
SmartFusion2 SoC FPGA - eNVM Initialization

Table of Contents

Purpose	1
Introduction	1
Design Requirements	2
Initializing eNVM using the Libero eNVM Client	2
Initializing the eNVM Using the Cortex-M3 Processor	4
Design Description	5
Hardware Implementation	5
Software Implementation	7
Setting Up the Design	9
Running the Design	9
Conclusion	12
Appendix A – Design Files	12
Appendix B – eNVM Driver APIs	13
List of Changes	16

Purpose

This application note describes different methods to initialize the embedded nonvolatile memory (eNVM) in SmartFusion[®]2 system-on-chip (SoC) field programmable gate array (FPGA) devices.

Introduction

SmartFusion[®]2 SoC FPGA devices have a maximum of two on chip 256 KB flash memories called eNVM. 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 different methods:

- Using the eNVM client of the eNVM configurator in the Libero[®] System-on-Chip (SoC) v11.3
- Writing into the eNVM using ARM[®] Cortex[™]-M3 processor
- In-application programming (IAP)
- Writing into the eNVM using custom logic in the FPGA fabric

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

Refer to the "eNVM" chapter in [SmartFusion2 Microcontroller Subsystem User Guide](#) for detailed description of eNVM.

Design Requirements

Table 1 • Design Requirements

Design Requirements	Description
Hardware Requirements	
SmartFusion2 Development Kit <ul style="list-style-type: none"> 12 V adapter FlashPro4 programmer USB A to Mini-B cable 	Rev C or later
Host PC or Laptop	Windows 64-bit Operating System
Software Requirements	
Libero SoC for viewing the design files <ul style="list-style-type: none"> FlashPro Programming Software v11.3 	11.3
SoftConsole	3.4
Host PC Drivers	USB to UART drivers
HyperTerminal/Tera Term/PuTTY	

Initializing 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 the microcontroller subsystem (MSS) in the SmartDesign window to open the MSS configurator.
2. Double-click the **eNVM Configurator** to open the **eNVM: Modify core - ENVM** window.

