

SmartFusion2 SoC FPGA SRAM Initialization from eNVM - Libero SoC v11.7

Table of Contents

Purpose	1
Introduction	1
References	2
Design Requirements	2
Embedded SRAM Blocks in SmartFusion2 SoC FPGAs	2
SmartFusion2 SoC FPGA eNVM Controller for Data Storage	3
SRAM to APB3 Wrapper	5
SRAM Initialization Reference Designs	6
Cortex-M3 Processor as Master	6
Fabric Master	7
Initializing SRAM Using Cortex-M3 Processor as Master	10
Hardware Implementation	10
Firmware and Application Code Software Implementation	12
Simulating Reference Design with Cortex-M3 Processor as Master	12
Running the Design with Cortex-M3 Processor as Master	13
Initializing SRAM using Fabric Master	16
Hardware Implementation	17
Simulating Reference Design with a Fabric as Master	18
Running the Design with a Fabric Master	20
Customizing Wrapper Interface	22
Conclusion	23
Appendix: Design and Programming Files	24
List of Changes	25

Purpose

This application note describes two different methods of initializing the large static random access memory (LSRAM) and micro SRAMs (uSRAM) using design examples where ARM® Cortex®-M3 processor or fabric logic is used as the master. The design examples describe initializing the fabric SRAM blocks after power-up with the initialization data from the embedded non-volatile memory (eNVM) block.

Introduction

The SmartFusion®2 system-on-chip (SoC) field programmable gate array (FPGA) devices have embedded SRAM blocks in fabric. There are two types of SRAM blocks in SmartFusion2 SoC FPGA fabric—LSRAMs and uSRAMs. The LSRAMs are used for storing large data or for creating large FIFOs. The LSRAM and uSRAM blocks are volatile memory types, the stored data disappears in the absence of power. After the device is powered-up, the content of SRAM is unknown. There are some applications which require the SRAM data to be initialized and validated after power-up.

There are several methods of initializing the LSRAM and uSRAM. This document provides two solutions for implementing this initialization method, and also provides the design examples. The design examples describe initializing the fabric SRAM blocks after power-up with the initialization data from the eNVM block using the Cortex-M3 processor or fabric logic as the master. The Cortex-M3 processor or the fabric master transfers the data from eNVM to the SRAM blocks after power-up.

Figure 4 on page 7 and Figure 5 on page 9 show block diagrams of the design examples. The reference designs use the SRAM block configured as a two-port memory, but this initialization approach can be used for all the variations of LSRAM and uSRAM in the SmartFusion2 SoC FPGA device. The reference design is simulated and tested on silicon using SmartFusion2 Security Evaluation Kit board.

References

The list of references are:

- [UG0331: SmartFusion2 Microcontroller Subsystem User Guide](#)
- [TU0530: SmartFusion2 and IGLOO2 SmartDebug Hardware Design Debug Tools Tutorial](#)
- [SmartFusion2 MSS Embedded Nonvolatile Memory \(eNVM\) Simulation](#)
- [UG0445: IGLOO2 FPGA and SmartFusion2 SoC FPGA Fabric User Guide](#)

Design Requirements

Table 1 lists the design requirements.

Table 1 • Design Requirements

Design Requirements	Description
Hardware Requirements	
SmartFusion2 Security Evaluation Kit	M2S090TS-1FGG484
Software Requirements	
Libero [®] System-on-Chip (SoC)	v11.7
FlashPro programming software	v11.7
SoftConsole	v3.4 SP1*

Note: *For this application note, SoftConsole v3.4 SP1 is used. For using SoftConsole v4.0, see the TU0546: SoftConsole v4.0 and Libero SoC v11.7 Tutorial.

Embedded SRAM Blocks in SmartFusion2 SoC FPGAs

This section describes the fabric SRAM blocks in various SmartFusion2 devices and clarifies their differences.

Table 2 lists the types of fabric SRAM blocks in various SmartFusion2 devices.

Table 2 • SRAM Blocks in Various SmartFusion2 Devices

Features	M2S005	M2S010	M2S025	M2S050	M2S060	M2S090	M2S150
LSRAM 18 K Blocks	10	21	31	69	69	109	236
uSRAM 1 K Blocks	11	22	34	72	72	112	240
Total RAM (Kbits)	191	400	592	1314	1314	2074	4488

The LSRAM blocks can be configured as a dual-port SRAM or two-port SRAM. LSRAM configured as dual-port SRAM provides two independent access ports— Port A and Port B. In dual-port mode, data can be transferred through these ports independently based on various parameters. Each port has its own address, data in, data out, clock, clock enable, and write enable. LSRAM configured as two-port SRAM has Port A dedicated to read operations, and Port B dedicated to write operations. The read and write operations in LSRAM are synchronous and require a clock edge.

The uSRAM has two read ports (Port A and Port B) and one write port (Port C). The read ports operate either in synchronous or asynchronous modes. The write operation is performed only in synchronous mode.

The SRAM blocks support rich variations in size and features of memory blocks for SmartFusion2 SoC FPGA devices. Although these variations require changes for a specific implementation of initializing the SRAM blocks, the changes are not significant enough to affect the fundamentals of the reference design. Therefore, the two reference designs target only the LSRAM block. The effects of feature and size variations on the reference designs are discussed in the ["Customizing Wrapper Interface" section on page 24](#).

SmartFusion2 eNVM Controller for Data Storage

The design example uses the eNVM array in microcontroller subsystem (MSS) as the source of the SRAM initialization. The flash memory block in the eNVM is used to store the SRAM initialization data, and it is loaded to SRAM after power-up. The eNVM controller is an advanced high-performance bus (AHB) slave that provides access to eNVM. It converts the logical AHB addresses to physical eNVM addresses, and allows to command the eNVM to perform specific tasks such as read, and write operations. For more information, see the Embedded eNVM Controller section in the [UG0331: SmartFusion2 Microcontroller Subsystem User Guide](#).

In the design examples, the data is defined first to be programmed into eNVM, which is used for the SRAM initialization. The user can define an eNVM "Data Client", which is configured as 64 × 8 using the eNVM configurator. [Figure 1](#) shows the eNVM configurator graphical user interface (GUI) in Libero SoC that is accessed through the System Builder tools.

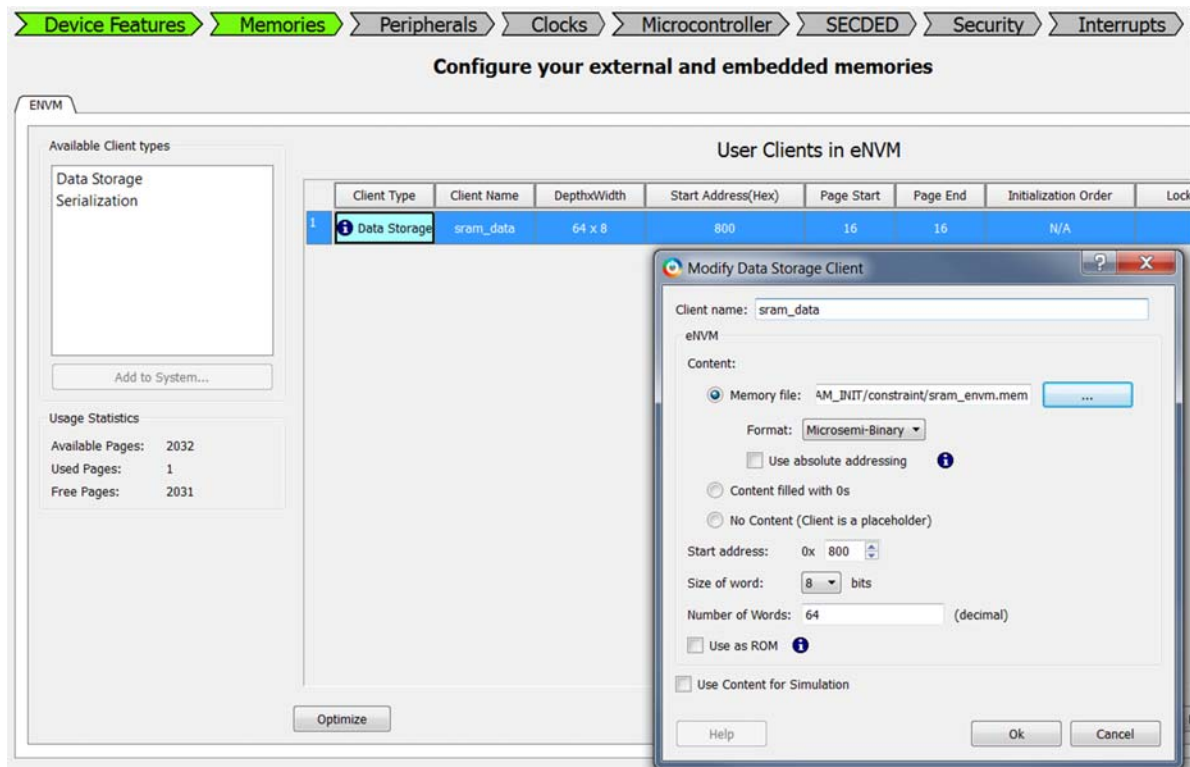


Figure 1 • Data Storage eNVM Client (System Builder)

Page 16 (start address 0x800) is used here for demonstration purposes. [Figure 2](#) shows an excerpt of the data storage client content using Microsemi binary scheme (sram_envm.mem) that is defined in the eNVM. The sram_envm.mem file is included in the Libero project under the constraint folder.

