SmartFusion2 SoC FPGA – Remapping eNVM, eSRAM, and DDR/SDR SDRAM Memories

Table of Contents

Purpose

This application note describes the remapping of the following memories to the ARM® Cortex™-M3 processor code region and explains how to execute the program code built with absolute addresses without remapping.

- Embedded nonvolatile memory (eNVM)
- Embedded random access memory (eSRAM)
- Double data rate (DDR)/single data rate (SDR) synchronous dynamic random access memory (SDRAM)

Introduction

SmartFusion®2 system-on-chip (SoC) field programmable gate array (FPGA) devices integrate an Cortex-M3 processor, up to 512 KB of eNVM, 64 KB of eSRAM, and memory interfaces for DDR/SDR SDRAM for program code, and data.

The Cortex-M3 processor has a predefined memory map for code space, data space, and system space with dedicated bus interfaces. The desired memory regions of the SmartFusion2 SoC FPGA can be mapped to the Cortex-M3 processor code space for the application program execution.

This application note explains how to remap the eNVM, eSRAM, and DDR/SDR SDRAM memories to the Cortex-M3 processor code region. This also explains how to execute the program code built with absolute addresses without remapping.

SmartFusion2 SoC FPGA Booting and Address Space Overview

This application note describes the SmartFusion2 SoC FPGA boot sequence, and how to remap the various memory regions to the Cortex-M3 processor code region, and to an optional softcore processor located in the FPGA fabric.

The Cortex-M3 processor is based on ARM architecture v-7M that includes a nested vectored interrupt controller (NVIC) for handling the interrupts, and includes a non-maskable interrupt. The NVIC contains the addresses of the initial stack pointer, exception handlers, and interrupt service routines (ISRs). The first entry in the NVIC should be the initial stack pointer and the second entry should be the address of

the reset exception handler. The Cortex-M3 processor eliminates the need for setting up the initial C runtime environment using assembly code. Developers can code entirely in the C language.
The Cortex-M3 processor upon reset reads two words from memory:

- At the address location 0x00000000 for the initial stack pointer
- At the address location 0x00000004 for the address of the reset handler exception

The reset handler performs the basic initialization and execution control which is given to the main application code. This execution flow is explained in Figure 1.

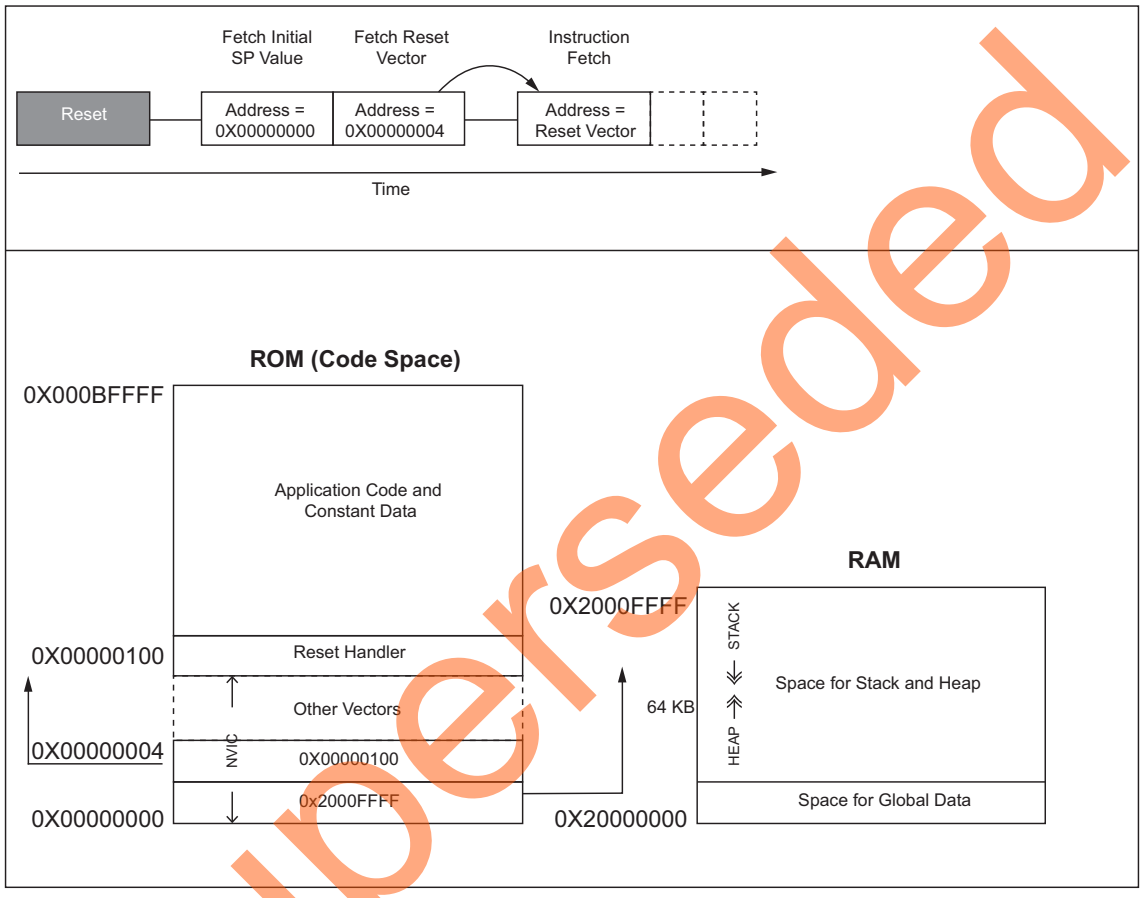

***Figure 1* • The Cortex-M3 Processor Execution Flow from the Reset**

SmartFusion2 SoC FPGA: The Cortex-M3 Processor Code Space Details

The address range from the 0x00000000 to 0x1FFFFFFF (0.5-GB space) is code space for the Cortex-M3 processor. Following are the SmartFusion2 SoC FPGA memory maps for the code/data space:

- On-chip eNVM (from 0x60000000 to 0x6007FFFF) of 256 KB for code and constant data regions
- On-chip eSRAM (from 0x20000000 to 0x2000FFFF) of 64 KB with SECDED for both code and data regions
- On-chip FPGA fabric RAM (FPGA Fabric FIC Region 0). This can be mapped via fabric interface controllers (FIC): FIC 0 or FIC 1. This region can be accessed by system bus for Instructions and data.
- External RAM memory interfaced through DDR or SDR interfaces (from 0xA0000000 to 0xDFFFFFFF) of 1 GB for both code and data regions

Any of the above memory regions with any offset from its base address, can be mapped to the Cortex-M3 processor code region space. On power-on, the eNVM region 0x60000000 is automatically remapped to the Cortex-M3 processor executable region start address (0x00000000). Hence, for every 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). Once the execution control goes to the default reset handler, the boot up sequence executes and execution control jumps to the user boot code.

The user boot code can be at the following locations based on the execution environment:

- **In Release mode:** It should be in read-only memory (ROM) region. The SmartFusion2 SoC FPGA after reset is initialized and remaps the eNVM address 0x60000000 to 0x00000000 of the Cortex-M3 processor address space.
- **In Debug mode:** It can either be in ROM or RAM. Choices/options are in the debugger command window to choose from where to debug (remap to 0x00000000) and in case of Debug mode, the SmartFusion2 SoC FPGA after reset is initialized through the flash bits and remaps the user boot code as follows:
  - eNVM address 0x60000000 to 0x00000000 of the Cortex-M3 processor address space, or
  - eSRAM address 0x20000000 to 0x00000000 of the Cortex-M3 processor address space

From the user boot code there can be multiple independent executable images in various parts of memories. The eNVM address locations can be remapped with any offset, eSRAM address locations with any offset, FPGA fabric RAM, or memory through DDR/SDRAM interface with any offset to the based address 0x00000000 of the Cortex-M3 processor code region.

Design Requirements

Table 1 lists the design requirements.

Table 1 • **Design Requirements**

Design Requirements	Description
Hardware Requirements	
SmartFusion2 Development Kit • FlashPro4 programmer • USB A to Mini-B cable • 12 V Adapter	Rev C or later
Host PC or Laptop	Any 64-bit Windows Operating System
Software Requirements	
Libero® System-on-Chip (SoC)	11.3

Table 1 • **Design Requirements (continued)**

SoftConsole	3.4
USB to UART drivers	-
One of the following serial terminal emulation programs: • HyperTerminal • TeraTerm • PuTTY	-

Design Description

The design examples in this application note use MMUART_0, GPIO, eSRAM, DDR and eNVM memory controllers. In the design examples, the System Builder Clock section is configured as shown in Figure 6 to run the M3_CLK at 111 MHz which drives the clock to Cortex-M3 processor.The independent executable images are created with required memory map. These executable images can be remapped to the starting address of the Cortex-M3 processor code space, or can be made executable for the Cortex-M3 processor.The implementation details are explained in hardware and software implementation sections.

