AC390
Application Note
SmartFusion2 SoC FPGA – Remapping eNVM, eSRAM, and DDR/SDR SDRAM Memories

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 10.0

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

- Updated the document for Libero SoC v2021.1.
- Removed the references to Libero version numbers.

1.2 Revision 9.0

Updated the document for Libero SoC v11.7 software release (SAR 76553).

1.3 Revision 8.0

Updated the document for Libero SoC v11.6 software release (SAR 71546).

1.4 Revision 7.0

Updated the document for Libero SoC v11.5 software release (SAR 64108).

1.5 Revision 6.0

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

- Updated the document for Libero SoC v11.4 software release (SAR 60315).
- Updated the document for Advanced Development Kit Board details (SAR 60315).

1.6 Revision 5.0

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

- Figure 2 is changed (SAR 57912).
- Added Figure 3 (SAR 57912).
- Added Figure 4 (SAR 57912).
- Added Figure 5 (SAR 57912).
- Updated the document for Libero SoC v11.3 software release (SAR 57912).

1.7 Revision 4.0

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

- Figure 6 is changed.
- Figure 3 is changed.

1.8 Revision 3.0

Updated the document for Libero SoC v11.0 software release (SAR 47617).

1.9 Revision 2.0

Updated the document for Libero SoC v11.0 beta SP1 software release (SAR 45398).

1.10 Revision 1.0

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

- Updated Remapping eNVM Address Space to Cortex-M3 Processor Code Space, page 12 (SAR 42911).
- Updated Remapping eSRAM to Cortex-M3 Processor Code Space, page 15 (SAR 42911).
- Updated Remapping External RAM (DDR/SDR SDRAM Interface) to Cortex-M3 Processor Code Space, page 18 section (SAR 42911).
- Updated Setting Up the Demo Design, page 20 (SAR 42911).
- Updated Appendix 2: Design Files, page 29 (SAR 42911).

2 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 Non-Volatile Memory (eNVM)
- embedded Static Random Access Memory (eSRAM)
- Double Data Rate (DDR)/Single Data Rate (SDR) Synchronous Dynamic Random Access Memory (SDRAM)

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

3.1 Introduction

The SmartFusion®2 System-on-Chip (SoC) Field Programmable Gate Array (FPGA) devices integrate 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. It also explains how to execute the program code built with absolute addresses without remapping.

3.1.1 SmartFusion2 Booting and Address Space Overview

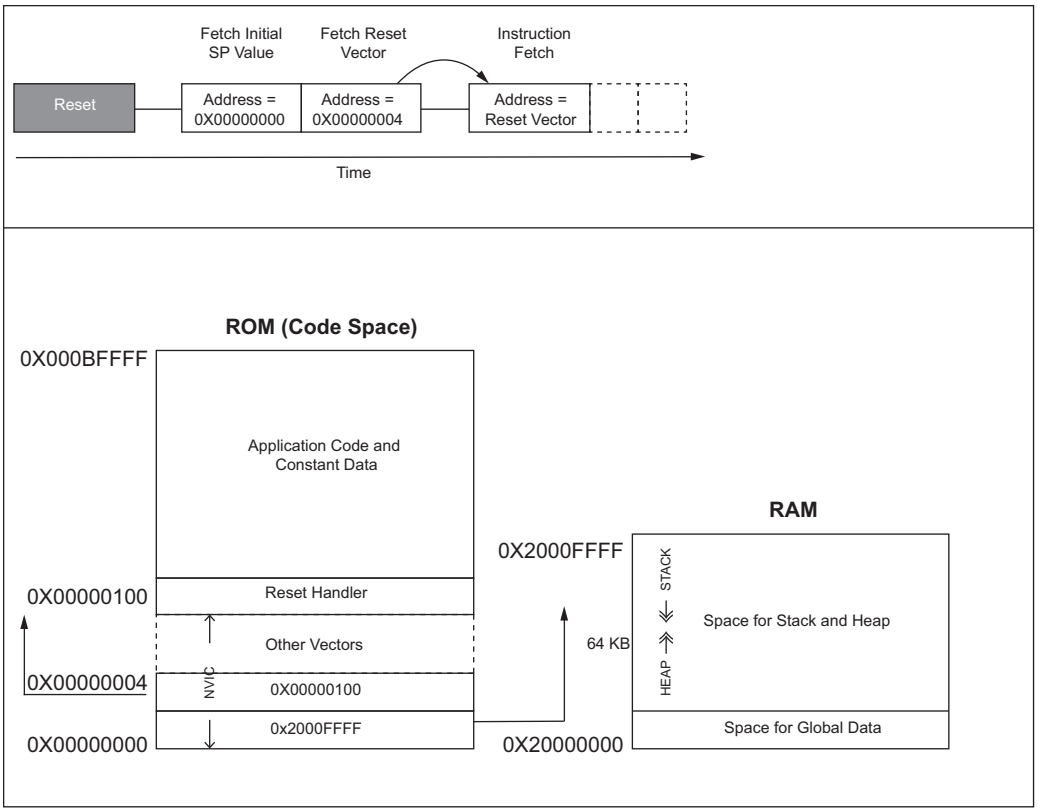
This application note describes the SmartFusion2 SoC FPGA boot sequence, and how to remap the various memory regions to the Cortex-M3 processor code region and an optional softcore processor located in the FPGA fabric.

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 must be the initial stack pointer and the second entry must 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. After reset, the Cortex-M3 processor reads two words from memory:

- At the address location 0×00000000 for the initial stack pointer
- At the address location 0×00000004 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. Figure 1 shows the execution flow.

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



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

The address range from 0×00000000 to 0×1FFFFFFF (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 0×60000000 to 0×6007FFFF) of 512 KB for code and constant data regions
- On-chip eSRAM (from 0×20000000 to 0×2000FFFF) 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 the system bus for instructions and data
- External RAM interfaced through DDR or SDR interface (from 0×A0000000 to 0×DFFFFFFF) of 1 GB for both code and data regions

Any of the preceding 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 0×60000000 is automatically remapped to the Cortex-M3 processor executable region start address (0×00000000). Therefore, for every power-on reset, the Cortex-M3 processor fetches the initial stack pointer from 0×00000000 (eNVM address 0×60000000) and the address of the reset handler from 0×00000004 (eNVM address 0×60000004). 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 must be in the Read-Only Memory (ROM) region. The SmartFusion2 SoC FPGA after reset is initialized and remaps the eNVM address 0×60000000 to 0×00000000 of the Cortex-M3 processor address space.

- **In Debug mode:** It can be either in ROM or RAM. Options are in the debugger command window to choose from where to debug (remap to 0×00000000) 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 0×60000000 to 0×00000000 of the Cortex-M3 processor address space, or
  - eSRAM address 0×20000000 to 0×00000000 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 base address 0×00000000 of the Cortex-M3 processor code region.

3.2 References

The following are the reference documents:

- *UG0446: SmartFusion2 and IGLOO2 FPGA High Speed DDR Interfaces User Guide*
- *SmartFusion2 MSS ARM Cortex-M3 Configuration Guide*
- *AC372: SmartFusion cSoC- Basic Bootloader and Field Upgrade eNVM Through IAP Interface App Note*
- *UG0451: IGLOO2 and SmartFusion2 Programming User Guide*
- *UG0450: SmartFusion2 SoC and IGLOO2 FPGA System Controller User Guide*
- *UG0331: SmartFusion2 Microcontroller Subsystem User Guide*
- *Configuring Serial Terminal Emulation Programs*

3.3 Design Requirements

Table 1 lists the hardware and software design requirements for running this demo design.

Table 1 • **Design Requirements**

Requirement	**Version**
Operating System	64 bit Windows 7 and 10
Hardware	
SmartFusion2 Advanced Development Kit: • FlashPro5 programmer • USB A to Mini-B cable • 12 V Adapter	Rev A or later
Host PC or Laptop	
Software	
FlashPro Express	Refer to the `readme.txt` file provided in the design files for the software versions used with this reference design.
Libero® System-on-Chip (SoC)	
SoftConsole	
USB to UART drivers	–

Note: Libero SmartDesign and configuration screen shots shown in this guide are for illustration purpose only. Open the Libero design to see the latest updates.

3.4 Prerequisites

Before you begin:

1. Download and install Libero SoC (as indicated in the website for this design) on the host PC from the following location:

   https://www.microsemi.com/product-directory/design-resources/1750-libero-soc

2. For demo design files download link:

   http://soc.microsemi.com/download/rsc/?f=m2s_ac390_df

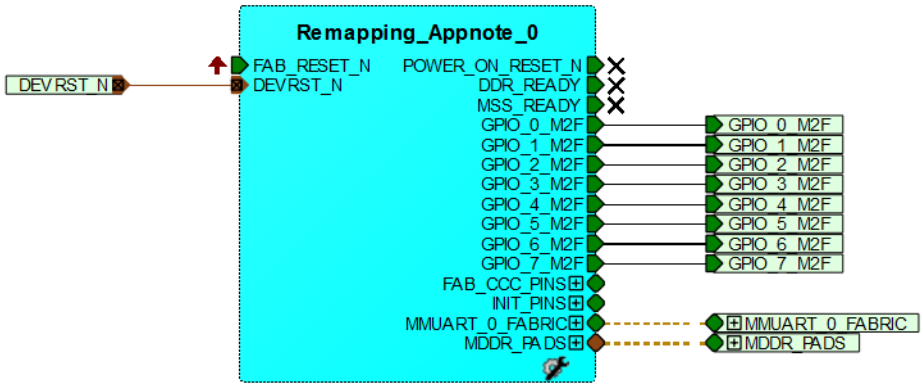
3.5 Design Description

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, page 10 to run the M3_CLK at 111 MHz, which drives the clock to the Cortex-M3 processor. The independent executable images are created with the 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 the hardware and software implementation sections.

