

---

# SmartFusion2 SoC FPGA SRAM Initialization from eNVM

---

## Table of Contents

---

References . . . . .	1
Introduction . . . . .	2
Tools Required . . . . .	2
Embedded SRAM Blocks in SmartFusion2 SoC FPGAs . . . . .	2
SRAM Initialization Reference Designs . . . . .	6
Initialize SRAM using Cortex-M3 Processor as the Master . . . . .	10
Initializing the SRAM using Fabric Master . . . . .	14
Customizing the Wrapper Interface . . . . .	21
Conclusion . . . . .	22
Appendix A - Design and Programming Files . . . . .	22
List of Changes . . . . .	23

---

## References

The following list of references is used in this document. The references complement and help in understanding the relevant Microsemi<sup>®</sup> SmartFusion<sup>®</sup> 2 system-on-chip (SoC) field programmable gate array (FPGA) device features and flows that are demonstrated in this document:

- [SmartFusion2 Microcontroller Subsystem User Guide](#)
- [SmartDebug – Hardware Design Debug Tools Tutorial](#)
- [SmartFusion2 Development Kit Board](#)
- [SmartFusion2 MSS Embedded Nonvolatile Memory \(eNVM\) Simulation](#)
- [SmartFusion2 SoC FPGA Fabric User Guide](#)

## Introduction

SmartFusion<sup>®</sup>2 system-on-chip (SoC) field programmable gate array (FPGA) devices have embedded static random access memory (SRAM) blocks in fabric. There are two types of SRAM blocks in SmartFusion2 FPGA fabric: Large SRAMs (LSRAMs) and Micro SRAMs (uSRAMs). The LSRAMs are used for storing large data or for creating big 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 offers 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 embedded non-volatile memory (eNVM) block using the ARM<sup>®</sup> Cortex<sup>™</sup>-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 and Figure 5 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 Development Kit board.

## Tools Required

Table 1 lists the reference design requirements and details.

**Table 1 • Reference Design Requirements and Details**

Reference Design Requirements and Details	Description
<b>Hardware Requirements</b>	
SmartFusion2 Development Kit	Rev C or later
<b>Software Requirements</b>	
Libero <sup>®</sup> System-on-Chip (SoC)	11.3
FlashPro programming software	11.3
SoftConsole	3.4
Terminal Emulator Program	HyperTerminal or Equivalent

## 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 SoC FPGA devices.

**Table 2 • SRAM Blocks in Various SmartFusion2 SoC FPGA Devices**

Features	M2S005	M2S010	M2S025	M2S050	M2S090	M2S100	M2S150
LSRAM 18K Blocks	10	21	31	69	109	160	236
uSRAM 1K Blocks	11	22	34	72	112	160	240
Total RAM (KBits)	191	400	592	1,314	2074	3040	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 the Wrapper Interface" section on page 21.

## SmartFusion2 SoC FPGA 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, refer to Embedded eNVM Controller section in the *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 64x8 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.

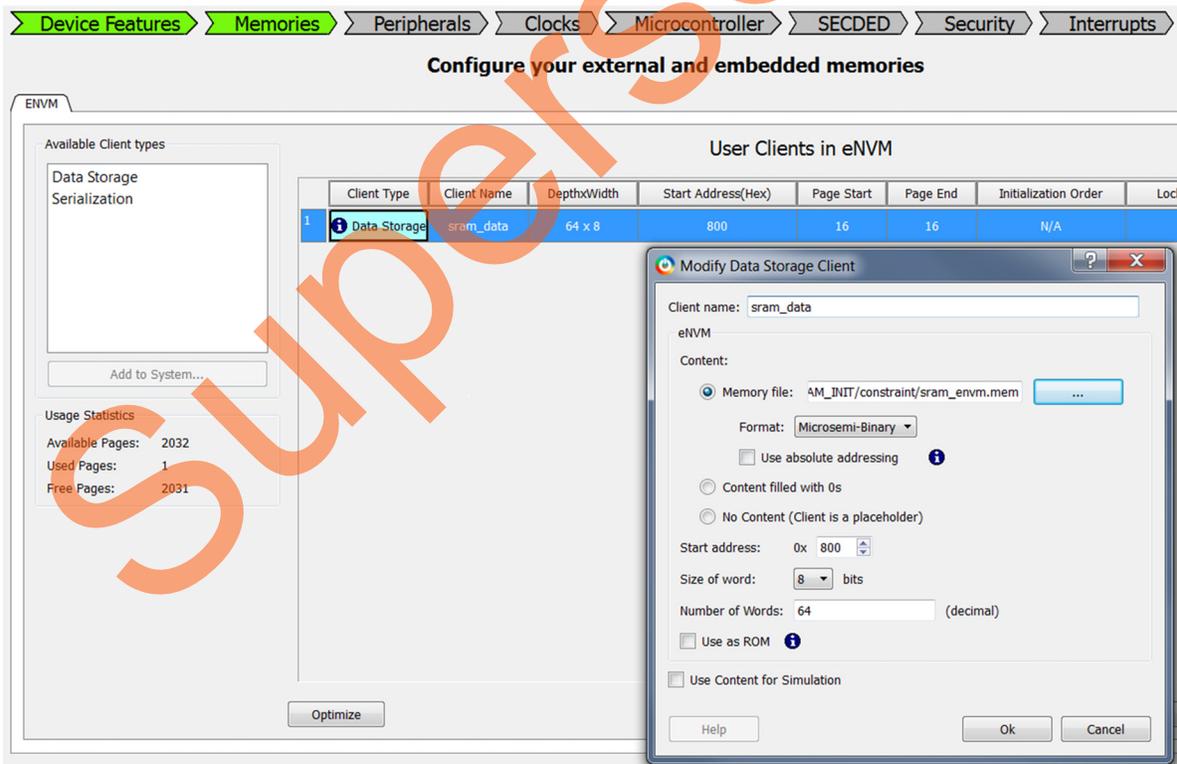
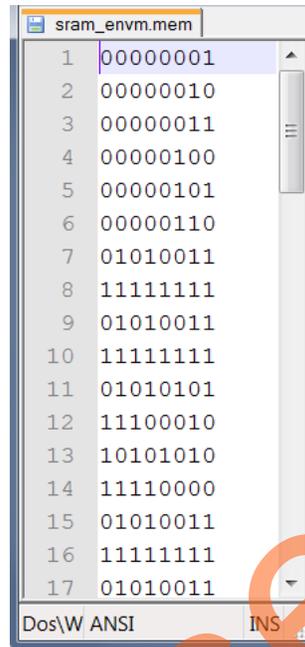


Figure 1 • Data Storage eNVM Client (System Builder)

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.



```
sram_envm.mem
1 00000001
2 00000010
3 00000011
4 00000100
5 00000101
6 00000110
7 01010011
8 11111111
9 01010011
10 11111111
11 01010101
12 11100010
13 10101010
14 11110000
15 01010011
16 11111111
17 01010011
Dos\W ANSI  INS
```

**Figure 2 • Memory file Content Saved into the eNVM**

## SRAM to APB3 Wrapper

The section describes connecting the SRAM block to the advanced microcontroller bus architecture (AMBA<sup>®</sup>) 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.