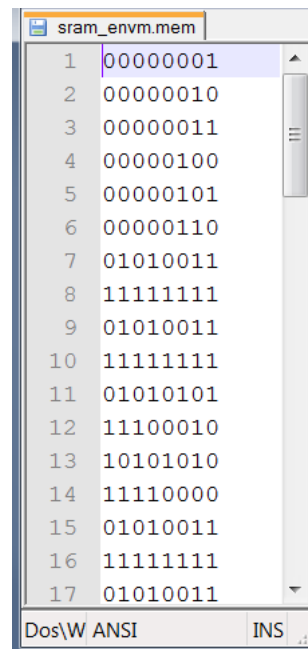


Figure 2 • Memory File Content Saved into eNVM

SRAM to APB3 Wrapper

The section describes connecting the SRAM block to the advanced microcontroller bus architecture (AMBA®) advanced peripheral bus 3 (APB3) bus system. To move the data from eNVM to SRAM using the Cortex-M3 processor as the master or a fabric master, the user needs to create a wrapper logic around the SRAM block. The wrapper generates the write enable and read enable for SRAM using the APB3 bus signals. Figure 3 shows the state diagram for the APB3 bus specification.

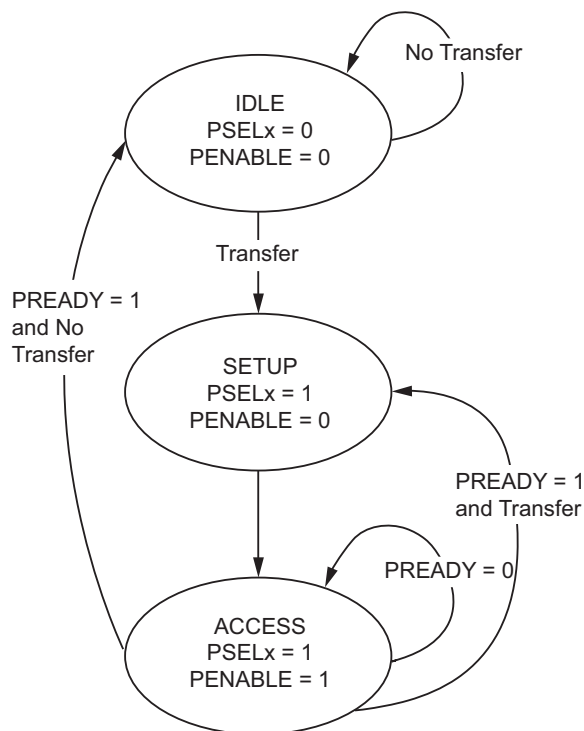


Figure 3 • APB3 State Diagram

Following are the three states:

- **IDLE:** This is the default state for the peripheral bus.
- **SETUP:** When a transfer is required, the bus moves to this state where the appropriate select signal PSELx is asserted. The bus remains in this state for one clock cycle only and always moves to the ACCESS state on the next rising edge of the clock.
- **ACCESS:** In this state, the enable signal PENABLE is asserted. The address, write, and select signals should be stable during the transition from SETUP to ACCESS state. The transition from the ACCESS state is controlled by the PREADY signal from the slave.
 - If PREADY is held low by the slave, then the peripheral bus remains in the ACCESS state.
 - If PREADY is held high by the slave and no more transfers are required, the bus transitions from the ACCESS state to the IDLE state. Alternatively, if another transfer follows, the bus moves directly to the SETUP state.

In this design example, the wrapper logic generates the write enable and read enable for SRAM using the PSEL, PWRITE, and PENABLE signals. The PREADY signal is used to insert the wait state.

SRAM Initialization Reference Designs

This document discusses two methods of initializing the fabric SRAM. The first method uses the Cortex-M3 processor as the master that transfers the data from eNVM to SRAM. The second method uses a master in the fabric to transfer the data from eNVM to SRAM. The two reference designs are described and analyzed in the following sections:

- **Cortex-M3 Processor as Master**—describes the method of initializing SRAM using the Cortex-M3 processor as the master.
- **Fabric Master**—describes the method of initializing SRAM using a fabric master.

Cortex-M3 Processor as Master

The SRAM block is configured as two-port memory with a depth of 64 and a width of 8. This design implements an APB3 slave wrapper interface on Port A and Port B of the SRAM block, and the APB3 wrapper is memory mapped to the MSS. The user can also implement the AHBLite wrapper instead of APB3 wrapper on the SRAM block and connect to the MSS. However, the APB3 interface is much simpler than the AHBLite interface, and it is easy to create this interface with the SRAM ports. This APB3 slave wrapper interface is connected to the MSS through the CoreAPB3, CoreAHBTOAPB3, CoreAHBLite and fabric interface controller (FIC_0) interface as shown in Figure 4. FIC_0 and FIC_1 enable the connectivity between the fabric and the MSS. The FIC_0 is part of the MSS, and performs a bridging functionality between MSS and FPGA fabric. The FIC can be configured either in the AHBLite mode or in the APB3 mode. In this design example, the FIC_0 is configured in the AHBLite, so that the other AHBLite blocks in the fabric can be connected to MSS through FIC. Figure 4 shows a top-level block diagram of the design example using the Cortex-M3 processor as the master.

The muxing arbiter block in the APB3 slave wrapper allows switching the SRAM ports as user-ports after the initialization is done. The Cortex-M3 processor in MSS acts as a master to read data from eNVM after powering-up and initializing the fabric SRAM block. After the initialization is done, the APB3 wrapper interface asserts a SEL signal for muxing arbiter to switch the SRAM ports as user-ports. After the initialization is done, the user reads/writes from/to SRAM block can be started. Figure 4 shows the design example block diagram using the Cortex-M3 processor as the master.

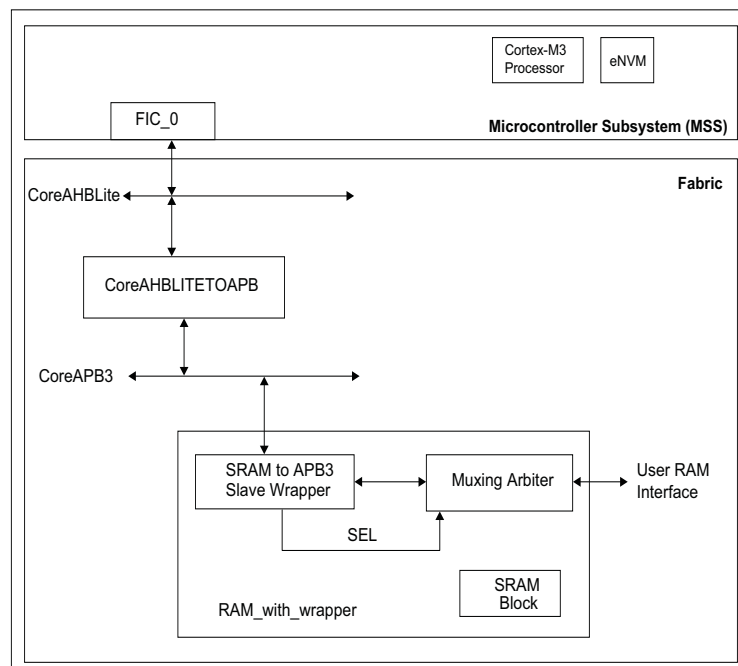


Figure 4 • Design Example Block Diagram

Interface Description

Table 3 shows the top-level Cortex-M3 processor as the master interface signal descriptions.

For more information about LSRAM and uSRAM functionalities and features, refer to [UG0445: IGLOO2 FPGA and SmartFusion2 SoC FPGA Fabric User Guide](#).

Table 3 • Top-Level Cortex-M3 Processor as the Master Interface Signals

Signal	Direction	Description
raddr_user[5:0]	Input	User read address
rclk_user	Input	User read clock
rd_enable_user	Input	User read enable
waddr_user[5:0]	Input	User write address
wclk_user	Input	User write clock
wdata_user[7:0]	Input	User write data
wr_enable_user	Input	User write enable
rdata_user[7:0]	Output	User read data
INIT_DONE	Output	Initialization complete
DEVRST_N	Input	Active low reset
MMUART_1_RXD	Input	Uart RX input (for debug only)
MMUART_1_TXD	Output	Uart TX output (for debug only)
SEL	Output	Selection for RAM muxing logic (for debug only)

Status Output

The INIT_DONE output of the reference design indicates the sequence of initialization done.

At power-up, it is asserted as low to indicate the start of initialization process. It remains low until the Cortex-M3 processor or a fabric master finishes reading the data from eNVM and writing it to SRAM. Once INIT_DONE output is asserted, the asserted state indicates the end of initialization process. Port A and Port B of SRAM interface are available to the user for read and write access operations.

Fabric Master

The design is similar to the design that is implemented using the Cortex-M3 processor as the master. The fabric acts as a master to read data from eNVM after powering-up and initializing the SRAM block. After the initialization is done, the APB3 wrapper interface asserts a SEL signal for muxing arbiter to switch the SRAM ports as user-ports. After the initialization is done, the write and read data to/from the SRAM block can be started. The INIT_DONE output of the reference design indicates the sequence of initialization done.

Figure 5 shows a top-level block diagram of the design example.