3.6 Hardware Implementation

The hardware implementation involves configuring the Microcontroller Subsystem (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, page 8 and Figure 4, page 8 show the MSS MDDR configuration settings. For DDR configuration file, refer to Prerequisites, page 7 (design files).

Figure 3 • **Select MDDR**

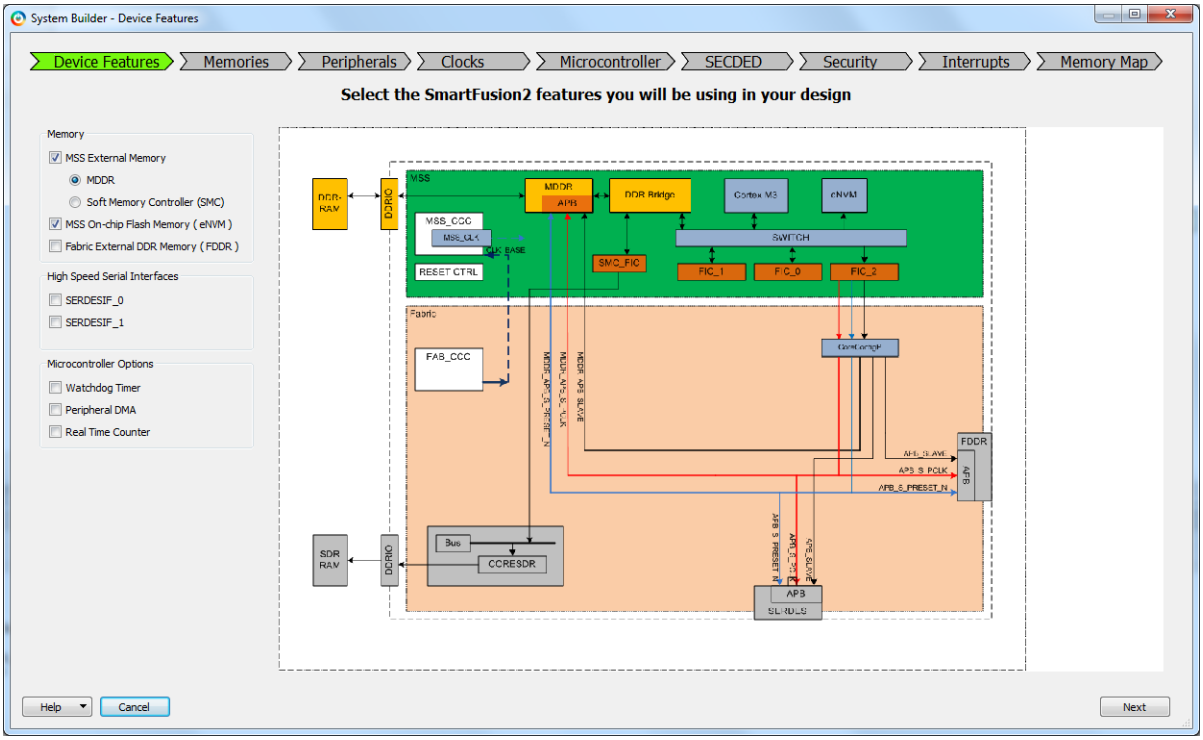
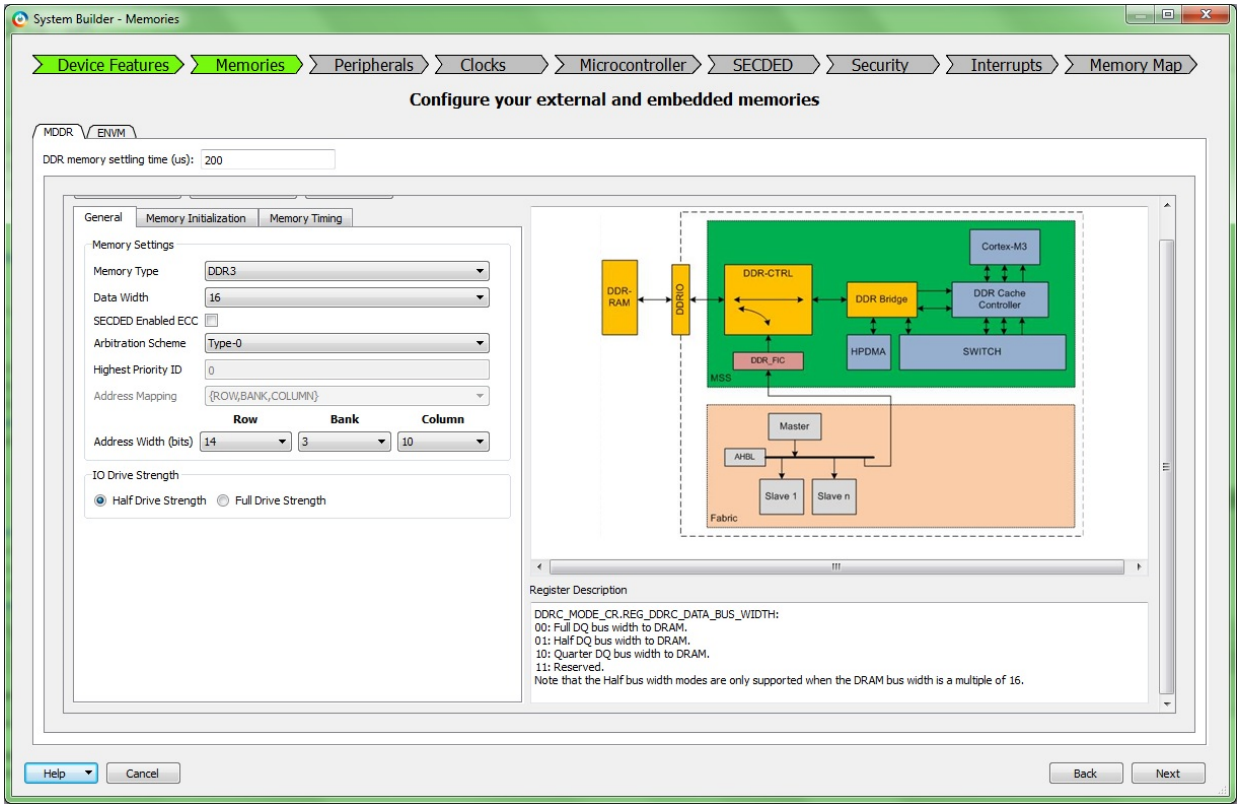
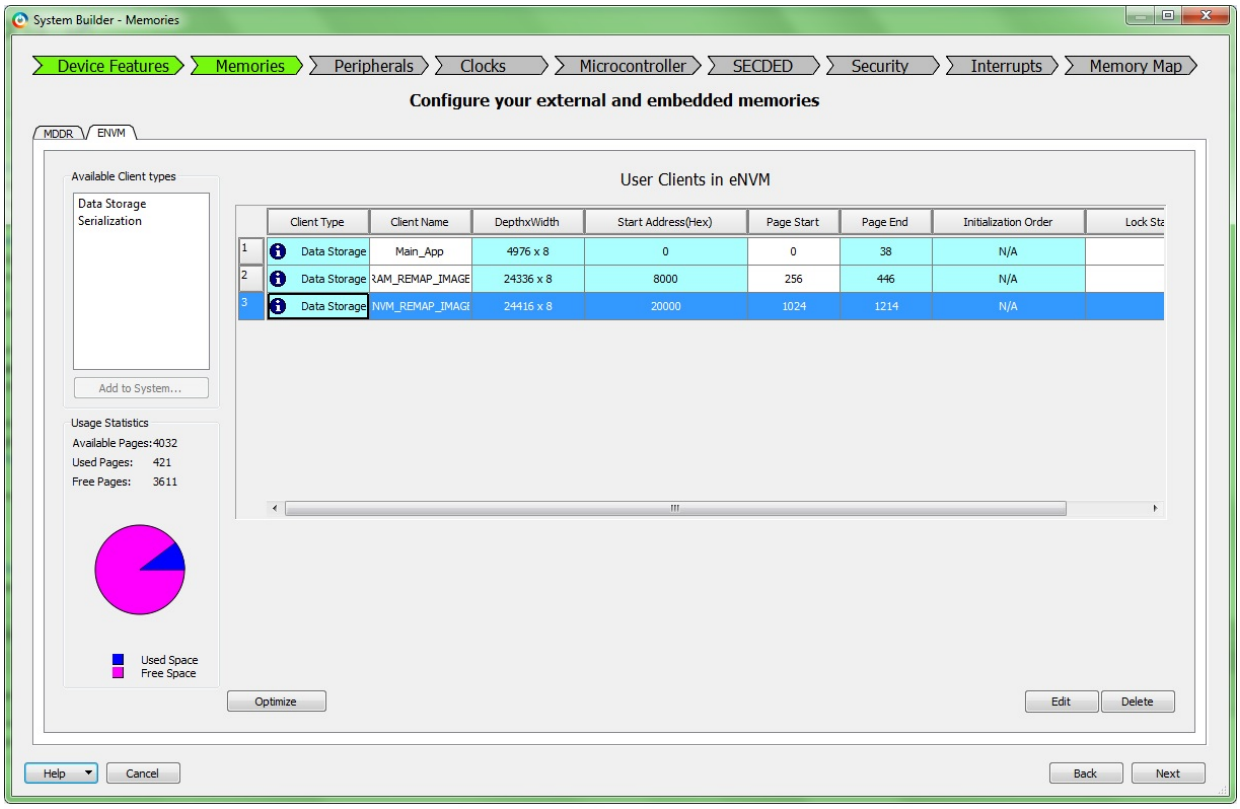