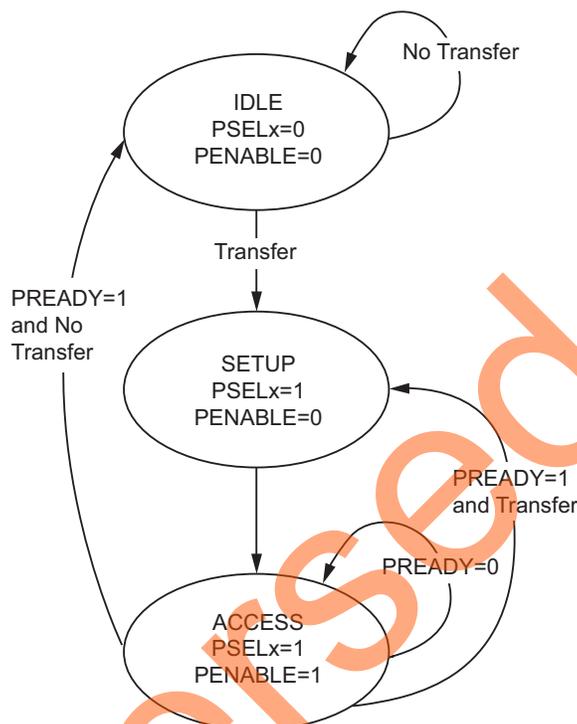


Figure 3 • APB3 State Diagram

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 the Master](#): This section describes the method of initializing SRAM using the Cortex-M3 processor as the master.
- [Fabric Master](#): This section describes the method of initializing SRAM using a fabric master.

### Cortex-M3 Processor as the Master

The SRAM block is configured as two-port memory with a depth of 64 and a width of 8. This design implements an advanced peripheral bus 3 (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.

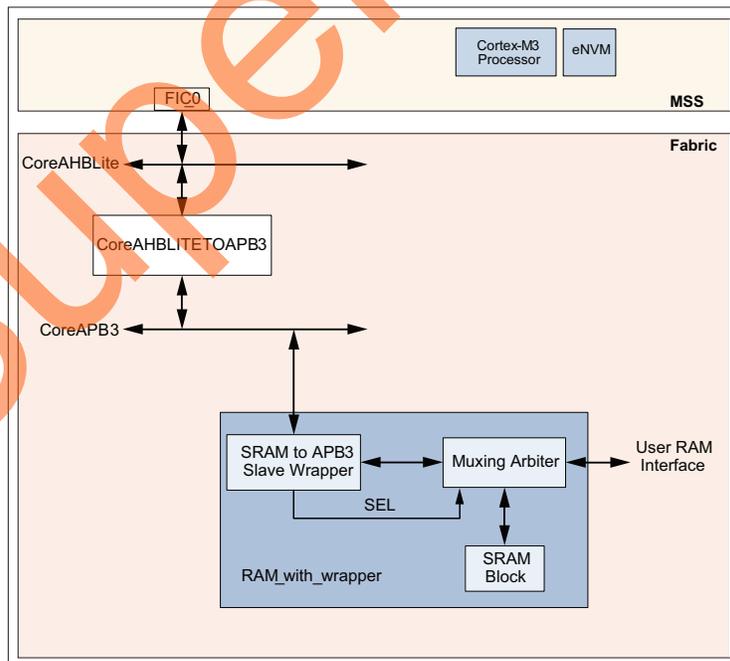


Figure 4 • Design Example Block Diagram

## Interface Description

Table 3 shows the top-level Cortex-M3 processor as the master interface signal descriptions. Refer to *SmartFusion2 SoC FPGA Fabric User Guide* for more details on the LSRAM and uSRAM functionalities and features.

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

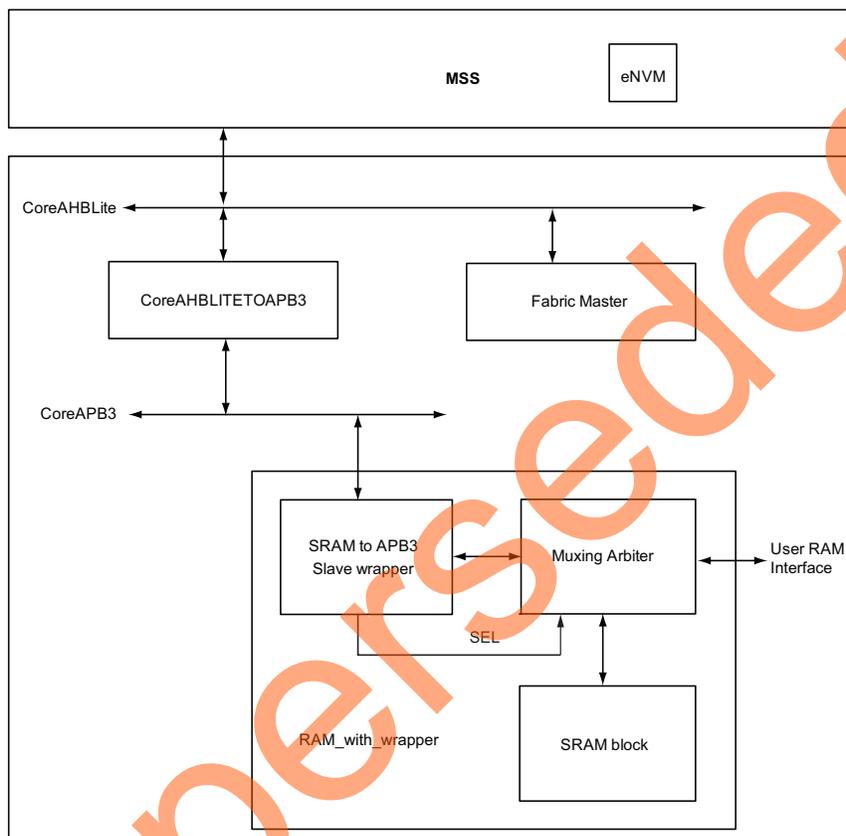
Signal	Direction	Description
raddr_user[5:0]	Input	User read address
rclk_user	Input	User read clock
rd_enable_user	Input	User read enable
waddr_user[5:0]	Input	User write address
wclk_user	Input	User write clock
wdata_user[7:0]	Input	User write data
wr_enable_user	Input	User write enable
rdata_user[7:0]	Output	User read data
INIT_DONE	Output	Initialization complete
DEVRST_N	Input	Active low reset
MMUART_0_RXD_F2M	Input	Uart RX input (for debug only)
MMUART_0_TXD_M2F	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. The 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 an AHB\_lite 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. Figure 6 shows the state diagram for the fabric master.