Hardware Implementation

The hardware implementation involves configuring MSS, Fabric, clocks and oscillator using System Builder. Figure 2 shows the top-level SmartDesign of the application.

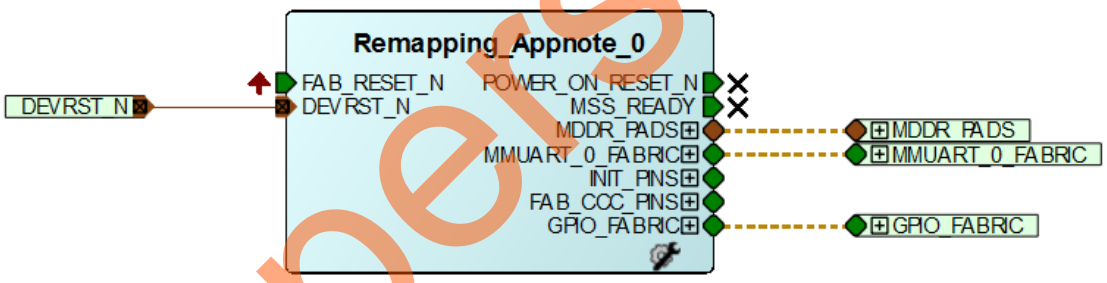

Figure 2 • **Top-Level SmartDesign**

The MDDR is configured for DDR3 at 333 MHz speed. Figure 3 and Figure 4 show the MSS MDDR configuration settings. Refer to "Appendix A – Design Files" on page 21 for DDR configuration file.

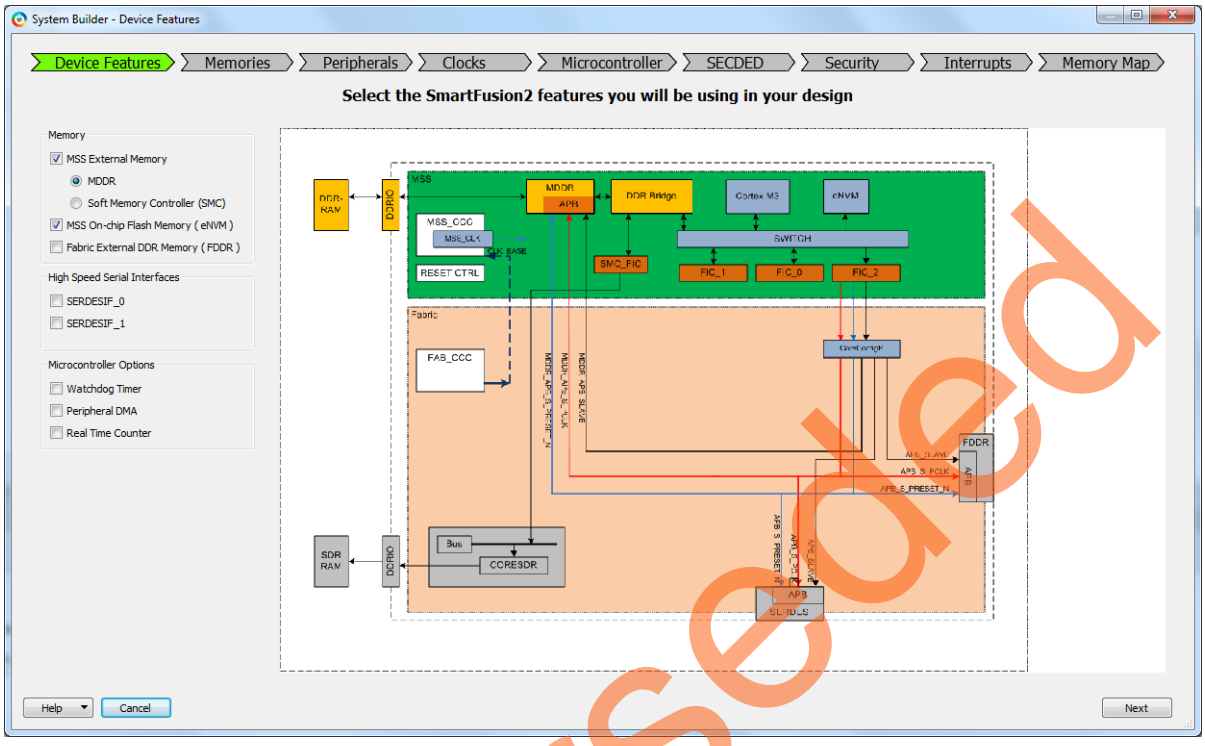

***Figure 3* • Select MDDR**

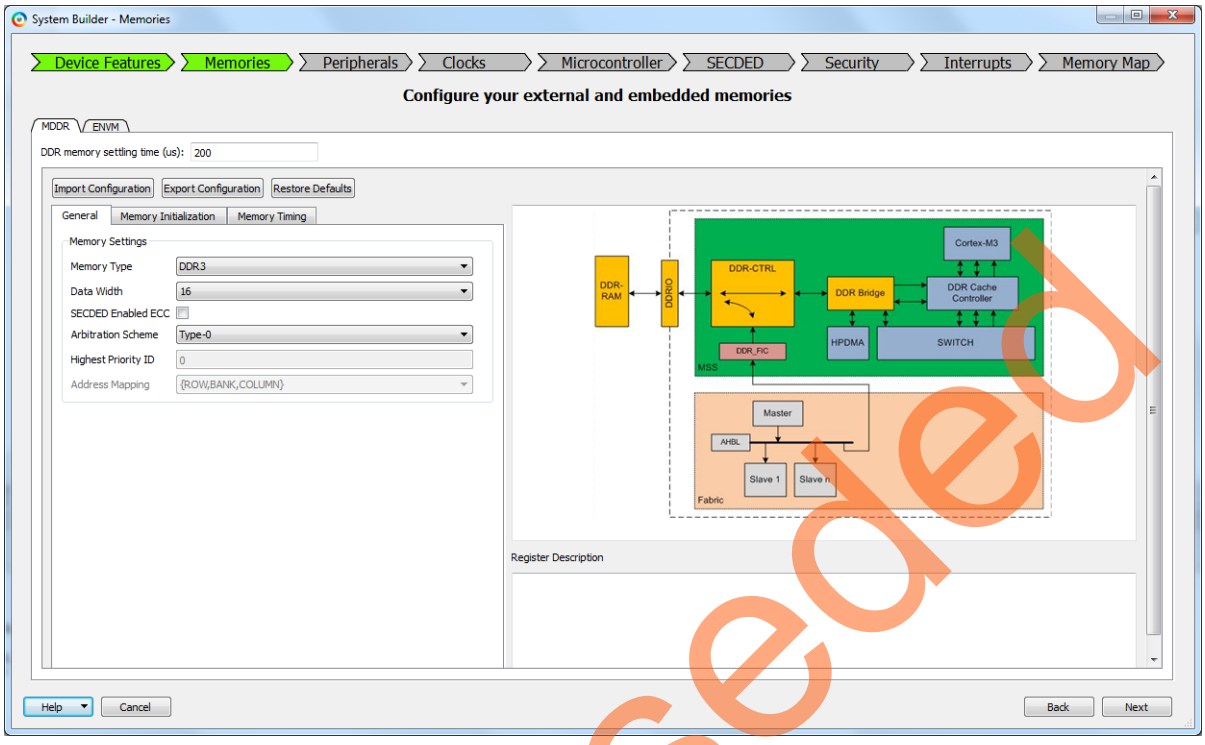

Figure 4 • **MDDR Configurator**

Add the eNVM user clients in ENVM configurator as shown in Figure 5.