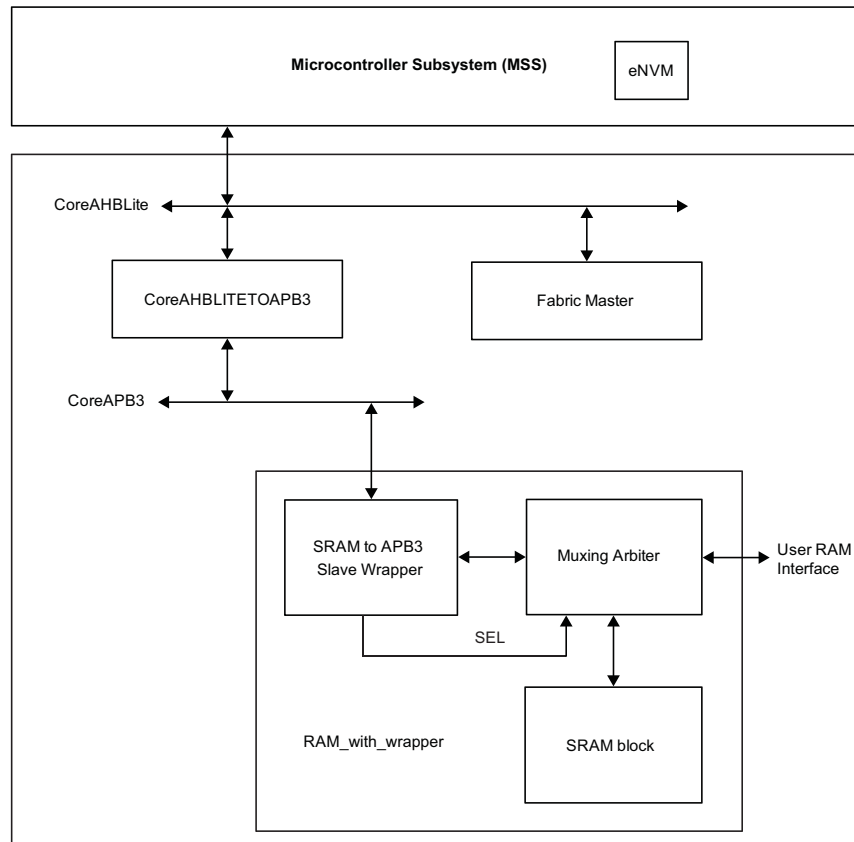
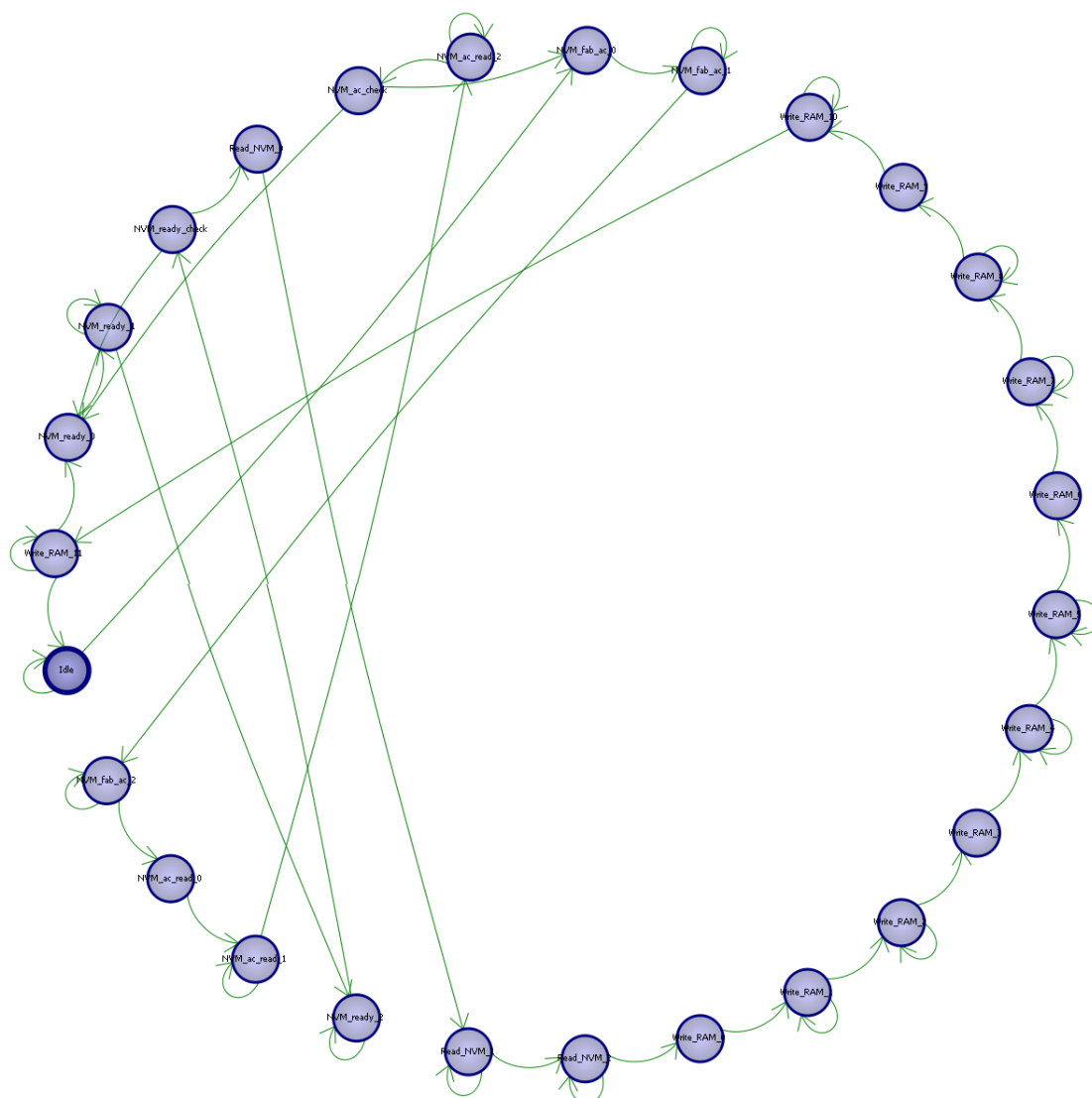


Figure 5 • Design Example Block Diagram using Fabric Master

The fabric Master block shown in Figure 5 acts as a master logic to read data from eNVM and write it to SRAM. The AHB-Lite master drives the address and controls the signals onto the bus after the rising edge of HCLK. If HREADY is in low state, the Fabric Master waits. If HREADY is in high state, the logic moves to the data phase. During the data phase, if HREADY is in low state, the AHB-Lite master holds the data stable throughout the extended cycle for a write operation, or read the data only after HREADY is in high state.



Interface Description for Fabric Master Design

Table 4 shows the top-level interface signal descriptions.

Table 4 • Top-Level Interface Signals

Signal	Direction	Description
raddr_user[5:0]	Input	User read address
rclk_user	Input	User read clock
rd_enable_user	Input	User read enable
waddr_user[5:0]	Input	User write address
rdata_user[7:0]	Output	User read data
wclk_user	Input	User write clock
wdata_user[7:0]	Input	User write data
wr_enable_user	Input	User write enable
INIT_DONE	Output	Initialization complete
DEVRST_N	Input	Active Low reset
MMUART_1_RXD	Input	Uart RX input (for debug only)
MMUART_1_TXD	Output	Uart TX output (for debug only)
RESP_err[1:0]	Output	Ahb error response
ram_init_done	Output	Initialization complete
SEL	Output	Selection for RAM muxing logic (for debug only)
ahb_busy	Output	Ahb busy indication

Initializing SRAM Using Cortex-M3 Processor as Master

This section explains the following topics:

- [Hardware Implementation](#)
- [Firmware and Application Code Software Implementation](#)
- [Simulating Reference Design with Cortex-M3 Processor as Master](#)
- [Running the Design with Cortex-M3 Processor as Master](#)

Hardware Implementation

The hardware implementation involves configuring the MSS along with the SRAM block configuration. The SRAM block is configured as two-port memory with a depth of 64 and a width of 8. The MSS along with FIC_0, MMUART, and the eNVM are configured using System Builder. Through the System Builder, the design is configured to use a 50 MHz RC oscillator as a reference clock for the fabric phase-locked loop (PLL). The fabric PLL then generates a 100 MHz clock that is used as the main system clock. The design example consists MSS, SRAM wrapper logic, and IP cores (CoreAHBToAPB3, CoreAPB3) as shown in [Figure 7 on page 12](#).