Figure 4 • **MDDR Configurator**



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

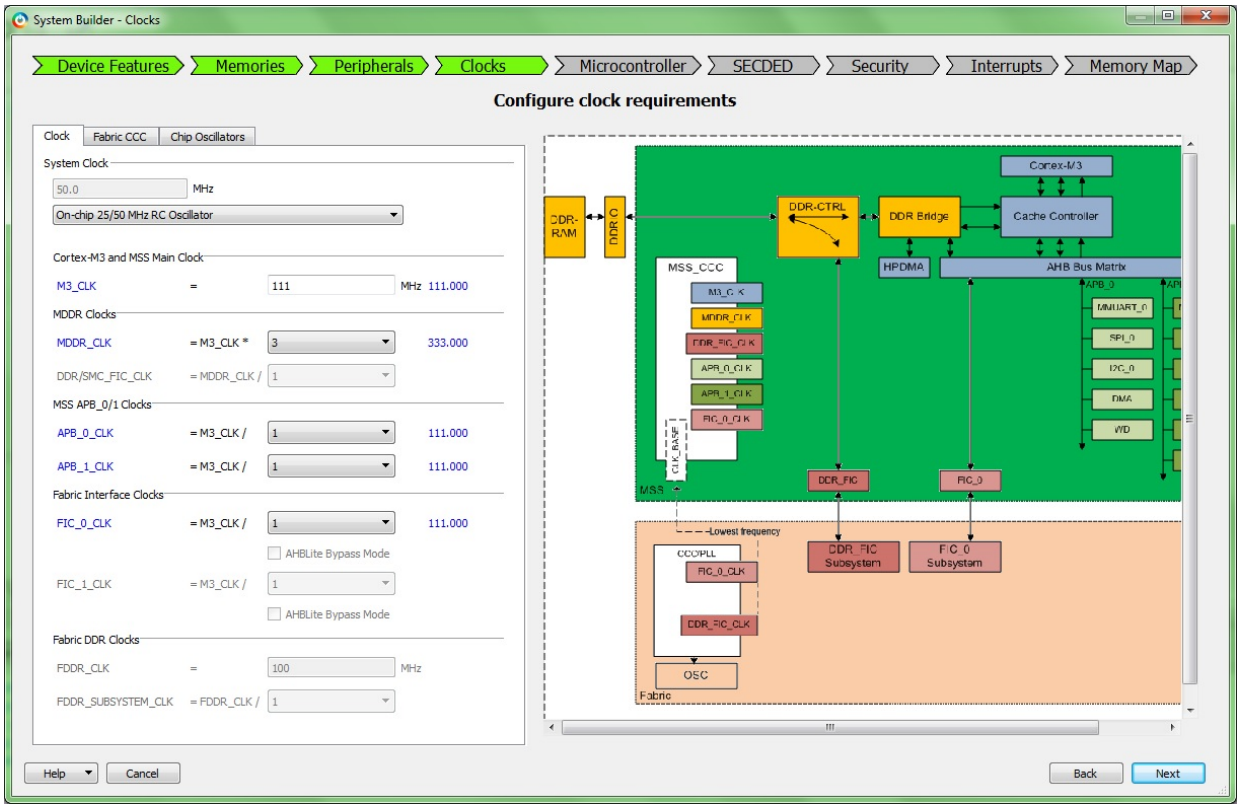
Figure 5 • **Memory Device Configuration**



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 Configuration**



3.7 Software Implementation

The following sections 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.

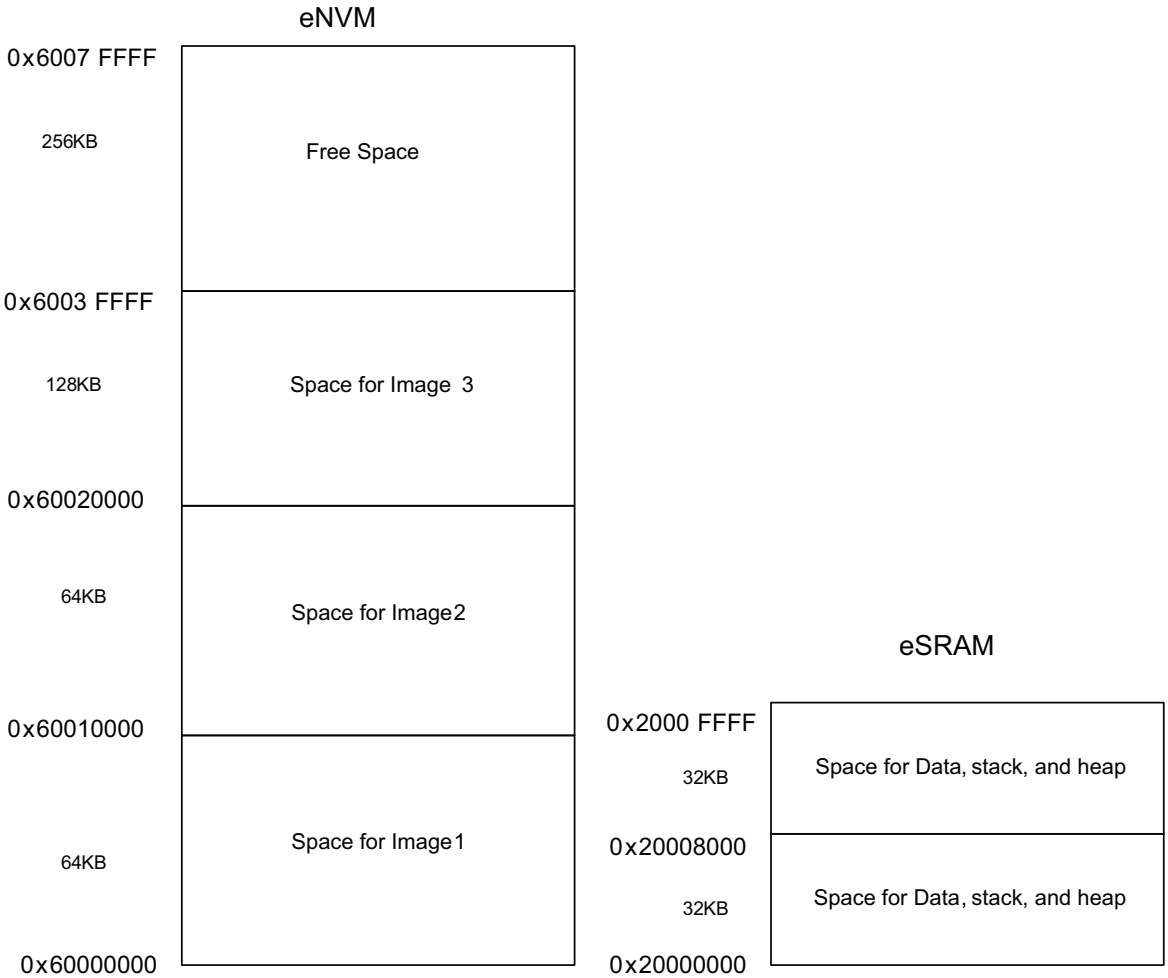
Figure 7 • **Cortex-M3 Processor Memory Map in SmartFusion2**

	Memory Map of Cortex-M3 Processor	Memory Map of System Controller, FPGA Fabric Master, Ethernet MAC, Peripheral DMA		
	FPGA Fabric FIC Region5	FPGA Fabric FIC Region5	0xF0000000 - 0xFFFFFFFF	
			0xE0000000 - 0xEFFFFFFF	
	DDR_0 Space 3	DDR_0 Space 3	0xD0000000 - 0xDFFFFFFF	
	DDR_0 Space 2	DDR_0 Space 2	0xC0000000 - 0xCFFFFFFF	
	DDR_0 Space 1	DDR_0 Space 1	0xB0000000 - 0xBFFFFFFF	
	DDR_0 Space 0	DDR_0 Space 0	0xA0000000 - 0xAFFFFFFF	
	FPGA Fabric FIC Region4	FPGA Fabric FIC Region4	0x90000000 - 0x9FFFFFFF	
	FPGA Fabric FIC Region3	FPGA Fabric FIC Region3	0x80000000 - 0x8FFFFFFF	
	FPGA Fabric FIC Region2	FPGA Fabric FIC Region2	0x70000000 - 0x7FFFFFFF	
			0x60100000 - 0x6FFFFFFF	
	AHB-to-eNVM_1 Registers	AHB-to-eNVM_1 Registers	0x600C0000 - 0x600FFFFF	
	AHB-to-eNVM_0 Registers	AHB-to-eNVM_0 Registers	0x60080000 - 0x600BFFFF	
	eNVM_1	eNVM_1	0x60040000 - 0x6007FFFF	
	eNVM_0	eNVM_0	0x60000000 - 0x6003FFFF	
	FPGA Fabric FIC Region1	FPGA Fabric FIC Region1	0x50000000 - 0x5FFFFFFF	
Peripheral Bit-band alias region of Cortex-M3 Processor			0x44000000 - 0x4FFFFFFF	
	Peripherals(BB View)		0x42000000 - 0x43FFFFFF	
			0x40410000 - 0x41FFFFFF	
	Cache Back door		0x40400000 - 0x4040FFFF	
			0x40044000 - 0x403FFFFF	
	USB	USB	0x40043000 - 0x40043FFF	
			0x40042000 - 0x40042FFF	
	Ethernet MAC Control	Ethernet MAC Control	0x40041000 - 0x40041FFF	
			0x40039000 - 0x40040FFF	
	SYSREG	SYSREG	0x40038000 - 0x40038FFF	
			0x40030000 - 0x40037FFF	
	Config DDR_1, PCIe_0, PCIe_1 etc	Config DDR_1, PCIe_0, PCIe_1 etc	0x40020400 - 0x4002FFFF	→ (63K space allocation for devices outside MSS)
	Config DDR_0	Config DDR_0	0x40020000 - 0x400203FF	
			0x40018000 - 0x4001FFFF	
	RTC	RTC	0x40017000 - 0x40017FFF	
	COMBLK	COMBLK	0x40016000 - 0x40016FFF	
	CAN	CAN	0x40015000 - 0x40015FFF	
	High Performance DMA	High Performance DMA	0x40014000 - 0x40014FFF	
	MSS GPIO	MSS GPIO	0x40013000 - 0x40013FFF	
	I2C_1	I2C_1	0x40012000 - 0x40012FFF	
	SPI_1	SPI_1	0x40011000 - 0x40011FFF	
	UART_1	UART_1	0x40010000 - 0x40010FFF	
			0x40007000 - 0x4000FFFF	
	Fabric Interface Interrupt Controller	Fabric Interface Interrupt Controller	0x40006000 - 0x40006FFF	
	Watchdog	Watchdog	0x40005000 - 0x40005FFF	
	Timer	Timer	0x40004000 - 0x40004FFF	
	Peripheral DMA Control	Peripheral DMA Control	0x40003000 - 0x40003FFF	
	I2C_0	I2C_0	0x40002000 - 0x40002FFF	
	SPI_0	SPI_0	0x40001000 - 0x40001FFF	
	UART_0	UART_0	0x40000000 - 0x40000FFF	
	FPGA Fabric FIC Region0	FPGA Fabric FIC Region0	0x30000000 - 0x3FFFFFFF	
SRAM Bit-band alias region of Cortex-M3 Processor			0x24000000 - 0x2FFFFFFF	
	eSRAM_0/eSRAM_1(BB View)		0x22000000 - 0x23FFFFFF	
			0x20014000 - 0x21FFFFFF	
	ECC eSRAM_1	ECC eSRAM_1	0x20012000 - 0x20013FFF	
	ECC eSRAM_0	ECC eSRAM_0	0x20010000 - 0x20011FFF	
	eSRAM_1	eSRAM_1	0x20008000 - 0x2000FFFF	
Cortex M3 Processor System Region	eSRAM_0	eSRAM_0	0x20000000 - 0x20007FFF	
			0x00080000 - 0x1FFFFFFF	
Cortex M3 Processor Code Region	eNVM (Cortex-M3) Virtual View	eNVM (Fabric) Virtual View	0x0007FFFF 0x00000000	Visible only to FPGA Fabric Master

