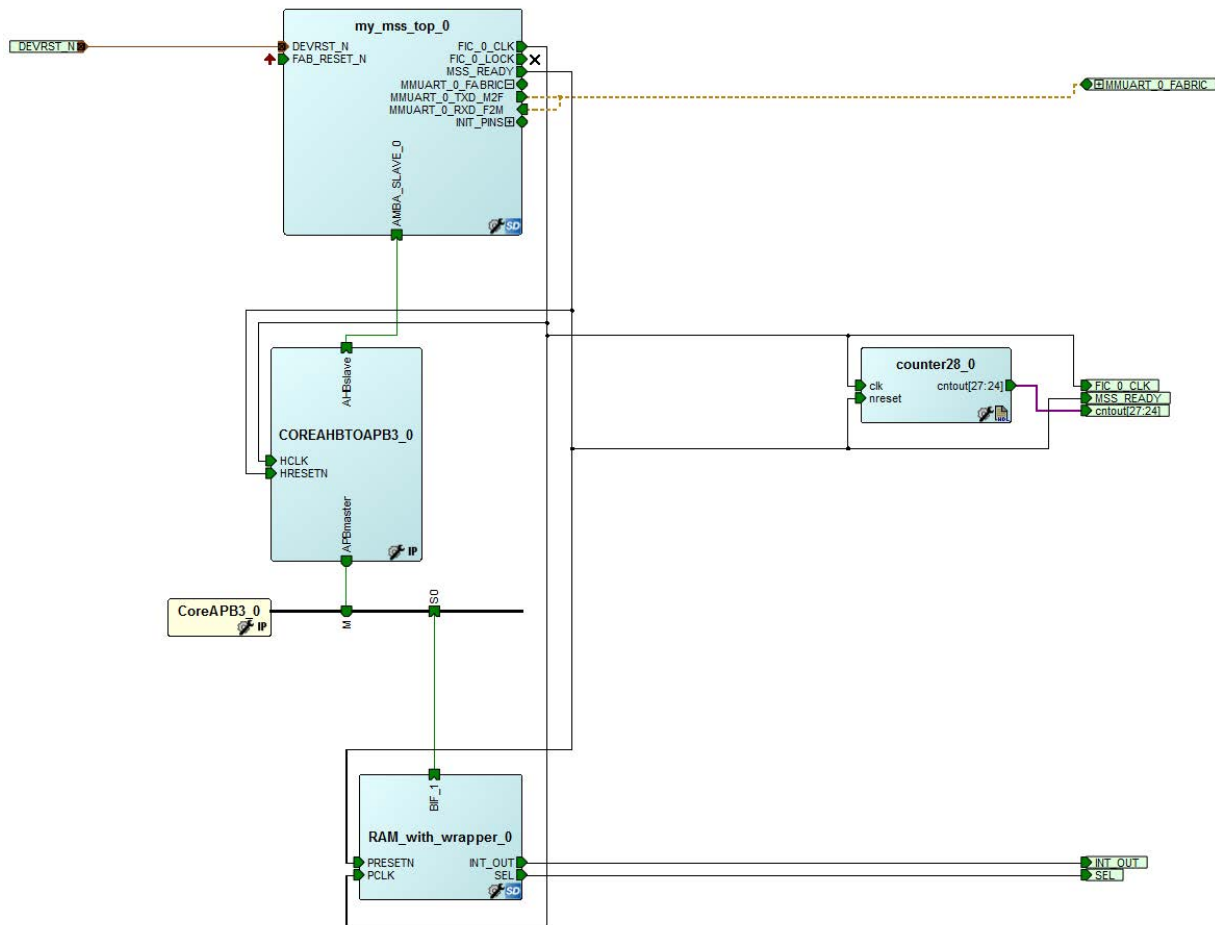


Figure 19 • Top_M3_Master_0 Block

Software Implementation

The software design example performs the following operations:

1. Initialize the System Controller Enable
2. Initialize MMUART_0
3. Move eNVM data to SRAM
4. Perform zeroize cryptography services

MSS_SYS_zeroize_device();

The MSS_SYS_zeroize_device() function initiates zeroization system service and destroys the FPGA contents.

nvm_access();

The nvm_access() function copies the data from the eNVM data client and moves it to Fabric SRAM.

Running the Design

The following steps describe how to run the design example on the M2GL_M2S-EVAL Kit board using the M2S090TS-1FGG484 device:

1. Connect the power supply to the M2GL_M2S-EVAL Kit board.
2. Plug the FlashPro4 ribbon cable into the JTAG Programming Header on the SmartFusion2 Security Evaluation Kit board.
3. Program the SmartFusion2 Evaluation board with the provided STAPL file. For more information about STAPL file, refer to ["Appendix A: Design Files" on page 26](#).
4. Connect the host PC to the J18 connector using the USB mini-B cable.

