Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer’s responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided “as is, where is” and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) 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. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 4,800 employees globally. Learn more at www.microsemi.com.
Contents

1 Revision History ................................................................. 1
  1.1 Revision 6.0 ........................................................................ 1
  1.2 Revision 5.0 ........................................................................ 1
  1.3 Revision 4.0 ........................................................................ 1
  1.4 Revision 3.0 ........................................................................ 1
  1.5 Revision 2.0 ........................................................................ 1
  1.6 Revision 1.0 ........................................................................ 1

2 Overview .............................................................................. 2
  2.1 Introduction ......................................................................... 2
  2.2 References .......................................................................... 2

3 Configuring SmartFusion2 and IGLOO2 Devices for Safety-Critical Applications . . . 3
  3.1 Protecting MSS Configuration in SmartFusion2 Devices ................. 5
    3.1.1 Using MSS Configurator to Disable Unused MSS Features .......... 6
    3.1.2 How to Set System Registers Write-Protection Bits ..................... 9
    3.1.3 Keeping the Cortex-M3 Processor in Reset ................................. 10
  3.2 Isolating MSS/HPMS/FDDR/SERDESIF Blocks ............................. 13
  3.3 Isolating System Controller ................................................... 14
    3.3.1 Enabling System Controller Suspend Mode Using Libero SoC Software .......... 18
    3.3.2 Reading the State of System Controller Suspend Mode Flash Bit through JTAG ... 20
    3.3.3 Monitoring System Controller Reset Status from FPGA Fabric ........... 22
  3.4 Conclusion ......................................................................... 23

4 Appendix: System Registers Description ..................................... 24
  4.1 Software Reset Control Register (Offset address 0x48) ................... 24
  4.2 Cache Configuration Register (Offset address 0x18) ....................... 26
  4.3 Watchdog Configuration Register (Offset address 0x6C) .................... 26
### Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>SmartFusion2 Architectural Block Diagram</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>IGLOO2 Architectural Block Diagram</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>SmartFusion2 MSS Block Diagram</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4</td>
<td>System Register Schematic</td>
<td>6</td>
</tr>
<tr>
<td>Figure 5</td>
<td>SmartFusion2 MSS Configurator</td>
<td>7</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Disabling Unused MSS Sub blocks</td>
<td>8</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Register Lock Bit Settings</td>
<td>9</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Lock Bit Configuration File</td>
<td>10</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Launching MSS RESET Controller Configurator</td>
<td>11</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Exposing M3_RESET_N to FPGA Fabric</td>
<td>11</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Launching MSS GPIO Configurator</td>
<td>12</td>
</tr>
<tr>
<td>Figure 12</td>
<td>MSS GPIO Configuration</td>
<td>13</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Device Initialization Sequence Performed by System Controller</td>
<td>15</td>
</tr>
<tr>
<td>Figure 14</td>
<td>System Controller Reset Generation Circuitry</td>
<td>16</td>
</tr>
<tr>
<td>Figure 15</td>
<td>System Controller Suspend Mode Activation after Device Initialization</td>
<td>17</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Enabling System Controller Suspend Mode in New Project- Device Settings Page</td>
<td>18</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Enabling System Controller Suspend Mode in Project Settings Window</td>
<td>19</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Selecting Configure Action/Procedures</td>
<td>20</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Selecting DEVICE_INFO Action</td>
<td>21</td>
</tr>
<tr>
<td>Figure 20</td>
<td>System Controller Suspend Mode Status in Reports Window</td>
<td>21</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Catalog Window</td>
<td>22</td>
</tr>
<tr>
<td>Figure 22</td>
<td>System Controller Reset Status Macro</td>
<td>22</td>
</tr>
</tbody>
</table>
# Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Hard IP Block Reset Signals</td>
<td>13</td>
</tr>
<tr>
<td>Table 2</td>
<td>SOFT_RESET_CR</td>
<td>24</td>
</tr>
<tr>
<td>Table 3</td>
<td>CC_CR</td>
<td>26</td>
</tr>
<tr>
<td>Table 4</td>
<td>WDOG_CR</td>
<td>26</td>
</tr>
</tbody>
</table>
1 Revision History

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

1.1 Revision 6.0

The following changes are made in revision 6.0 of this document:

• Title of the section, MSS Configurator to Set System Registers Write-Protection Bits is changed to How to Set System Registers Write-Protection Bits, page 9
• Added content on how to write-protect system registers using Configure Register Lock Bits option available in the Libero design flow. For more information, see How to Set System Registers Write-Protection Bits, page 9

1.2 Revision 5.0

Updated a note in revision 5.0 of this document. For more information, see Isolating MSS/HPMS/FDDR/SERDESIF Blocks, page 13.

1.3 Revision 4.0

Updated a note and added a note in revision 4.0 of this document. For more information, see Isolating MSS/HPMS/FDDR/SERDESIF Blocks, page 13.