3. Select **Data Storage** under **Available client types** and click **Add to System**, as shown in Figure 1.

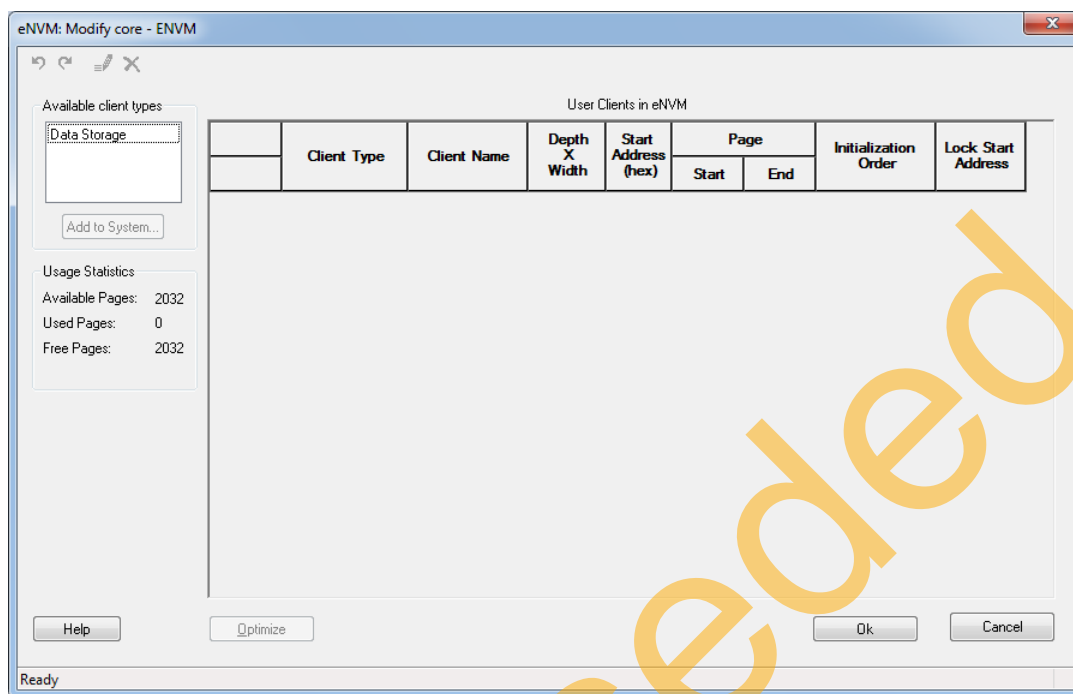


Figure 1 • Adding the Client Type

4. The **Add Data Storage Client** window is displayed, as shown in Figure 2 on page 4. It supports four types of memory file formats:
 - Intel-Hex
 - Motorola-S
 - Actel-Hex
 - Actel-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 3.4 with the linker script "production-execute-in-place.ld".

