AC391
Application Note
SmartFusion2 SoC FPGA - eNVM Initialization

Microsemi
a Microchip company

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

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 77068).

1.3 Revision 8.0

Updated the document for Libero SoC v11.6 software release (SAR 71726).

1.4 Revision 7.0

Updated the document for Libero SoC v11.5 software release (SAR 64416).

1.5 Revision 6.0

The following are the changes made in revision 6.0 of this document.

- Updated the document for Libero SoC v11.4 software release (SAR 59913).
- Updated the document for SmartFusion2 Evaluation Kit details (SAR 59913).

1.6 Revision 5.0

Updated the document for Libero SoC v11.3 software release (SAR 57098).

1.7 Revision 4.0

Updated the document for Libero SoC v11.2 software release (SAR 52884).

1.8 Revision 3.0

Updated the document for Libero SoC v11.0 software release (SAR 47576).

1.9 Revision 2.0

Updated the document for Libero SoC v11.0 beta SP1 software release (SAR 44871).

1.10 Revision 1.0

Updated the document for Libero SoC v11.0 beta SPA software release (SAR 42847).

2 Purpose

This application note describes the different methods to initialize the embedded Non-Volatile Memory (eNVM) in the SmartFusion®2 System-on-Chip (SoC) Field Programmable Gate Array (FPGA) devices.

3 SmartFusion2 SoC FPGA - eNVM Initialization

3.1 Introduction

The SmartFusion2 SoC FPGA devices have a maximum of two on-chip 256 KB eNVM flash memories. The eNVM is used to store the application code image or used to store data, which can be used by the end application. The eNVM can be initialized by these methods:

- Using the eNVM client of the eNVM configurator in the Libero® System-on-Chip (SoC)
- Writing into the eNVM using ARM® Cortex®-M3 processor
- In-application programming (IAP)
- Writing into the eNVM using custom logic in the FPGA fabric

For more information about how to initialize the eNVM using the eNVM client in Libero and the ARM Cortex-M3 processor, refer to the *AC429: SmartFusion2 and IGLOO2 - Accessing eNVM and eSRAM from FPGA Fabric Application Note.*

For detailed description about eNVM, refer to the "eNVM" chapter in *UG0331: SmartFusion2 Microcontroller Subsystem User Guide*.

3.2 References

The list of references used are:

- *UG0331: SmartFusion2 Microcontroller Subsystem User Guide*
- *SmartFusion2 System Builder User Guide*

3.3 Design Requirements

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

Table 1 • **Design Requirements**

Requirement	Version
Operating system	64 bit Windows 7 and 10
Hardware	
SmartFusion2 Security Evaluation Kit: • FlashPro4 programmer • 12 V adapter • USB A to Mini-B cable	Rev D or later
Host PC or Laptop	Windows 64-bit Operating System
Software	
FlashPro Express	Refer to the `readme.txt` file provided in the design files for the software versions used with this reference design.
Libero SoC for viewing the design files	
SoftConsole	
Host PC Drivers	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:

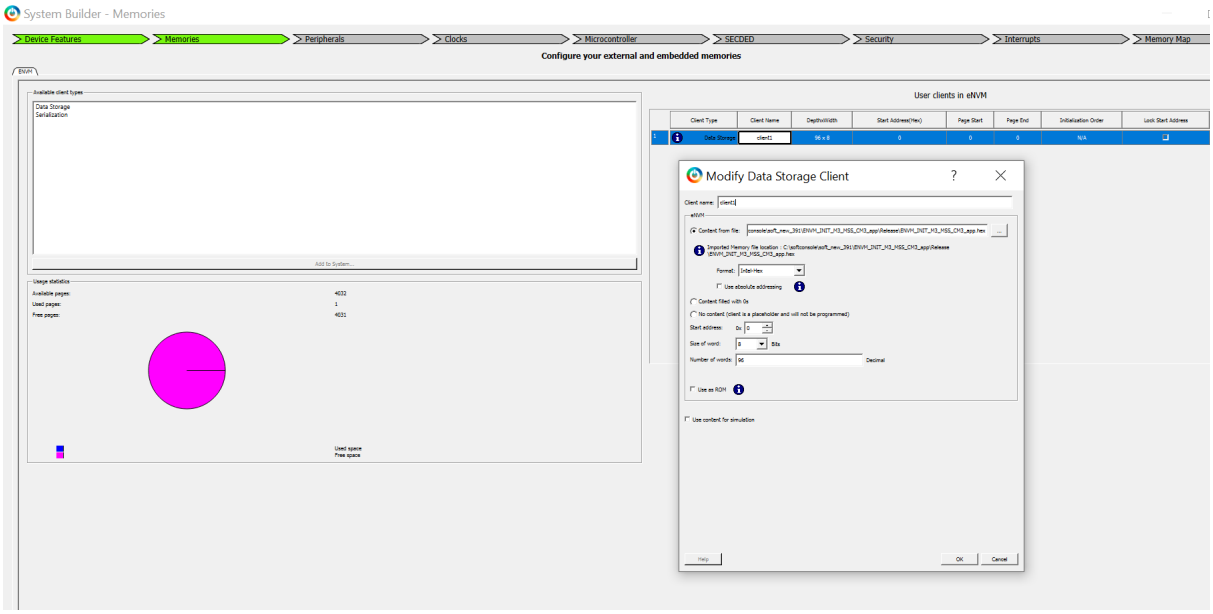
   http://soc.microsemi.com/download/rsc/?f=m2s_ac391_df