1.4 Revision 3.0

Updated the document for Libero SoC v11.5 software release changes in revision 3.0 of this document.

1.5 Revision 2.0

Updated the document for Libero SoC v11.4 software release changes in revision 2.0 of this document.

1.6 Revision 1.0

Revision 1.0 was the first publication of this document.
2 Overview

This application note explains the SmartFusion®2 system-on-chip (SoC) field programmable gate array (FPGA) and IGLOO®2 FPGA features that require customer action for implementing safety-critical designs.

2.1 Introduction

Designers intending to deploy integrated circuits in safety-critical applications need to understand the effects of atmospheric radiation. The single event upset (SEU) phenomenon was first discovered in 1979 by Intel and Bell Labs as failures in dynamic random-access memory (DRAMs). SEU is attributed to stray alpha particles or neutrons changing the contents of storage elements such as registers and memory cells. In aviation applications, the operational altitudes have a higher neutron flux. However, the SEU phenomenon has increasingly become a concern at sea level as well. The continuous drive to smaller node processes reduces the charge at each base cell. Therefore, the likelihood of SEU errors at sea level has increased. These SEU errors are of concern to designers of safety-critical systems.

The next generation Microsemi SmartFusion2 SoC FPGAs and IGLOO2 FPGAs are the only devices that address fundamental requirements for advanced security, high reliability and low power in safety-critical industrial, military, aviation, communications, and medical applications.

2.2 References

Following are the references used in this document:

- UG0331: SmartFusion2 Microcontroller Subsystem User Guide
- UG0451: IGLOO2 and SmartFusion2 Programming User Guide
- UG0448: IGLOO2 FPGA High Performance Memory Subsystem User Guide
- AC393: Board Design Guidelines for SmartFusion2 SoC and IGLOO2 FPGAs Application Note
- Single Event Effects
SmartFusion2 and IGLOO2 devices integrate various hard IP blocks and FPGA fabric on a single die. The key elements of these devices include microcontroller subsystem (MSS) in case of SmartFusion2 devices or high performance memory subsystem (HPMS) in case of IGLOO2 devices, FPGA fabric, double data rate (DDR) memory controllers, and high-speed transceiver lanes. SmartFusion2 and IGLOO2 devices do not require an external configuration device, reducing the component count and improving reliability.

The following image shows the architectural block diagram of SmartFusion2 devices.

Figure 1 • SmartFusion2 Architectural Block Diagram
The SmartFusion2 and IGLOO2 device families address critical high-reliability requirements with the following features:

- SEU immune FPGA fabric configuration
- SEU immune hard IP blocks (MSS/HPMS/FDDR, and SERDESIFs) configuration
- SEU immune I/O configuration
- SEU protected memories
- Hard 667 Mbps DDR2/3 controllers with single error correction and double error detection (SECDED) protection
- Built-in self-test (BIST)

SmartFusion2 and IGLOO2 devices have dedicated flash cells to configure the FPGA fabric, hard IP blocks, and IOMUXes. These flash cells are programmed at the time of device programming and cannot be changed during run-time. The bits programmed into these flash cells are referred to as flash bits throughout the remainder of the document. These flash bits are proven to be SEU immune with zero failure in time (FIT) rate due to the high voltage charge level required to reprogram the flash cell. For more information about SEU, see [Single Event Effects](#) web page.

SmartFusion2 devices include a variety of built-in functionality that can be enabled or disabled to provide the functions required for an application. Designers implementing safety-critical designs using SmartFusion2 SoC FPGAs have the option of disabling certain built-in features if these features are not required in the design.

SmartFusion2 and IGLOO2 devices support write-protection for MSS/HPMS configuration registers, FDDR configuration registers, SERDESIF configuration registers, and MSS IOMUX configuration registers. The write-protection feature enables immunity against SEUs and unintentional writes to the configuration registers.

This application note describes the methodology used to isolate critical functions in SmartFusion2 SoC FPGAs and IGLOO2 FPGAs for design assurance purposes in the safety-critical applications.

**Note:** All unused user I/Os are tri-stated with internal weak pull-up resistors.
3.1 Protecting MSS Configuration in SmartFusion2 Devices

SmartFusion2 devices integrate a hardened MSS. The MSS consists of multiple hard IP blocks such as, ARM Cortex-M3 processor subsystem, embedded memories, direct memory access (DMA) engines, communication peripherals, timers, and DDR memory controller. See UG0331: SmartFusion2 Microcontroller Subsystem User Guide for more information.

The following figure shows the SmartFusion2 MSS block diagram.