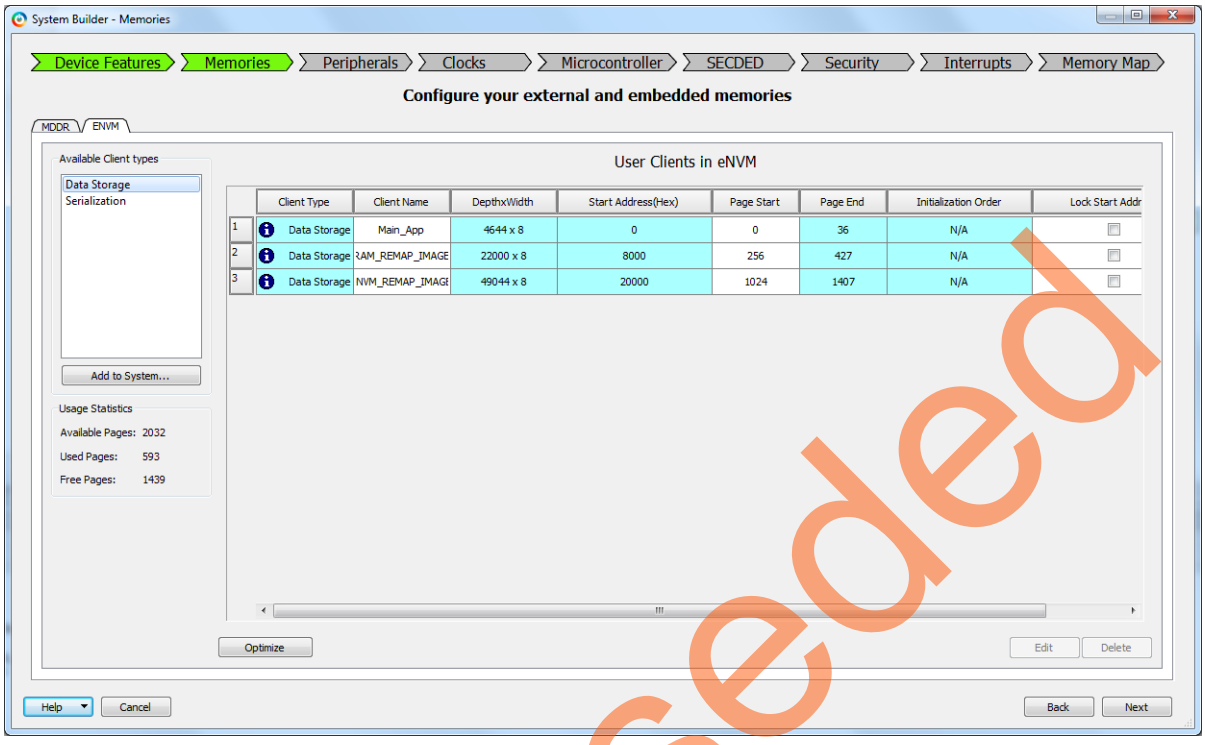

Figure 5 • **Memory Device Configuration**

The MMUART_0 is routed through FPGA fabric to communicate with the serial terminal program. The MSS_CCC clock is sourced from the FCCC via the CLK_BASE port. The FCCC is configured to provide the 100 MHz clock using GL0. Figure 6 shows the system clocks configurations for the M3_CLK, MDDR_CLK, and APB_0_CLK/APB_1_CLK.

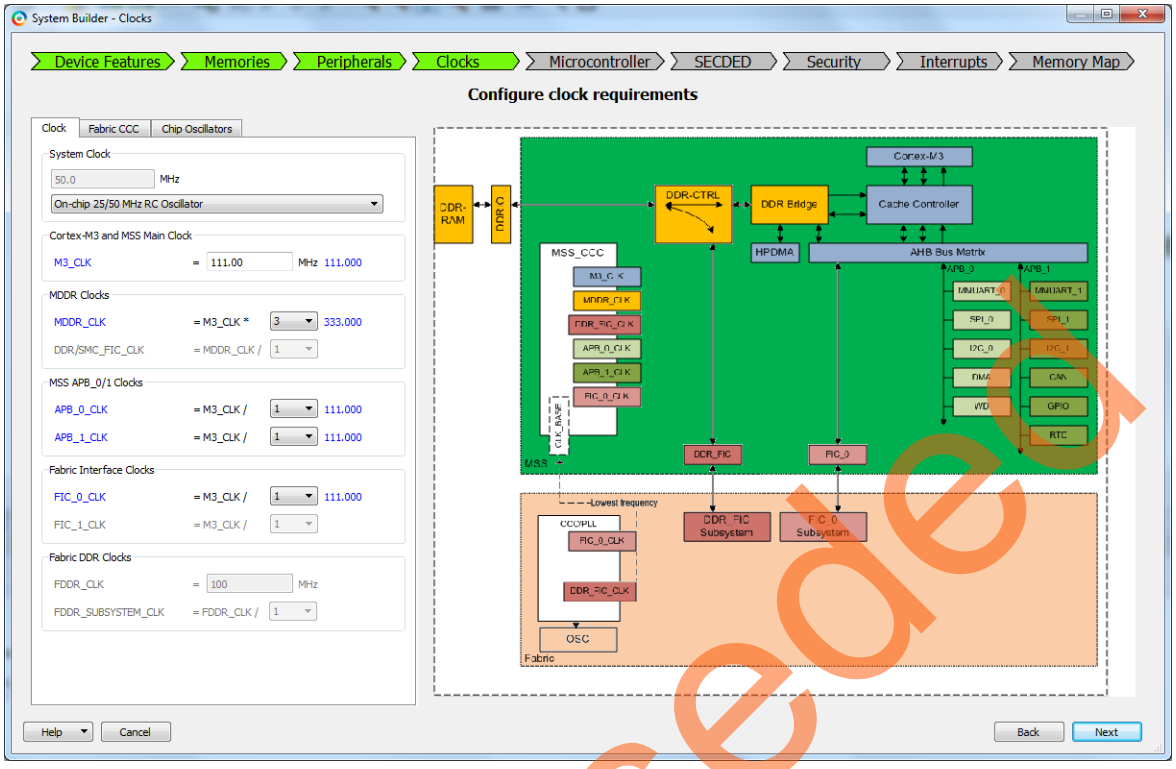

Figure 6 • **Clock Configurations**

Software Implementation

The following sections of the application note describe how to remap the various memory regions of the SmartFusion2 SoC FPGA to the Cortex-M3 processor code space. Figure 7 describes the memory map for the Cortex-M3 processor.