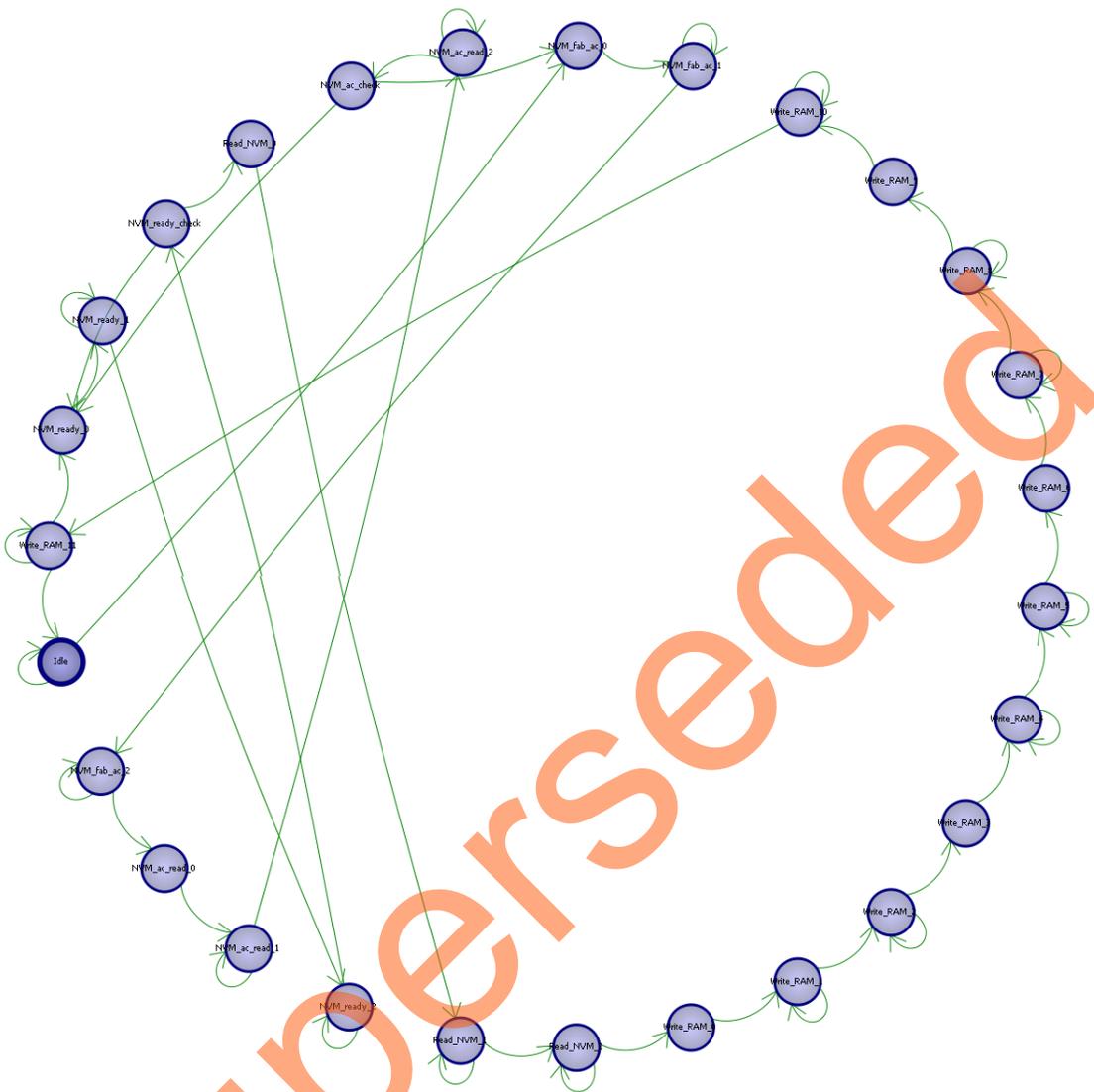


Figure 6 • Fabric Master State Diagram

## 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_0_RXD_F2M	Input	Uart RX input (for debug only)
MMUART_0_TXD_M2F	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

## Initialize SRAM using Cortex-M3 Processor as the Master

This section explains the following topics:

- Hardware Implementation
- Firmware and Application Code Software Implementation
- Simulating Reference Design with the Cortex-M3 Processor as the Master
- Running the Design with the Cortex-M3 Processor as the 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 of MSS, SRAM wrapper logic, and IP cores (CoreAHBToAPB3, CoreAPB3) as shown in Figure 7.

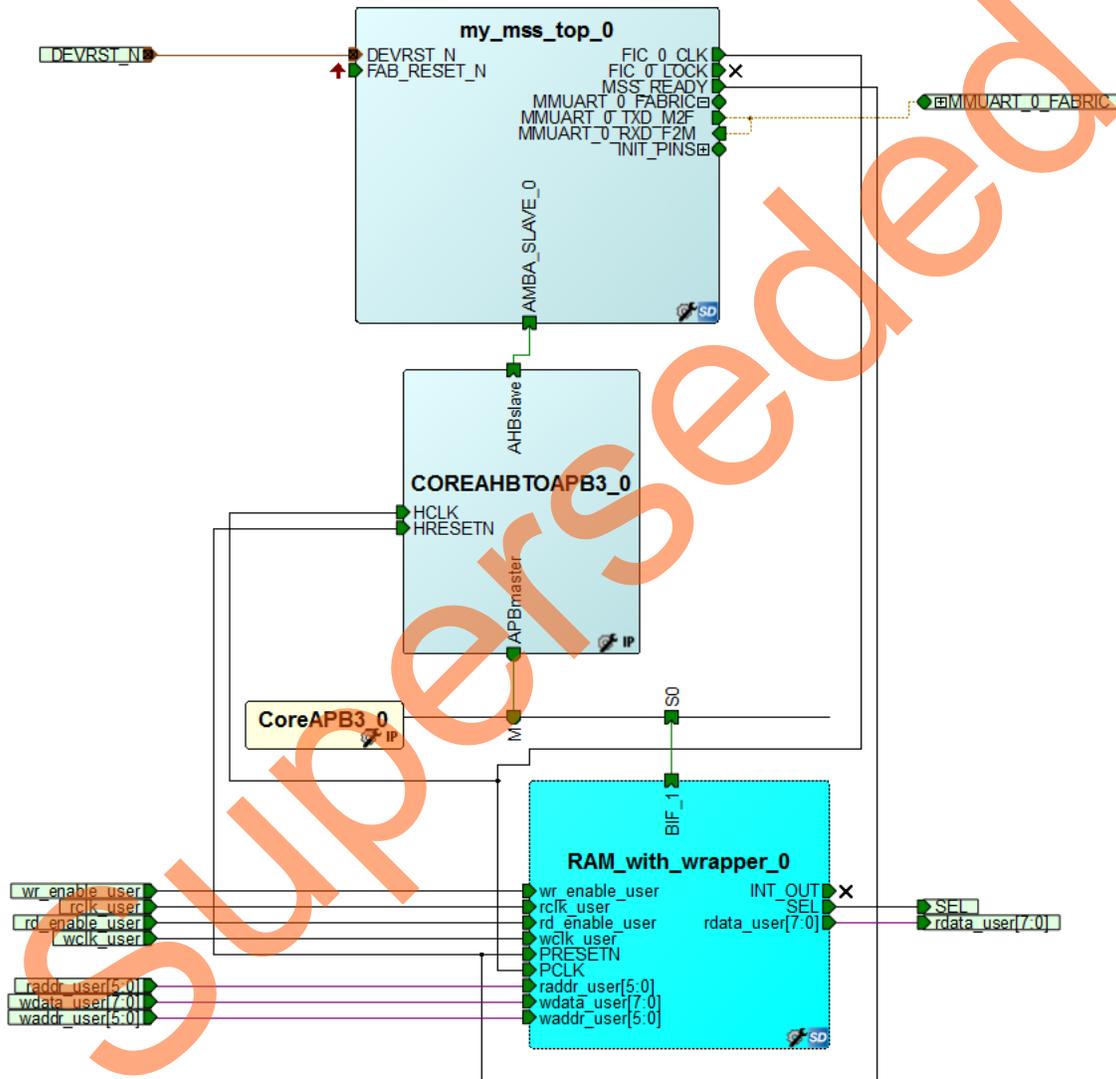


Figure 7 • Top-Level Hardware Design for Cortex-M3 Processor as the Master

CoreAHBLite 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\_0 block. The MMUART\_0 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 the Cortex-M3 Processor as the Master

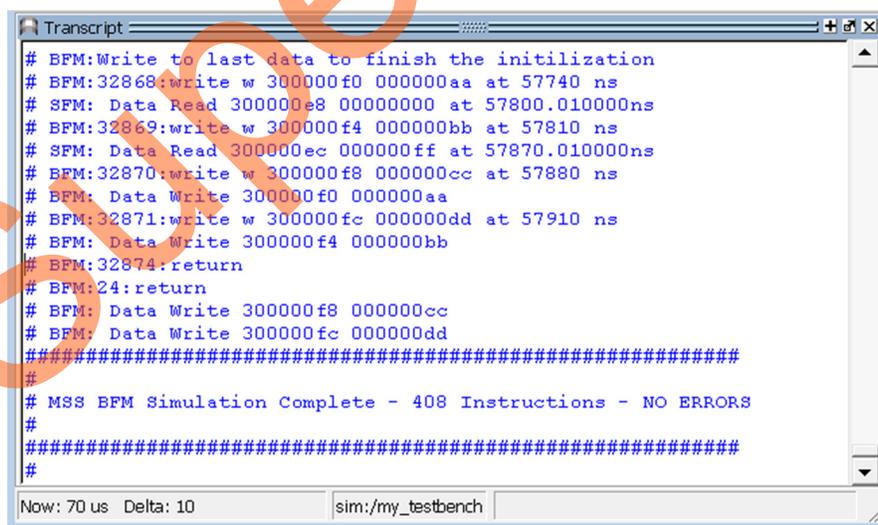
The design file includes the test bench files to run simulation in the Libero SoC. The simulation uses the bus functional model (BFM) command to exercise data transfer between the MSS and the fabric.