**Figure 3 • SmartFusion2 MSS Block Diagram**

The state of the MSS sub-blocks is controlled by system registers (SYSREG). System registers are initialized by the user-configurable flash bits at power-up. After initialization, the system registers’ content can be protected from SEUs or unintended run-time writes by enabling the write-protection feature associated with system registers. The write-protection feature of system registers is set by user-configurable flash bits. Write protection can be assigned to all bits in a register, to a specific field in a register, or to each specific bit in a register.
The following figure shows the system register schematic.

Figure 4 • System Register Schematic

If a write-protection flash bit (DYN_REG[x]) is set, then the associated SYSREG flip-flop is held in continuous reset or preset depending upon its initialization flash bit value (MSS_P[x]). Therefore, even if there is an SEU hit on the SYSREG flip-flop, the SYSREG flip-flop immediately goes back to the initialized flash bit value. See How to Set System Registers Write-Protection Bits, page 9 for more information.

The system registers content can be read back at run-time (using the Cortex-M3 processor or fabric logic) to ensure that system registers are initialized properly and they have not changed over time. It is required to enable at least one of the fabric interface controllers (FIC) to read the SYSREG content from the FPGA fabric.

See the Fabric Interface Controller chapter of UG0331: SmartFusion2 Microcontroller Subsystem User Guide for more information on how to interface a fabric master to MSS to read the SYSREG address space.

Note: The complete MSS can be reliably disabled prior to device operation. The status of the MSS can be monitored using FPGA fabric logic to verify that the MSS remains disabled. When MSS_RESET_N_F2M is tied to 0, the MSS is held in reset and the state of the MSS can be monitored by polling MSS_RESET_N_M2F signal from the FPGA fabric. The MSS_RESET_N_M2F signal gets asserted (drives 0), when the MSS is held in reset.

3.1.1 Using MSS Configurator to Disable Unused MSS Features

If the MSS is used in a system design, it is still possible to disable the various sub-blocks that are not used. The MSS sub-blocks can be held in reset from power-up by configuring the flash bits associated with the following system registers — SOFT_RESET_CR, CC_CR, and WDOG_CR. It is also required to enable the write protection to protect the configuration from SEUs or unintended writes.


The SmartFusion2 MSS Configurator allows enabling or disabling and configuring of each MSS sub-block according to the user’s application requirements.

The following figure shows the SmartFusion2 MSS Configurator window.
The SmartFusion2 MSS Configurator allows enabling or disabling of the following sub-blocks:

- Cache Controller
- MDDR Controller
- USB Controller
- Ethernet MAC
- MMUART_0 and MMUART_1
- SPI_0 and SPI_1
- I2C_0 and I2C_1
- PDMA
- CAN
- Watchdog Timer
- GPIO
- FIC_0 and FIC_1
The following figure shows how to disable the MSS sub-blocks by selecting the check box provided in the lower-right corner of the sub-blocks. At the time of device programming, Libero software programs the SOFT_RESET_CR system register with the appropriate values based on the enable or disable configuration in the MSS Configurator.

The following figure shows how to disable unused MSS sub-blocks.

**Figure 6 • Disabling Unused MSS Sub blocks**

Disabling a sub-block causes it to be held in reset when the device is powered up. While most of the MSS sub-block states can be configured through MSS configurator, the following blocks are not provided in MSS configurator to select their state:

- HPDMA
- Timers
- COMM_BLK
- Embedded memories (eNVM_0, eNVM_1, eSRAM_0, eSRAM_1)

By default, the Libero SoC software configures high-performance DMA (HPDMA) and Timers to be held in reset using appropriate flash bits, so no action is required to disable them. The appropriate system registers have to be configured to bring HPDMA and Timers out of reset. COMM_BLK and embedded memories are active from power-up if MSS is used in the design. The eNVM cannot be held in reset from power-up as the system controller needs to perform eNVM self-test during device initialization.

**Note:** There is no flash bit to disable the real time clock (RTC). Microsemi recommends masking the RTC wakeup interrupt to the Cortex-M3 processor and FPGA fabric by un-checking the RTC sub-block in the MSS configurator. RTC wakeup interrupt masking is done through flash bits, and can be protected from SEUs by enabling write protection to the RTC_WAKEUP_CR system register.
3.1.2 How to Set System Registers Write-Protection Bits

The **Configure Register Lock Bits** option available in the Libero Design Flow is used to write-protect or lock the MSS, SERDES, and FDDR system registers to prevent them from being overwritten by masters that have access to these registers. Register write-protection bits or lock bits are set in a text (*.txt) file, which is then imported into the SmartFusion2 project. From the **Design Flow** window, click **Configure Register Lock Bits** option to open the configurator. Click **Browse...** to navigate to the text file (*.txt) that contains the register lock bits settings (see the following figure). The Libero SoC software generates the configuration data for write-protection flash bits based on user inputs. These flash bits are programmed at the time of device programming.

**Figure 7**  Register Lock Bit Settings

![Register Lock Bit Settings](image)

### 3.1.2.1 Lock Bit File

An initial, default lock bit file can be generated by clicking **Generate FPGA Array Data** in the **Design Flow** window.