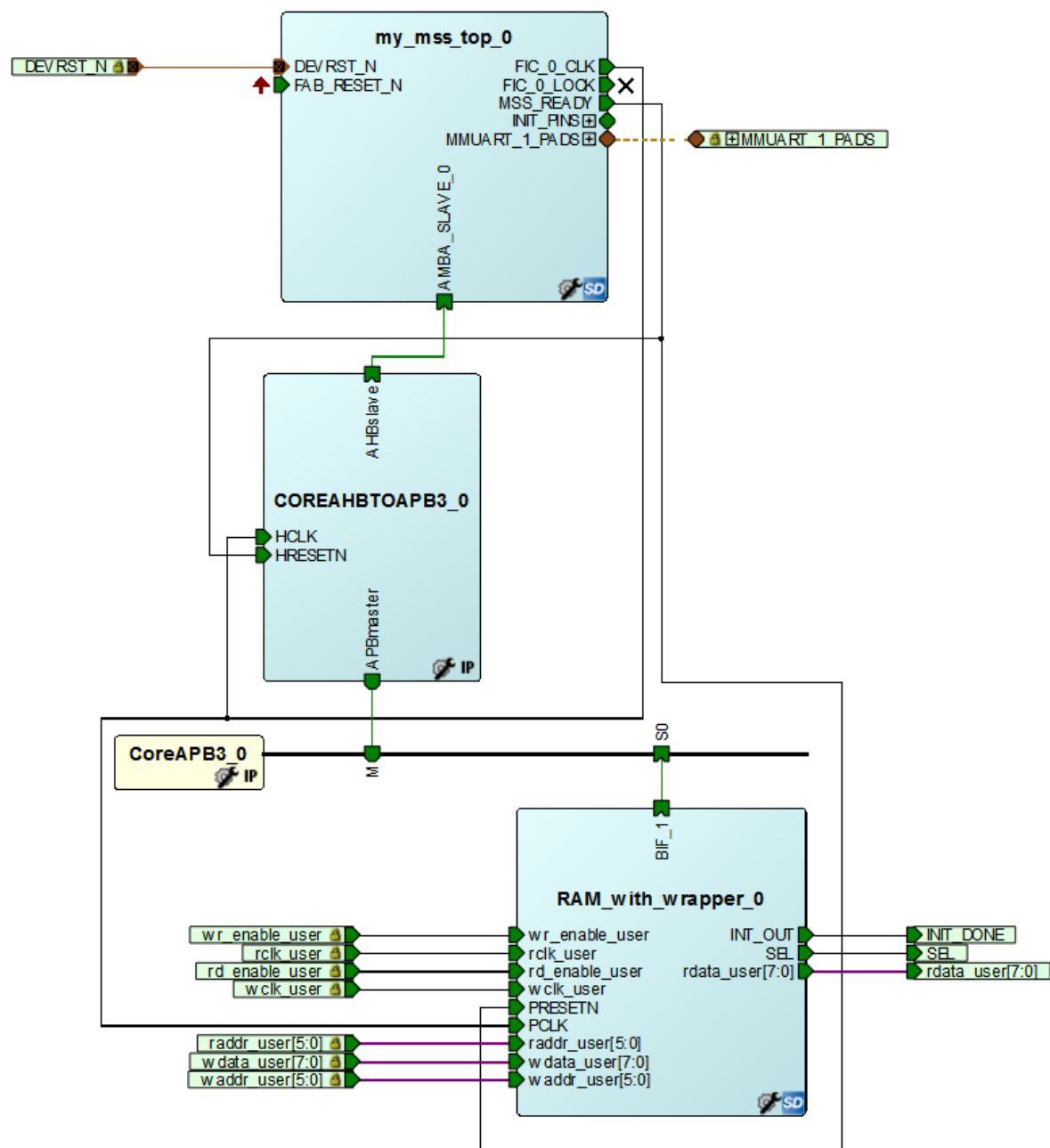


Figure 7 • Top-Level Hardware Design for Cortex-M3 Processor as Master

CoreAHLite IP is generated and used automatically inside the System Builder block. The IP cores along with the SRAM wrapper are used to initialize the fabric SRAM by moving the data from eNVM to the fabric SRAM through the FIC_0 AHB master interface. A Data Storage client is defined in the eNVM with the data to be written to the SRAM.

Firmware and Application Code Software Implementation

Firmware and application code is required only while using the Cortex-M3 processor as the master. This design example includes the MSS MMUART_1 block. The MMUART_1 block is used so that the initialization sequence and the debug of SRAM block can be viewed through HyperTerminal. The software design includes an initialization function (nvm_access()) that reads the eNVM content and writes it to the SRAM block.

nvm_access ()

This function reads the eNVM content which is loaded during SmartFusion2 SoC FPGA device programming. Each read output is 64-bit data. It converts the 64-bit data to four sets of 8-bit data, and then writes each set of 8-bit data to four SRAM locations. This process (read, convert, and write) continues until the last SRAM address is initialized. It also reads back the SRAM content to check the data.

Note: Once the last address location is written, the SEL signal is generated and the SRAM interface is switched to User mode, so the last address read back should be seen as zeros.

Simulating Reference Design with Cortex-M3 Processor as Master

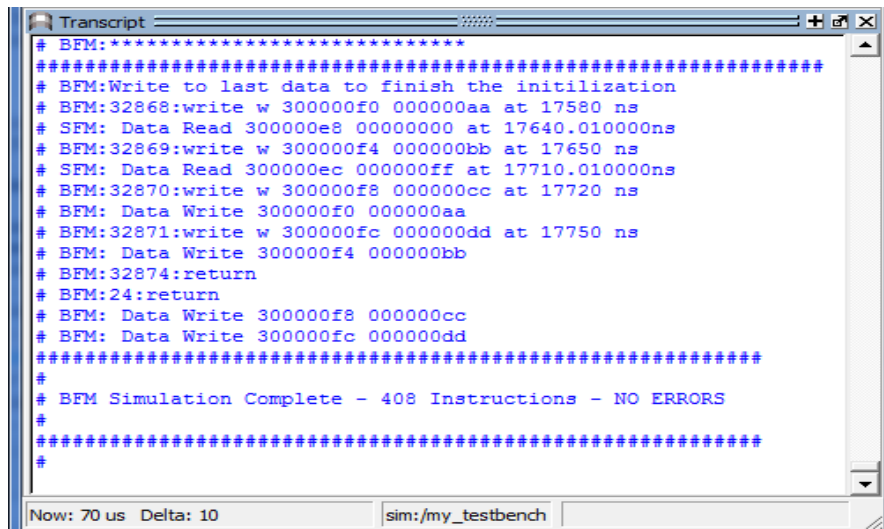
The design file includes the test bench files to run simulation in the Libero SoC. The simulation uses the bus functional model (BFM) command to exercise data transfer between the MSS and the fabric.

Note: After system reset, the BFM has several commands to load the eNVM content, which is not needed for software implementation.

The BFM has the following sequence:

1. Setting access privileges to eNVM
2. Writing the initialization data to eNVM (for simulation only)
3. Reading from eNVM and then write to SRAM Reading SRAM through the MSS and check the data

Figure 8 shows the BFM simulation transcript results and Figure 9 on page 14 shows the ModelSim presynthesis simulation waveform results.



```

# BFM:*****
#####
# BFM:Write to last data to finish the initialization
# BFM:32868:write w 300000f0 000000aa at 17580 ns
# SFM: Data Read 300000e8 00000000 at 17640.010000ns
# BFM:32869:write w 300000f4 000000bb at 17650 ns
# SFM: Data Read 300000ec 000000ff at 17710.010000ns
# BFM:32870:write w 300000f8 000000cc at 17720 ns
# BFM: Data Write 300000f0 000000aa
# BFM:32871:write w 300000fc 000000dd at 17750 ns
# BFM: Data Write 300000f4 000000bb
# BFM:32874:return
# BFM:24:return
# BFM: Data Write 300000f8 000000cc
# BFM: Data Write 300000fc 000000dd
#####
# BFM Simulation Complete - 408 Instructions - NO ERRORS
#
#####
#

```

Now: 70 us Delta: 10 sim:/my_testbench

Figure 8 • BFM Transcript Simulation Results



This section describes running Cortex-M3 processor as master design example in SmartFusion2 Security Evaluation Kit.

1. Open the **MSS_MSTR_RAM_INIT** Libero project (refer to "[Appendix: Design and Programming Files](#)" on page 25).
2. Update the **eNVM client memory** file path. For more information, refer to <http://soc.microsemi.com/kb/article.aspx?id=SL5657>.
3. Program the SmartFusion2 Security Evaluation Kit board by selecting **Run PROGRAM Action** option in the Libero **Design Flow** window or with the provided Cortex-M3 processor as master STAPL file (refer to "[Appendix: Design and Programming Files](#)" on page 25) using FlashPro4.
4. Connect the USB to PC.
5. Launch the **SoftConsole v4.0** and browse the SoftConsole folder project where the Libero project is created as shown in [Figure 10 on page 15](#).

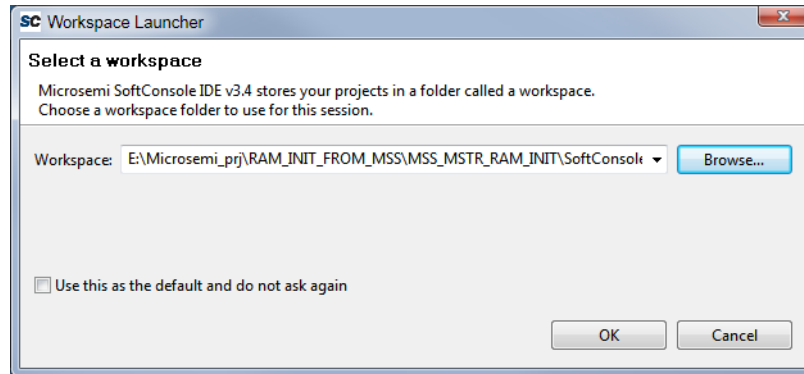


Figure 10 • Specifying the SoftConsole Workspace Location

6. Click **OK**.

SoftConsole opens with the project automatically loaded as shown in [Figure 11](#).