3.5 Initializing the eNVM Using the Libero eNVM Client

The Libero eNVM client creates the necessary programming information that FlashPro uses to initialize the eNVM during the programming. The following steps describe how to generate a programming file with the eNVM client:

1. In SmartFusion2 SoC FPGA Libero project, double-click **ENVM_INIT_M3_0** in the Libero **SmartDesign** window to open the System Builder Configuration window.
2. Go to the **Memories** tab to open the **ENVM** window.
3. Select **Data Storage** under **Available Client types** and click **Add to System**, as shown in Figure 1.

Figure 1 • **Adding the Client Type**

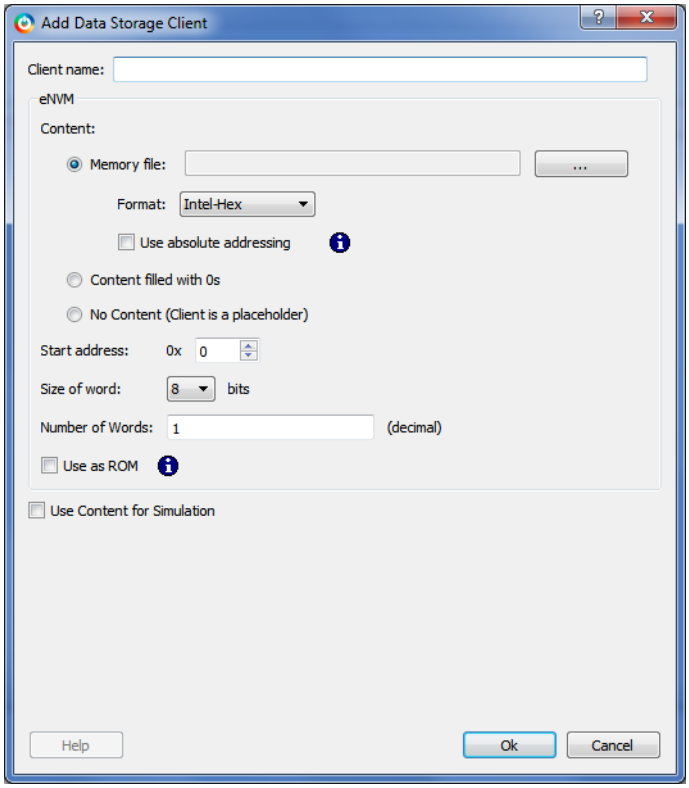


The **Add Data Storage Client** window is displayed, as shown in Figure 2, page 5. It supports four types of memory file formats:

- Intel-Hex
- Motorola-S
- Microsemi-Hex
- Microsemi-Binary

Create the memory file in any one of the above formats with your code or data. You can create the memory file for your code using the SoftConsole v(x.x) with the linker script `debug-in-microsemi-smartfusion2-esram.ld`.

Figure 2 • **Add Data Storage Client**



4. Enter the **Client name**.
5. Browse to the created **Memory file**, and click **Ok** to add the eNVM client.

   The Modify core - ENVM window (displayed next) shows the client and its size. You can also add more than one client with a different start address.

6. After adding the eNVM clients, click **Next** and **Generate System Builder Component**.
7. Save and generate the SmartDesign in Libero using the **Generate Component**.
8. Double-click the **Run Program Action** in the Libero **Design Flow** window to program the SmartFusion2 Security Evaluation Kit to initialize the eNVM with the memory file.

3.6 Initializing the eNVM Using the Cortex-M3 Processor

The following sections describe how to initialize eNVM using the Cortex-M3 processor with an example design. The design example describes how to write, read, and verify the data to or from different locations within the eNVM using the Cortex-M3 processor.

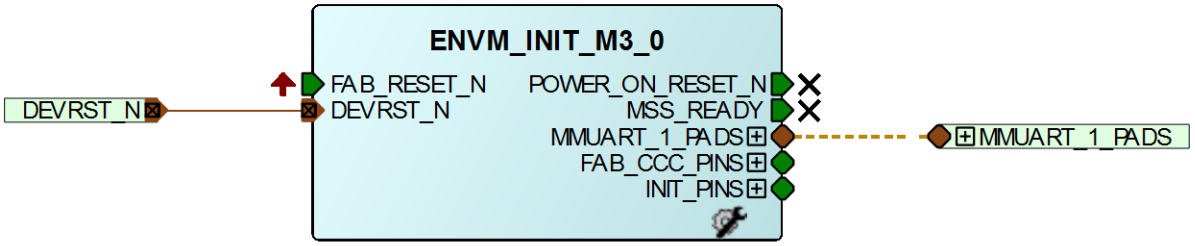
3.7 Design Description

The design example included with this application note uses RC oscillator and Fabric CCC to generate the base clock to MSS CCC. In the design example, the MSS CCC is configured to run the M3_CLK at 100 MHz, which drives the clock to the Cortex-M3 processor. The MMUART_1 is routed for communicating with the serial terminal program. The design receives the user given commands for read, write, and verify operations and a corresponding address, length, and data through the serial terminal program. After completing every operation, it displays the status (success/fail) of operation on the serial terminal program.