3.7.1 Remapping eNVM Address Space to 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**



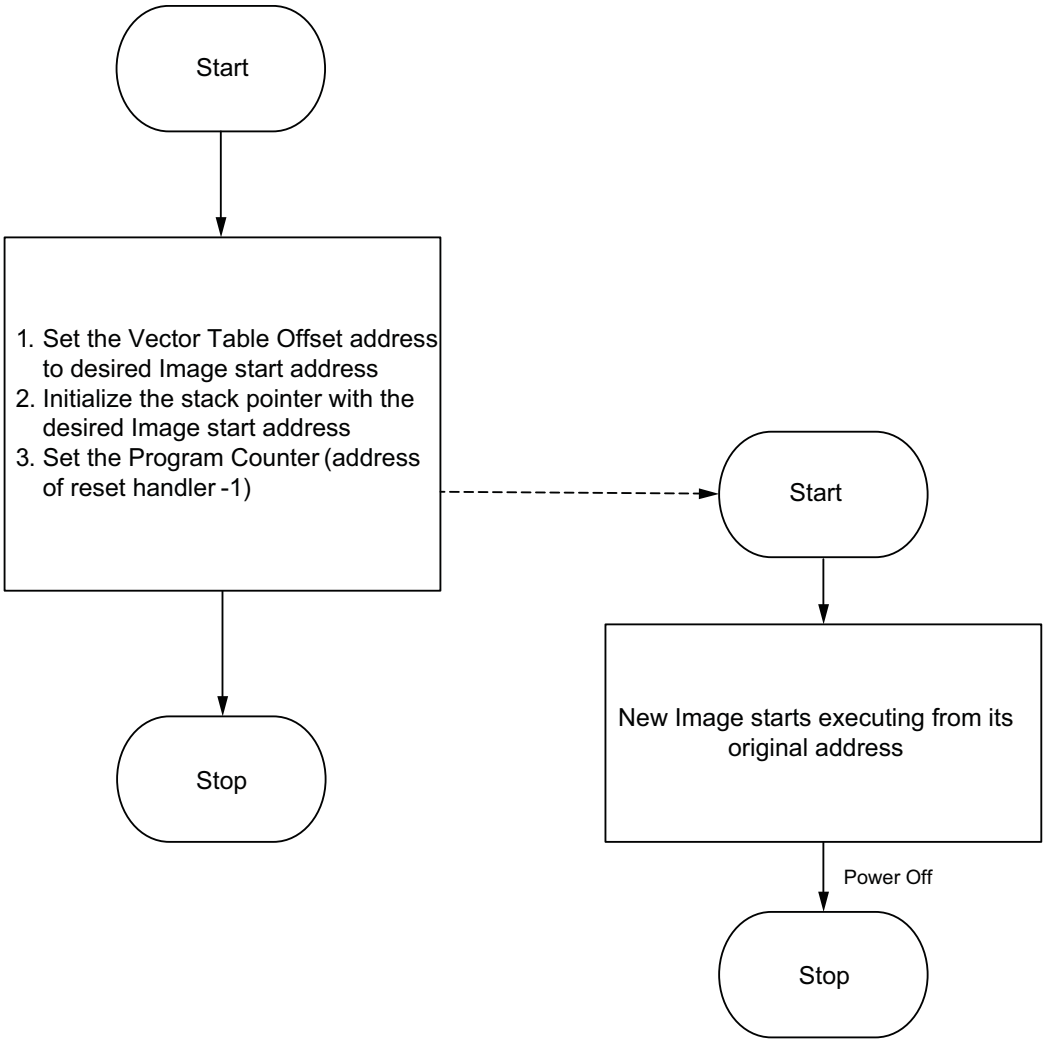
In the example scenario (as shown in Figure 8), there are three images, which can be remapped to the starting address of the Cortex-M3 processor code space or can be made executable for the Cortex-M3 processor. To create the independent executable images with the required memory map, it is required to create the linker scripts with the required memory map. Once the executable images are created for the required memory map in Production mode, these images are added to the programming file using the eNVM clients in the Libero 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 (0×00000000). In such cases, without remapping approach has to be used, which is explained as follows.

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

- **Without Remapping:** By default, the eNVM base address 0×60000000 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 must 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 Execution Control to New Image in eNVM without Remapping**



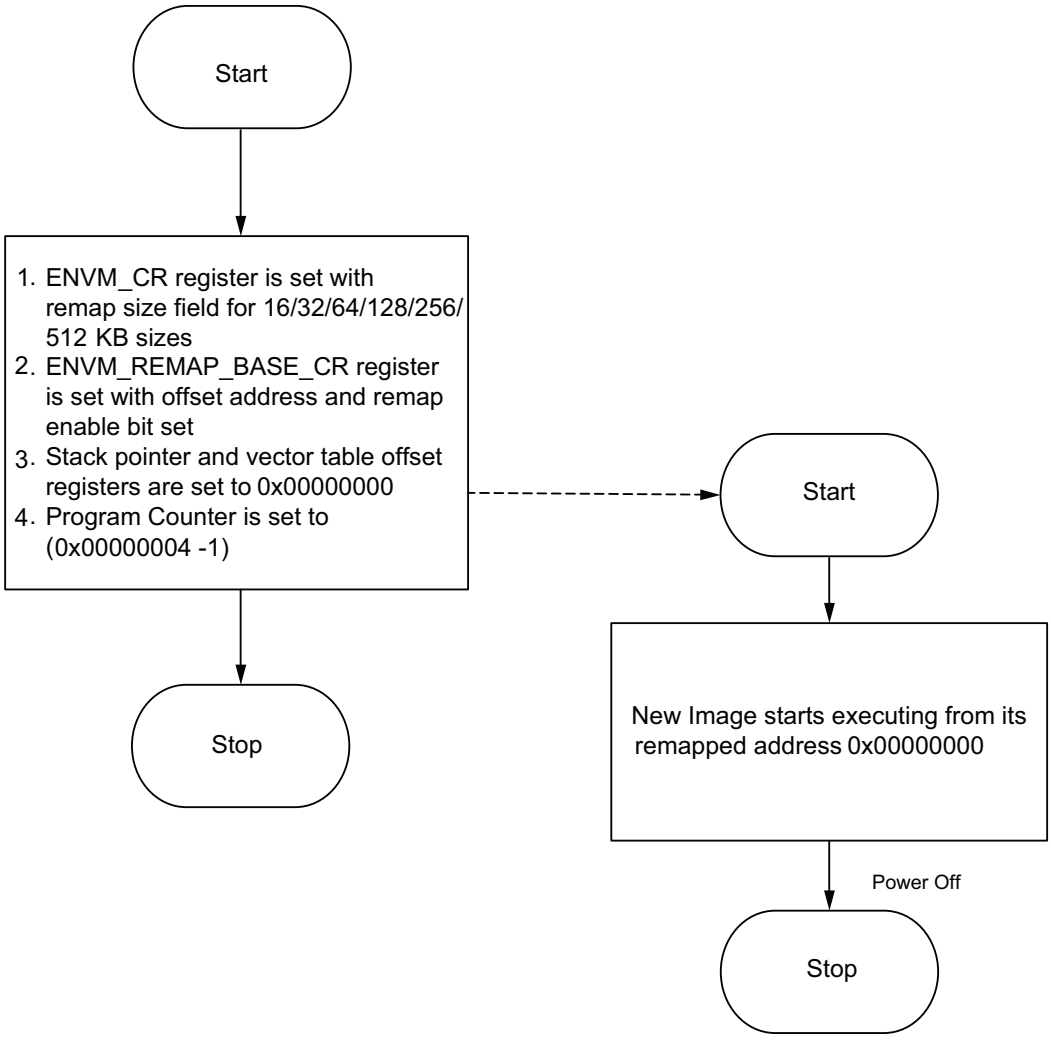
For example, for the memory map of the different images explained in Figure 8, page 12, the images are built with the base address as shown in Figure 8, page 12. To run Image 2, while Image 1 is running, use the following steps, as shown in Figure 9:

1. Set the vector table offset address register is set to 0×60010000
2. Initialize the stack pointer with the content of 0×60010000
3. Change the program counter to the reset handler of Image 2, that is, PC = (0×60010004 -1)

With the all preceding 3 steps, Image 2 starts executing from 0×60010000.