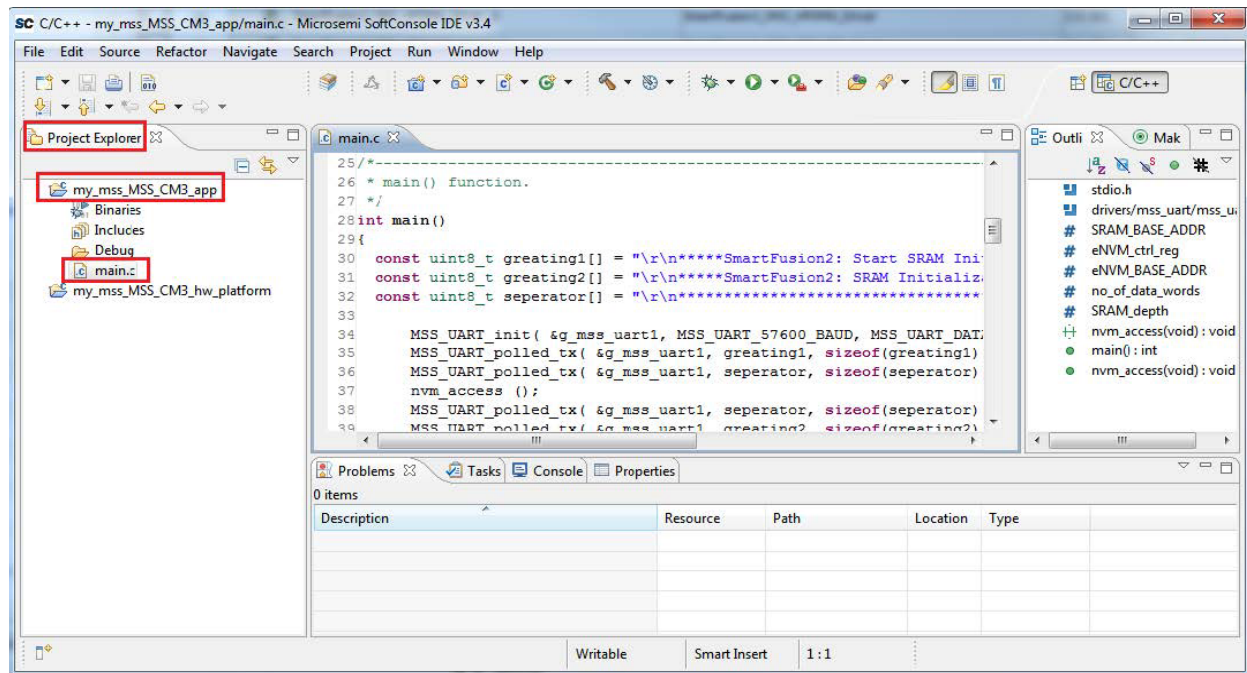


Figure 11 • SoftConsole Window

7. In SoftConsole, click the **Project Explorer** tab and click the **SF2_GNU_SC4_MMUART_polled_uart** folder on the left pane.
8. Inspect the main code by double-clicking the **main.c** file as shown in [Figure 11](#).
9. Choose **Project > Clean** to perform a clean build of the code.
10. Retain the default settings in the **Clean** dialog box and click **OK**.

Note: Ensure that errors are not displayed throughout the design configuration and build flow.

11. Choose **Run > Debug Configurations > Debug** option. The **Debugger** window is displayed as shown in [Figure 12](#). Click on **Debug** from the Debugger window.

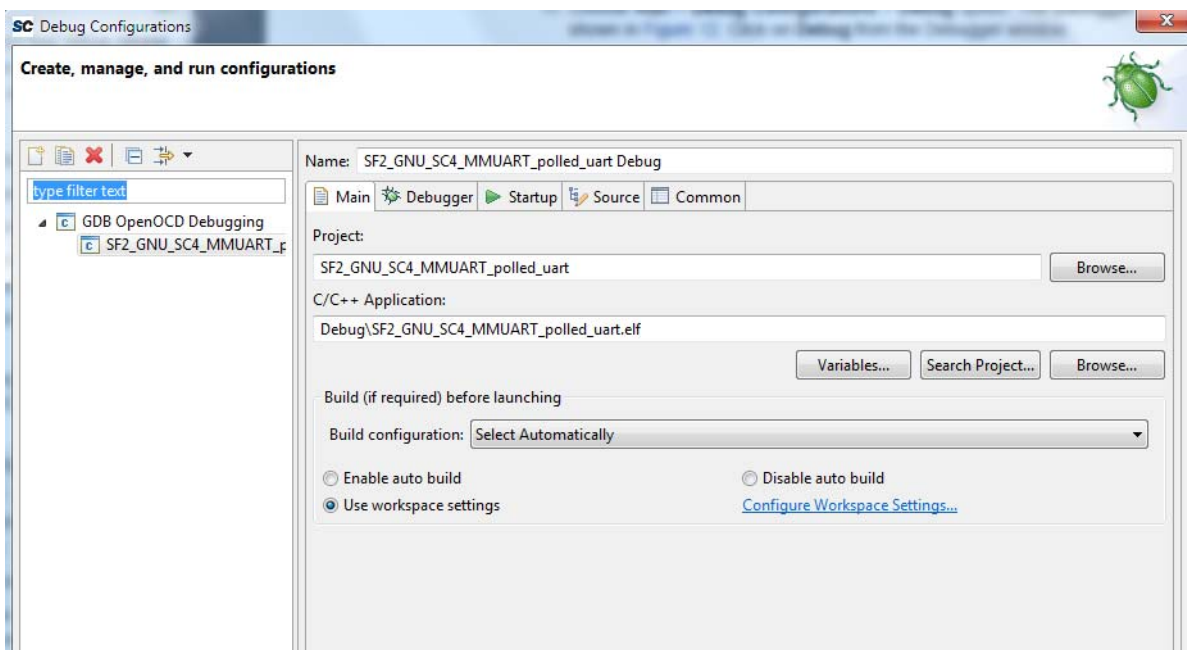


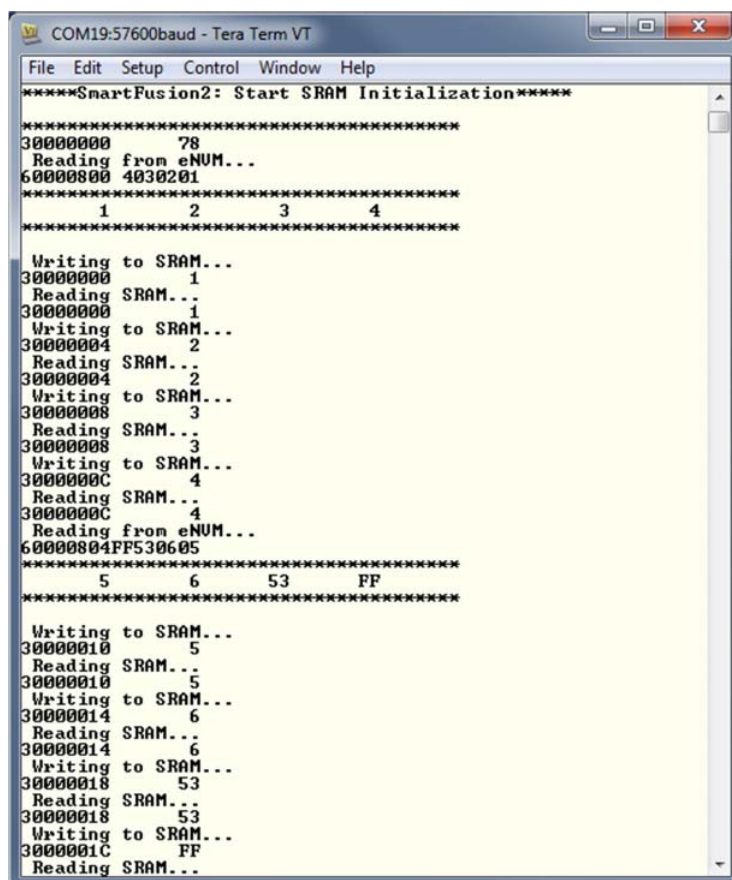
Figure 12 • Launch the Debugger

12. 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 Tera Term can be used. Refer to the [Configuring Serial Terminal Emulation Programs Tutorial](#) for configuring HyperTerminal, Tera Term, or PuTTY.
13. Run the debugger by pressing the **F8** (function key) on the keyboard or double-click the **Resume** icon as shown in [Figure 13](#).



Figure 13 • Resume Icon

The HyperTerminal window shows the initialization sequence by reading eNVM and writing to SRAM. Figure 14 shows the screenshot of HyperTerminal.



```

COM19:57600baud - Tera Term VT
File Edit Setup Control Window Help
*****SmartFusion2: Start SRAM Initialization*****
*****
30000000 78
Reading from eNVM...
60000800 4030201
*****
1 2 3 4
*****
Writing to SRAM...
30000000 1
Reading SRAM...
30000000 1
Writing to SRAM...
30000004 2
Reading SRAM...
30000004 2
Writing to SRAM...
30000008 3
Reading SRAM...
30000008 3
Writing to SRAM...
3000000C 4
Reading SRAM...
3000000C 4
Reading from eNVM...
60000804 FF530605
*****
5 6 53 FF
*****
Writing to SRAM...
30000010 5
Reading SRAM...
30000010 5
Writing to SRAM...
30000014 6
Reading SRAM...
30000014 6
Writing to SRAM...
30000018 53
Reading SRAM...
30000018 53
Writing to SRAM...
3000001C FF
Reading SRAM...
  
```

Figure 14 • Screenshot of HyperTerminal Showing the Design Example

Initializing SRAM using Fabric Master

The fabric master design implementation is similar to the Cortex-M3 processor master design except that the master is responsible for moving the initialization data from the eNVM to SRAM master in the fabric.

The following section describes the hardware implementation using a fabric master. It also details how to simulate the provided design along with the steps on how to run the design on the SmartFusion2 Security Evaluation Kit board.

Hardware Implementation

The hardware implementation involves configuring the MSS along with the SRAM block configuration. The SRAM block is configured as two-port memory with a depth of 64 and a width of 8. Through the System Builder, the design is configured to use a 50 MHz RC oscillator as a reference clock for the fabric phase-locked loop (PLL). The fabric PLL then generates a 100 MHz clock that is used as the main system clock. The design example consists MSS, SRAM wrapper logic, fabric master (AHBMASTER_FIC_0) as shown in Figure 15.