3.8 Hardware Implementation

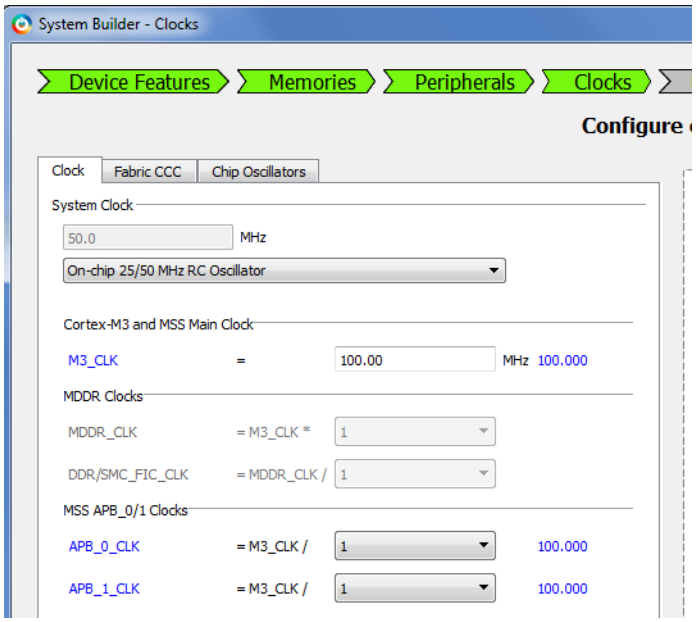
The hardware implementation involves configuring System Builder. Figure 3 shows the top-level hardware design in SmartDesign.

Figure 3 • **Top-Level SmartDesign**



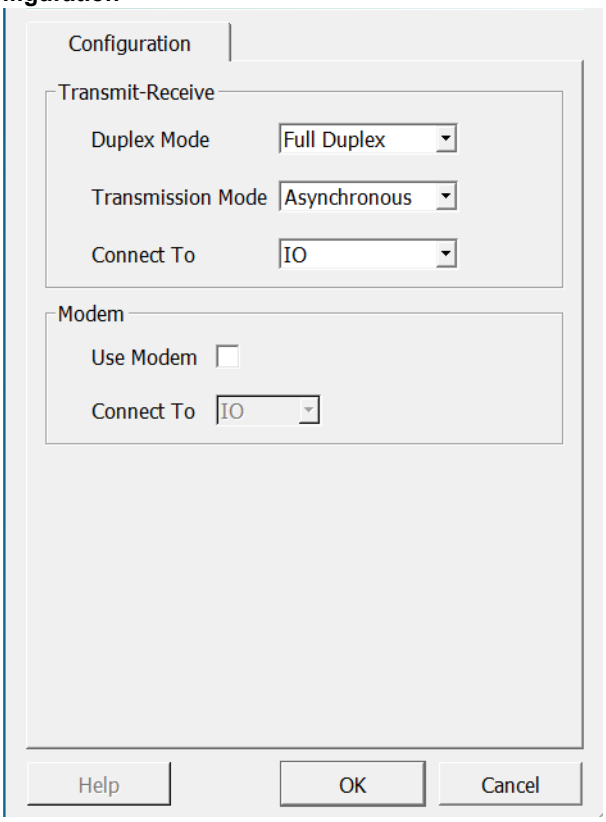
The MSS_CCC clock source is sourced from the FCCC through the **On-Chip Oscillator**. The FCCC is configured to provide the 100 MHz clock using GL0. Figure 4 shows the system clock configurations for the M3_CLK and APB_0_CLK clock settings.

Figure 4 • **Clock Configurations**



The MMUART_1 is used for reading and writing to the HyperTerminal window. On the SmartFusion2 Security Evaluation Kit board, the MMUART_1 TX and RX are connected to the mini-B USB through I/Os. Figure 5, page 7 shows the MMUART_1 configuration.

Figure 5 • **MMUART_1 Configuration**



3.9 Software Implementation

The software design example performs the write, read, and verify tasks on receiving commands from user through HyperTerminal.

The design uses the SmartFusion2 MSS MMUART driver to communicate with the serial terminal program running on the host PC.

The design implements APIs to read, write, and verify the data. The API implementation and usage are described in the following sections. For the API code, refer to Appendix 2: eNVM Driver APIs, page 16.

3.9.1 Write Operation

The design uses the **NVM_write()** API to write or program the data to eNVM over any memory range within the limits of 256 KB. This function supports programming data that spans across multiple pages. The following is the function prototype:

```
nvm_status_t
NVM_write
(
    uint32_t start_addr,
    const uint8_t * pidata,
    uint32_t length,
    uint32_t lock_page
);
```

The data is written from the memory location specified by the first parameter *start_addr*. This address is the relative address, which is added to the eNVM base address 0x60000000. The pidata parameter is the byte aligned starting address of the input data. The length parameter is the number of data bytes that are to be programmed. On successful execution, this function returns SUCCESS, otherwise it returns INVALID_PARAMETER.