- **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 (0×0000_0000) of the Cortex-M3 processor code region, the linker scripts take care of building the images from the bottom (0×0000_0000) code region. The eNVM offset address must not be used in this approach. This approach is explained in the flow chart shown in Figure 10.

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



For example, for the memory map of the different images explained in Figure 8, page 12, the images are built with 0×00000000 as a base address. To run 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 0×00010000
3. Set the Stack Pointer to 0×00000000
4. Set the PC to 0×00000004 - 1

Note: To set the ENVM_CR register to 16/32/64/128/256/512 KB as remap image size, refer to the eNVM Controller Chapter in *UG0331: SmartFusion2 Microcontroller Subsystem User Guide*.

With all the preceding steps, the new Image 2 starts executing from 0×00000000, which is mapped to 0×60010000.

The reference design is provided with this application note with remapping and without remapping. For design files, refer to Prerequisites, page 7 and follow the steps explained in Setting Up the Demo Design, page 20 for executing the reference design.

Table 2 describes the registers, which are required to be set for the eNVM remapping to the bottom (0×0000_0000) of the Cortex-M3 processor. The SYSREG block is located at address 0×40038000 in the Cortex-M3 processor address space.

Table 2 • **eNVM Remap Register**

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
ENVM_CR	0×C	RW-P	Register	sysreset_n	eNVM Configuration register
ENVM_REMAP_BASE_CR	0×10	RW-P	Register	sysreset_n	eNVM remap configuration register for the Cortex-M3 processor.

3.7.2 Remapping eNVM to Soft Core Processor Memory Map

The SoftCore processor implemented in SmartFusion2 SoC FPGA fabric can access the eNVM for 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 of in such a way that these two partitions are mutually exclusive. The remapping of the eNVM offset address to the SoftCore processor bottom (0×0000_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 0×40038000 in the Cortex-M3 processor address space.

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

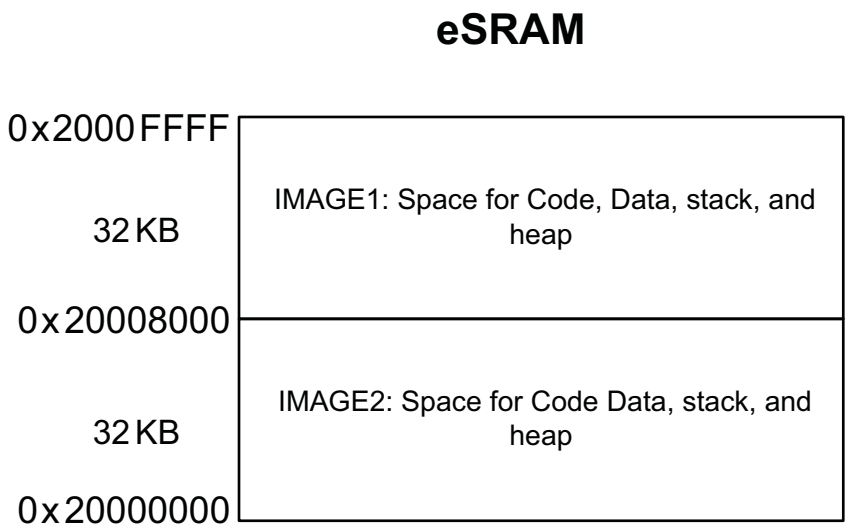
Table 3 • **eNVM Remap Register to Fabric SoftCore Processor Address Space**

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
ENVM_REMAP_FAB_CR	0×14	RW-P	Register	sysreset_n	NVM remap configuration register for the soft processor in the FPGA

3.7.3 Remapping eSRAM to 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 scenario shown in Figure 11, page 15, describes two images, which can be remapped to the bottom (0×0000_0000) of the Cortex-M3 processor or can be made executable for the Cortex-M3 processor. To create the independent executable images with the required memory map, the linker scripts need to be created with the required memory map. After creating the images required for memory map in Production mode, copy the images to an external memory such as SPI Flash, eNVM, and so on; and are code shadowed by the bootloader to the eSRAM whenever it is required to execute the new images.

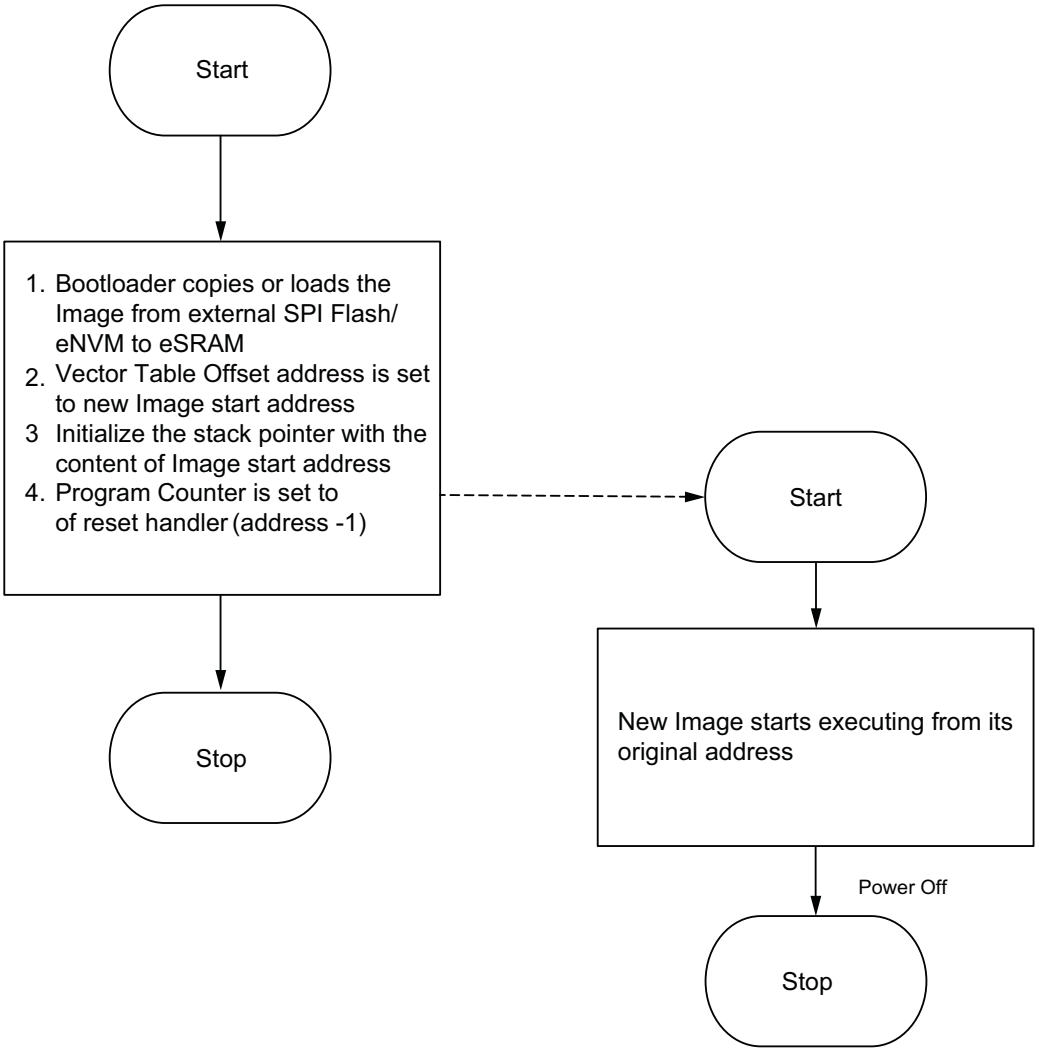
After copying the images to eSRAM by bootloader, the execution control can be allowed to the desired image by using any of the following two approaches:

If the executable images are built with an absolute address, the execution control needs to be allowed without using the remapping to the starting address of the code space (0×00000000). In such cases, without remapping approach, which is explained as follows (Point 1), has to be used.

If the executable images are built with the address 0×00000000, the execution control needs to be allowed by using remapping to starting address of the code space (0×00000000). In such cases, the remapping approach, which is explained as follows (Point 2), has to be used.