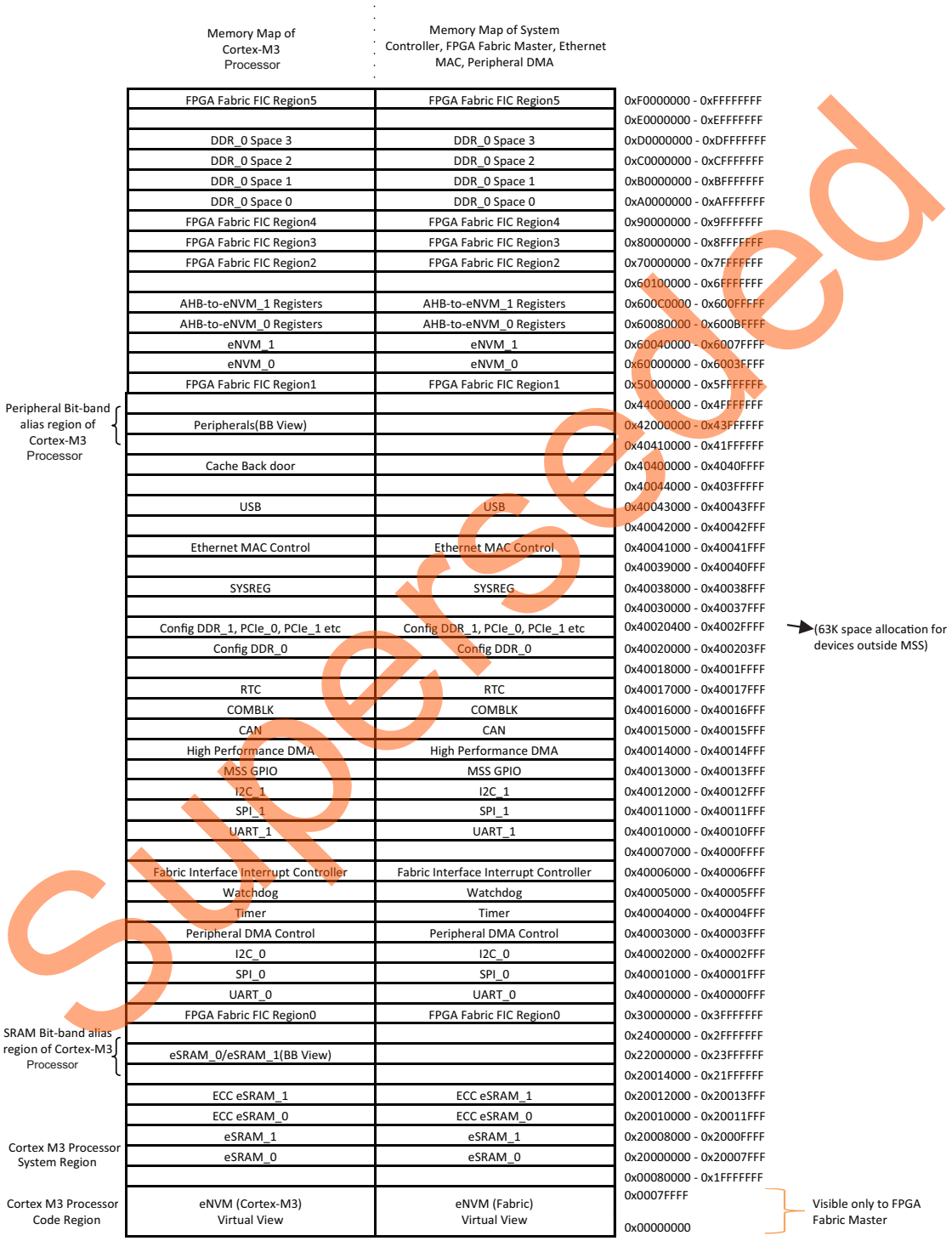

***Figure 7* • The Cortex-M3 Processor Memory Map in SmartFusion2 SoC FPGA**

Remapping eNVM Address Space to the Cortex-M3 Processor Code Space

Figure 8 shows an example scenario with multiple executable images in the eNVM regions.

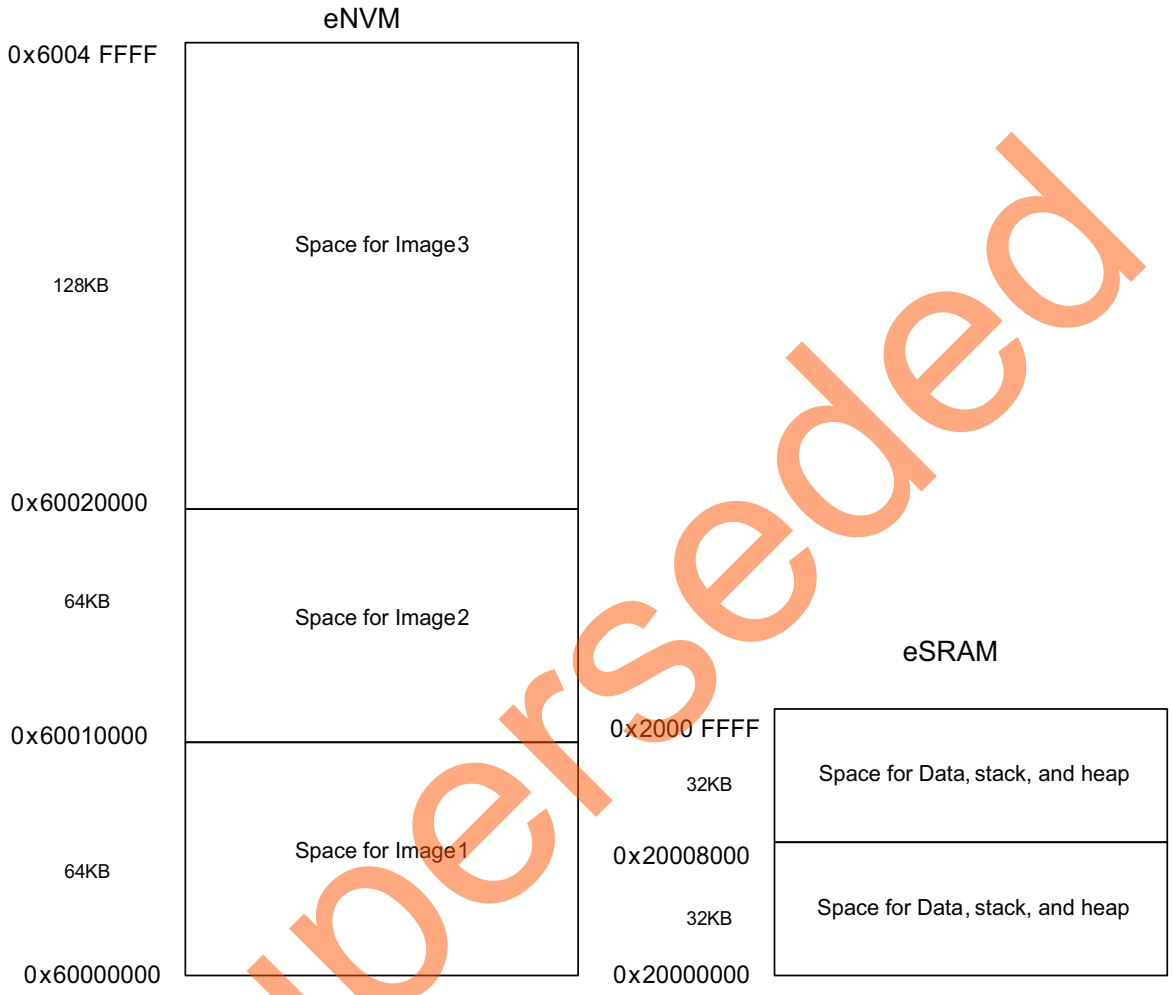

***Figure 8* • Example Scenario of Multiple Executable Images in eNVM**

In the example scenario (as shown in Figure 8), there are three images, which can be remapped to the starting address of the Cortex-M3 processor code space, or can be made executable for the Cortex-M3 processor. To create the independent executable images with the required memory map, it is required to create the linker scripts with the required memory map. The linker scripts are provided in the "Appendix A – Design Files" section.Once the executable images are created for the required memory map in Production mode, these images are added in the programming file using the eNVM clients in the Libero® System-on-Chip (SoC) hardware (HW) creation flow.

If the executable images are built with an absolute address, it is required to allow the execution control without using the remapping to the starting address of the code space (0x00000000). In such cases without remapping approach has to be used as explained below.

The execution control should be allowed to the desired image by using the following two approaches:

- **Without remapping:** By default, the eNVM base address 0x60000000 is remapped to the starting address of the code space of the Cortex-M3 processor. The vector table address of the desired image can be set by using the vector table offset register in the system registers, and pointing the stack pointer (SP) and program counter to the reset handler address of the desired image. This allows the Cortex-M3 processor to execute the new image. The eNVM offset address should be used in the linker script generation for the executable images in this approach. This approach is explained in the flow chart shown in Figure 9.

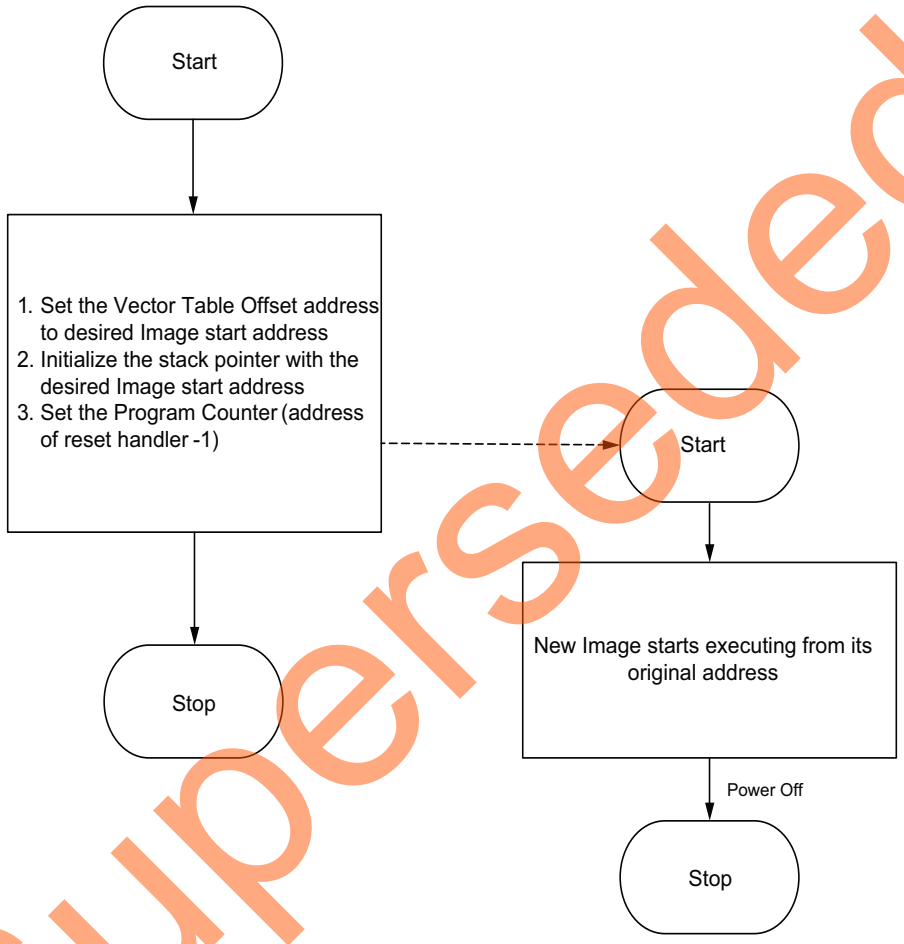

***Figure 9* • Logic for Moving the Execution Control to New Image in eNVM without Remapping**

For example, for the memory map of the different images explained in the Figure 8, the images are built with the base address as shown in Figure 8. If you need to run the Image 2 while Image 1 is running, use the following steps (explained in Figure 9):