3.9.1.1 Example:

```
uint8_t idata[815] = {"Z"};

nvm_status_t status = NVM_write((0x0, idata, sizeof(idata),
NVM_DO_NOT_LOCK_PAGE);
```

The **NVM_write()** API calls the **write_nvm()** API to perform the page write into eNVM after aligning the input data into pages. The **write_nvm()** API uses the eNVM controller's page-wise write command. It uses the following sequence to write or program the eNVM page:

1. Request access to eNVM by writing the 0x1 to the controller register REQ_ACCESS of eNVM.
2. Poll to the REQ_ACCESS for 0x5 (Cortex-M3 processor access to eNVM is granted).
3. Fill the WDBUFFER with the data that needs to be written into eNVM.
4. To write the data to the eNVM array, write the CMD control register with the page program and the address of the page.
5. Poll for eNVM busy bit in the STATUS control register of eNVM for 1. The 0 for this bit indicates that eNVM is busy in programming the data to the eNVM array. On programming, the eNVM controller makes busy bit to '0'.
6. Release the Cortex-M3 processor access to eNVM by writing 0x0 to the controller register REQ_ACCESS of eNVM.

The page program command programs the entire page with the data in the WDBUFFER.

Note: The eNVM frequency range (NV_FREQRNG field of ENVM_CR System Register) value must be set to the maximum value 15 to ensure the correct programming of the eNVM. After programming eNVM, restore the original frequency range value for eNVM read or verify operations.

3.9.2 Read Operation

The design uses the **MSS_NVM_read()** API to read the data from eNVM over any memory range within the limits of 256 KB. The following is the function prototype:

```
nvm_status_t

MSS_NVM_read

(

uint8_t * addr,

uint8_t * podata,

uint32_t len

);
```

The data is read from the memory location specified by the first parameter *addr*. This address is the relative address which is added to the eNVM base address 0x60000000. The *addr* parameter is the byte aligned address of eNVM from which the data is read. The *podata* parameter is the byte aligned address of the output buffer in which the read data is stored. The len parameter is the number of data bytes that are to be read. On successful execution, this function returns SUCCESS, otherwise it returns INVALID_PARAMETER.

3.9.2.1 Example:

```
uint8_t outbuf[815] = {0};

nvm_status_t status = MSS_NVM_read(0, outbuf, sizeof(outbuf));
```

The read API reads the data from eNVM similar to that of reading from any other memory location because the eNVM controller supports RAM type of accessing for the read operation. This API also checks for the 2-bit error while reading eNVM.

3.9.3 Verify Operation

The design uses the **NVM_verify** API to verify the eNVM memory against the reference data provided. This function supports verification that spans across multiple pages. The following is the function prototype:

```
nvm_status_t

NVM_verify

(

    uint32_t addr,

    const uint8_t * pidata,

    uint32_t length

);
```

The data is verified from the memory location specified by the first parameter addr. This address is the relative address which is added to the eNVM base address 0x60000000. The addr parameter is the byte aligned address of eNVM from which the data is to be verified. The pidata parameter is the byte aligned starting address of the reference input data against which the verification should be performed. The length parameter is the number of data bytes that are to be verified. On successful execution, this function returns SUCCESS, otherwise it returns INVALID_PARAMETER.

3.9.3.1 Example:

```
uint8_t idata[815] = {"Z"};

nvm_status_t status = NVM_write((0x0, idata, sizeof(idata),
NVM_DO_NOT_LOCK_PAGE);

status = NVM_verify(0x0, idata, sizeof(idata));
```

The **NVM_verify()** API calls the **verify_nvm()** API to perform the page verify to eNVM after aligning the input data into pages. The **verify_nvm()** API uses the eNVM controller's page-wise verify command. It uses the following sequence to verify the data on the eNVM page:

1. Request access to eNVM by writing the 0x1 to the controller register REQ_ACCESS of eNVM.
2. Poll to the REQ_ACCESS for 0x5 (Cortex-M3 processor access to eNVM is granted).
3. Fill the WDBUFFER with the data to verify the data in the eNVM array.
4. To verify the data in the eNVM array, write the CMD control register to verify the page program and the address of the page.
5. Poll for eNVM busy bit in the STATUS control register of eNVM for '1'. The '0' for this bit indicated eNVM is busy in programming the data to eNVM array. On programming, the eNVM controller makes busy bit to '0'.
6. Check the bit[1] of the STATUS register for '0' which indicates verify success. It is '1' in case of verify failure.
7. Release the Cortex-M3 processor access to eNVM by writing 0x0 to the controller register REQ_ACCESS of eNVM.

3.10 Setting Up the Design

Connect the following jumpers on the SmartFusion2 Security Evaluation Kit, as described in Table 2. Switch OFF the power supply switch SW7 while connecting the jumpers.

Table 2 • **SmartFusion2 Security Evaluation Kit Jumper Settings**

Jumper	Pin (From)	Pin (To)	Comments
J22, J23, J24, J8, J3	1	2	These are the default jumper settings of the Evaluation Kit board. Make sure these jumpers are set accordingly.