**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 shows the Modelism presynthesis simulation waveform results.



```

Transcript
# BFM:Write to last data to finish the initialization
# BFM:32868:write w 300000f0 000000aa at 57740 ns
# SFM: Data Read 300000e8 00000000 at 57800.010000ns
# BFM:32869:write w 300000f4 000000bb at 57810 ns
# SFM: Data Read 300000ec 000000ff at 57870.010000ns
# BFM:32870:write w 300000f8 000000cc at 57880 ns
# BFM: Data Write 300000f0 000000aa
# BFM:32871:write w 300000fc 000000dd at 57910 ns
# BFM: Data Write 300000f4 000000bb
# BFM:32874:return
# BFM:24:return
# BFM: Data Write 300000f8 000000cc
# BFM: Data Write 300000fc 000000dd
#####
#
# MSS BFM Simulation Complete - 408 Instructions - NO ERRORS
#
#####
#
Now: 70 us Delta: 10 sim:/my_testbench
    
```

Figure 8 • BFM Transcript Simulation Results

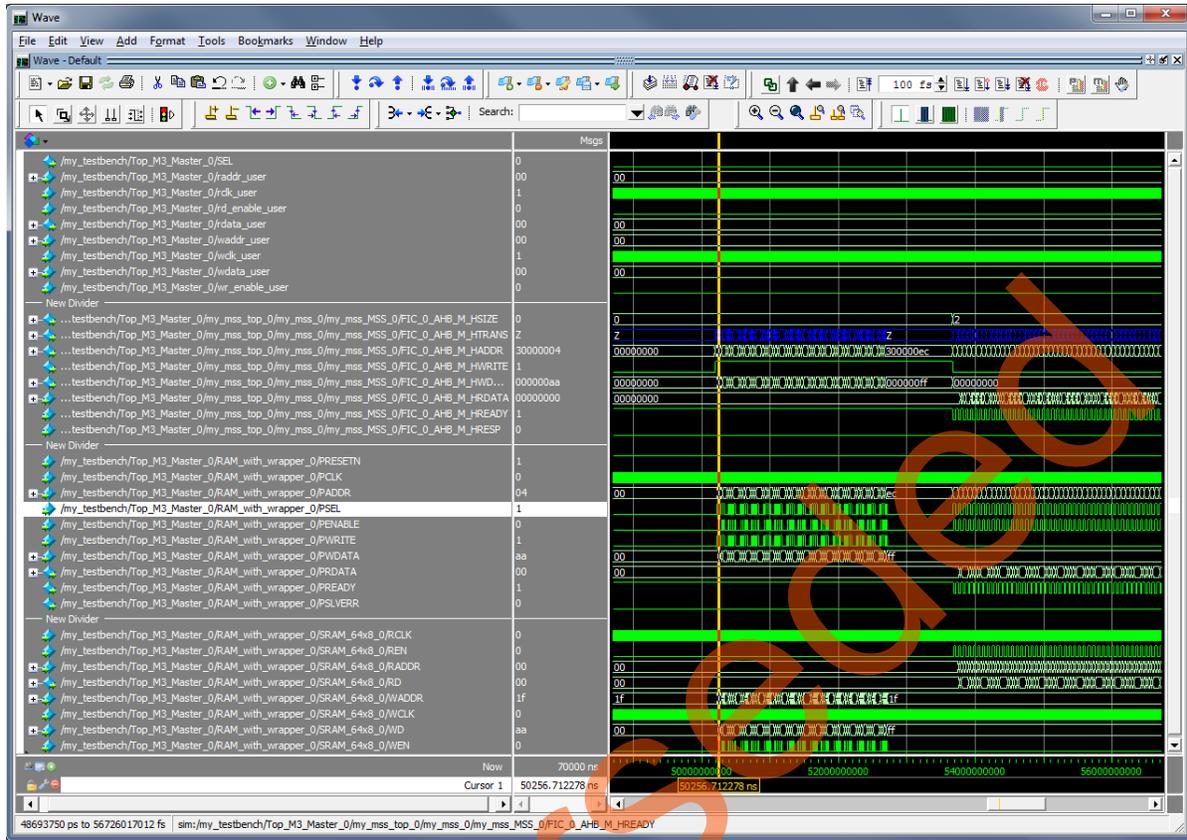


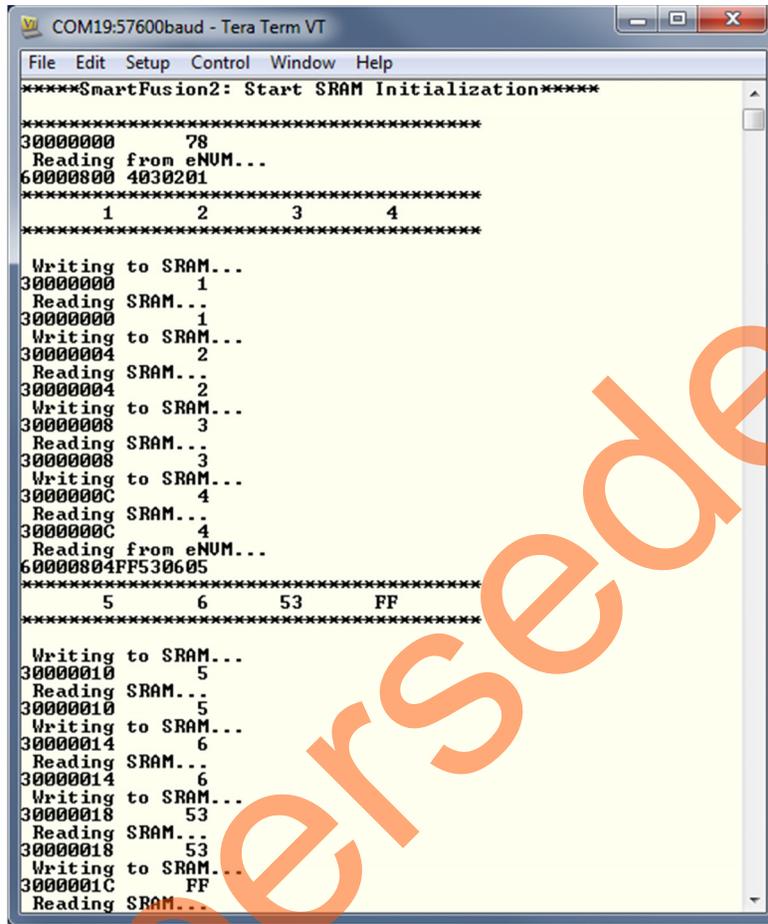
Figure 9 • MSS Master Design Example Waveform

## Running the Design with the Cortex-M3 Processor as the Master

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

1. Program the SmartFusion2 SoC FPGA device Development Kit board with the provided Cortex-M3 processor as the master STAPL file (refer to "Appendix A - Design and Programming Files" on page 22) using FlashPro4.
2. Set the jumpers on the Development Kit board as:
  - a. J129 (Pin 2 - Pin 3)
  - b. J133 (Pin 2 - Pin 3)
3. Connect the USB to PC.
4. Invoke the SoftConsole integrated design environment (IDE) from Libero SoC project and launch the debugger.
5. 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 Configuring Serial Terminal Emulation Programs tutorial for configuring HyperTerminal, Tera Term, or PuTTY.