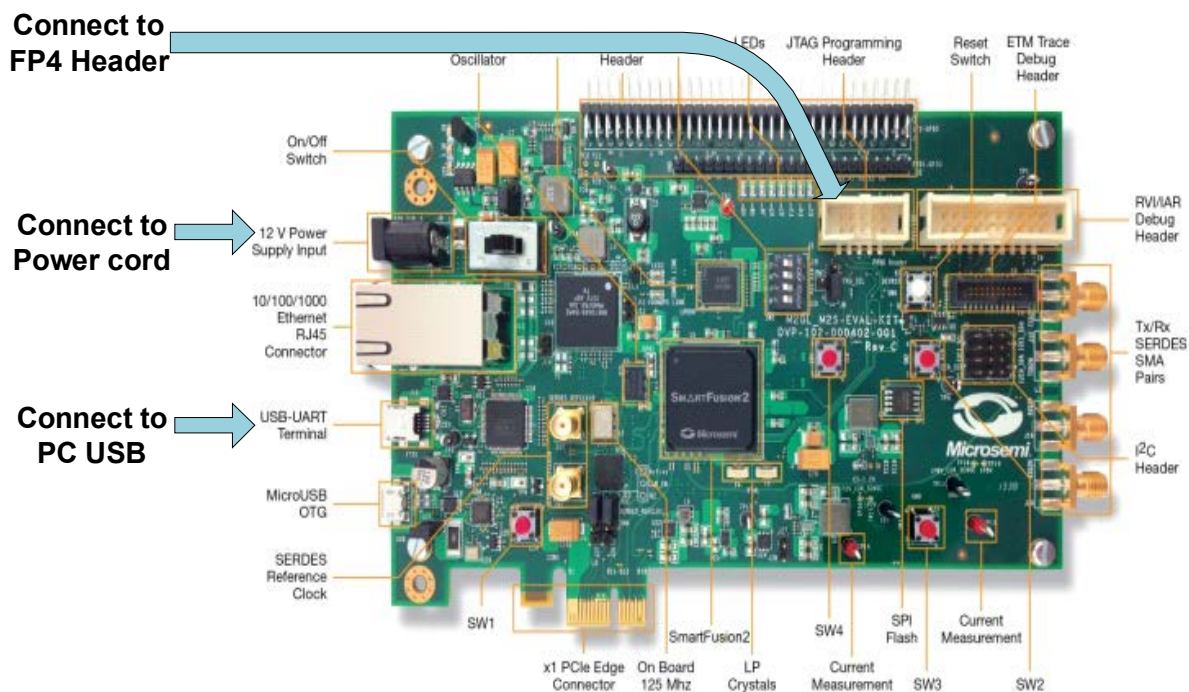
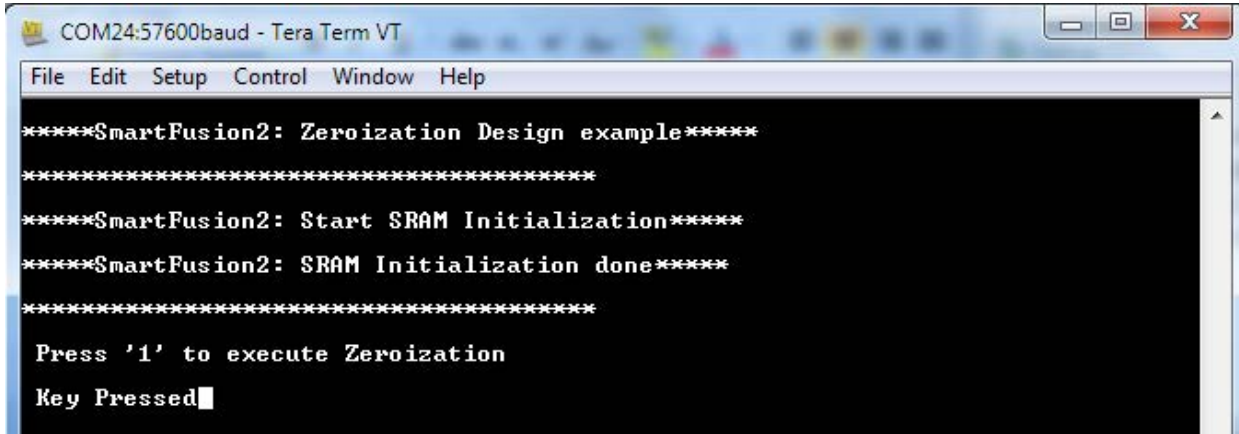