1. Set the vector table offset address register is set to 0x60010000
2. Initialize the stack pointer with the content of 0x60010000
3. Change the program counter to the reset handler of Image 2 that is, PC=(0x60010004 -1)

With the all above 3 steps Image 2 starts executing from 0x60010000.

- **With remapping:** In this approach, the new image address can be remapped to the starting address of the code region of the Cortex-M3 processor by using the ENVM_CR, ENVM_REMAPSIZE, and ENVM_REMAP_BASR_CR registers. As the new image address is remapped to the bottom (0x0000_0000) of the Cortex-M3 processor code region, the linker scripts take care of building the images from the bottom (0x0000_0000) code region. The eNVM offset address should not be used in this approach. This approach is explained in the flow chart shown Figure 10.

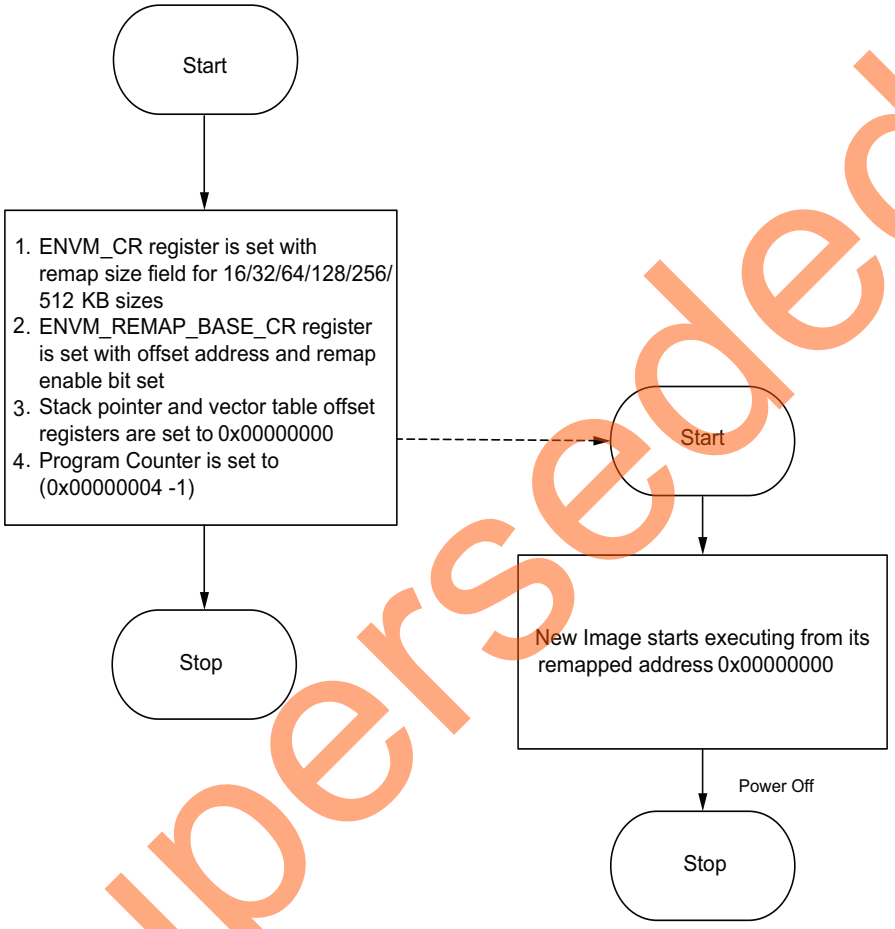

Figure 10 • **Logic for Moving the Execution Control to New Image in eNVM with Remapping**

For example, for the memory map of the different images explained in the Figure 8, the images are built with 0x00000000 as base address.If you need to run the Image 2 while Image 1 is running, use the following steps (explained in Figure 10):

1. Set the ENVM_CR register to 64KB as remap image size
2. Set the ENVM_REMAP_BASE_CR register with 0x00010000
3. Set the Stack Pointer to 0x00000000
4. Set the PC to 0x00000004 - 1

With all the above steps the new Image 2 starts executing from 0x00000000 which is mapped to 0x60010000.

Reference design is provided with this application note with remapping and without remapping. Refer to "Appendix A – Design Files" section for design files and follow the steps explained in "Running the Design" section for executing the reference design.

Table 2 describes the registers which are required to be set for the eNVM remapping to the bottom (0x0000_0000) of the Cortex-M3 processor. The SYSREG block is located at address 0x40038000 in the Cortex-M3 processor address space.

Table 2 • **eNVM Remap Registers**

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
ENVM_CR	0XC	RW-P	Register	sysreset_n	eNVM Configuration register
ENVM_REMAP_BASE_CR	0x10	RW-P	Register	sysreset_n	eNVM remap configuration register for the Cortex-M3 processor.

Remapping eNVM to Soft Core Processor Memory Map

Soft core processor implemented in SmartFusion2 SoC FPGA fabric can access the eNVM for the code execution purposes. For this use case the fabric interface controller (FIC_0 or FIC_1) and the eNVM AHB controller need to be set properly. The eNVM partitioning between the Cortex M3 and SoftCore processor needs to be taken care in such a way that these two partitions are mutually exclusive. The remapping of the eNVM offset address to the soft core processor bottom (0x0000_0000) address map is very similar to the remapping of the eNVM address to the Cortex-M3 processor. ENVM_REMAP_FAB_CR register has to be used instead of ENVM_REMAP_BASE_CR register. The SYSREG block is located at address 0x40038000 in the Cortex-M3 processor address space.

Table 3 describes the eNVM remap register to fabric SoftCore processor address space.

Table 3 • **eNVM Remap Register to Fabric SoftCore Processor Address Space**

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
ENVM_REMAP_FAB_CR	0X14	RW-P	Register	sysreset_n	NVM remap configuration register for the soft processor in the FPGA

Remapping eSRAM to the Cortex-M3 Processor Code Space

Figure 11 shows the example scenario of the executable images in eSRAM regions.

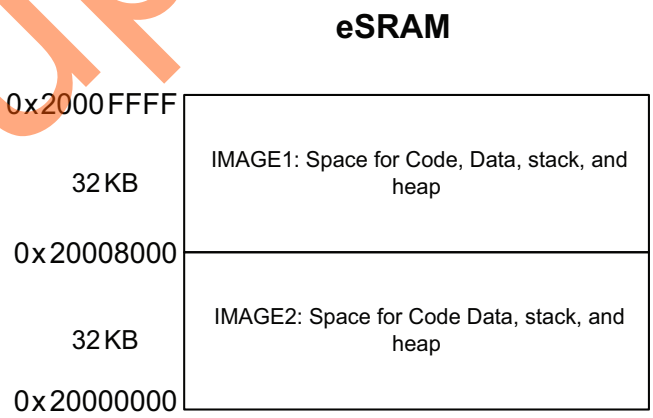

Figure 11 • **Example Scenario of Multiple Executable Images in eSRAM**

The above scenario, shown in Figure 11, describes two images which can be remapped to the bottom (0x0000_0000) of the Cortex-M3 processor or can be made executable for the Cortex-M3 processor. To create the independent executable images with required memory map, the linker scripts need to be created with required memory map. The linker scripts are provided in the "Appendix A – Design Files" section. Once the images are created for the required memory map in Production mode, these images are to be copied to an external memory like SPI Flash/eNVM, and are code shadowed by the bootloader to the eSRAM whenever it is required to execute the new images.

