

UG0837
User Guide
IGLOO2 and SmartFusion2 FPGA
System Services Simulation

June 2018



Contents

1	Revision History	1
1.1	Revision 1.0	1
2	IGLOO2 and SmartFusion2 FPGA System Services Simulation	2
2.1	Types of Available System Services	3
2.2	IGLOO2 System Service Simulation	3
2.3	SmartFusion2 System Service Simulation	4
2.4	Simulation Examples	5
2.5	IGLOO2 Serial Number Service Simulation	5
2.6	SmartFusion2 Serial Number Service Simulation	8
2.7	IGLOO2 Zeroization Service Simulation	13
2.8	SmartFusion2 Zeroization Service Simulation	16
3	Appendix: Types Of System Services	19
3.1	Simulation Message Services	19
3.1.1	Flash*Freeze	19
3.1.2	Zeroization	19
3.2	Data Pointer Services	19
3.2.1	Serial Number	19
3.2.2	Usercode	19
3.3	Data Descriptor Services	19
3.3.1	AES	20
3.3.2	SHA 256	20
3.3.3	HMAC	20
3.3.4	DRBG Generate	20
3.3.5	DRBG Reset	20
3.3.6	DRBG Self Test	21
3.3.7	DRBG Instantiate	21
3.3.8	DRBG Uninstantiate	21
3.3.9	DRBG Reseed	21
3.3.10	KeyTree	21
3.3.11	Challenge Response	21
3.4	Other Services	22
3.4.1	Digest Check	22
3.4.2	Unrecognized Command Response	22
3.4.3	Unsupported Services	22
3.5	System Services Simulation Support File	22
3.5.1	Forcing Error Responses	22
3.5.2	Parameter Setting	23

3.5.3 Device Priority 23

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 1.0

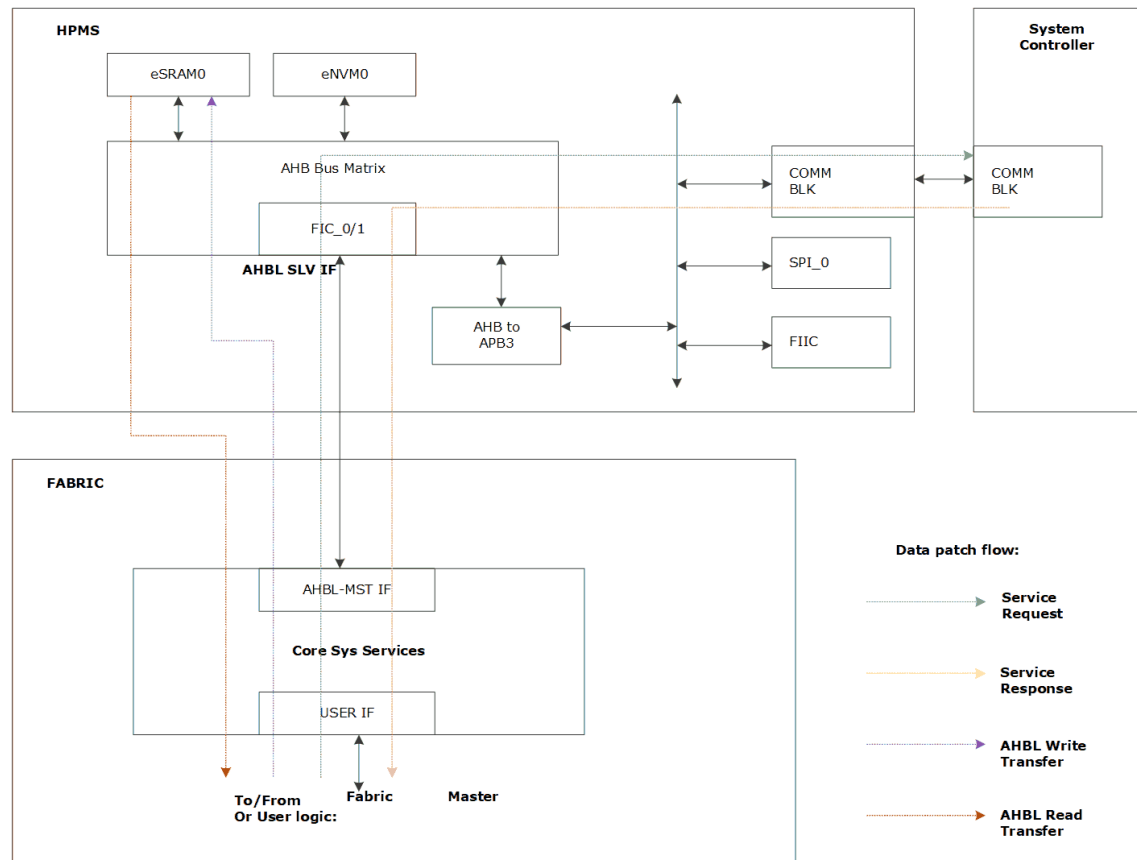
Revision 1.0 was published in June 2018. It was the first publication of this document.