The default file located at `<proj_location>/designer/<root>/<root>_init_config_lock_bits.txt` can be used to make the required changes.

**Note:** Save the file using a different name if you modify the text file to set the lock bits.

### 3.1.2.2 Lock Bit File Syntax

A valid entry in the lock bit configuration file is defined as a `<lock_parameters> < lock bit value>` pair format.

The lock parameters are structured as follows:

- Lock bits syntax for a register: `<Physical block name>_<register name>_LOCK`
- Lock bits syntax for a specific field: `<Physical block name>_<register name>_<field name>_LOCK`
- The following are the physical block names (varies with device family and die):
  - MSS
  - FDDR
  - SERDES_IF_x (where x is 0,1,2,3 to indicate the physical SERDES location) for SmartFusion2 M2S010/025/050/150 devices
  - SERDES_IF2 for SmartFusion2 M2S060/090 devices (only one SERDES block per device)
- Set the lock bit value to 1 to indicate that the register can be written to (unlocked) and to 0 to indicate that the register cannot be written to (locked)

**Note:** Lines starting with # or ; are comments. Empty lines are allowed in the lock bit configuration file.
The following figure shows the lock bit configuration file.

**Figure 8 • Lock Bit Configuration File**

```plaintext
#Register Lock Bits Configuration File for MSS, SERDES(s) and Fabric DDR
#Microsemi Corporation - Microsemi Libero Software Release v11.7 SP1 (Version 11.7.1.2)
#Date: Tue Mar 29 13:24:34 2016

# sb_sh_0/shb_sh_MSS_0/MSS_ADLIB_INST/INST_MSS_050_TLP
MSS_ESRAM_CONFIG_LOCK 0
MSS_ESRAM_MAXLAT_LOCK 1
MSS_DDR_CONFIG_LOCK 1
MSS_ENVN_CONFIG_LOCK 0
MSS_ENVN_REMAP_BASE_LOCK 1
MSS_ENVN_FAB_REMAP_LOCK 1
MSS_CC_FEATURE_LOCK 0
MSS_CC_CACHE_REGION_LOCK 1
MSS_CC_LOCKBASEADDR_LOCK 1
MSS_CC_FLUSHINDEX_LOCK 0
MSS_DDR_BUF_TIMER_LOCK 1
MSS_DDR_NR_ADR_LOCK 1
MSS_DDR_NR_SIZE_LOCK 0
MSS_DDR_CONFIG_LOCK 1
MSS_EDAC_ENABLE_LOCK 1
MSS_MASTER_WEIGHT_CONFIG_LOCK 1
MSS_MASTER_WEIGHT_CONFIGI_LOCK 1
MSS_SOFT_INTERRUPT_LOCK 1
MSS_SOFTRESET_ENV0_SOFTRESET_LOCK 1
MSS_SOFTRESET_ENV1_SOFTRESET_LOCK 1
MSS_SOFTRESET_ESRAM0_SOFTRESET_LOCK 1
MSS_SOFTRESET_ESRAM1_SOFTRESET_LOCK 1
MSS_SOFTRESET_MAC_SOFTRESET_LOCK 1
MSS_SOFTRESET_PDMA_SOFTRESET_LOCK 1
MSS_SOFTRESET_TIMER_SOFTRESET_LOCK 1
MSS_SOFTRESET_MMUART0_SOFTRESET_LOCK 1
MSS_SOFTRESET_MMUART1_SOFTRESET_LOCK 1
MSS_SOFTRESET_GSPI0_SOFTRESET_LOCK 1
MSS_SOFTRESET_GSPI1_SOFTRESET_LOCK 1
MSS_SOFTRESET_I2C0_SOFTRESET_LOCK 1
MSS_SOFTRESET_I2C1_SOFTRESET_LOCK 1
MSS_SOFTRESET_CAN_SOFTRESET_LOCK 1
MSS_SOFTRESET_USB_SOFTRESET_LOCK 1
MSS_SOFTRESET_COMBLK_SOFTRESET_LOCK 1
MSS_SOFTRESET_FPGA_SOFTRESET_LOCK 1
MSS_SOFTRESET_DPMA_SOFTRESET_LOCK 1
MSS_SOFTRESET_FIC32_0_SOFTRESET_LOCK 1
MSS_SOFTRESET_FIC32_1_SOFTRESET_LOCK 1
MSS_SOFTRESET_MSS_GPIO_SOFTRESET_LOCK 1
MSS_SOFTRESET_MSS_GPIO_7_0_SOFTRESET_LOCK 1
MSS_SOFTRESET_MSS_GPIO_15_8_SOFT_RESET_LOCK 1
MSS_SOFTRESET_MSS_GPIO_23_16_SOFT_RESET_LOCK 1
MSS_SOFTRESET_MSS_GPIO_31_24_SOFT_RESET_LOCK 1
MSS_SOFTRESET_MDDR_CTRL_SOFTRESET_LOCK 1
MSS_SOFTRESET_MDDR_FIC64_SOFTRESET_LOCK 1
MSS_M_CONFIG_LOCK 1
```

### 3.1.2.3 Locking and Unlocking a Register

A register can be locked or unlocked by setting the appropriate lock bit value in the lock bit configuration.txt file.