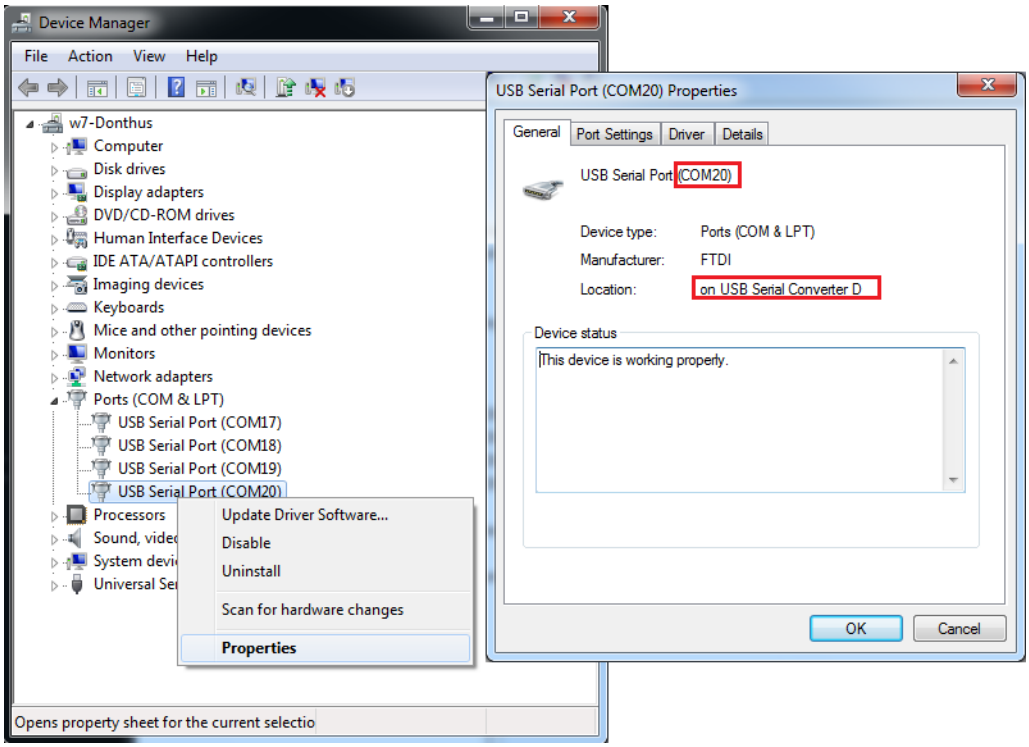
3.11 Running the Design

The following steps describe how to run the design:

1. Connect the FlashPro4 programmer to the J5 connector of the SmartFusion2 Security Evaluation Kit.
2. Connect the J18 connector on the SmartFusion2 Security Evaluation Kit to the host PC using the USB mini-B cable. Ensure that the USB to UART bridge drivers are automatically detected by verifying the Device Manager, as shown in Figure 6.

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 6.

Figure 6 • **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 J6 connector and change the power supply switch SW7 to ON.
5. Program the SmartFusion2 Security Evaluation 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 13.
6. Invoke the standalone SoftConsole v(x.x) Integrated Design Environment (IDE).
7. Load the SoftConsole project from the provided design files.
8. Launch the debugger in SoftConsole.

9. Start a **HyperTerminal** with the baud rate set to 57600, 8 data bits, 1 stop bit, no parity, and no flow control.

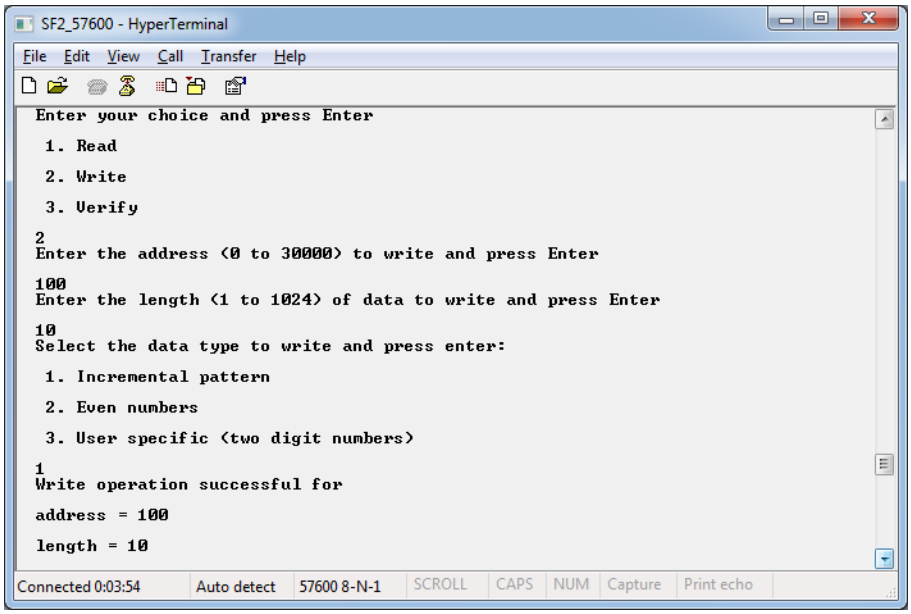
   If your PC does not have a HyperTerminal program, use any free serial terminal emulation program such as PuTTY or TeraTerm. For configuring HyperTerminal, TeraTerm, and PuTTY, refer to the *Configuring Serial Terminal Emulation Programs Tutorial.*

   When you run the debugger in SoftConsole, the HyperTerminal window shows a message to enter your choice.