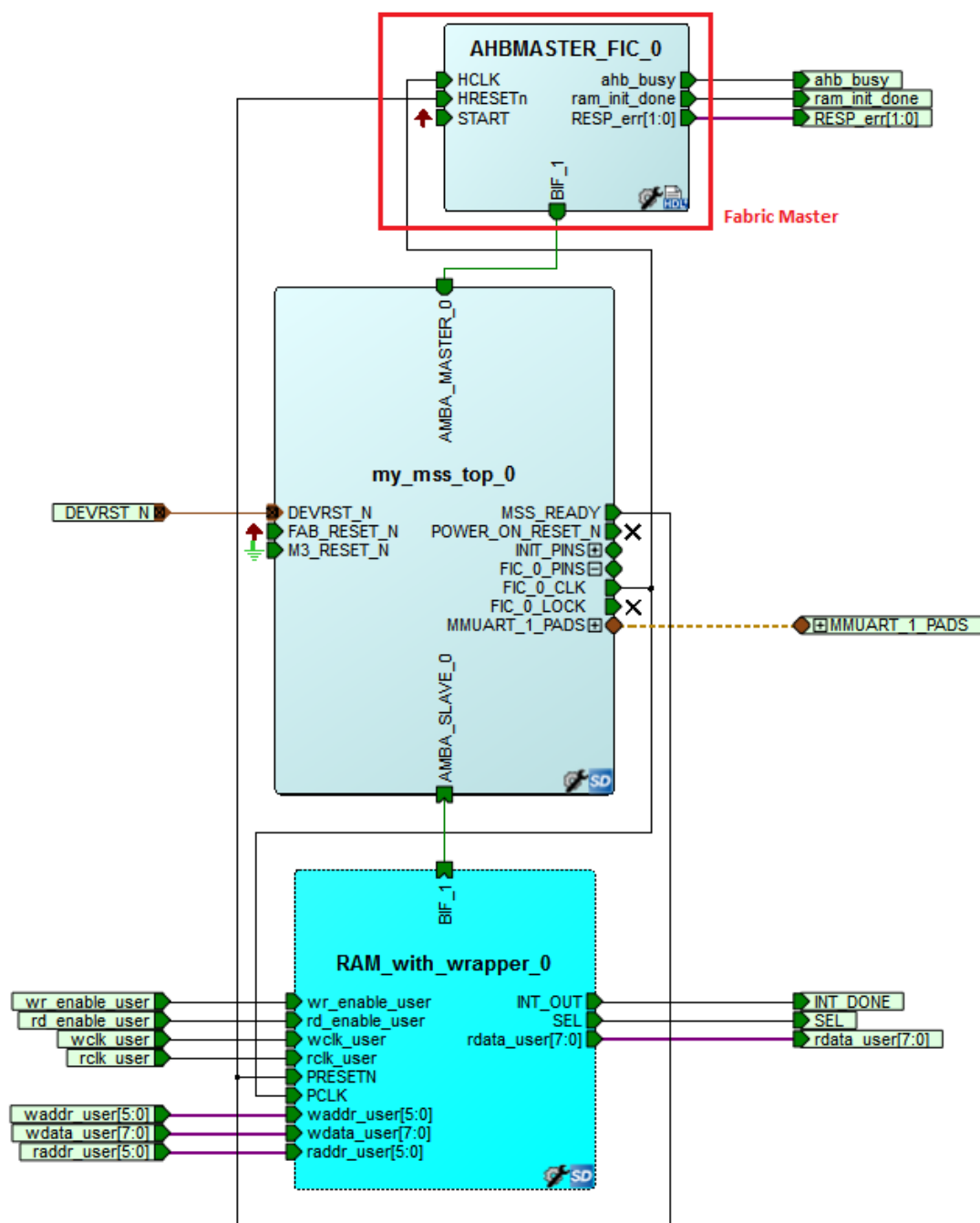


Figure 15 • Top-Level Hardware Design for Fabric Master

The SRAM wrapper along with the fabric master is used to initialize the fabric SRAM by moving data from the eNVM to the fabric SRAM through the FIC_0 AHB master interface. The System Builder is mainly used to configure the MSS, eNVM Data Storage client, and FIC interface. A Data Storage client is defined in the eNVM with the data to write to SRAM. Refer to [Figure 1 on page 4](#) and [Figure 2 on page 5](#) for more details.

At power-up or at power-on reset, the Cortex-M3 processor fetches the initial stack pointer from 0x00000000 (eNVM address 0x60000000) and address of the reset handler from 0x00000004 (eNVM address 0x60000004). If the execution control goes to the default reset handler, the boot up sequence is executed and the execution control moves to the user boot code. The Cortex-M3 processor is not used for this particular design since there is no user boot code implemented for it. The user can expose the reset signal M3_RESET_N and tie it LOW to keep the Cortex-M3 in reset as shown in [Figure 15 on page 18](#).

Note: To expose the M3_RESET_N signal, the System Builder block is re-opened as SmartDesign block. Refer to the [SmartFusion2 System Builder User Guide](#) for more information on Modifying/Inspecting Your System Builder Design.

Simulating Reference Design with a Fabric as Master

This section describes the pre-synthesis simulation detail of simulating the fabric master design using the top-level test bench, Top_Fabric_Master, and Use Content for Simulation option in the Data Storage Client Configurator, as shown in [Figure 16](#).

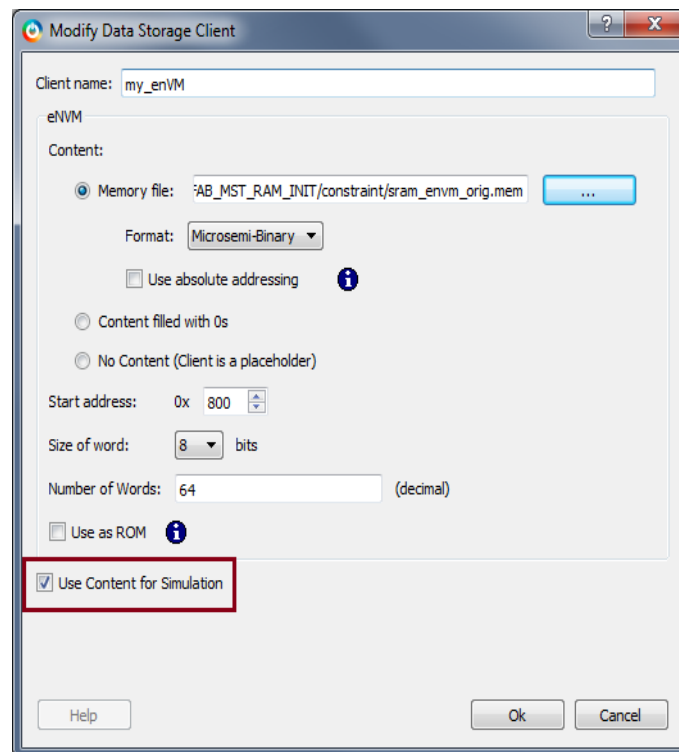


Figure 16 • Use Content for Simulation Data Storage for Client Option

Using the **User Content for Simulation** option, the Data Client mem file content is automatically used by the simulation model and the user do not have to emulate the process of writing into eNVM.

Figure 17 shows the simulation transcript waveform results showing the eNVM read data at the equivalent eNVM address.

```

Transcript
# NVM_0: Write into CMD Reg Addr: 1fc : CMD: 00 : Page#: 00 : Sector#: 00
# NVM_0: User Read Data: 32'h00000006 : Mem Address: 1fc : Time: 3060 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 3130 ns
# NVM_0: User Read Data: 32'h04030201 : Mem Address: 800 : Time: 3260 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 3440 ns
# NVM_0: User Read Data: 32'hff530605 : Mem Address: 804 : Time: 3510 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 3690 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 808 : Time: 3820 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 4000 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 80c : Time: 4070 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 4250 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 810 : Time: 4380 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 4560 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 814 : Time: 4630 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 4810 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 818 : Time: 4940 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 5120 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 81c : Time: 5190 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 5370 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 820 : Time: 5500 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 5680 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 824 : Time: 5750 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 5930 ns
# NVM_0: User Read Data: 32'h0201ff53 : Mem Address: 828 : Time: 6060 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 6240 ns
# NVM_0: User Read Data: 32'h06050403 : Mem Address: 82c : Time: 6310 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 6490 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 830 : Time: 6620 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 6800 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 834 : Time: 6870 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 7050 ns
# NVM_0: User Read Data: 32'h0201ff53 : Mem Address: 838 : Time: 7180 ns
# NVM_0: User Read Data: 32'hc00000001 : Mem Address: 120 : Time: 7360 ns
# NVM_0: User Read Data: 32'h06050403 : Mem Address: 83c : Time: 7430 ns
# NVM_0: Write into CMD Reg Addr: 1fc : CMD: 00 : Page#: 00 : Sector#: 00
V$IM 2>

```

Figure 17 • Transcript eNVM Data and Address Results

Figure 18 shows the ModelSim presynthesis simulation waveform results.