1. Browse to locate the lock bit configuration.txt file.
2. Do one or both of the following:
   - Set the lock bit value to 0 for the registers you want to lock.
   - Set the lock bit value to 1 for the registers you want to unlock.
3. Save the file, and import the file into the project (Design Flow window > Configure Register Lock Bits), see Figure 7, page 9.
4. Regenerate the bitstream.

See the System Register Block chapter of **UG0331: SmartFusion2 Microcontroller Subsystem User Guide** for more information on system registers.

### 3.1.3 Keeping the Cortex-M3 Processor in Reset

In SmartFusion2 devices, the system controller keeps the Cortex-M3 processor in reset during device initialization. The state of the Cortex-M3 processor after device initialization is defined by the M3_RESET_N signal. The Cortex-M3 processor can be held in reset while the FPGA is powered by tying the M3_RESET_N signal to 0. The M3_RESET_N signal must be exposed to the FPGA fabric to tie it to
0. Tying the M3_RESET_N signal to ‘0’ in the fabric means that the M3_RESET_N signal is directly driven from a flash bit and immune to SEUs.

The reset controller block in the MSS configurator GUI allows the user to access the M3_RESET_N signal via the FPGA fabric.

Double-click the RESET controller sub-block (Figure 9, page 11) of the MSS configurator to launch MSS RESET controller configurator.

**Figure 9 • Launching MSS RESET Controller Configurator**

The following figure shows how to configure the RESET controller sub-block to expose the M3_RESET_N signal to the FPGA fabric.

**Figure 10 • Exposing M3_RESET_N to FPGA Fabric**
The following steps describe how to monitor the state of the Cortex-M3 processor from the FPGA fabric using GPIOs:

1. In the Libero project, double-click the GPIO sub-block (see, Figure 9, page 11) of the MSS configurator to launch the MSS GPIO configurator.

Figure 11 • Launching MSS GPIO Configurator

2. In the MSS GPIO configurator, configure several MSS GPIOs as outputs by selecting Output as Direction from the drop-down list, as shown in Figure 12, page 13.

3. Promote the configured MSS GPIOs to FPGA fabric by selecting Fabric_A from the drop-down list, as shown in Figure 12, page 13.
4. Build a Cortex-M3 executable application program to drive a known pattern on these MSS GPIOs. For more information about how to build an Cortex-M3 executable application program using Microsemi tools, see **TU0546: SoftConsole v4.0 and Libero SoC v11.7 Tutorial**.

5. Add a data storage client to the eNVM with the above MSS GPIO application program as its content. This application program is automatically executed when the Cortex-M3 processor is out of reset. For information about how to add a eNVM data storage client, see **AC426: Implementing Production Release Mode Programming for SmartFusion2 Application Note**.

6. During device run-time, monitor these MSS GPIOs from FPGA fabric logic. If any activity is detected on these MSS GPIOs, it indicates that the Cortex-M3 processor is not in reset, and the design must take action as required.

### 3.2 Isolating MSS/HPMS/FDDR/SERDESIF Blocks

The unused hard IP blocks (MSS in SmartFusion2, HPMS in IGLOO2, FDDR, and SERDESIFs) can be held in reset to isolate them from rest of the system. If any of the hard IP blocks are not used in a design, then Libero SoC software automatically configures them to be in reset by asserting appropriate control signals as tabulated in the following table. These control signals are directly driven from the flash bits, which are immune to SEUs.

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Reset Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSS (SmartFusion2)/HPMS (IGLOO2)</td>
<td>M3_RESET_N, M2F_RESET_N, M3F_RESET_N, MDDR_DDR_CORE_RESET_N, MDDR_APB_S_RESET_N</td>
</tr>
<tr>
<td>FDDR</td>
<td>FDDR_CORE_RESET_N, FDDR_APB_S_PRESET_N</td>
</tr>
<tr>
<td>SERDESIF</td>
<td>SERDESIFx_CORE_RESET_N, SERDESIFx_PHY_RESET_N, SERDESIFx_APB_S_PRESET_N, SERDESIFx_EPCS_0/1_RESET_N</td>
</tr>
</tbody>
</table>
Note: The status of the MSS/HPMS can be monitored using FPGA fabric logic to verify that the MSS or HPMS remain disabled. When MSS_RESET_N_F2M/HPMS_RESET_N_F2M is tied to 0, the MSS/HPMS is held in reset and the state of the MSS or HPMS can be monitored by polling the MSS_RESET_N_M2F/HPMS_RESET_N_M2F signal from the FPGA fabric. The MSS_RESET_N_M2F/HPMS_RESET_N_M2F signal gets asserted (drives to 0), when the MSS/HPMS is held in reset.