Figure 20 • M2GL_M2S-EVAL Kit Board

5. Invoke the SoftConsole integrated design environment (IDE), open the softconsole project, and launch the debugger.

6. Start a HyperTerminal session with 57600 baud rate, 8 data bits, 1 stop bit, no parity, and no flow control. If the computer does not have the HyperTerminal program, any free serial terminal emulation program such as PuTTY or TeraTerm can be used. Refer to [Configuring Serial Terminal Emulation Programs Tutorial](#) for configuring HyperTerminal, TeraTerm, or PuTTY.
7. Run the debugger in the SoftConsole tool. The HyperTerminal window shows the options to run the zeroize operation. Follow the instructions as shown in [Figure 21](#) to run the example.



```
COM24:57600baud - Tera Term VT
File Edit Setup Control Window Help
*****SmartFusion2: Zeroization Design example*****
*****
*****SmartFusion2: Start SRAM Initialization*****
*****SmartFusion2: SRAM Initialization done*****
*****
Press '1' to execute Zeroization
Key Pressed█
```

Figure 21 • Zeroization System Service Design Example using HyperTerminal

The flow for zeroizing the device using tamper macro is similar to the IGLOO2 design example.

Conclusion

This application note describes the zeroization usage, features, and options of the SmartFusion2 and IGLOO2 devices. Zeroize_M2GL090 and Zeroize_M2S090 design examples are also provided.

Appendix A: Design Files

The design files can be downloaded from the Microsemi SoC Products Group website:
http://soc.microsemi.com/download/rsc/?f=m2s_m2gl_ac433_liberov11p6_df.

The design file consists of Libero SoC Verilog project, SoftConsole software project, and programming files (*.stp) for SmartFusion2 Security Evaluation Kit board. Refer to the `Readme.txt` file included in the design file for the directory structure and description.

Appendix B: Running SmartDebug

The following steps describe how to read the eNVM or SRAM content.

1. Launch SmartDebug by selecting the SmartDebug Design option from the Design Flow window as shown in Figure 22. The SmartDebug window is displayed.

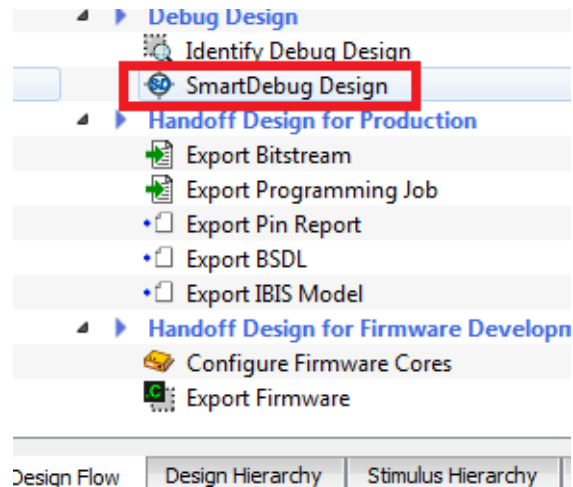


Figure 22 • SmartDebug Design Option in Design Flow Window

2. Click **View Flash Memory Content** to retrieve the eNVM content from the device using the SmartDebug window as shown in Figure 23. The Flash Memory window is displayed.