10. Enter the choice to write. It prompts for address, length, and data consequently. Enter the values, as shown in Figure 7.

On writing, the message **write operation successful** is displayed.

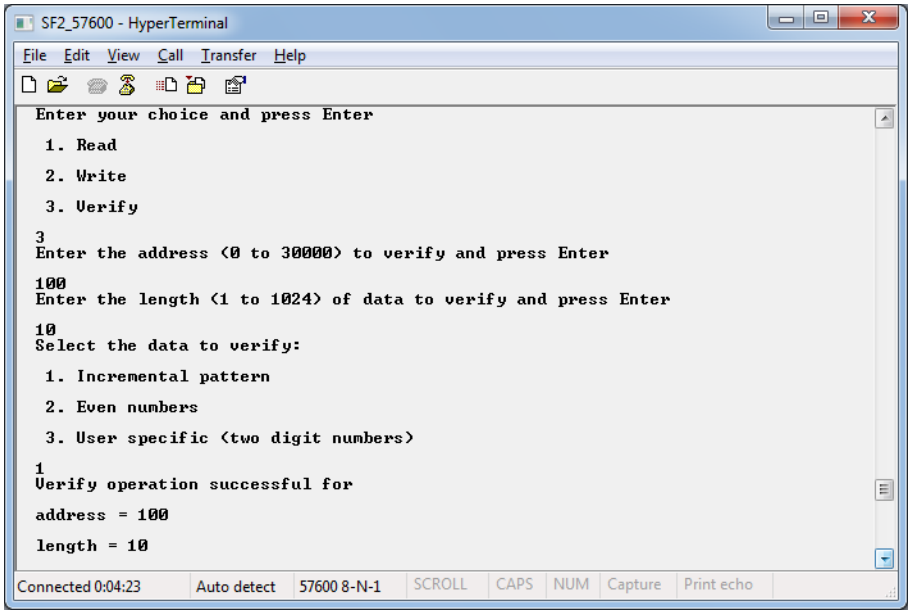
Figure 7 • **Write Operation**



11. Enter the choice to verify. It prompts for address, length, and data consequently. Enter the values, as shown in Figure 8.

    On writing, the message **verify operation successful** is displayed.

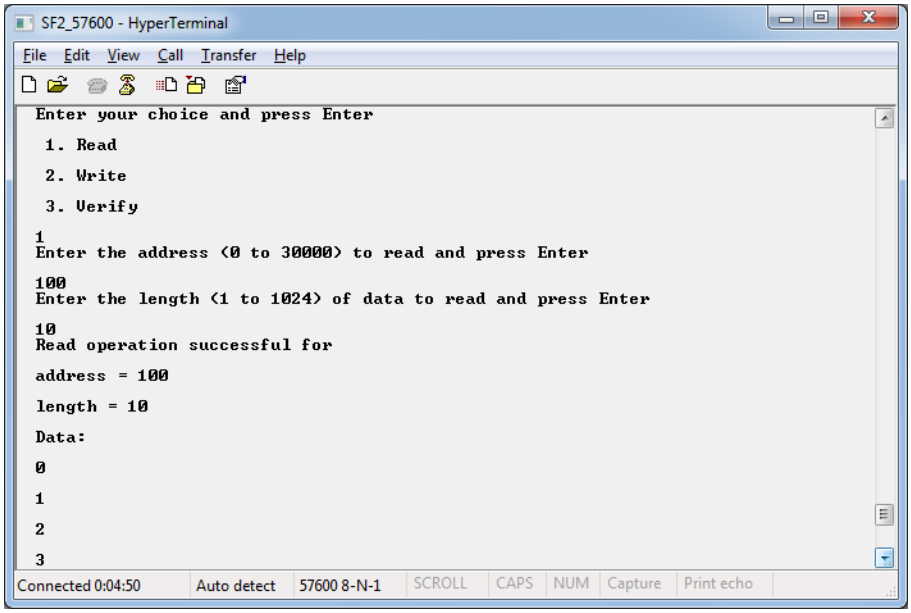
Figure 8 • **Verify Operation**



12. Enter the choice to read. It prompts for address and length consequently. Enter the values as shown in Figure 9.

    On reading, the read values are displayed.

Figure 9 • **Read Operation**



3.12 Conclusion

This application note describes how to initialize eNVM using the eNVM client of the eNVM configurator in the Libero SoC and using the Cortex-M3 processor.

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.