Note: In IGLOO2 devices, the Cortex-M3 processor is held in reset by a factory-programmed SEU immune flash bit. If you are using an IGLOO2 device and have requirements to monitor the Cortex-M3 being held in reset then the only way to accomplish this is to actually use a SmartFusion2 device and follow the steps described at Keeping the Cortex-M3 Processor in Reset, page 10. SmartFusion2 and IGLOO2 devices are pin compatible.

Note: In cases where certain interfaces are not used, the associated pins need to be configured properly, see AC393: Board Design Guidelines for SmartFusion2 SoC and IGLOO2 FPGAs Application Note for more information.

Note: Unused Fabric phase-locked loops (PLLs) are configured to be held in power-down mode by the Libero SoC software.

3.3 Isolating System Controller

In SmartFusion2 and IGLOO2 devices, the system controller manages device initialization, programming operations, and handles the system service requests. After power-on-reset or device reset (DEVRST_N) events, the system controller performs the initialization sequence of the I/O banks, FPGA fabric, and MSS or HPMS.
The following image shows the flow chart of sequence of operations performed by the system controller.

**Figure 13 • Device Initialization Sequence Performed by System Controller**
The system controller can be held in suspend mode after the completion of device initialization to protect the device from unintended device programming or zeroization of the device due to SEUs.

The system controller suspend mode is designed to provide an SEU immune reset state for the system controller. The system controller reset generation circuitry is designed with a triple modular redundancy (TMR) self-refreshing latch to provide SEU immunity. In this mode, the system controller is held in reset while its output ports to the rest of the system are forced to known and well-determined states.

System controller suspend mode is controlled by a flash bit (SC_SUSPEND_MODE_DISABLE), which is set during device programming, and is not accessible either by external pin or from within the design. It is only accessed by the programming file loaded into the device, during programming. Since the SC_SUSPEND_MODE_DISABLE control bit is stored in a flash cell, it is immune to SEUs.

- If SC_SUSPEND_MODE_DISABLE = 1, the system controller suspend mode is disabled.
- If SC_SUSPEND_MODE_DISABLE = 0, the system controller suspend mode is enabled.

In system controller suspend mode, the system controller is held in a reset state and cannot provide any system services such as Flash*Freeze service, cryptographic services, or programming services.

The following image shows the system controller reset generation circuitry.

Figure 14 • System Controller Reset Generation Circuitry
The following image shows the activation sequence for the system controller suspend mode.

**Figure 15 • System Controller Suspend Mode Activation after Device Initialization**

The system controller becomes active if the device is power-cycled or if a hard reset (DEVRST_N) is applied, but it returns to suspend mode after the initialization cycle is completed. To restore normal operation, the device must be reprogrammed with the system controller suspend mode turned off (SC_SUSPEND_MODE_DISABLE = 1).

After the device has entered the suspend mode, the system controller is held in reset and cannot respond to the reprogramming requests. To facilitate reprogramming of the device, the JTAG_TRST_N pin is used to gate the internal FORCE signal and releases the system controller from reset. In an avionics environment, JTAG_TRST_N must be held asserted to prevent JTAG circuitry from affecting the I/Os due to SEUs. Releasing JTAG_TRST_N puts the system controller out of reset and allows the device to be reprogrammed.

When a programming mode instruction is loaded, the system controller sends a pulse on SUSPEND_MODE_RESET_N to clear the TMR latch so that the device can re-execute a normal boot sequence after programming is completed.

Reprogramming via the system controller SPI (SC_SPI) interface is also possible. However, JTAG_TRST_N must be controlled by the external host.

As shown in Figure 14, page 16, the state of the system controller can be monitored by the FPGA fabric logic by reading the state of the SYSCTRL_RESET_STATUS signal.

- If SYSCTRL_RESET_STATUS = 1, the system controller suspend mode is enabled.
- If SYSCTRL_RESET_STATUS = 0, the system controller suspend mode is disabled.

**Note:** The SYSCTRL_RESET_STATUS signal is available in all SmartFusion2 and IGLOO2 devices except M2S050 and M2GL050 variants.

For information on system controller reset status monitoring, see Monitoring System Controller Reset Status from FPGA Fabric, page 22.