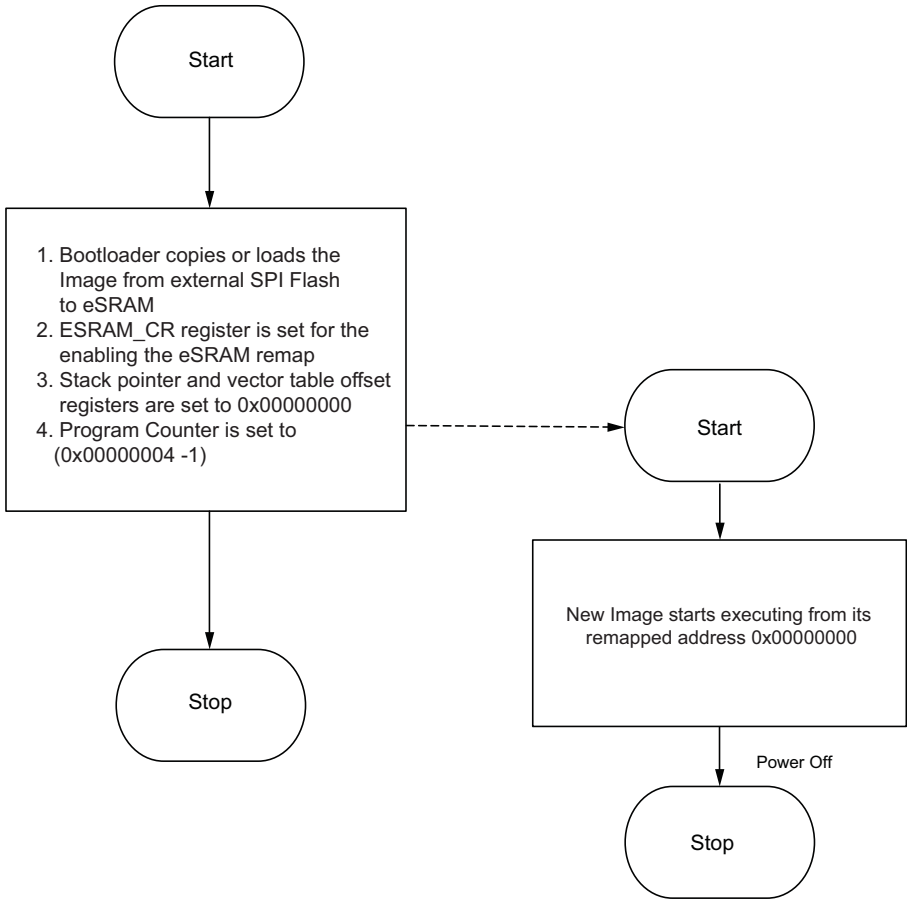
- **Without Remapping:** Using the vector table offset register in the system registers, the vector table address of the desired image can be set for execution, and point the Stack Pointer (SP) and the program counter to the reset handler of the desired image. This allows the Cortex-M3 processor to execute the new image. The eSRAM address must 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 Execution Control to New Image in eSRAM without Remapping**



- **With Remapping:** In this approach, the new image address can be remapped to the bottom (0×0000_0000) of the Cortex-M3 processor by using the ESRAM_CR registers. As the new image address is remapped to the bottom (0×0000_0000) of the Cortex-M3 processor code region, the linker scripts take care of building the images from the bottom (0×0000_0000) code region. The eSRAM address must not be used. Instead, an 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 Execution Control to New Image in eSRAM with Remapping**



For example, for the memory map of the different images explained in Figure 11, page 15, the images are built with 0×00000000 as a base address. If it is required to jump from Image 2 to Image 1, use the following steps (as explained in Figure 13).

1. Copy the image1 from Flash to eSRAM starting address 0×20008000
2. Set the ESRAM_CR register to enable the eSRAM remapping to 0×00000000
3. Set the Stack Pointer to 0×00008000 and Vector Table offset register to 0×00008000
4. Set the PC to 0×00008004 -1

With all the preceding steps, the new Image1 starts executing from 0×00008000, which is then mapped to address 0×20008000.

The reference design is provided with this application note with remapping and without remapping. For design files, refer to Prerequisites, page 7 and follow the steps explained in Setting Up the Demo Design, page 20 for executing the reference design.

Table 4 lists the registers that are required to be set for the eSRAM remapping. The SYSREG block is located at address 0×40038000 in the Cortex-M3 processor address space.

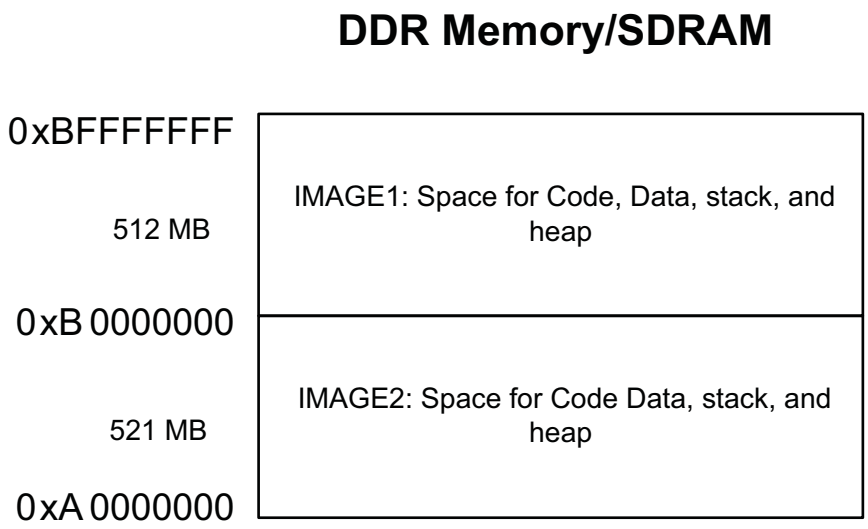
Table 4 • **Registers Required to eSRAM Remapping**

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

3.7.4 Remapping External RAM (DDR/SDR SDRAM Interface) to 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 (0×0000_0000) of the Cortex-M3 processor or can be made executable for the Cortex-M3 processor.

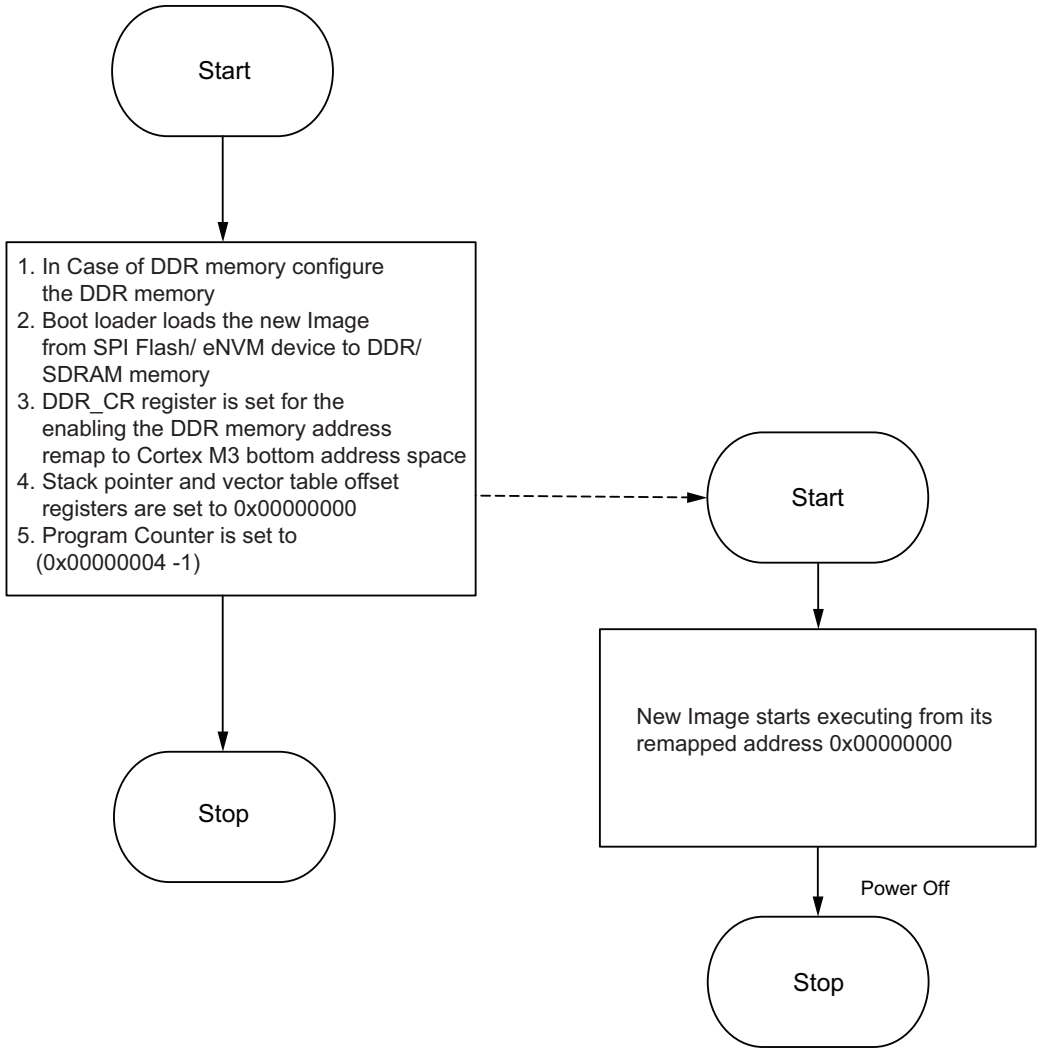
To create the independent executable images with the required memory map, the linker scripts need to be created with the required memory map. After creating the required images for the memory map in Production mode, these images are to be copied to an external memory such as SPI Flash, and code shadowed by the bootloader to DDR memory or SDRAM whenever the execution of the new images is required.

After copying the image 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 (0×0000_0000) of the Cortex-M3 processor code region. As the new image start address is re-mapped to the Cortex-M3 processor code region 0×0000_0000, the linker scripts take care of building the images from the code region 0×0000_0000. The DDR memory or SDRAM addresses (0×A000_0000) must 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 the 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**



The reference design is provided with this application note with remapping. For design files, refer to Prerequisites, page 7 and follow the steps explained in Setting Up the Demo Design, page 20 for executing the reference design.

Table 5 lists the registers required to be set for the DDR/SDR SDRAM remapping. The SYSREG block is located at address 0×40038000 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	0×8	RW-P	Register	sysreset_n	DDR control Register. Configures DDR Space.