- Run the debugger in the SoftConsole, the HyperTerminal window shows the initialization sequence by reading eNVM and writing to SRAM. Figure 10 shows the screenshot of HyperTerminal.



```

COM19:57600baud - Tera Term VT
File Edit Setup Control Window Help
*****SmartFusion2: Start SRAM Initialization*****
*****
30000000      78
Reading from eNUM...
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 eNUM...
60000804FF530605
*****
      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 10 • Screenshot of HyperTerminal Showing the Design Example

## Initializing the 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 details the hardware implementation using a fabric master. In addition, it also details how to simulate the provided design along with the steps on how to run the design on the SmartFusion2 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 of MSS, SRAM wrapper logic, fabric master (AHBMASTER\_FIC\_0) as shown in Figure 11.

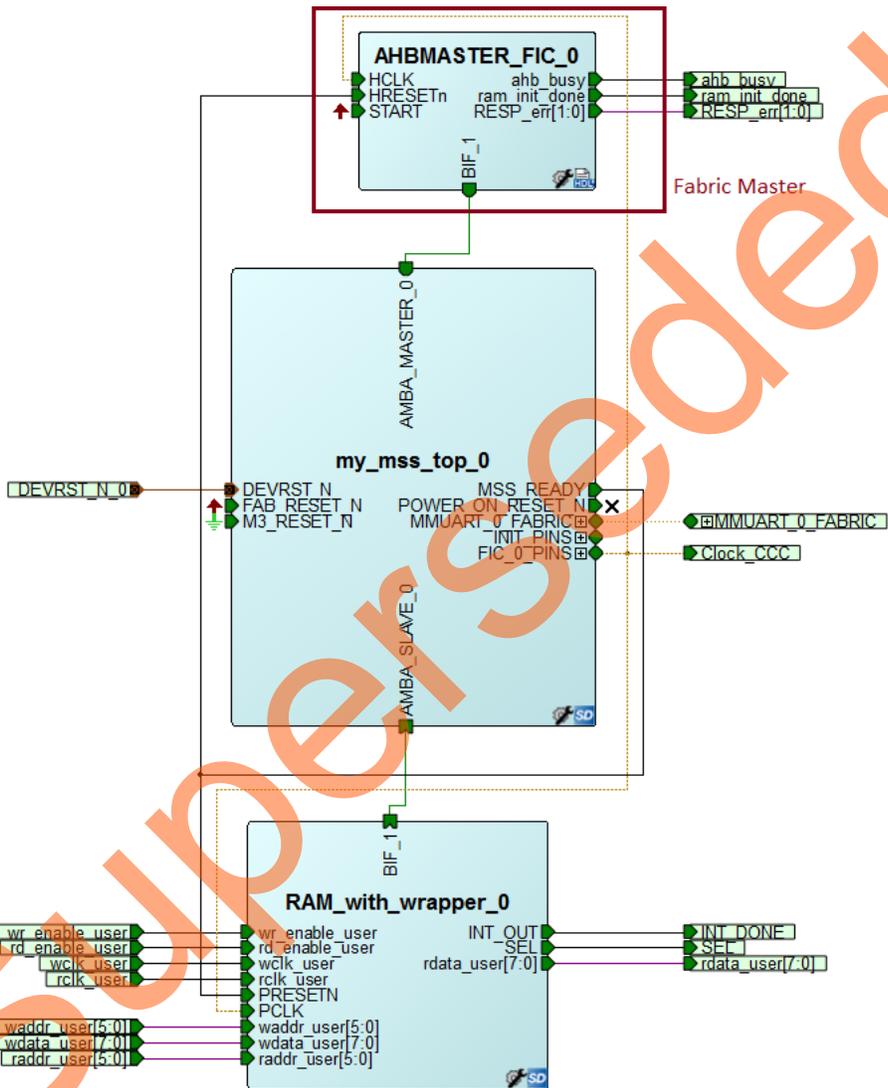


Figure 11 • Top-Level Hardware Design for Fabric Master

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 and Figure 2 for more details.

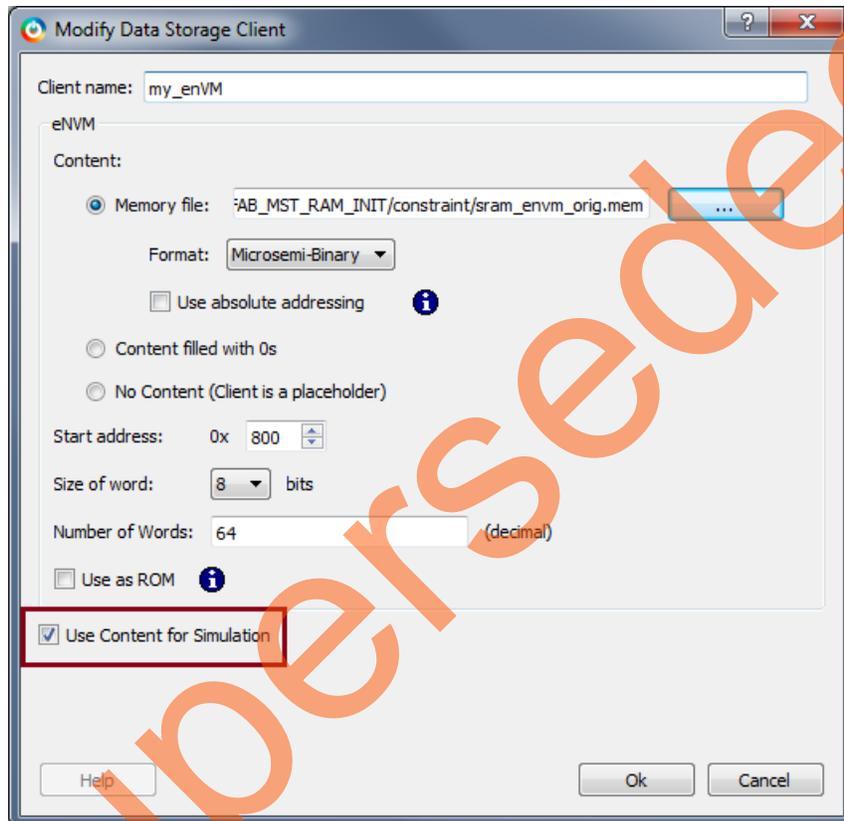
At power-up or at power-on reset, the Cortex-M3 processor fetches the initial stack pointer from 0x00000000 (eNVM address 0x60000000) and address of the reset handler from 0x00000004 (eNVM address 0x60000004). If the execution control goes to the default reset handler, the boot up sequence is executed 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 11](#).

**Note:** To expose the M3\_RESET\_N signal, the System Builder block is re-opened as SmartDesign block. Refer to "Modifying/Inspecting Your System Builder Design" section in *SmartFusion2 System Builder User's Guide* for more details.

## Simulating Reference Design with a Fabric as Master

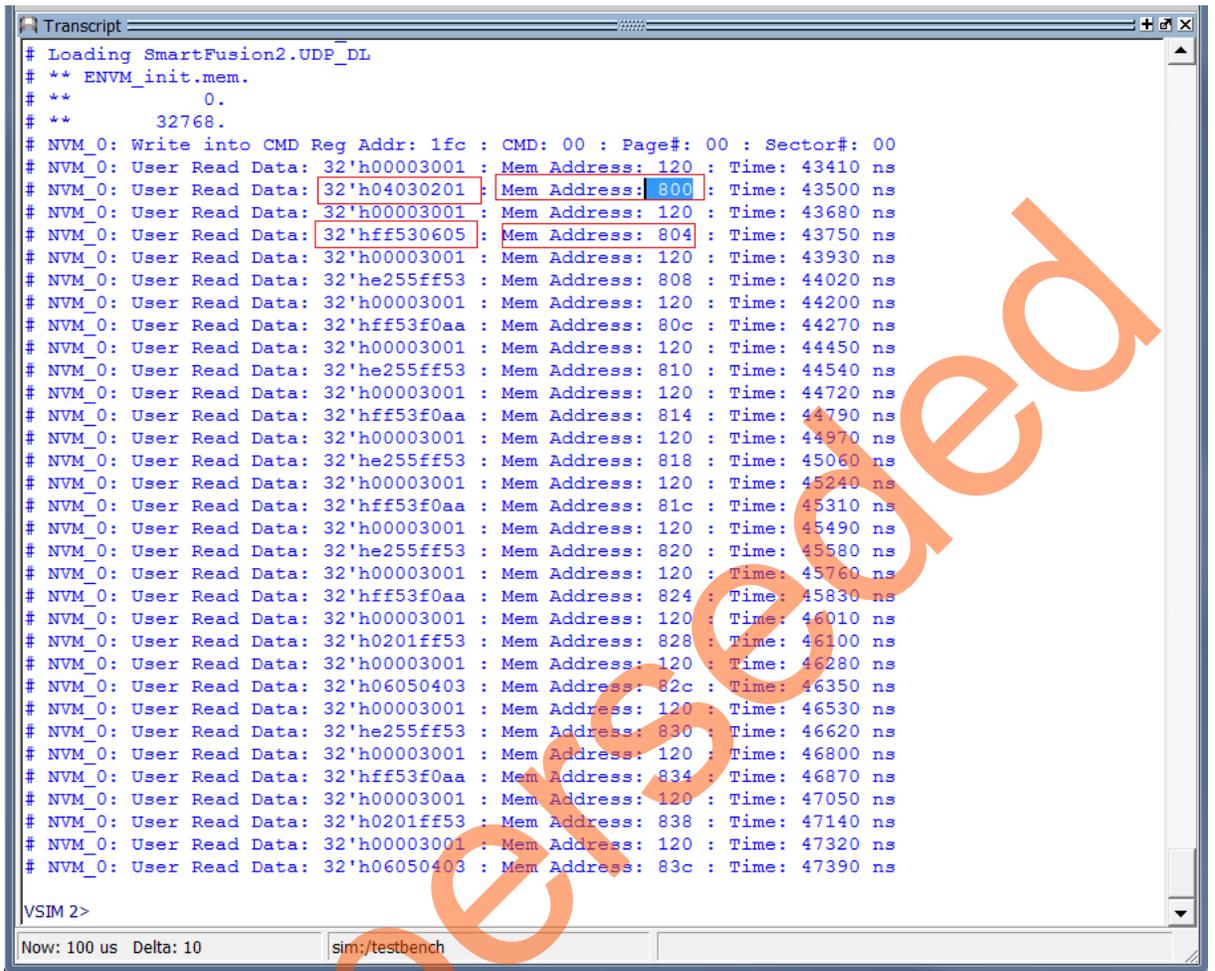
This section describes the detail of simulating the fabric master design using the top-level test bench, which is automatically generated by SmartDesign for Top\_Fabric\_Master component using the "Use Content for Simulation" option in the Data Storage Client Configurator as shown in [Figure 12](#).



**Figure 12 • Use Content for Simulation Data Storage for Client Option**

By using "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 13 shows the simulation transcript waveform results showing the eNVM read data at the equivalent eNVM address.