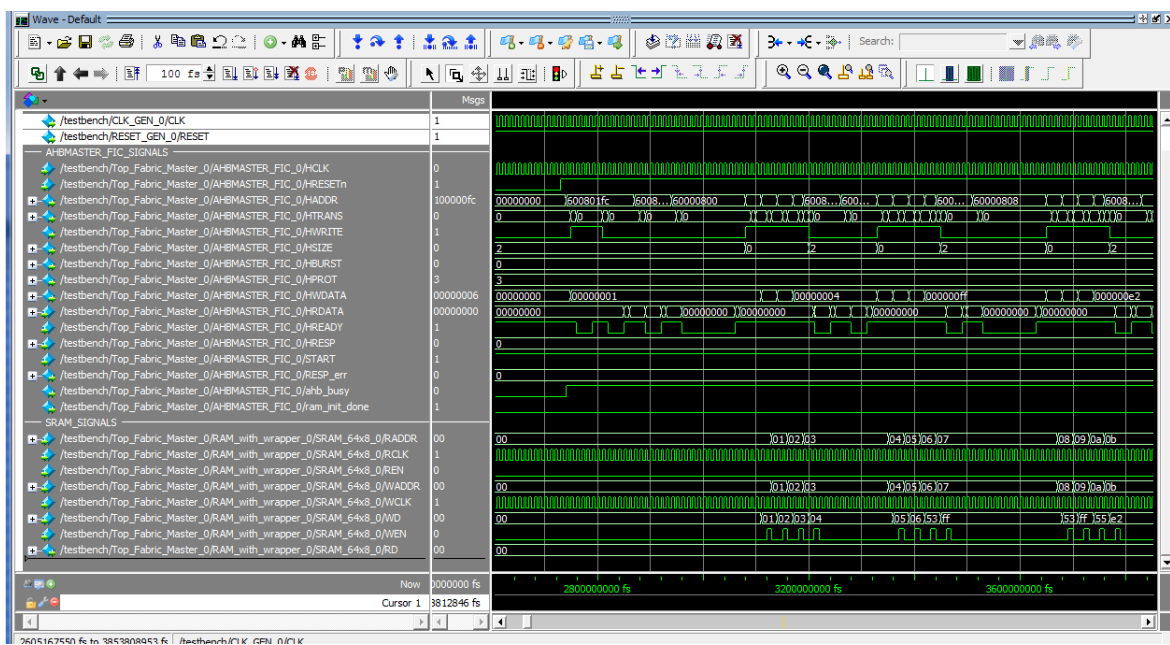


Figure 18 • Fabric Master Design Example Simulation Waveform (1)

Figure 19 shows the HRDATA is 04030201 at the eNVM address 800 which matches with the SRAM read data on WD.

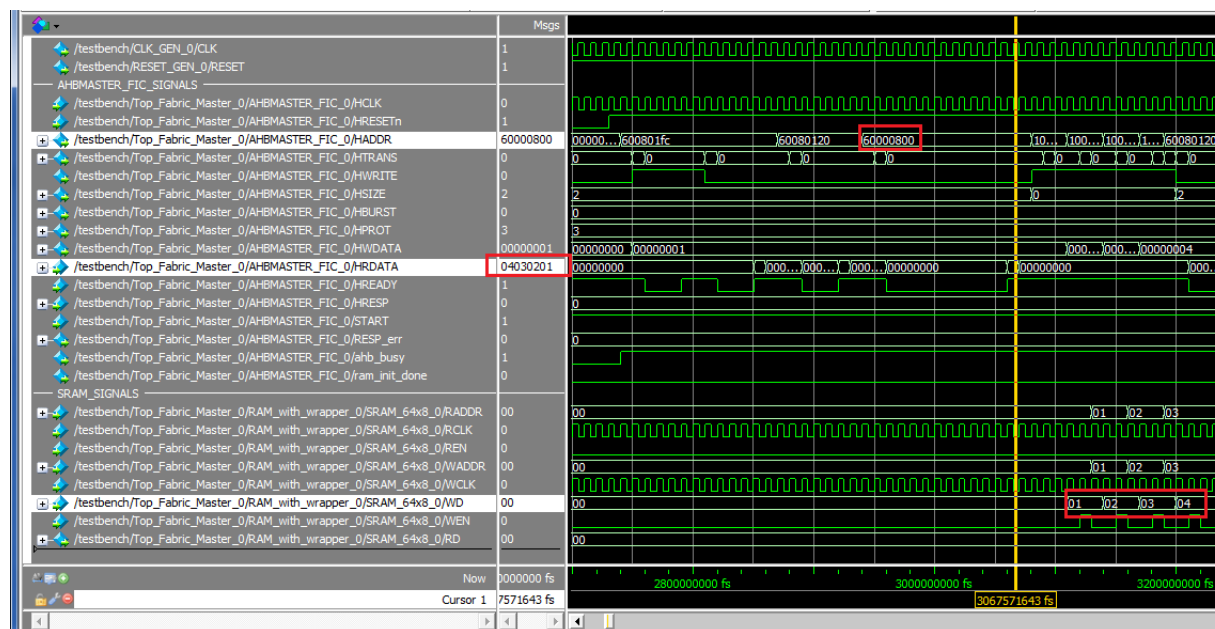


Figure 19 • Fabric Master Design Example Simulation Waveform (2)

Running the Design with a Fabric Master

This section describes running the design example in SmartFusion2 Security Evaluation Kit board where SRAM is initialized using a master in the fabric instead of the Cortex-M3 processor. The content of eNVM and SRAM is checked with real-time data using the SmartDebug tools as shown in the following steps:

1. Open the **FAB_MSTR_RAM_INIT** Libero project (refer to "Appendix: Design and Programming Files" on page 25).
2. Update the **eNVM client memory** file path, if needed. For more information, refer to <http://soc.microsemi.com/kb/article.aspx?id=SL5657>.
3. Program the SmartFusion2 Security Evaluation Kit board by selecting **Run PROGRAM** Action option in the Libero **Design Flow** window or with the provided fabric master version of STAPL file (refer to "Appendix: Design and Programming Files" on page 25) using FlashPro4.

4. Launch **SmartDebug** by selecting the **SmartDebug Design** option from the **Design Flow** window as shown in Figure 20. The **SmartDebug** window is displayed.

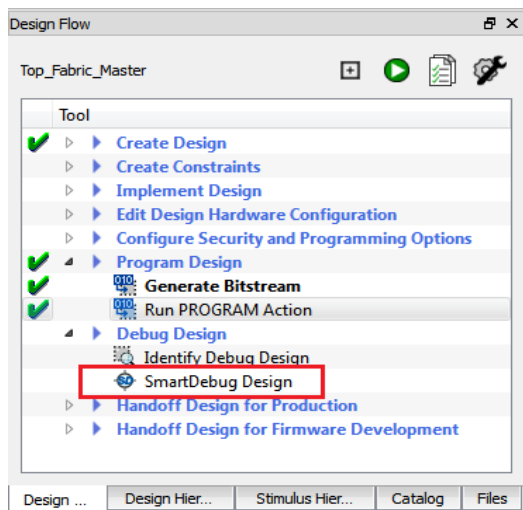


Figure 20 • SmartDebug Window Debug Options

5. From the SmartDebug window, click **View Flash Memory Content** to retrieve the eNVM content from the device. The **Flash Memory** window is displayed as shown in Figure 21 on page 22.
6. Enter the **Start Page** and **End Page** as 16 because the data storage client is stored in page16. Page 16 is used for demonstration purposes.
7. Click **Read from Device** as shown in Figure 21.

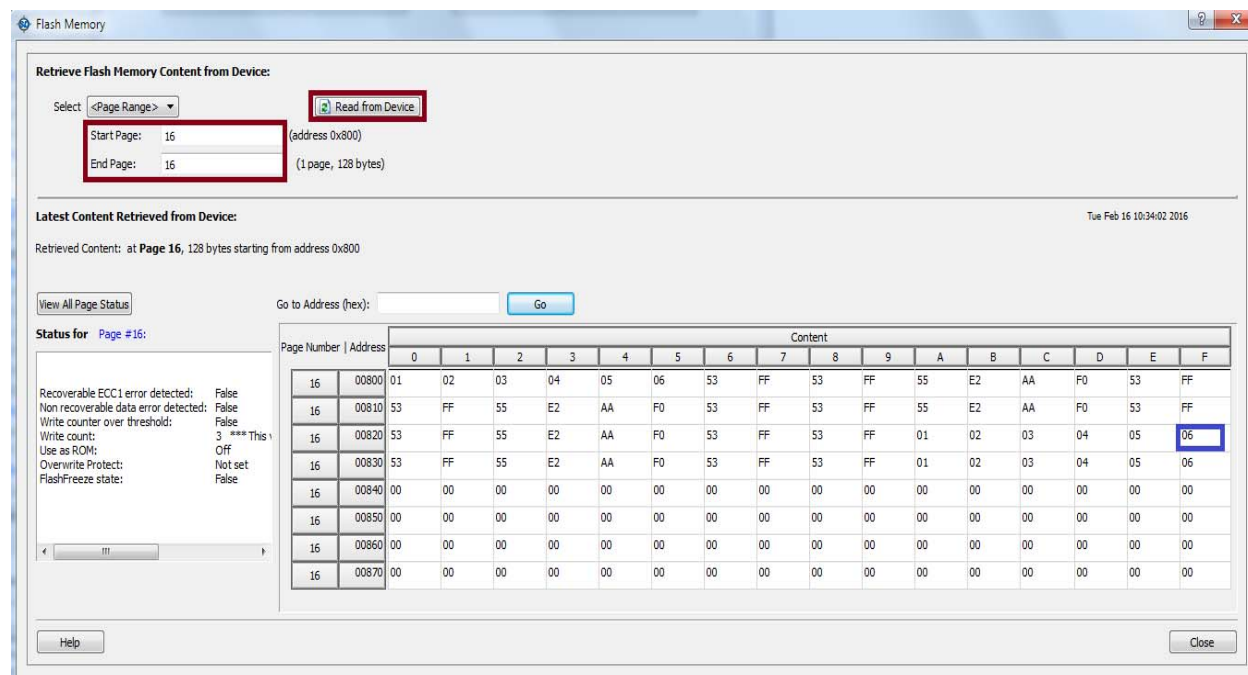


Figure 21 • Flash Memory (eNVM) Content Read from the Device

8. From the SmartDebug window, click **Debug FPGA Array**. The **Debug FGPA Array** window opens.

Note: Libero SoC generates the Debug File, <projectName>_debug.txt, during Place and Route and stores the file into the <project path>\designer folder. The Debug File contains information used by SmartDebug mainly for mapping the user design names to their respective physical addresses on the device. It also contains other information used during the debug process. SmartDebug when launched, automatically points to the debug file.