2 IGLOO2 and SmartFusion2 FPGA System Services Simulation

The SmartFusion®2 FPGA family's System Services block features a collection of services responsible for various tasks. These include simulation message services, data pointer services, and data descriptor services. The system services can be accessed through the Cortex-M3 in SmartFusion2 and from the FPGA fabric through the fabric interface controller (FIC) for both SmartFusion2 and IGLOO®2. These access methods are sent to the system controller through the COMM_BLK. The COMM_BLK has an advanced peripheral bus (APB) interface and acts as a message passing conduit to exchange data with the system controller. System service requests are sent to the system controller and system service responses are sent to the CoreSysService through the COMM_BLK. The address location for the COMM_BLK is available inside the microcontroller sub-system (MSS)/high performance memory sub-system (HPMS). For details, see the [UG0450: SmartFusion2 SoC and IGLOO2 FPGA System Controller User Guide](#).

The following illustration shows system services data flow.

Figure 1 • System Service Data Flow Diagram



For both IGLOO2 and SmartFusion2 system service simulation, you need to send out system service requests and check the system service responses to verify that the simulation is correct. This step is necessary to access the system controller, which provides the system services. The way to write to and read from the system controller is different for IGLOO2 and SmartFusion2 devices. For SmartFusion2, the Cortex-M3 is available and you can write and read from the system controller using bus functional model (BFM) commands. For IGLOO2, the Cortex-M3 is not available and the system controller is not accessible using BFM commands.

2.1 Types of Available System Services

Three different types of system services are available and each type of service has different sub-types.

- Simulation message services
- Data pointer services
- Data descriptor services

The [Appendix –System Services Types \(see page 19\)](#) chapter of this guide describes the different types of system services. For more information on system services, see [UG0450: SmartFusion2 SoC and IGLOO2 FPGA System Controller User Guide](#).

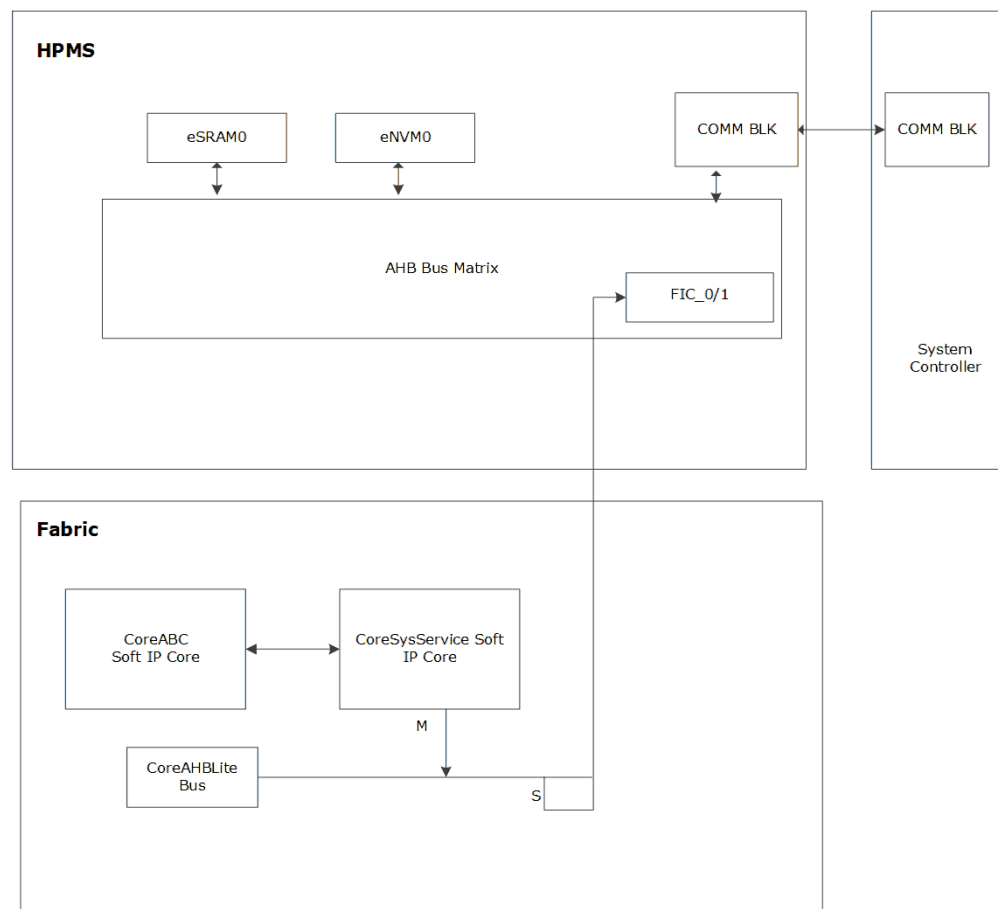
2.2 IGLOO2 System Service Simulation

System services involve writing to and reading from the system controller. To write to and read from the system controller for simulation purposes, you need to perform the steps as follows.