Once the images are copied to eSRAM by bootloader, the execution control can be allowed to the desired image by using any of the following two approaches:

If the executable images are built with an absolute address, the execution control needs to be allowed without using the remapping to starting address of the code space (0x00000000). In such cases, without remapping approach explained below (Point 1) has to be used.

If the executable images are built with the address 0x00000000, the execution control needs to be allowed by using remapping to starting address of the code space (0x00000000). In such cases, remapping approach explained below (Point 2) has to be used.

1. **Without remapping:** Using the vector table offset register in the system registers, the vector table address of the desired image can be set for execution, and point the stack pointer (SP) and the program counter to the reset handler of the desired image. This allows the Cortex-M3 processor to execute the new image. The eSRAM address should be used in the linker script generation for the executable images in this approach. This approach is explained in the flow chart shown in Figure 12.

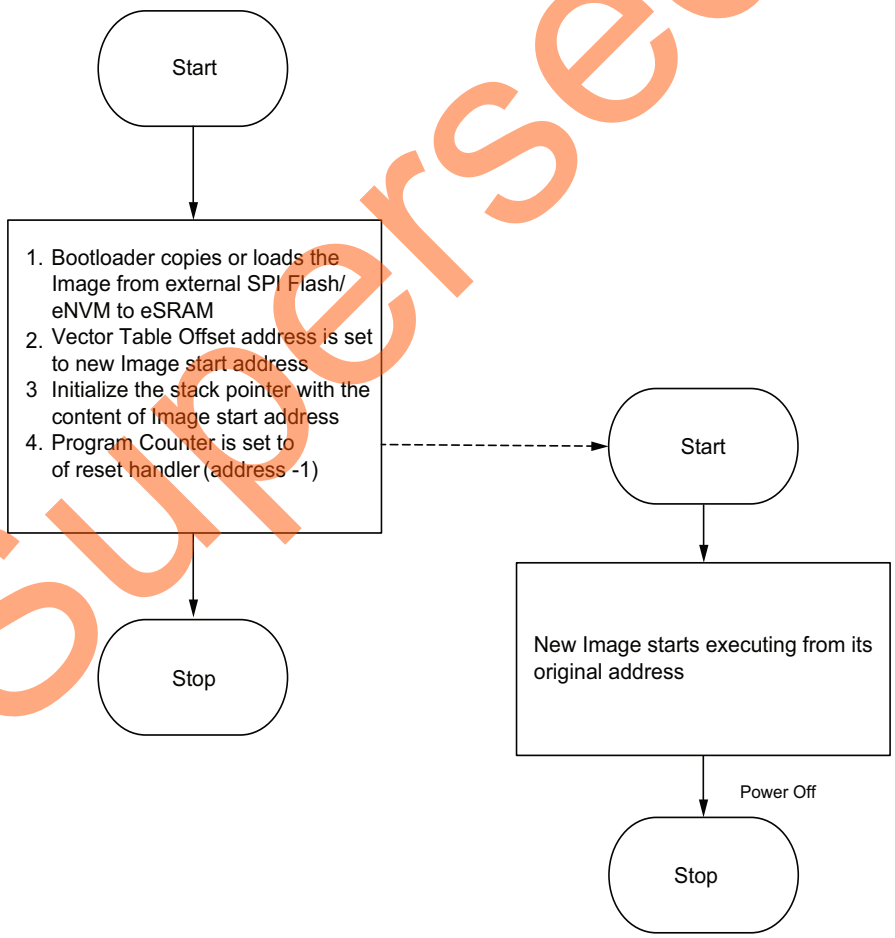

***Figure 12* • Logic for Moving the Execution Control to New Image in eSRAM without Remapping**

- **With Remapping:** In this approach, the new image address can be remapped to the bottom (0x0000_0000) of the Cortex-M3 processor by using the ESRAM_CR registers. As the new image address is remapped to bottom (0x0000_0000) of the Cortex-M3 processor code region the linker scripts take care of building the images from the bottom (0x0000_0000) code region. The eSRAM address should not be used instead offset address from zero has to be used in the linker scripts for this approach. This approach is explained in the flow chart shown in Figure 13.

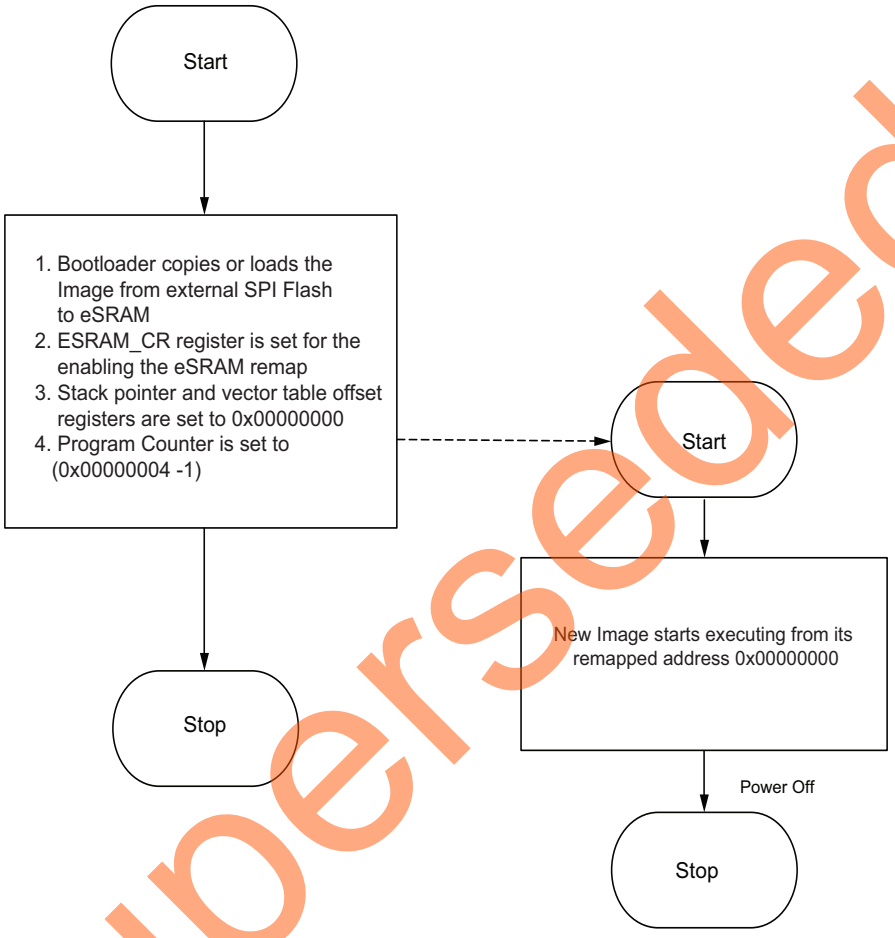

***Figure 13* • Logic for Moving the Execution Control to New Image in eSRAM with Remapping**

For example, for the memory map of the different images explained in the Figure 11, the images are built with 0x00000000 as base address. If it is required to jump from the Image 2 to Image 1 then use the following steps (as explained in the Figure 13).

1. Copy the image1 from Flash to eSRAM starting address 0x20008000
2. Set the ESRAM_CR register to enable the eSRAM remapping to 0x00000000
3. Set the Stack Pointer to 0x00008000 and Vector Table offset register to 0x00008000
4. Set the PC to 0x00008004 -1

With all the above steps the new Image1 starts executing from 0x00008000 which is the mapped to address 0x20008000.

Reference design is provided with this application note with remapping and without remapping. Refer to the "Appendix A – Design Files" section for the design files and follow the steps explained in "Running the Design" section for executing the reference design.

Table 4 explains the registers that are required to be set for the eSRAM remapping. The SYSREG block is located at address 0x40038000 in the Cortex-M3 processor address space.

Table 4 • **Registers Required to eSRAM Remapping**

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
ESRAM_CR	0x0	RW-P	Register	sysreset_n	Controls address mapping of the eSRAMs

Remapping External RAM (DDR/SDR SDRAM Interface) to the Cortex-M3 Processor Code Space

Figure 14 shows the scenario of the multiple executable images in DDR/SDRAM interface memory regions.