```
# Loading SmartFusion2.UDP_DL
# ** ENVM_init.mem.
# **      0.
# **      32768.
# NVM_0: Write into CMD Reg Addr: 1fc : CMD: 00 : Page#: 00 : Sector#: 00
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 43410 ns
# NVM_0: User Read Data: 32'h04030201 : Mem Address: 800 : Time: 43500 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 43680 ns
# NVM_0: User Read Data: 32'hff530605 : Mem Address: 804 : Time: 43750 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 43930 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 808 : Time: 44020 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 44200 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 80c : Time: 44270 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 44450 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 810 : Time: 44540 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 44720 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 814 : Time: 44790 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 44970 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 818 : Time: 45060 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 45240 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 81c : Time: 45310 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 45490 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 820 : Time: 45580 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 45760 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 824 : Time: 45830 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 46010 ns
# NVM_0: User Read Data: 32'h0201ff53 : Mem Address: 828 : Time: 46100 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 46280 ns
# NVM_0: User Read Data: 32'h06050403 : Mem Address: 82c : Time: 46350 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 46530 ns
# NVM_0: User Read Data: 32'he255ff53 : Mem Address: 830 : Time: 46620 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 46800 ns
# NVM_0: User Read Data: 32'hff53f0aa : Mem Address: 834 : Time: 46870 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 47050 ns
# NVM_0: User Read Data: 32'h0201ff53 : Mem Address: 838 : Time: 47140 ns
# NVM_0: User Read Data: 32'h00003001 : Mem Address: 120 : Time: 47320 ns
# NVM_0: User Read Data: 32'h06050403 : Mem Address: 83c : Time: 47390 ns