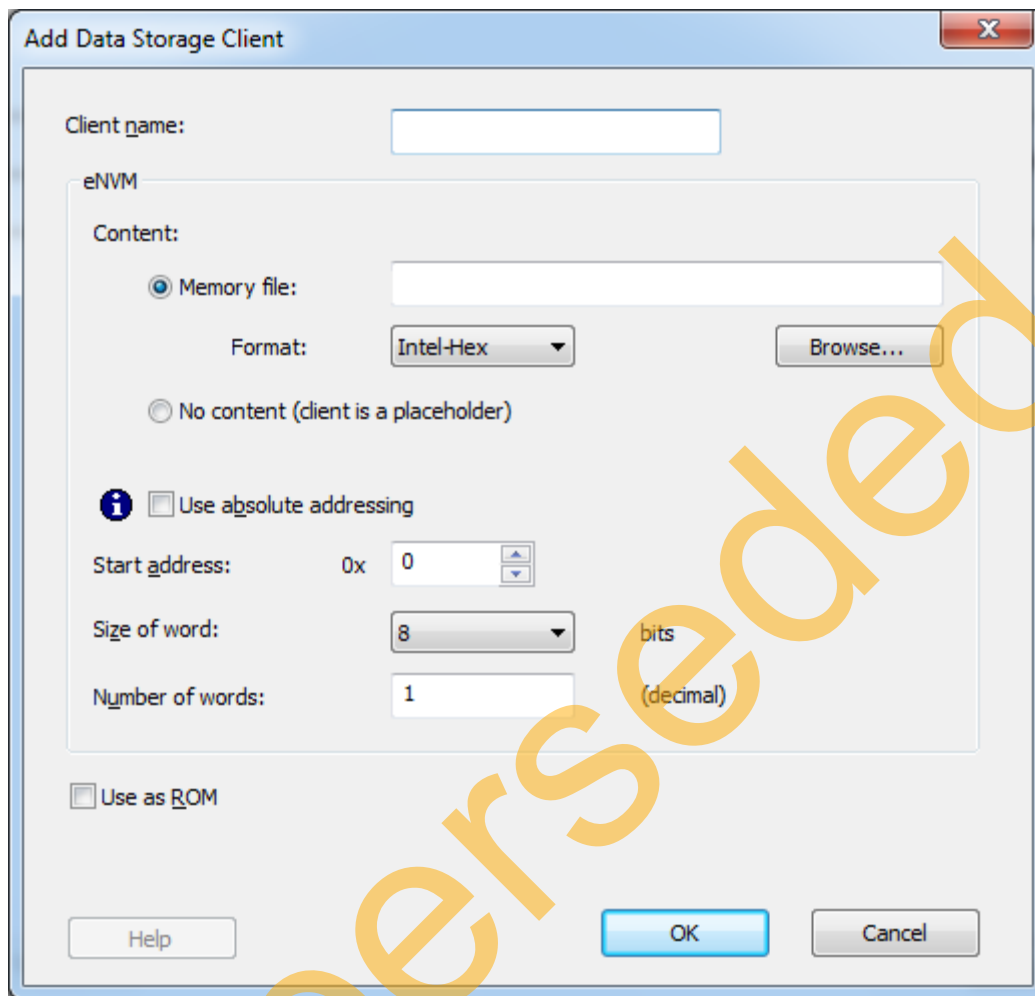


Figure 2 • Add Data Storage Client

5. Enter the **Client name**.
6. Browse to the created **Memory file**, and click **OK** to add the eNVM client.
7. The **Modify core - ENVm** window (displayed next) shows the client and its size. You can also add more than one client with the different start address.
8. After adding the eNVM clients, click **OK**.
9. Save and generate the SmartDesign in Libero using the **Generate Component**.
10. Double-click the **Program Device** in the Libero **Design Flow** window to program the SmartFusion2 SoC FPGA to initialize the eNVM with the memory file.

Initializing the eNVM Using the Cortex-M3 Processor

This application note describes how to initialize eNVM using Cortex-M3 processor with an example design. The design describes how to write, read, and verify the data to or from different locations within the eNVM using the Cortex-M3 processor. The design description and implementation details are described in the following sections.

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 Cortex-M3 processor. The MMUART_0 is routed through FPGA fabric for communicating with the serial terminal program. The design receives the user given commands for read, write, and verify operations and 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.

Hardware Implementation

The hardware implementation involves configuring MSS, Fabric CCC and oscillator. Figure 3 shows the top level hardware design in SmartDesign.

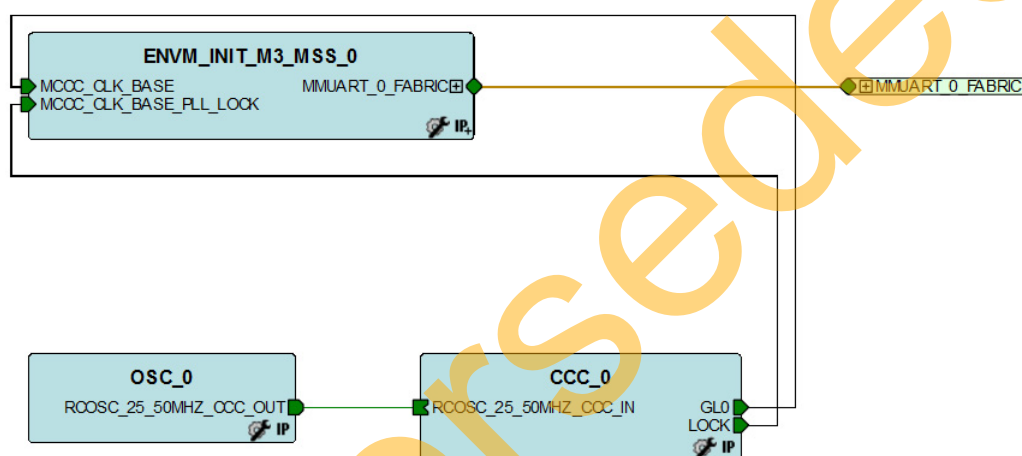


Figure 3 • Top Level SmartDesign

The MSS_CCC clock source is sourced from the FCCC via the CLK_BASE port. The FCCC is configured to provide the 100 MHz clock using GL0. Figure 4 shows the system clocks configurations for the M3_CLK and APB_0_CLK clock settings.

The screenshot shows the 'MSS Clock Conditioning Circuitry Configurator' window. It has two tabs: 'System Clocks' and 'Advanced Options'. Under 'System Clocks', the 'Clock Source' section shows 'CLK_BASE' set to 100 MHz, with a checked box for 'Monitor FPGA Fabric PLL Lock (CLK_BASE_PLL_LOCK)'. The 'Cortex-M3 and MSS Main Clock' section shows 'M3_CLK' set to 100.000 MHz. The 'MDDR Clocks' section shows 'MDDR_CLK' as M3_CLK * 2 and 'DDR_SMC_FIC_CLK' as MDDR_CLK / 1. The 'MSS APB_0/1 Sub-busses Clocks' section shows 'APB_0_CLK' and 'APB_1_CLK' both set to M3_CLK / 1, resulting in 100.000 MHz. There is a 'FPGA Fabric Interface Clocks' section at the bottom.