9. Select the **Memory Blocks** tab under **FPGA Array Debug Data** as shown in [Figure 22 on page 23](#).

The fabric master locks and gets exclusive access to the eNVM as it fetches the data from the eNVM to SRAM. In that case, no other masters, for example, SmartDebug, can access the eNVM until the fabric master completes the access operation and releases the lock on the eNVM.

In the fabric master example, SmartDebug is used to read the data, not the Cortex-M3 processor. SmartDebug reads the data from the eNVM and SRAM separately and validates both the data to be the same.

Before the fabric master unlocking the access on the eNVM if SmartDebug accesses the eNVM, the following error message is displayed:

Error: Unable to access embedded Flash Memory for your selected device: The firmware was unable to obtain exclusive access to the eNVM within the allotted time.

To release the lock on the eNVM after the fabric master has completed its access operations, you need to write 0x00 to REQACCESS register in eNVM control registers (address 0x600801FC) to release the access. Refer to the fabric master code (AHBMASTER_FIC.v) for more information.

- j. Click **Read Block** to read the SRAM content in real-time from the device. The content of the SRAM is displayed as shown in [Figure 22](#). This is the same eNVM data, as shown in [Figure 2 on page 5](#), that is used to initialize the SRAM.

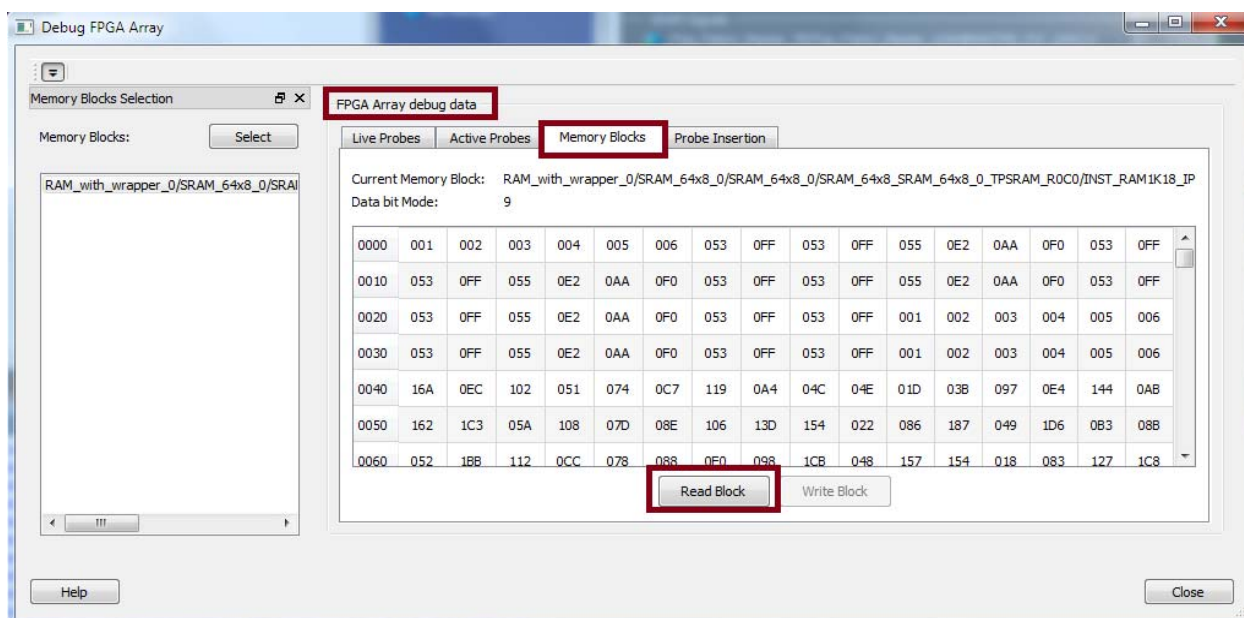


Figure 22 • SRAM Content Read from the Device

Customizing Wrapper Interface

This section describes how to customize SRAM initialization block.

The RAM_with_wrapper block presented in the design example can be modified based on the user SRAM configuration. In addition, the software code needs to be modified based on the user SRAM setting. Figure 23 on page 24 shows the RAM_with_wrapper block. It has three blocks as mentioned below:

- SRAM_64x8_0: Two-port SRAM block with depth 64 and width 8.
- mem_apb_wrp_0: Creates APB3 wrapper on SRAM port.
- mux_blk_0: Creates the Muxing arbiter.

Depending on the user SRAM block configuration, the SRAM64x8_0 setting needs to be updated. In addition, the DATA_WIDTH and ADDR_WIDTH parameter in mem_apb_wrp, and mux_blk file should be modified according to their design requirement and the blocks should be re-connected, if needed.

Note: The wrapper interface used in the design example supports up to 32-bit DATA_WIDTH.

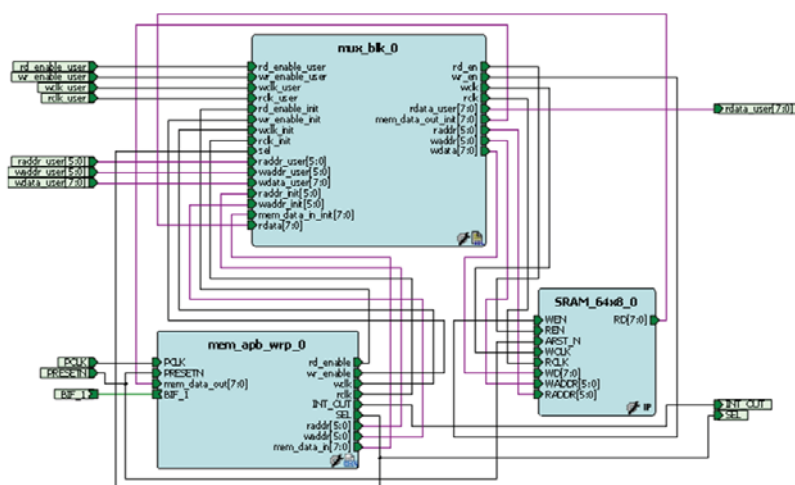


Figure 23 • RAM_with_wrapper Block

Conclusion

This design example shows how the SRAM blocks in SmartFusion2 SoC FPGA fabric can be initialized after power-up either by using the Cortex-M3 processor as the master or by using a master in the fabric. This example application uses an eNVM to initialize the SRAM after power-up. The eNVM can also be updated using the methods of programming, flash loader, or writing to eNVM, if needed. This application note presents an interface that can be instantiated into the user's design, performing the initialization at power-up. The reference design utilizes a very small portion of the FPGA logic for implementation, and does not affect the performance of the main design. The design in this document initializes a 64x8 SRAM block, but can be easily modified to support memory organizations of different width and depth.

Appendix: Design and Programming Files

The user can download the design files from the Microsemi website:

http://soc.microsemi.com/download/rsc/?f=m2s_ac392_liberov11p7_df_pf

The design file consists Libero Verilog projects, SoftConsole software project, and programming files (*.stp) for SmartFusion2 Security Evaluation Kit. Two programming files are included: the Cortex-M3 processor as the master (`Top_M3_Master.stp`), and the fabric master (`Top_Fabric_Master.stp`) files. Refer to the `Readme.txt` file included in the design file for the directory structure and description.

List of Changes

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

Revision*	Changes	Page
Revision 10 (March 2016)	Updated the document for Libero v11.7 software release (SAR 76443).	NA
Revision 9 (October 2015)	Updated the document for Libero v11.6 software release (SAR 68375).	NA
Revision 8 (January 2015)	Updated the document for Libero v11.5 software release (SAR 62937).	NA
Revision 7 (December 2014)	Removed all instances of and references to M2S100 device from Table 2 (SAR 62858).	2
Revision 6 (September 2014)	Updated the document for Libero v11.4 software release (SAR 59071).	NA
	Updated the document for SmartFusion2 Evaluation Kit details (SAR 59071).	NA
Revision 5 (March 2014)	Added " Purpose " section (SAR 51324)	1
	Updated Figure 1 , Figure 2 , Figure 5 , Figure 6 , and Figure 8 (SAR 51324)	4 , 5 , 9 , 10 , and 13
	Updated " SRAM Initialization Reference Designs " section (SAR 51324)	7
	Added " Cortex-M3 Processor as Master " section (SAR 51324)	7
	Updated " Running the Design with Cortex-M3 Processor as Master " section (SAR 51324)	14
	Added " Initializing SRAM using Fabric Master " section (SAR 51324)	17
	Added " Simulating Reference Design with a Fabric as Master " section (SAR 51324)	19
	Added " Running the Design with a Fabric Master " section (SAR 51324)	21
	Updated " Appendix: Design and Programming Files " section (SAR 51324)	25
Revision 4 (December 2013)	Updated Figure 1 and Figure 8 (SAR 51324).	4 , 13
Revision 3 (June 2013)	Modified " Introduction " section (SAR 48177).	1
	Modified " SmartFusion2 eNVM Controller for Data Storage " section (SAR 48177).	3
	Modified " SRAM Initialization Reference Designs " section (SAR 48177).	7
	Modified " Fabric Master " section (SAR 48177).	8
	Modified " Appendix: Design and Programming Files " section (SAR 48177).	25
	Modified Table 2 (SAR 48177).	2
	Added Figure 5 , Figure 6 and Figure 8 (SAR 48177).	9 , 10 , 13
Revision 2 (March 2013)	Updated the document for Libero SoC v11.0 beta SP1 release and made required changes for better usage of the term 'SEL' (SAR 45591).	NA
Revision 1 (November 2012)	Updated " Introduction " section. (SAR 42893)	1
	Updated " Appendix: Design and Programming Files " section (SAR 42893)	25



Microsemi Corporate 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

E-mail: sales.support@microsemi.com

© 2016 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace 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 Solution; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 4,800 employees globally. Learn more at 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.