VSIM 2>
Now: 100 us Delta: 10 sim:/testbench
```

Figure 13 • Transcript eNVM Data and Address Results

Figure 14 and Figure 15 show the Modelsim presynthesis simulation waveform results.

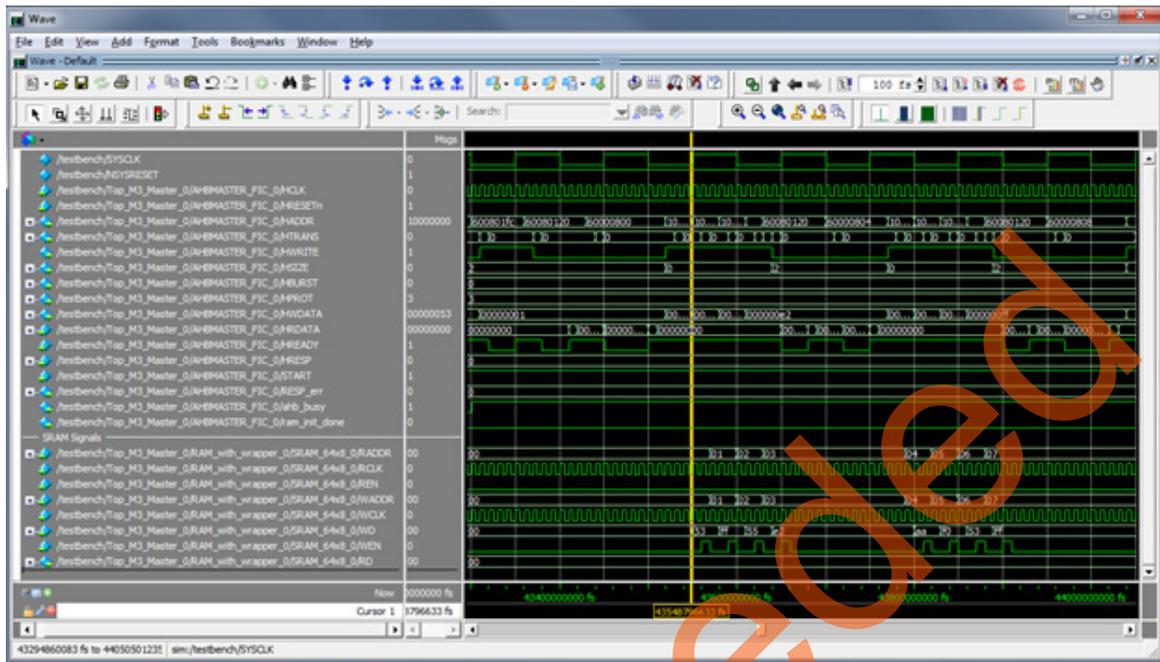


Figure 14 • Fabric Master Design Example Simulation Waveform (1)

Figure 15 shows the HRDATA is 04030201 at the eNVM address 800 which matches with the SRAM read data on WD.

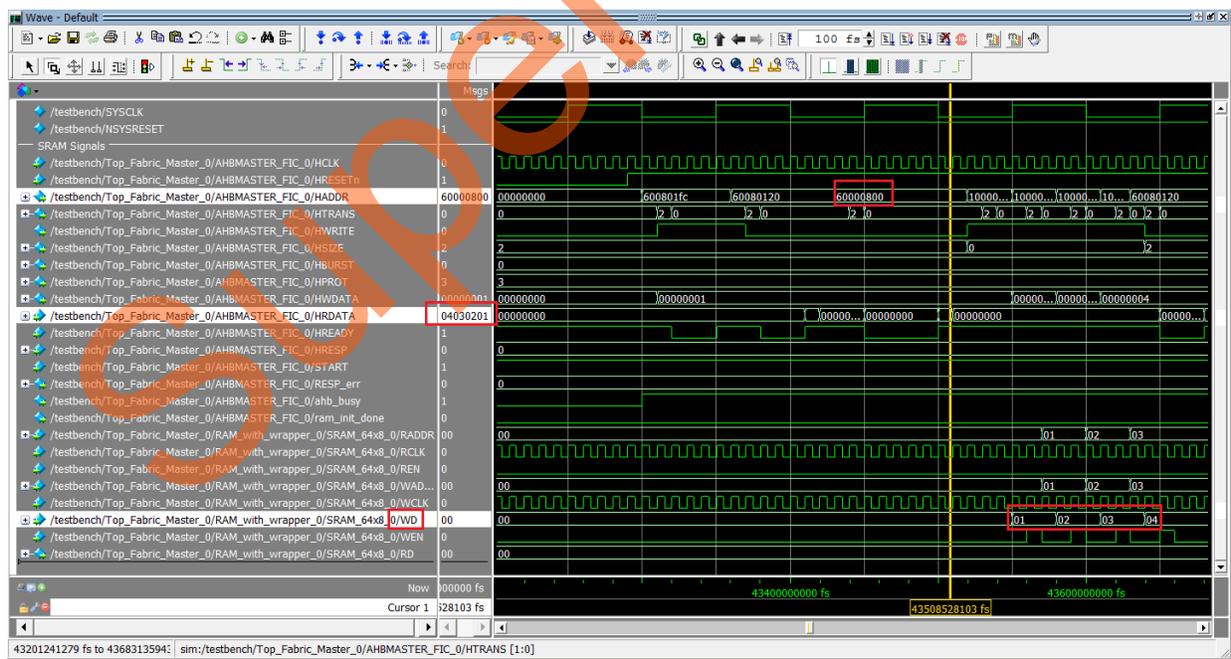


Figure 15 • Fabric Master Design Example Simulation Waveform (2)

## Running the Design with a Fabric Master

This section describes running the design example in SmartFusion2 Development 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. Program the SmartFusion2 SoC FPGA device Development Kit Board with the provided fabric master version of STAPL file (refer to "Appendix A - Design and Programming Files" on page 22) using FlashPro4.
2. Launch SmartDebug by selecting the SmartDebug Design option from the Design Flow window as shown in Figure 16. The SmartDebug window is displayed.

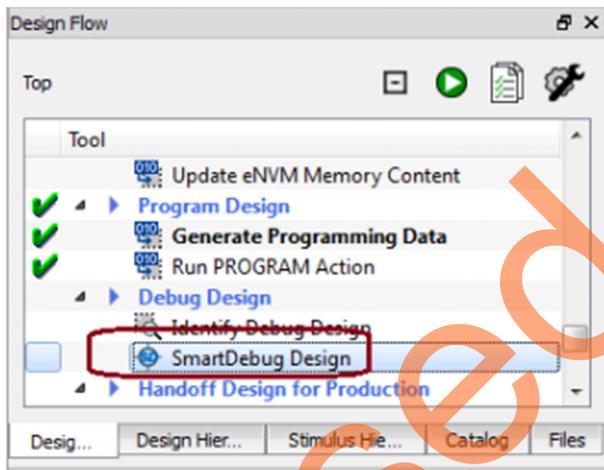


Figure 16 • Launching SmartDebug Design Tools

3. Click **View Flash Memory Content** to retrieve the eNVM content from the device using the SmartDebug window as shown in Figure 17. The Flash Memory window is displayed.

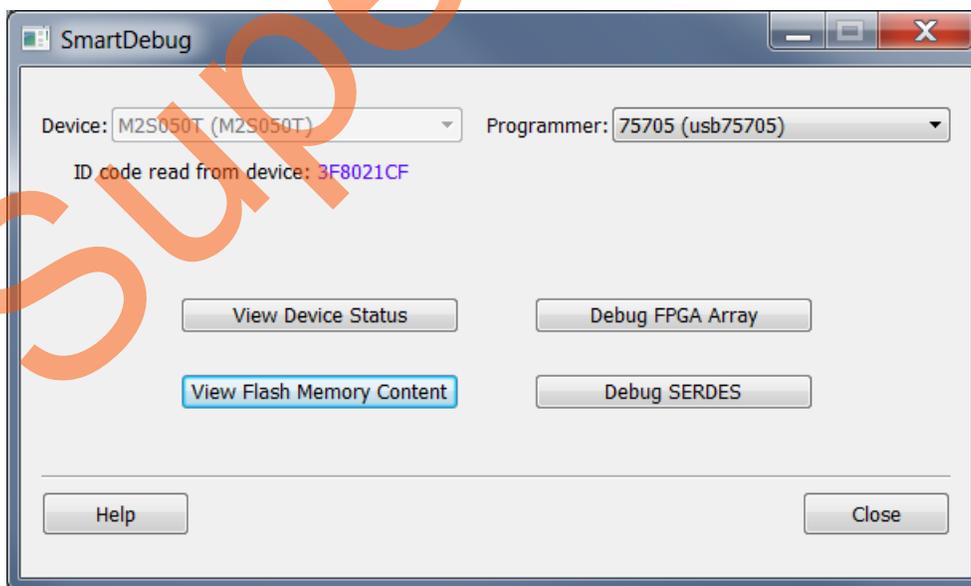
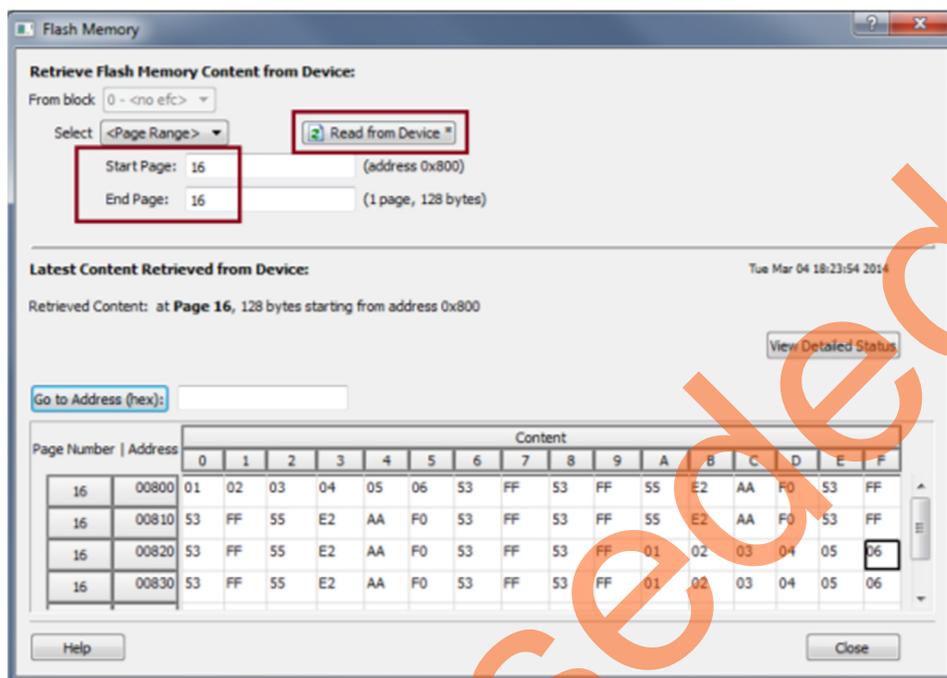


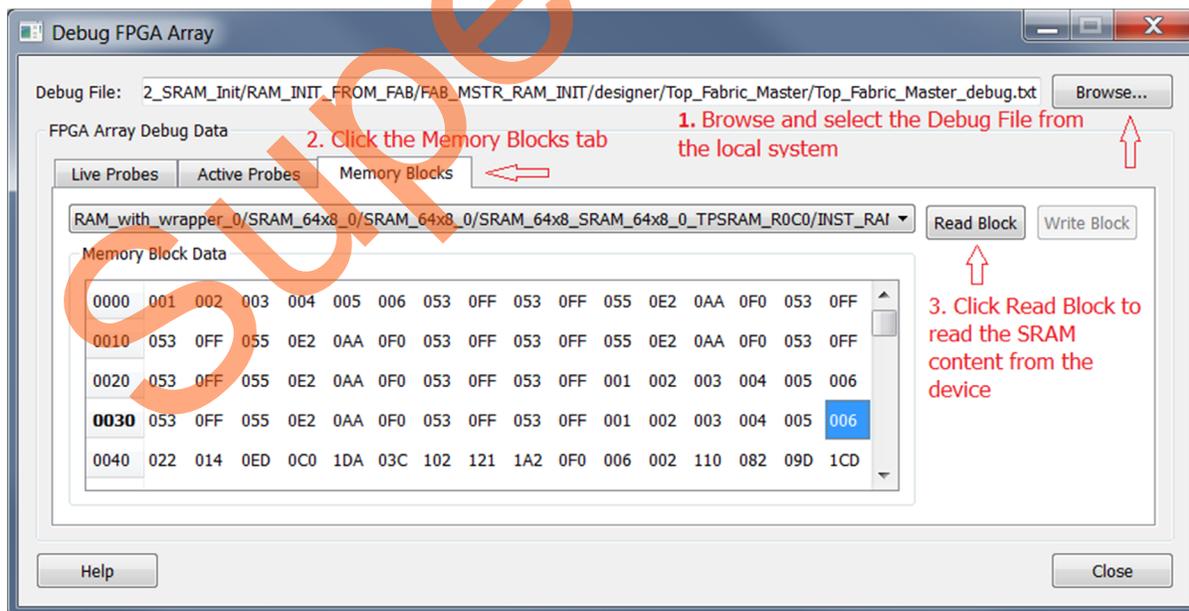
Figure 17 • SmartDebug Window Debug Options

4. 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.
5. Click **Read from Device** as shown in Figure 18.



**Figure 18 • Flash Memory (eNVM) Content Read from the Device**

6. Click **Debug FPGA Array** as shown in Figure 17 to open the Debug FGPA Array window as shown in Figure 19.



**Figure 19 • SRAM Content Read from the Device**

- a. Browse to select the Top\_Fabric\_Master\_debug.txt file. The Debug File must be specified before starting the FPGA Array Debug as shown in Figure 19. For example, the Debug File = <project root>/designer/Top/Top\_Fabric\_Master\_debug.txt  
 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.
- b. Select the **Memory Blocks** tab.
- c. 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 19. The SRAM data that is stored into eNVM which is used to initialize the SRAM block.

## Customizing the 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 20 shows the RAM\_with\_wrapper block. It has three blocks as mentioned below:

- SRAM64x8\_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.

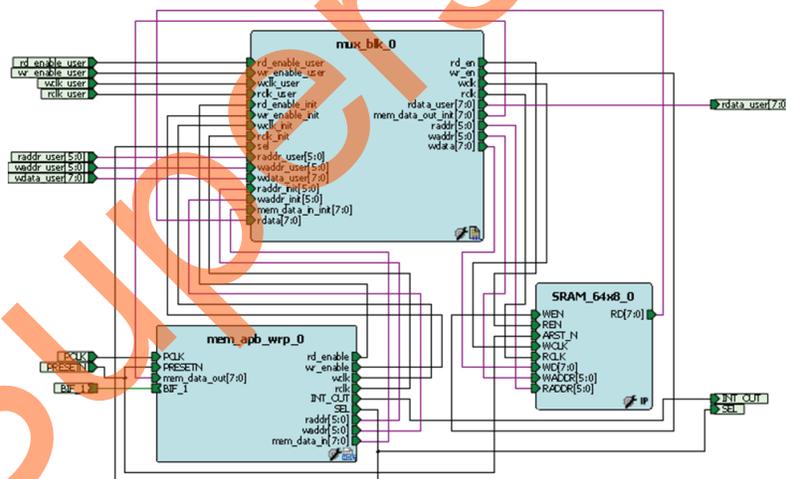


Figure 20 • RAM\_with\_wrapper Block