Figure 4 • Clock Configurations

The MMUART_0 is used for reading and writing to the HyperTerminal window. On the SmartFusion2 Development board, the MMUART_0 TX and RX are connected to the mini-B USB through the fabric and fabric I/Os. Figure 5 shows the MMUART_0 configuration.

The screenshot shows the 'MSS MMUART_0 Configurator' window. It has two tabs: 'Configuration' and 'Advanced Options'. Under 'Configuration', 'Duplex Mode' is set to 'Full Duplex', 'Async/Sync Mode' is set to 'Asynchronous', and 'Use Modem Interface' is unchecked. The 'Assignment' section contains a table with columns: MSIO, Direction, Main Connection, and Package Pin. The table is titled 'Full Duplex' and has two rows: RXD (IN) and TXD (OUT), both with 'Fabric' as the main connection.

MSIO	Direction	Main Connection	Package Pin
Full Duplex			
RXD	IN	Fabric	
TXD	OUT	Fabric	

Figure 5 • MMUART_0 configuration

Software Implementation

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

The design uses following firmware drivers:

- SmartFusion2 MSS MMUART driver:
 - To communicate with the Serial terminal program running on Host PC.

The design implements APIs to read, write, and, verify the data. The API implementation and usage is described in the following sections. Refer to "[Appendix B – eNVM Driver APIs](#)" on page 13 for the API C code.

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 function prototype is shown below.

```
nvm_status_t
NVM_write
(
    uint32_t start_addr,
    const uint8_t * pldata,
    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 *pldata* 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.

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 the 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 eNVM array, write the CMD control register with 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 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.

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 function prototype is shown below.

```
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 to be read. The *podata* parameter is the byte aligned address of the output buffer in which the read data is to be 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.

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 read operation. This API also checks for the 2-bit error while reading eNVM.