**Note:** To keep the system controller in suspend mode, the FLASH_GOLDEN_N pin must be tied to high at power-up or during assertion of the device reset pin (DEVRST_N). If the FLASH_GOLDEN_N pin is set to low at power-up or during assertion of the DEVRST_N pin, the device enters the auto-programming mode. For more information on auto-programming mode, see UG0451: IGLOO2 and SmartFusion2 Programming User Guide.
3.3.1 Enabling System Controller Suspend Mode Using Libero SoC Software

The system controller suspend mode feature can be configured (enabled/disabled) using the Libero SoC software in the following two ways:

- At the time of project creation, select the System Controller Suspended Mode check box in the New Project- Device Settings page as shown in the following figure.

*Figure 16 • Enabling System Controller Suspend Mode in New Project- Device Settings Page*
In an existing project, the system controller suspend mode can be enabled by selecting the **System Controller Suspended Mode** check box under **Device Settings** of the **Project Settings** window as shown in the following figure.

**Figure 17** • Enabling System Controller Suspend Mode in Project Settings Window

The programming file generated with this configuration sets the suspend mode flash bit (SC_SUSPEND_MODE_DISABLE = 0) when the device is programmed.
3.3.2 Reading the State of System Controller Suspend Mode Flash Bit through JTAG

The state of the system controller suspend mode flash bit can be read through JTAG by running DEVICE_INFO programming action using the Libero SoC software. Following are the steps to run the DEVICE_INFO programming action:

1. Right-click Run PROGRAM Action in the Design Flow tab of Libero SoC software, and select Configure Action/Procedures, as shown in the following figure.

Figure 18 • Selecting Configure Action/Procedures
2. In the **Select Action and Procedures** dialog box, select **DEVICE_INFO** as Action, and click **OK**, as shown in the following figure.

*Figure 19 • Selecting DEVICE_INFO Action*

3. Right-click **Run PROGRAM Action** in the **Design Flow** tab and select **Run**. After the successful completion of this action, the system controller suspend mode status is displayed in the **Reports** window, as shown in the following figure.

*Figure 20 • System Controller Suspend Mode Status in Reports Window*
3.3.3 Monitoring System Controller Reset Status from FPGA Fabric

At run-time, the status of the system controller can be monitored from the FPGA fabric using System Controller Reset Status (SYSCTRL_RESET_STATUS) macro.

To monitor the system controller reset status, drag-and-drop the SYSCTRL_RESET_STATUS macro from the Catalog window on to Libero SoC software canvas, as shown in Figure 21, page 22 and Figure 22, page 22.

The RESET_STATUS port from the macro can be connected to the fabric logic to monitor the system controller reset status.

- If RESET_STATUS = 1, the system controller suspend mode is enabled.
- If RESET_STATUS = 0, the system controller suspend mode is disabled.

*Figure 21* • Catalog Window

*Figure 22* • System Controller Reset Status Macro
3.4 Conclusion

SmartFusion2 SoC FPGAs and IGLOO2 FPGAs provide several advantages to the designers of safety-critical systems. The inherent immunity of the flash technology to terrestrial and atmospheric radiation effects makes the parts suitable for safety-critical applications in industrial control, medical, aviation, and military applications. The high reliability of tried-and-trusted 65 nm flash brings more assurance. Flexibility of SmartFusion2 and IGLOO2 devices comes from its embedded system controller and hard IP blocks. To fully protect the device from radiation induced faults, this application note describes the specific actions that must be taken to avoid unintentional reprogramming or unintentional activity in the hard IP blocks.
4 Appendix: System Registers Description

The SYSREG block is located at address 0x40038000 in the SmartFusion2 system memory map.

4.1 Software Reset Control Register (Offset address 0x48)

The following table shows the bit definitions of the SOFT_RESET_CR system registers.