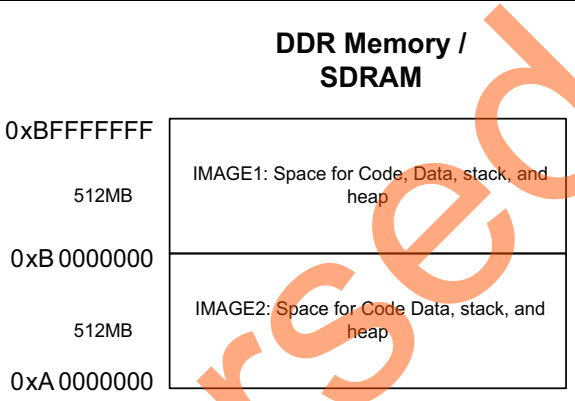

Figure 14 • **Example Scenario of Multiple Executable Images in DDR/SDR SDRAM**

In this scenario there are two images which can be remapped to the bottom (0x0000_0000) of the Cortex-M3 processor, or can be made executable for the Cortex-M3 processor.

To create the independent executable images with required memory map, the linker scripts need to be created with the required memory map. The links scripts are provided in "Appendix A – Design Files" section. Once the images are created for the required memory map in Production mode, these images are to be copied to an external memory like SPI Flash, and code shadowed by the bootloader to DDR memory or SDRAM whenever the execution of the new images is required.

Once the image is copied to the DDR memory and SDRAM by the bootloader, the execution control can be allowed to the desired image by using the following approach.

The new image address can be remapped by using the DDR_CR register to the bottom (0x0000_0000) of the Cortex-M3 processor code region. As the new image start address is re-mapped to the Cortex-M3 processor code region 0x0000_0000, the linker scripts take care of building the images from the code region 0x0000_0000. The DDR memory or SDRAM addresses (0xA000_0000) should not be used. Instead, the offset address from zero has to be used in the linker scripts for this approach.

As the DDR memory or SDRAM memory address range cannot be used in the Vector table offset register, so it is required to remap these memories to start address of the Cortex-M3 processor code space for the execution from these memories. This approach is explained in the flow chart shown in Figure 15.

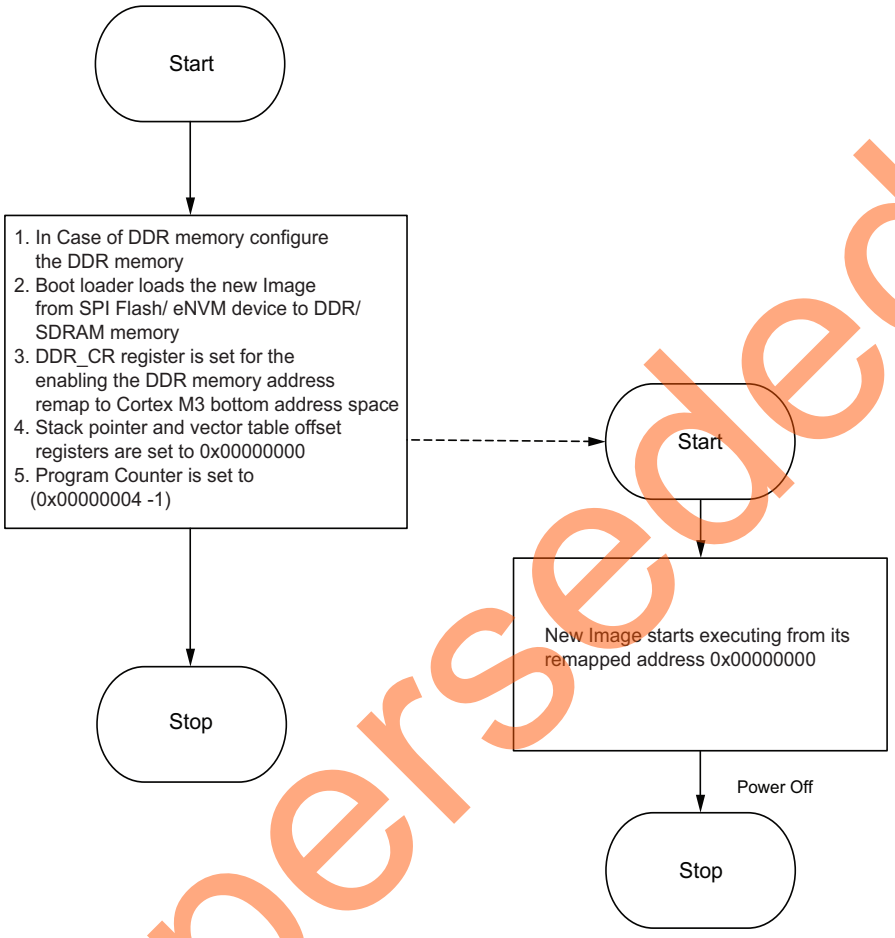

Figure 15 • **Logic for Moving the Execution Control to New Image in DDR/SDR SDRAM with Remapping**

Reference design is provided with this application note with remapping. Refer to the "Appendix A – Design Files" section for the design files and follow the steps explained in "Running the Design" section for executing the reference design.

Figure 5 shows the registers required to be set for the DDR/SDR SDRAM remapping. The SYSREG block is located at address 0x40038000 in the Cortex-M3 processor address space.

Table 5 • **Registers Required to DDR/SDR SDRAM Remapping**

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
DDR_CR	0x8	RW-P	Register	sysreset_n	DDR control Register. Configures DDR Space.

Firmware Drivers

The following firmware drivers are used in this application.

- MSS MMUART driver
  - To communicate with serial terminal program on the Host PC
- MSS GPIO driver
  - To drive onboard LED's.

Running the Design

This application note provides the design files for all the scenarios and describes the hardware and software requirements, board settings and steps to run the design:

Board Settings

Connect the following jumpers on the SmartFusion2 SoC FPGA Development Kit, as described in Table 6. While making the jumper connections, the power supply switch SW7 on the board should be in OFF position.

Table 6 • **SmartFusion2 SoC FPGA Development Kit Jumper Settings**

Jumper	Pin (From)	Pin (To)
J70, J93, J94, J117, J123, J142, J157, J160, J167, J225, J226, J227	1 (default)	2
J2	1 (default)	3
J23	2 (default)	3
J129, J133	2	3

Steps to Run the Design

The following steps describe how to run the design:

1. Connect the FlashPro4 programmer to the J59 connector of the SmartFusion2 SoC FPGA Development Kit.
2. Connect one end of the USB mini-B cable to the J24 connector provided on the SmartFusion2 SoC FPGA Development Kit. Connect the other end of the USB cable to the host PC. Make sure that the USB to UART bridge drivers are automatically detected (can be verified in the Device Manager), as shown in Figure 16.

Note: Copy the COM port number for serial port configuration. Ensure that the COM port location is specified as "on USB Serial Converter D", as shown in Figure 16.