Verify Operation

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

```
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.

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 in the eNVM page.

1. Request the 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 verify 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 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.

Setting Up the Design

Connect the following jumpers on the SmartFusion2 SoC FPGA Development Kit, as described in [Table 2](#). While making the jumper connections, the power supply switch SW7 on the board should be in OFF position.

Table 2 • 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

Running the Design

The following steps describe how to run the design:

1. Connect the FlashPro4 programmer to the J59 connector of 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 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.

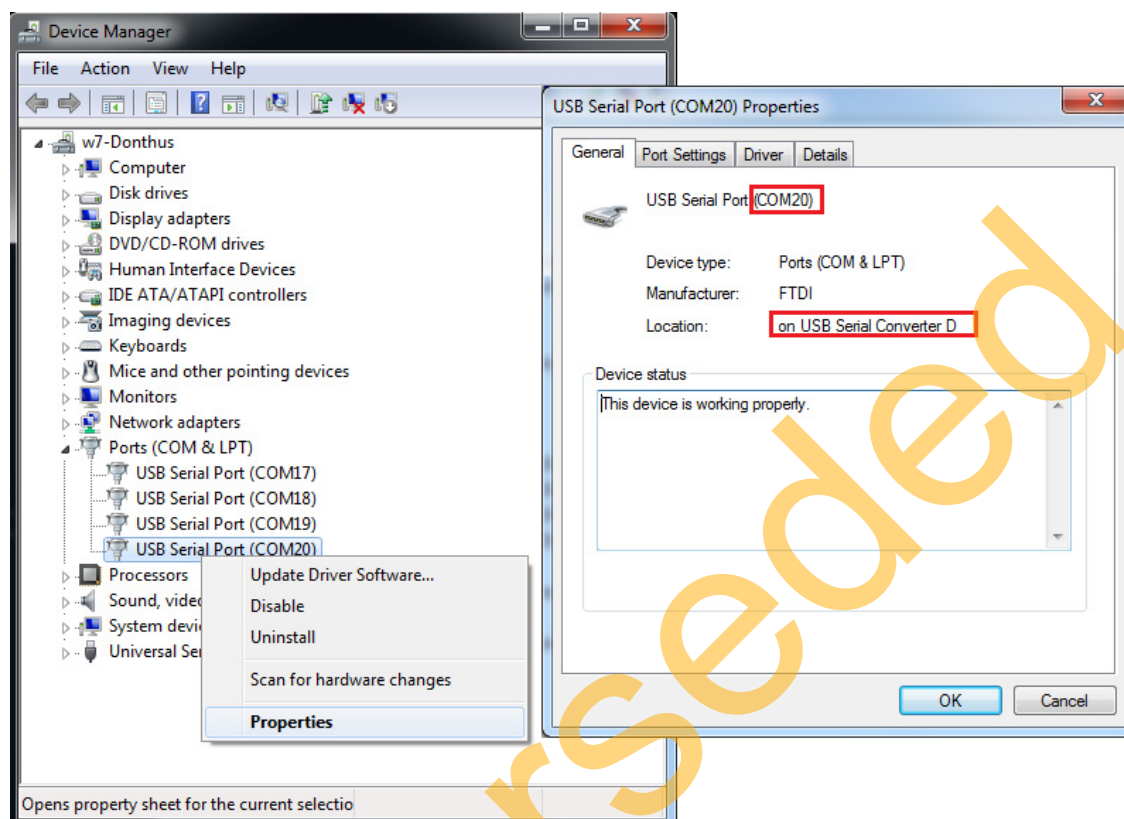


Figure 6 • USB to UART Bridge Drivers

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 J18 connector and change the power supply switch SW7 to ON.
5. Program the SmartFusion2 SoC FPGA Development Kit Board with the generated or provided *.stp file (refer to "Appendix A – Design Files" on page 12) using FlashPro.