3.7.4.1 Firmware Drivers

The following firmware drivers are used in this application:

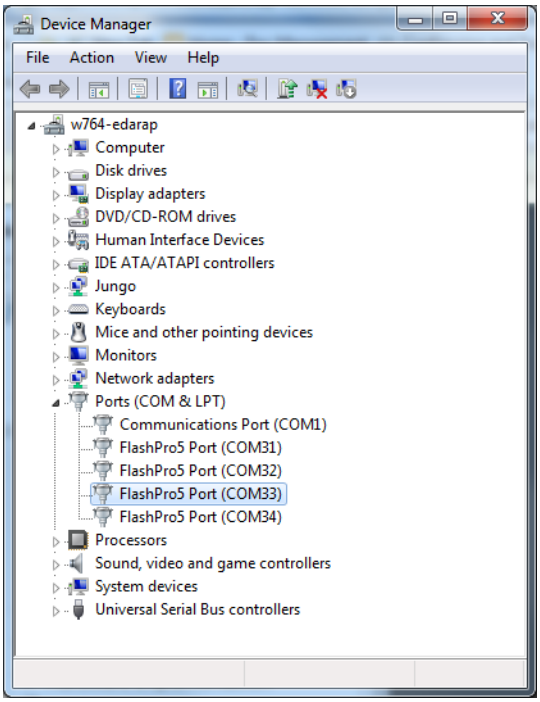
- MSS MMUART driver
  - To communicate with serial terminal program on the Host PC
- MSS GPIO driver
  - To drive onboard LEDs

3.8 Setting Up the Demo Design

The following steps describe how to set up the demo for the SmartFusion2 Advanced Development Kit board:

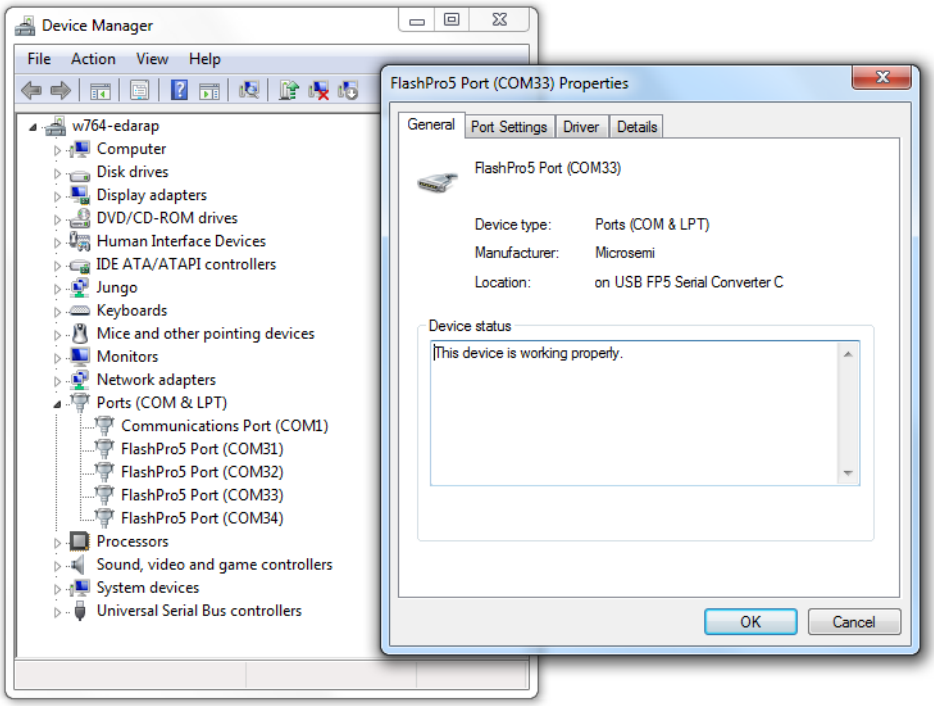
1. Connect the Host PC to the J33 Connector using the USB A to mini-B cable. The USB to UART bridge drivers are automatically detected. Download and install the drivers from *www.microsemi.com/soc/documents/CDM_2.08.24_WHQL_Certified.zip*, if the drivers are not installed or detected automatically. Verify, if the detection is made in the device manager, as shown in Figure 16.

Figure 16 • **Device Manager**



2. Select one of the four COM ports with **Location** as on **USB FP5 Serial Converter C**. Figure 17, page 21 shows the **Device Manager** window and its properties that display the **USB Serial Port** details. The COM port number is required to run the demo design.

Figure 17 • **Device Manager - FlashPro5 Properties**



3. Connect the jumpers on the SmartFusion2 Advanced Dev Kit board, as shown in Table 6.

CAUTION: The power supply switch **SW7** on the board should be in OFF position, while making the jumper connections.

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

Jumper	Pin (From)	Pin (To)	Comments
J116, J353, J354, J54	1	2	These are the default jumper settings of the Advanced Dev Kit board. Ensure these jumpers are set accordingly.
J123	2	3	
J124, J121, J32	1	2	JTAG programming via FTDI

4. Connect the power supply to the J42 connector on the SmartFusion2 Advanced Development Kit board.

3.8.1 Board Setup

Snapshots of the SmartFusion2 Advanced Development Kit board with the complete set up is given in the Appendix 2: SmartFusion2 Advanced Development Kit Board, page 26.

3.8.2 Programming the Device

Program the SmartFusion2 Advanced Development Kit board with the job file provided as part of the design files using FlashPro Express software, refer to Appendix 1: Programming the Device Using FlashPro Express, page 23.

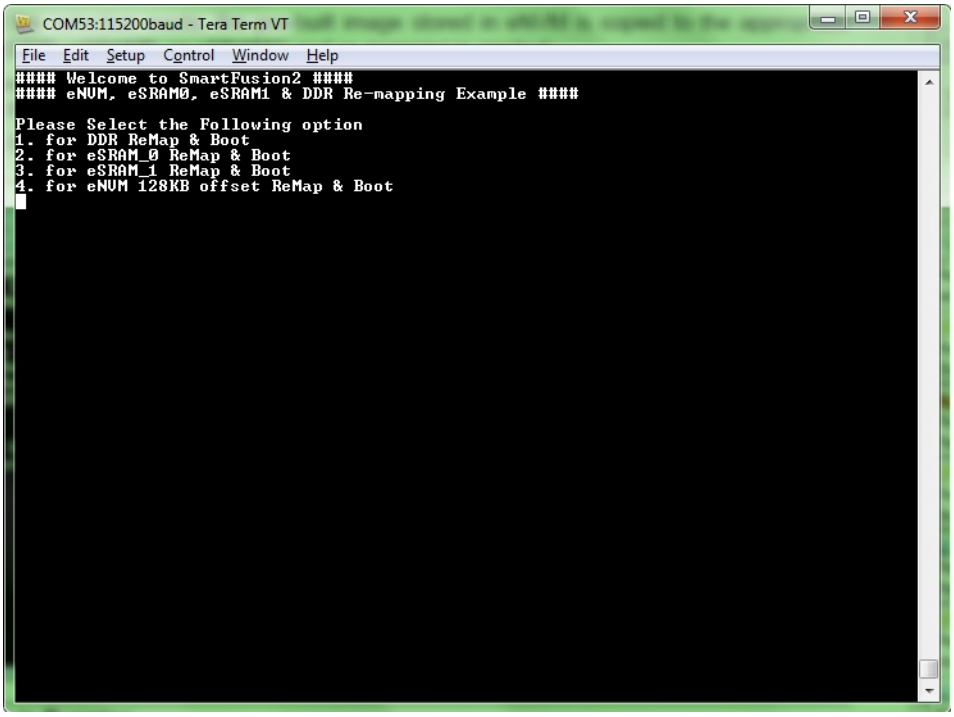
3.8.3 Running the Design

The following steps describe how to run the design.

1. Press **SW9** switch to reset the board after successful programming.

Figure 18 shows the TeraTerm window.

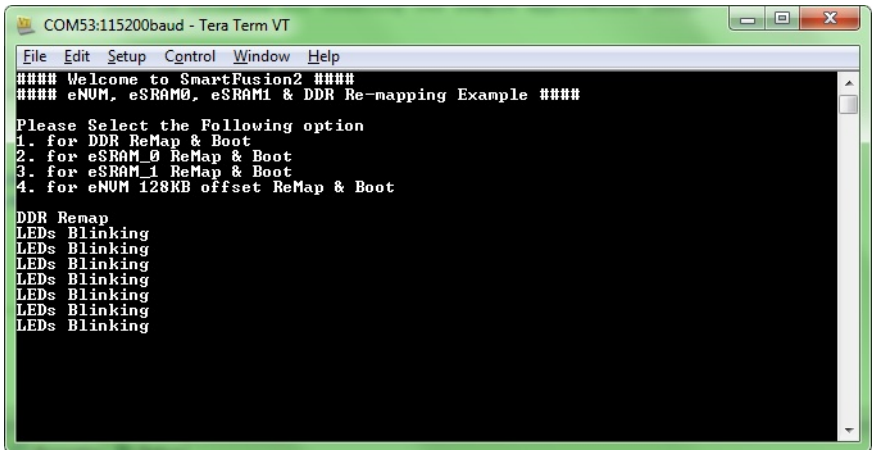
Figure 18 • **Main Menu of Re-Mapping Application Note**



2. 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.
3. 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 SmartFusion2 Advanced Development Kit.

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

Figure 19 • **Re-Mapped Image is Running**



For booting multiple images without remapping, refer to the *AC372: SmartFusion SoC: Basic Bootloader and Field Upgrade eNVM Through IAP Interface* application note.

3.9 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.

4 Appendix 1: Programming the Device Using FlashPro Express

This section describes how to program the SmartFusion2 device with the programming job file using FlashPro Express.