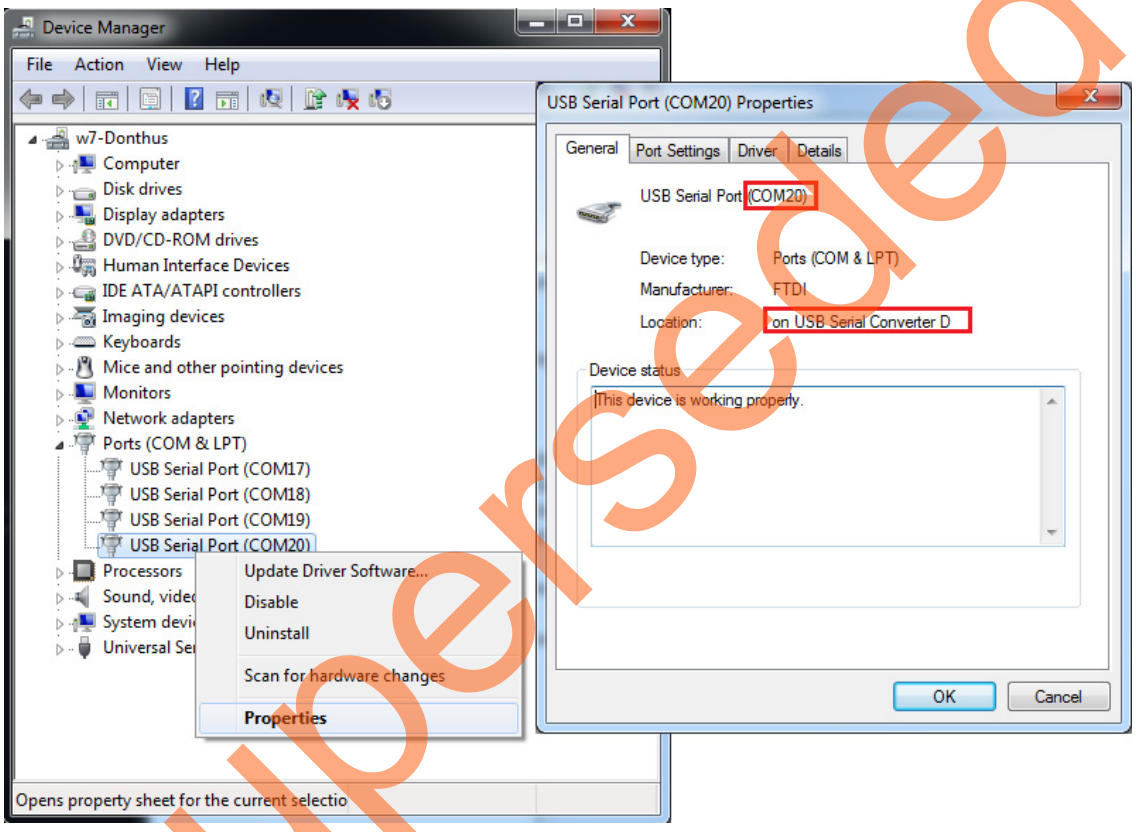

Figure 16 • **USB to UART Bridge Drivers**

3. If USB to UART bridge drivers are not installed, download and install the drivers from *www.microsemi.com/soc/documents/CDM_2.08.24_WHQL_Certified.zip*.
4. Connect the power supply to the J18 connector and change the power supply switch SW7 to ON. Start HyperTerminal program with a baud rate of 57600, 8 data bits, 1 stop bit, no parity, and no flow control. If your computer does not have HyperTerminal program, use any free serial terminal emulation program such as PuTTY or Tera Term. Refer to the *Configuring Serial Terminal Emulation Programs* tutorial for configuring the HyperTerminal, Tera Term, and PuTTY.
5. Program the SmarFusion2 device with the programming file provided in the design files (\Programming Files\Remapping_Appnote.stp) using FlashPro design software.
6. Press **SW9** switch to reset the board after successful programming.

7. Figure 17 shows the serial Terminal:

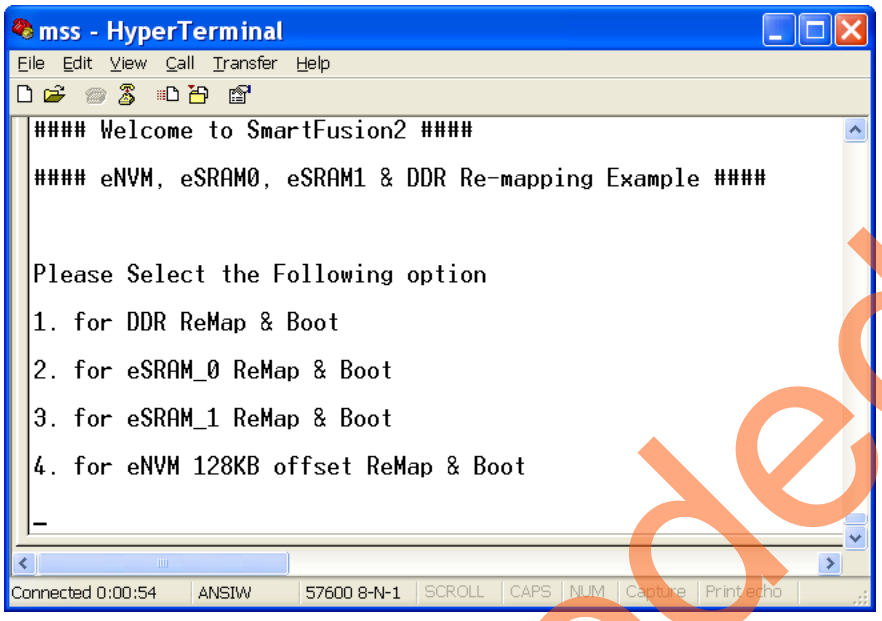

Figure 17 • **Main Menu of the Re-Mapping Application Note**

8. Based on the selection made, the pre-built image stored in eNVM is copied to the appropriate locations (DDR, eSRAM0, or eSRAM1) and re-mapping is applied.
9. Once the re-mapping is completed, the new Image starts booting and the following messages are shown on the serial terminal and LED starts blinking on the SF2 Development Kit.

Note: Reset the SmartFusion2 Development Kit board to switch among the application images.

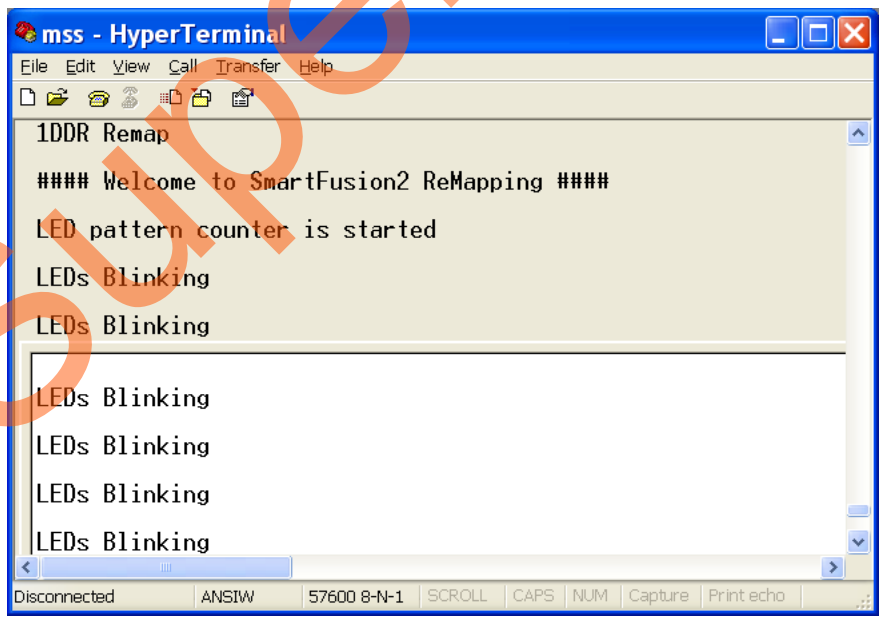

Figure 18 • **Re-mapped Image is Running**

For booting multiple images without remapping, refer to the *SmartFusion SoC: Basic Bootloader and Field Upgrade eNVM Through IAP Interface* application note.

Conclusion

This application note explains the remapping of the eNVM, eSRAM, and DDR/SDR SDRAM memories to the Cortex-M3 processor code region. It also explains how to execute the program code which is built with absolute addresses without remapping in case of eNVM and eSRAM.

List of Changes

The following table lists critical changes that were made in the current version of the chapter.

Date	Changes	Page
Revision 5 (May 2014)	Figure 2 is changed (SAR 57912).	4
	Added Figure 3 (SAR 57912).	5
	Added Figure 4 (SAR 57912).	6
	Added Figure 5 (SAR 57912).	7
	Updated the document for Libero SoC v11.3 software release (SAR 57912).	NA
Revision 4 (January 2014)	Figure 6 is changed.	8
	Figure 3 is changed.	5
	Table 6 is updated	18
Revision 3 (May 2013)	Updated the document for Libero SoC v11.0 software release (SAR 47617).	NA
Revision 2 (March 2013)	Updated the document for Libero SoC v11.0 beta SP1 software release (SAR 45398)	NA
Revision 1 (November 2012)	Updated "Remapping eNVM Address Space to the Cortex-M3 Processor Code Space" section. (SAR 42911).	10
	Updated "Remapping eSRAM to the Cortex-M3 Processor Code Space" section (SAR 42911).	13
	Updated "Remapping External RAM (DDR/SDR SDRAM Interface) to the Cortex-M3 Processor Code Space" section (SAR 42911).	16
	Updated "Running the Design" section (SAR 42911).	18
	Updated "Appendix A – Design Files" section (SAR 42911).	21

Appendix A – Design Files

The design files (DF), programming files (PF), and linker scripts (LD) can be downloaded from the Microsemi® SoC Products Group website:

www.microsemi.com/soc/download/rsc/?f=M2S_AC390_DF

www.microsemi.com/soc/download/rsc/?f=M2S_AC390_PF

www.microsemi.com/soc/download/rsc/?f=M2S_AC390_LD

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

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

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**.

51900258-5/05.14