6. Invoke the SoftConsole3.4 Integrated Design Environment (IDE) by clicking the Write Application code under Develop Firmware in Libero SoC v11.3 tool and launch the debugger.
7. 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 as PuTTY or Tera Term. Refer to the [Configuring Serial Terminal Emulation Programs Tutorial](#) for configuring HyperTerminal, Tera Term, and PuTTY.
8. When you run the debugger in SoftConsole, HyperTerminal window shows a message to enter your choice.
9. Enter the choice to write. It prompts for address, length, and data consequently. Enter the values, as shown in Figure 7.

10. On writing, it displays the message as "write operation successful".

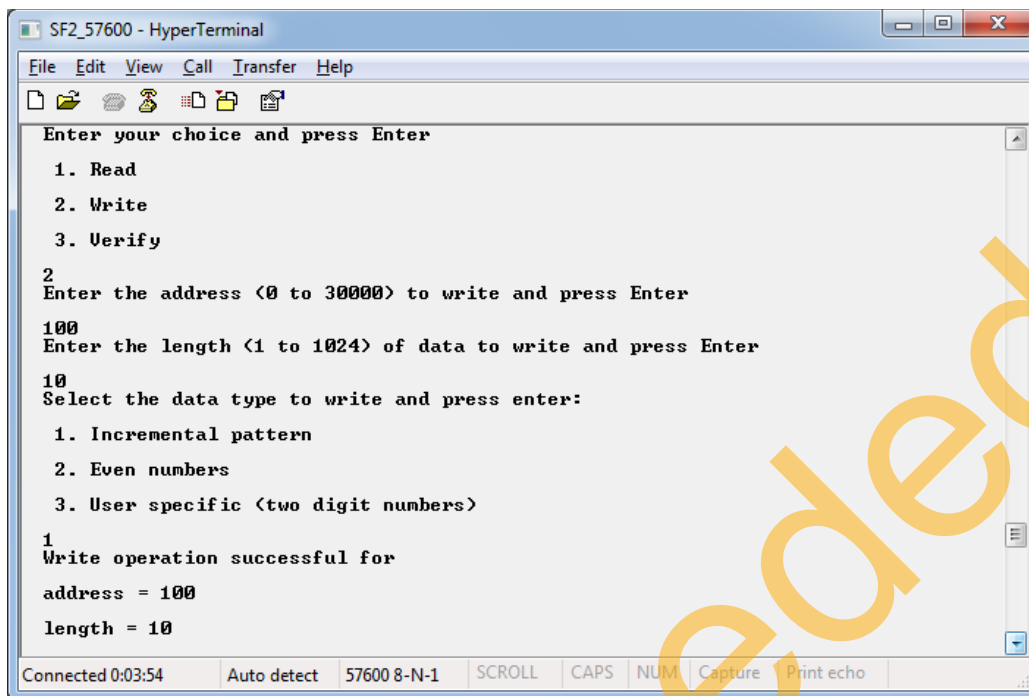


Figure 7 • Write Operation

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

12. On writing, it displays the message as "verify operation successful".

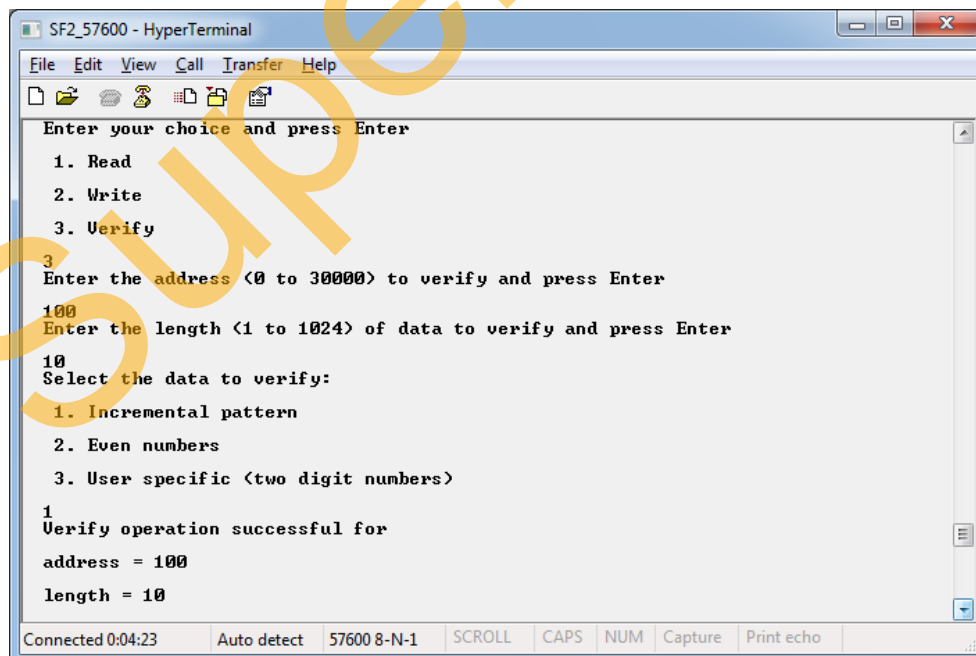
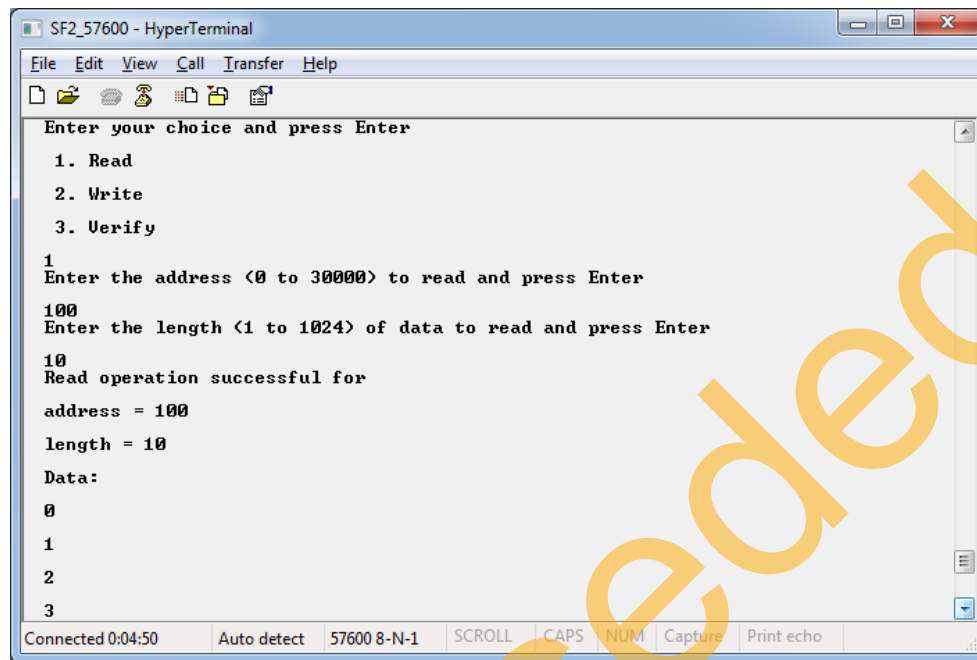


Figure 8 • Verify Operation

13. Enter the choice to read. It prompts for address and length consequently. Enter the values as shown in [Figure 9](#).
14. On reading, it displays all the read values.