1. Instantiate the CoreSysServices soft IP core, available in the SmartDesign catalog.
2. Write the HDL code for a finite state machine (FSM).

The HDL FSM interfaces with the CoreSysServices Core, which serves as the fabric master of the AHBLite bus. The CoreSysServices core initiates system service request to the COMM BLK and receives system service responses from the COMM BLK through the FIC_0/1, fabric interface controller as shown in the following illustration.

Figure 2 • IGLOO2 System Services Simulation Topology

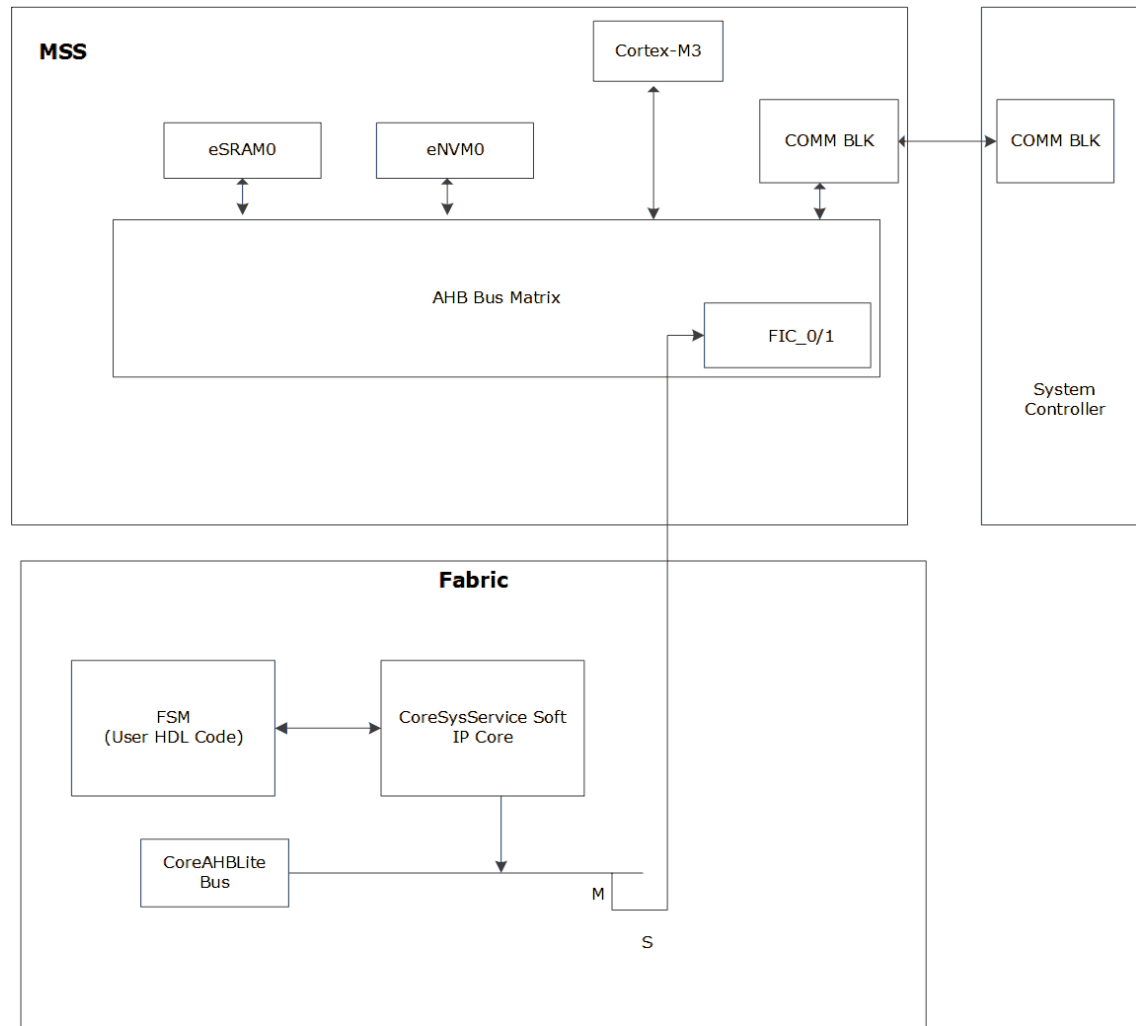


2.3 SmartFusion2 System Service Simulation

To simulate system services in SmartFusion2 devices, you need to write to and read from the system controller. Two options are available to access the system controller for simulation purposes.