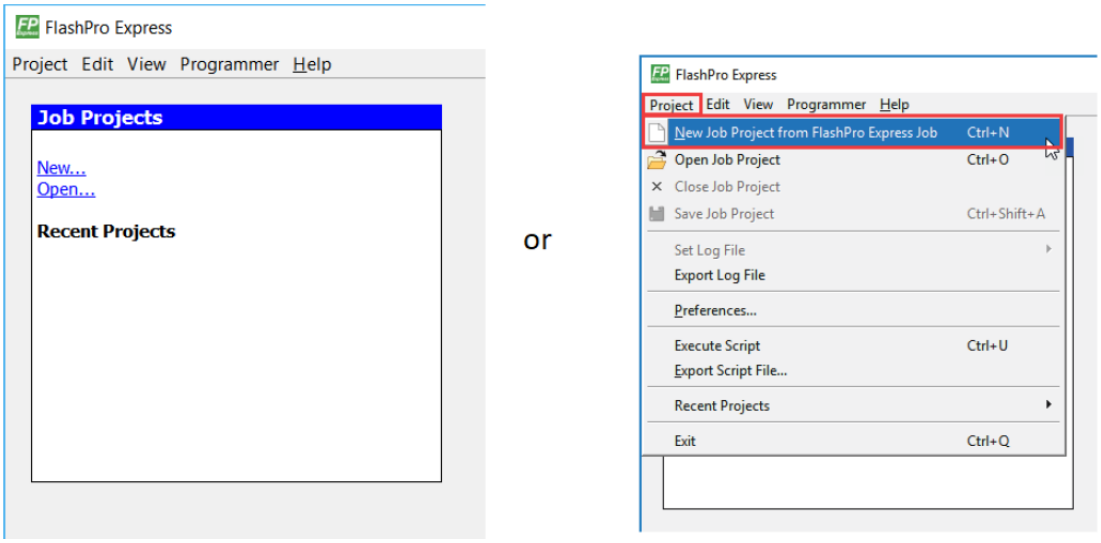
To program the device, perform the following steps:

1. Ensure that the jumper settings on the board are the same as those listed in Table 6, page 21.

Note: The power supply switch must be switched off while making the jumper connections.

2. Connect the power supply cable to the **J42** connector on the board.
3. Power **ON** the power supply switch **SW7**.
4. On the host PC, launch the **FlashPro Express** software.
5. Click **New** or select **New Job Project from FlashPro Express Job** from **Project** menu to create a new job project, as shown in Figure 20.

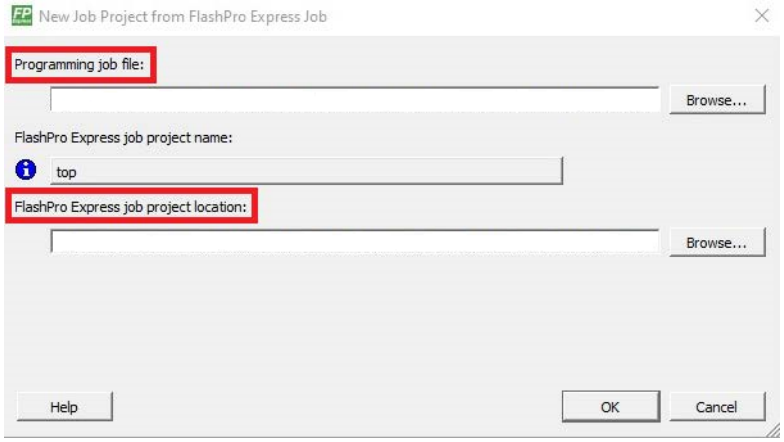
Figure 20 • **FlashPro Express Job Project**



6. Enter the following in the **New Job Project from FlashPro Express Job** dialog box:

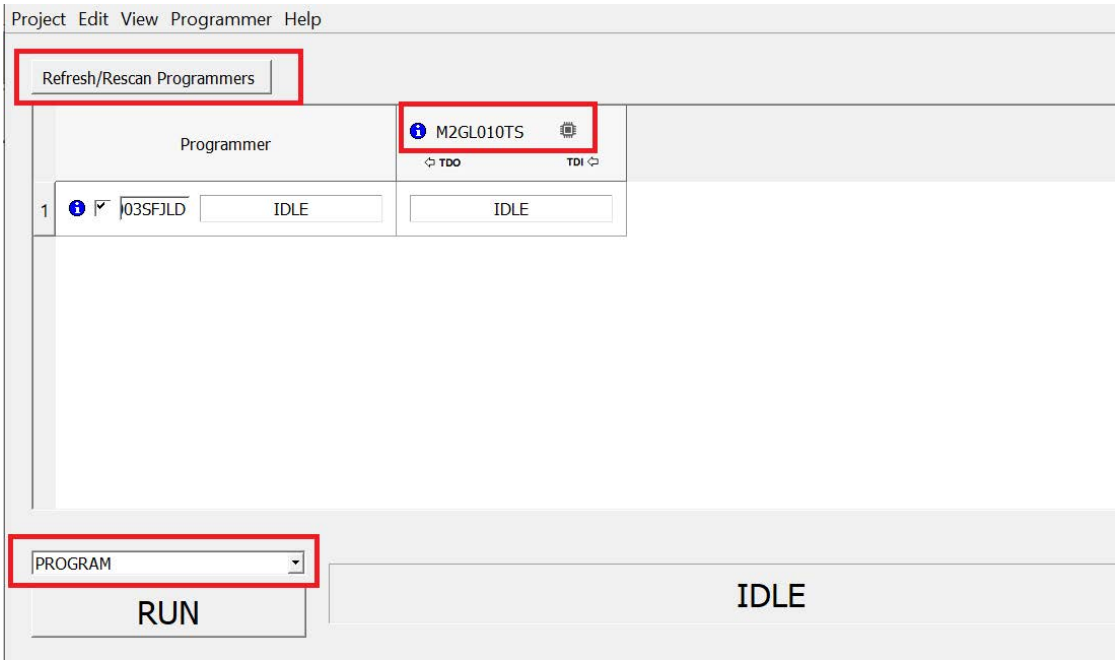
- **Programming job file:** Click **Browse**, and navigate to the location where the .job file is located and select the file. The default location is:
  <download_folder>\m2s_ac390_df\Programming_Job
- **FlashPro Express job project name:** Click **Browse** and navigate to the location where you want to save the project.

Figure 21 • **New Job Project from FlashPro Express Job**



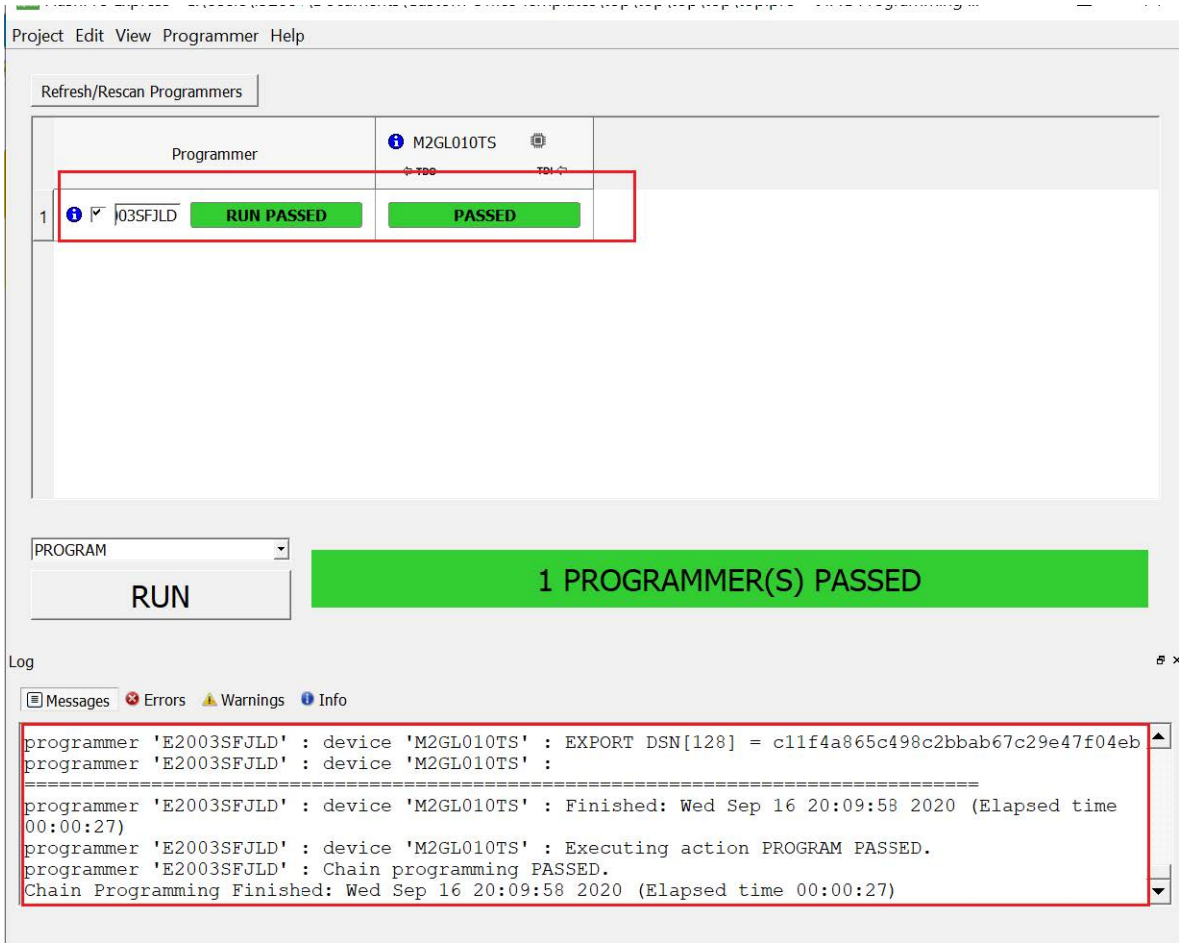
7. Click **OK**. The required programming file is selected and ready to be programmed in the device.
8. The FlashPro Express window appears, as shown in Figure 22. Confirm that a programmer number appears in the Programmer field. If it does not, confirm the board connections and click **Refresh/Rescan** Programmers.

Figure 22 • **Programming the Device**



9. Click **RUN**. When the device is programmed successfully, a **RUN PASSED** status is displayed as shown in Figure 23.

Figure 23 • **FlashPro Express—RUN PASSED**



10. Close **FlashPro Express** or in the Project tab, click **Exit**.

5 Appendix 2: SmartFusion2 Advanced Development Kit Board

Figure 24 shows the SmartFusion2 Advanced Development Kit board.

Figure 24 • **SmartFusion2 Advanced Development Kit Board**