```
Enter your choice and press Enter
1. Read
2. Write
3. Verify
1
Enter the address <0 to 30000> to read and press Enter
100
Enter the length <1 to 1024> of data to read and press Enter
10
Read operation successful for
address = 100
length = 10
Data:
0
1
2
3
Connected 0:04:50  Auto detect  57600 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
```

Figure 9 • Read Operation

Conclusion

This application note describes how to initialize the eNVM using the eNVM client of the eNVM configurator in the Libero SoC v11.3 and using the Cortex-M3 processor.

Appendix A – Design Files

You can download the design files from the Microsemi SoC Products Group website: http://soc.microsemi.com/download/rsc/?f=M2S_AC391_DF

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

Appendix B – eNVM Driver APIs

write_nvm()

```
static uint32_t
write_nvm
(
    uint32_t addr,
    const uint8_t * padata,
    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(padata, 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 occurred 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 occurred by returning 0 a a number of bytes written. */
length_written = 0u;
*p_status = g_nvm[block]->STATUS;
}
}

return length_written;
}

```

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 * nvmmaddr = 0u;

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

```

verify_nvm()

```
static uint32_t
verify_nvm
(
    uint32_t addr,
    const uint8_t * pldata,
    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(pldata, 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 occurred by returning 0 a a number of bytes written. */
            length_verified = 0u;
            *p_status = g_nvm[block]->STATUS;
        }
    }
    return length_verified;
}
```

List of Changes

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

Revision*	Changes	Page
Revision 5 (May 2014)	Updated the document for Libero SoC v11.3 software release (SAR 57098).	NA
Revision 4 (November 2013)	Updated the document for Libero SoC v11.2 software release (SAR 52884).	NA
Revision 3 (May 2013)	Updated the document for Libero SoC v11.0 software release (SAR 47576).	NA
Revision 2 (March 2013)	Updated the document for Libero SoC v11.0 beta SP1 software release (SAR 44871).	NA
Revision 1 (November 2012)	Updated the document for Libero SoC v11.0 beta SPA software release (SAR 42847).	NA

Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.

Superseded

Superseded



Microsemi

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.

© 2014 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.