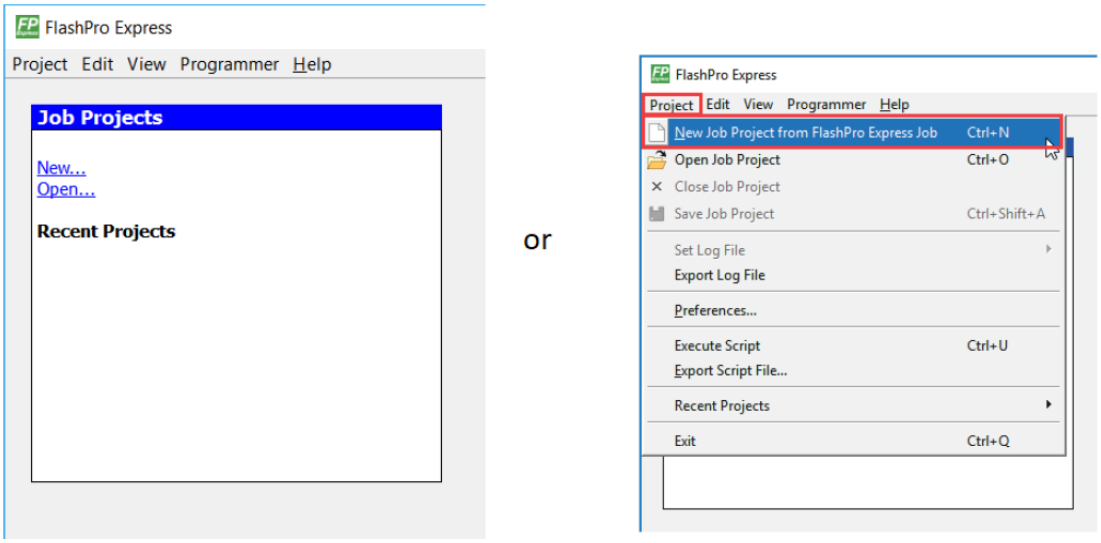
To program the device, perform the following steps:

1. Ensure that the jumper settings on the board are the same as those listed in Table 2, page 10.

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

2. Connect the power supply cable to the **J6** 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 10.

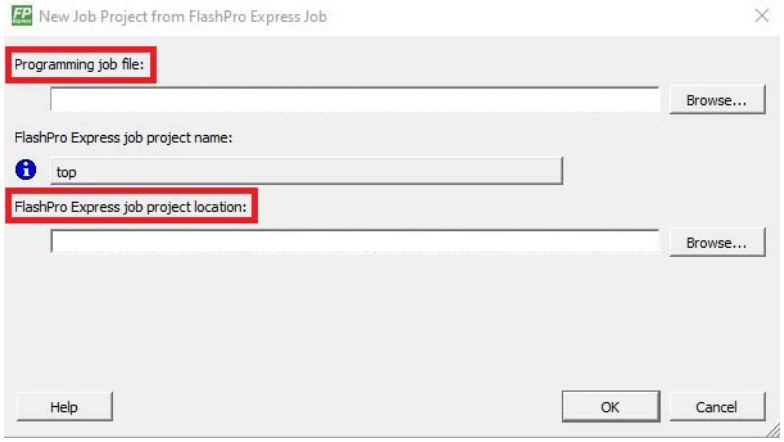
Figure 10 • **FlashPro Express Job Project**



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_ac391_df\Programming_Job
- **FlashPro Express job project name:** Click **Browse** and navigate to the location where you want to save the project.

Figure 11 • **New Job Project from FlashPro Express Job**

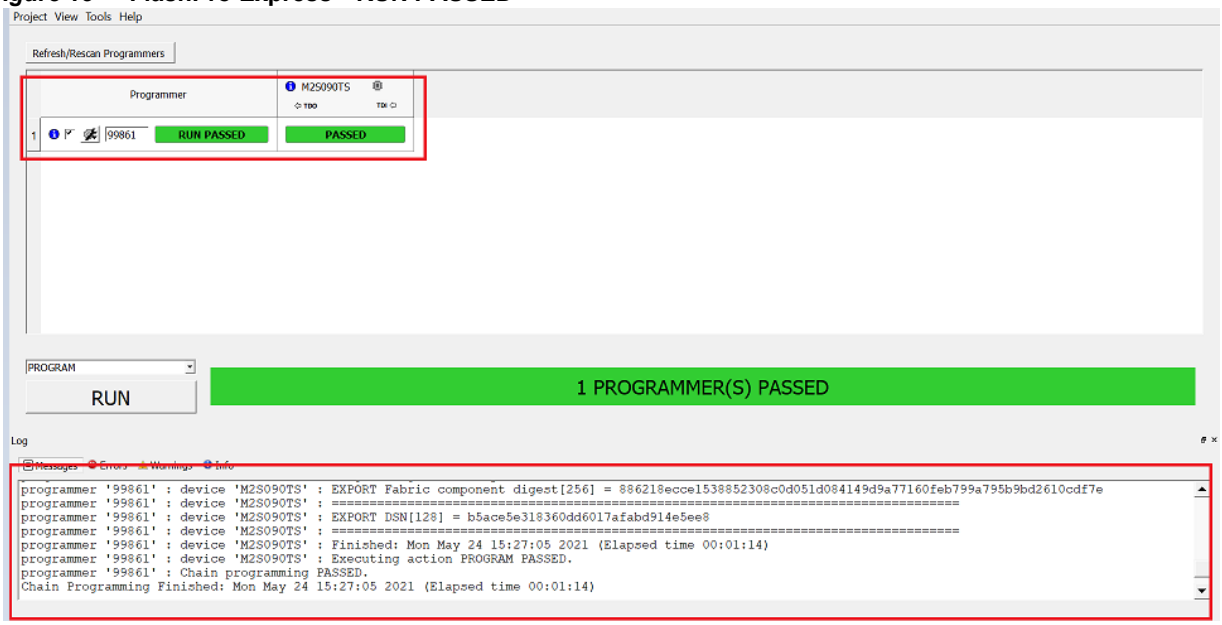


7. Click **OK**. The required programming file is selected and ready to be programmed in the device.
8. The FlashPro Express window appears as shown in Figure 12. Confirm that a programmer number appears in the Programmer field. If it does not, confirm the board connections and click **Refresh/Rescan** Programmers.