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Name</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:27]</td>
<td>Reserved</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>MDDR_FIC64_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases DDR_FIC controller from reset. 1: Keeps DDR_FIC controller in reset.</td>
</tr>
<tr>
<td>25</td>
<td>MDDR_CTLR_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases MDDR controller from reset. 1: Keeps MDDR controller in reset.</td>
</tr>
<tr>
<td>21</td>
<td>MSS_GPOUT_7_0_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases GPIO_OUT[7:0] from reset. 1: Keeps GPIO_OUT[7:0] in reset.</td>
</tr>
<tr>
<td>20</td>
<td>MSS_GPIO_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases the GPIO from reset, as long as it isn't being held in reset by some other means. 1: Keeps the GPIO to be held in reset. Asserting this soft reset bit holds APB register, GPIO input, interrupt generation logic in reset. GPIO OUT logic is not affected by this reset.</td>
</tr>
<tr>
<td>19</td>
<td>FIC32_1_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases FIC_1 from reset. 1: Keeps FIC_1 in reset.</td>
</tr>
<tr>
<td>18</td>
<td>FIC32_0_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases FIC_0 from reset. 1: Keeps FIC_0 in reset.</td>
</tr>
<tr>
<td>17</td>
<td>HPDMA_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases HPDMA from reset. 1: Keeps HPDMA in reset.</td>
</tr>
<tr>
<td>16</td>
<td>FPGA_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases FPGA from reset. 1: Keeps FPGA in reset.</td>
</tr>
<tr>
<td>15</td>
<td>COMBLK_SOFTRESET</td>
<td>0</td>
<td>0: Releases COMM_BLK from reset. 1: Keeps COMMUNICATION BLOCK (COMM_BLK) in reset.</td>
</tr>
<tr>
<td>14</td>
<td>USB_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases USB from reset. 1: Keeps USB in reset.</td>
</tr>
<tr>
<td>Bit Number</td>
<td>Name</td>
<td>Reset Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>-------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>13</td>
<td>CAN_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases CAN from reset. 1: Keeps CAN in reset.</td>
</tr>
<tr>
<td>12</td>
<td>I2C1_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases I2C_1 from reset. 1: Keeps I2C_1 in reset.</td>
</tr>
<tr>
<td>11</td>
<td>I2C0_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases I2C_0 from reset. 1: Keeps I2C_0 in reset.</td>
</tr>
<tr>
<td>10</td>
<td>SPI1_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases SPI1 from reset. 1: Keeps SPI1 in reset.</td>
</tr>
<tr>
<td>9</td>
<td>SPI0_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases SPI0 from reset. 1: Keeps SPI0 in reset.</td>
</tr>
<tr>
<td>8</td>
<td>MMUART1_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases MMUART_1 from reset. 1: Keeps MMUART_1 in reset.</td>
</tr>
<tr>
<td>7</td>
<td>MMUART0_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases MMUART_0 from reset. 1: Keeps MMUART_0 in reset.</td>
</tr>
<tr>
<td>6</td>
<td>TIMER_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases the system timer from reset. 1: Keeps the system timer in reset.</td>
</tr>
<tr>
<td>5</td>
<td>PDMA_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases the PDMA from reset. 1: Keeps the PDMA in reset.</td>
</tr>
<tr>
<td>4</td>
<td>MAC_SOFTRESET</td>
<td>0x1</td>
<td>0: Releases the Ethernet MAC from reset. 1: Keeps the Ethernet MAC in reset.</td>
</tr>
<tr>
<td>3</td>
<td>ESRAM1_SOFTRESET</td>
<td>0</td>
<td>0: Releases the ESRAM_1 memory controller from reset. 1: Keeps the ESRAM_1 memory controller in reset.</td>
</tr>
<tr>
<td>2</td>
<td>ESRAM0_SOFTRESET</td>
<td>0</td>
<td>0: Releases the ESRAM_0 memory controller from reset. 1: Keeps the ESRAM_0 memory controller in reset.</td>
</tr>
<tr>
<td>1</td>
<td>ENVM1_SOFTRESET</td>
<td>0</td>
<td>0: Releases the ENVM_1 memory controller from reset. 1: Keeps the ENVM_1 memory controller in reset.</td>
</tr>
<tr>
<td>0</td>
<td>ENVM0_SOFTRESET</td>
<td>0</td>
<td>0: Releases the ENVM_0 memory controller from reset. 1: Keeps the ENVM_0 memory controller in reset.</td>
</tr>
</tbody>
</table>


4.2 Cache Configuration Register (Offset address 0x18)

The following table shows the bit definitions of the CC_CR system registers.

Table 3 • CC_CR

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Name</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:3]</td>
<td>Reserved</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>CC_CACHE_LOCK</td>
<td>0</td>
<td>This signal allows the cache lock to be enabled. The allowed values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Cache lock disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Cache lock enabled</td>
</tr>
<tr>
<td>1</td>
<td>CC_SBUS_WR_MODE</td>
<td>0</td>
<td>This signal allows debug mode SBUS writes to cache memory to be enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The allowed values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Debug mode SBUS write disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Debug mode SBUS write enabled</td>
</tr>
<tr>
<td>0</td>
<td>CC_CACHE_ENB</td>
<td>0</td>
<td>This signal allows the cache to be disabled. The allowed values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Cache disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Cache enabled</td>
</tr>
</tbody>
</table>

4.3 Watchdog Configuration Register (Offset address 0x6C)

The following table shows the bit definitions of the WDOG_CR system registers.

Table 4 • WDOG_CR

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Name</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:2]</td>
<td>Reserved</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>WDOGMODE</td>
<td>0</td>
<td>This bit is the reset/interrupt mode selection bit from the system register. This value can be read from the WDOGCONTROL register within the WatchDog module.</td>
</tr>
<tr>
<td>0</td>
<td>WDOGENABLE</td>
<td>0</td>
<td>This is the enable bit for the Watchdog module. The status of this bit can be monitored in the WDOGENABLE register within the WatchDog module.</td>
</tr>
</tbody>
</table>