## 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 programming, or flash loader, or by 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 A - Design and Programming Files

The user can download the design files from the Microsemi SoC Products Group website:

[www.microsemi.com/soc/download/rsc/?f=M2S\\_AC392\\_DF](http://www.microsemi.com/soc/download/rsc/?f=M2S_AC392_DF).

The design file consists of Libero Verilog projects, SoftConsole software project, and programming files (\*.stp) for SmartFusion2 SoC FPGA Development 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.

Superseded

## List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision*	Changes	Page
Revision 5 (March 2014)	Added "References" section (SAR 51324)	1
	Updated Figure 1, Figure 2, Figure 5, Figure 6, and Figure 8 (SAR 51324)	3, 4, 8, 9, and 12
	Updated "SRAM Initialization Reference Designs" section (SAR 51324)	6
	Added "Cortex-M3 Processor as the Master" section (SAR 51324)	6
	Updated "Running the Design with the Cortex-M3 Processor as the Master" section (SAR 51324)	13
	Added "Initializing the SRAM using Fabric Master" section (SAR 51324)	14
	Added "Simulating Reference Design with a Fabric as Master" section (SAR 51324)	16
	Added "Running the Design with a Fabric Master" section (SAR 51324)	19
	Updated "Appendix A - Design and Programming Files" section (SAR 51324)	22
Revision 4 (December 2013)	Updated Figure 1 and Figure 8 (SAR 51324).	3, 12
Revision 3 (June 2013)	Modified "Introduction" section (SAR 48177).	2
	Modified "SmartFusion2 SoC FPGA eNVM Controller for Data Storage" section (SAR 48177).	3
	Modified "SRAM Initialization Reference Designs" section (SAR 48177).	6
	Modified "Fabric Master" section (SAR 48177).	8
	Modified "Appendix A - Design and Programming Files" section (SAR 48177).	22
	Modified Table 2 (SAR 48177).	2
	Added Figure 5, Figure 6 and Figure 8 (SAR 48177).	8, 9, 12
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)	2
	Updated "Appendix A - Design and Programming Files" section (SAR 42893)	22
<i>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.</i>		

Superseded



**Microsemi**<sup>®</sup>

**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](mailto:sales.support@microsemi.com)

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense and 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; security technologies and scalable anti-tamper products; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 3,400 employees globally. Learn more at [www.microsemi.com](http://www.microsemi.com).

© 2014 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.