Figure 12 • **Programming the Device**

9. Click **RUN**. When the device is programmed successfully, a **RUN PASSED** status is displayed as shown in Figure 13.

Figure 13 • **FlashPro Express—RUN PASSED**



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

5 Appendix 2: eNVM Driver APIs

5.1 write_nvm()

```
static uint32_t
write_nvm
(
uint32_t addr,
const uint8_t * pidata,
uint32_t length,
uint32_t lock_page,
uint32_t * p_status
){
uint32_t length_written;
uint32_t offset;
*p_status = 0u;
offset = addr & NVM_OFFSET_SIGNIFICANT_BITS; /* Ignore remapping. */
ASSERT(offset <= NVM1_TOP_OFFSET);
/* Adjust length to fit within one page. */
length_written = get_remaining_page_length(offset, length);
if(offset <= NVM1_TOP_OFFSET)
{
uint32_t block;
volatile uint32_t ctrl_status;
uint32_t errors;
if(offset < NVM1_BOTTOM_OFFSET)
{
block = NVM_BLOCK_0;
}
else
{
block = NVM_BLOCK_1;
offset = offset - NVM1_BOTTOM_OFFSET;
}
fill_wd_buffer(pidata, length_written, block, offset);
/* Set requested locking option. */
g_nvm[block]->PAGE_LOCK = lock_page;
/* Issue program command */
```

```
g_nvm[block]->CMD = PROG_ADS | (offset & PAGE_ADDR_MASK);
/* Wait for NVM to become ready. */
ctrl_status = wait_nvm_ready(block);
/* Check for errors. */
errors = ctrl_status & WRITE_ERROR_MASK;
if(errors)
{
/* Signal that an error occured by returning 0 a a number of bytes written. */
length_written = 0u;
*p_status = g_nvm[block]->STATUS;
}
else
{
/* Perform a verify. */
g_nvm[block]->CMD = VERIFY_ADS | (offset & PAGE_ADDR_MASK);
/* Wait for NVM to become ready. */
ctrl_status = wait_nvm_ready(block);
/* Check for errors. */
errors = ctrl_status & WRITE_ERROR_MASK;
if(errors)
{
/* Signal that an error occured by returning 0 a a number of bytes written. */
length_written = 0u;
*p_status = g_nvm[block]->STATUS;
}
}
}
return length_written;
}
```

5.2 MSS_NVM_read()

```
nvm_status_t
MSS_NVM_read
(
uint8_t * addr,
uint8_t * podata,
uint32_t len
){
nvm_status_t status = NVM_SUCCESS;
```

```
uint8_t * nvmaddr = 0u;
/* add read offset to read the data */
nvmaddr = ((uint8_t *) (NVM_BASE_ADDRESS + addr));
while((len > 0) && (NVM_SUCCESS == status))
{
len--;
podata[len] = nvmaddr[len];
if((g_nvm[NVM_BLOCK_0]->STATUS & MSS_NVM_ECC2))
status = FAILED;
}
return status;
}
```

5.3 verify_nvm()

```
static uint32_t
verify_nvm
(
uint32_t addr,
const uint8_t * pidata,
uint32_t length,
uint32_t * p_status
){
uint32_t length_verified;
uint32_t offset;
*p_status = 0u;
offset = addr & NVM_OFFSET_SIGNIFICANT_BITS; /* Ignore remapping. */
ASSERT(offset <= NVM1_TOP_OFFSET);
/* Adjust length to fit within one page. */
length_verified = get_remaining_page_length(offset, length);
if(offset <= NVM1_TOP_OFFSET)
{
uint32_t block;
volatile uint32_t ctrl_status;
uint32_t errors;
if(offset < NVM1_BOTTOM_OFFSET)
{
block = NVM_BLOCK_0;
}
else
```

```
{
block = NVM_BLOCK_1;
offset = offset - NVM1_BOTTOM_OFFSET;
}
fill_wd_buffer(pidata, length_verified, block, offset);
/* Perform a verify. */
g_nvm[block]->CMD = VERIFY_ADS | (offset & PAGE_ADDR_MASK);
/* Wait for NVM to become ready. */
ctrl_status = wait_nvm_ready(block);
/* Check for errors. */
errors = ctrl_status & WRITE_ERROR_MASK;
if(errors)
{
/* Signal that an error occured by returning 0 a a number of bytes written. */
length_verified = 0u;
*p_status = g_nvm[block]->STATUS;
}
}
return length_verified;
}
```