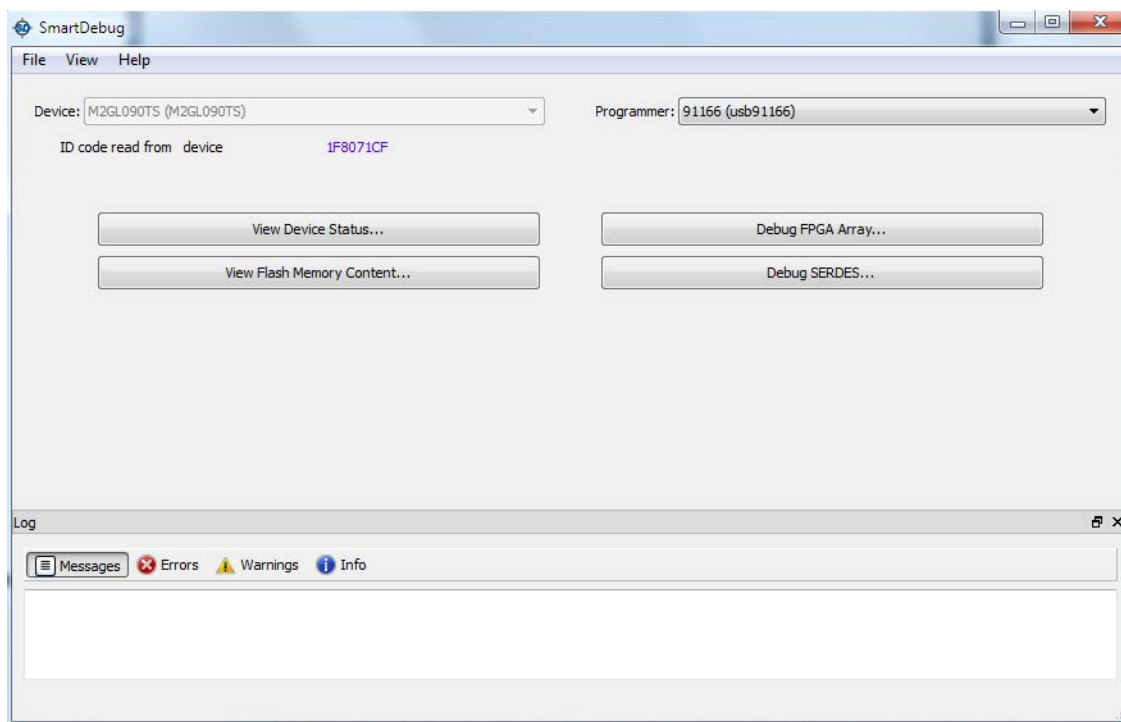


Figure 23 • SmartDebug GUI

3. Enter the **Start Page** and **End Page** as 16, because the data storage client is stored in page 16. Page 16 is used for demonstration purposes.
4. Click **Read from Device** as shown in Figure 24. SmartDebug reads eNVM content and display it in the GUI.

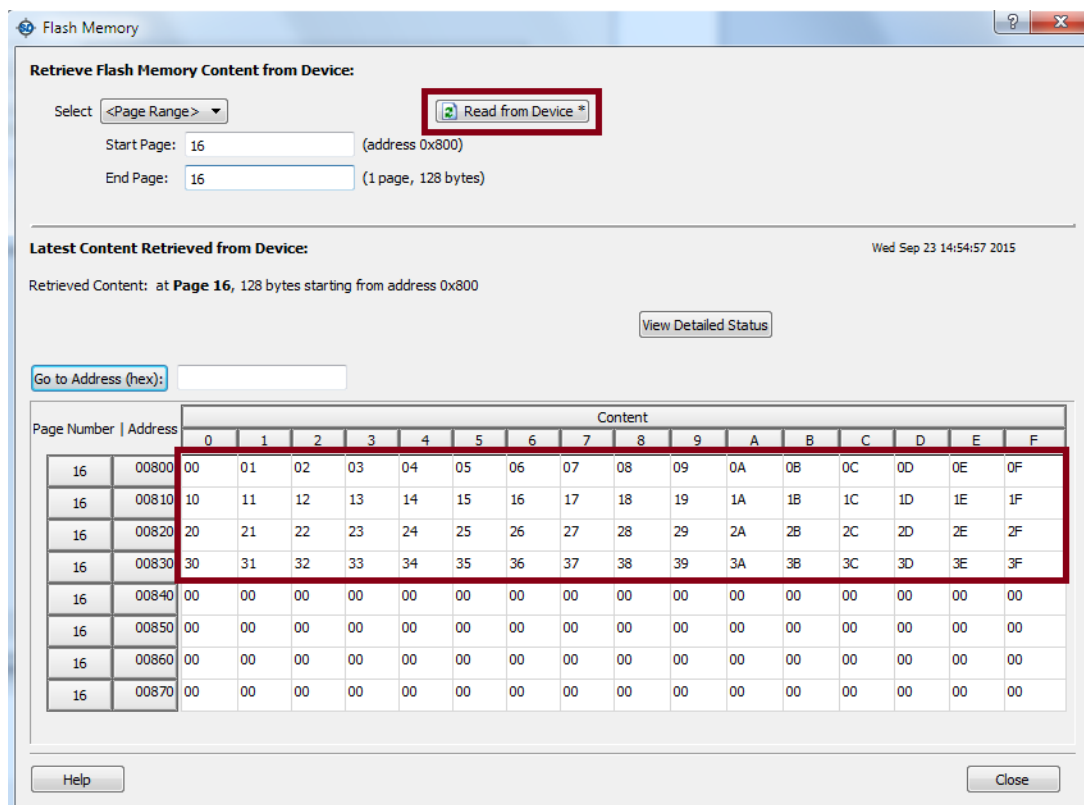


Figure 24 • SmartDebug GUI Showing Flash Memory

5. Click **Close** to close the Flash Memory window.
6. Click **Close** to close the Debug FGPA Array window.

List of Changes

The following table shows important changes made in this document for each revision.

Revision*	Changes	Page
Revision 3 (May 2024)	Updated the document for Libero SoC v2024.2 software release. De-featured Recoverable option in Libero SoC v2024.2. Accordingly deleted Recoverable instances from the document.	3, 5, 6
Revision 2 (October 2015)	Updated the document for Libero v11.6 software release (SAR 71463)	N/A
Revision 1 (February 2015)	Initial release.	N/A

Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.



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

www.microsemi.com

©2024 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.