- Option 1 — Write the HDL code for an FSM to interface with the CoreSysService soft IP core, which serves as an AHB Lite fabric master and initiates system service request to the COMM BLK and receives system service responses from the COMM BLK through the FIC_0/1 fabric interface as shown in the following illustration.

Figure 3 • SmartFusion2 System Services Simulation Topology



- Option 2 — As the Cortex-M3 is available for SmartFusion2 devices, you may use BFM commands to directly write to and read from the memory space of the system controller.

Using BFM commands (option 2) saves the need to write the HDL codes for the FSM. In this user guide, option 2 is used to show system services simulation in SmartFusion2. With this option, the system controller's memory space is accessed to find out the memory map of the COMM BLK and the fabric interface interrupt controller (FIIC) block when you write your BFM commands.

2.4 Simulation Examples

The user guide covers the following simulations.

- [IGLOO2 Serial Number Service Simulation \(see page 5\)](#)
- [SmartFusion2 Serial Number Service Simulation \(see page 8\)](#)
- [IGLOO2 Zeroization Service Simulation \(see page 13\)](#)
- [SmartFusion2 Zeroization Service Simulation \(see page 16\)](#)

Similar simulation methods can be applied to other system services. For a complete list of the different system services available, go to [Appendix – System Services Types \(see page 19\)](#).

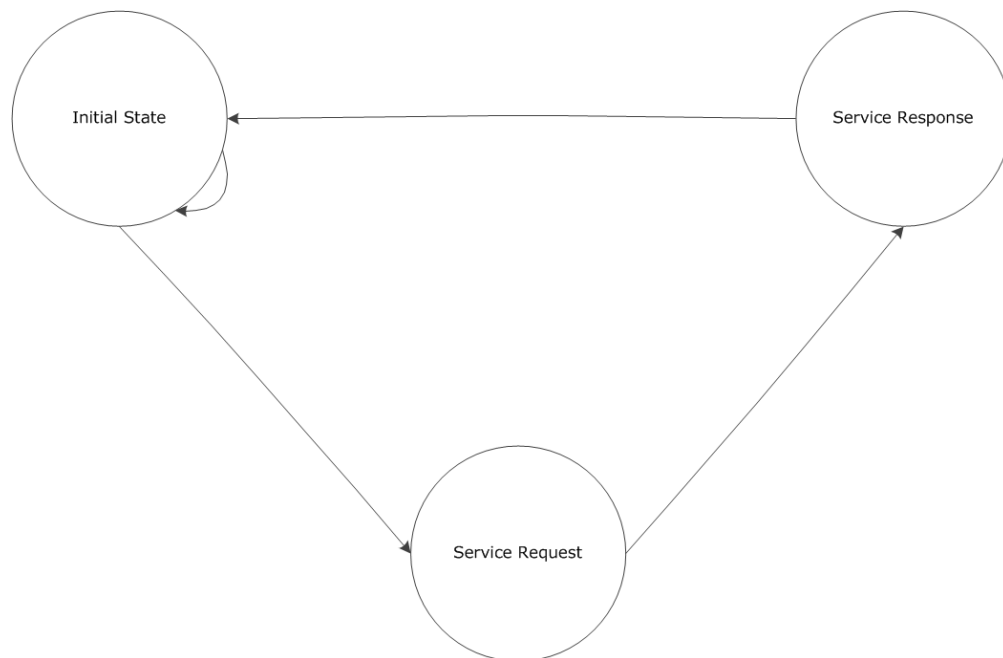
2.5 IGLOO2 Serial Number Service Simulation

To prepare for IGLOO2 serial number service simulation, perform the steps as follows.

1. Invoke system builder to create your HPMS block.
2. Check the HPMS System Services checkbox in the Device Features page. This will instruct the system builder to expose the HPMS_FIC_0 SYS_SERVICES_MASTER bus interface (BIF).
3. Leave all other checkboxes unchecked.
4. Accept the default in all other pages and click **Finish** to complete the system builder block. In the Libero® SoC's HDL editor, write the HDL code for the FSM (**File > New > HDL**). Include the following three states in your FSM.
 - INIT state (initial state)
 - SERV_PHASE (service request state)
 - RSP_PHASE (service response state).

The following figure shows the three states of FSM.

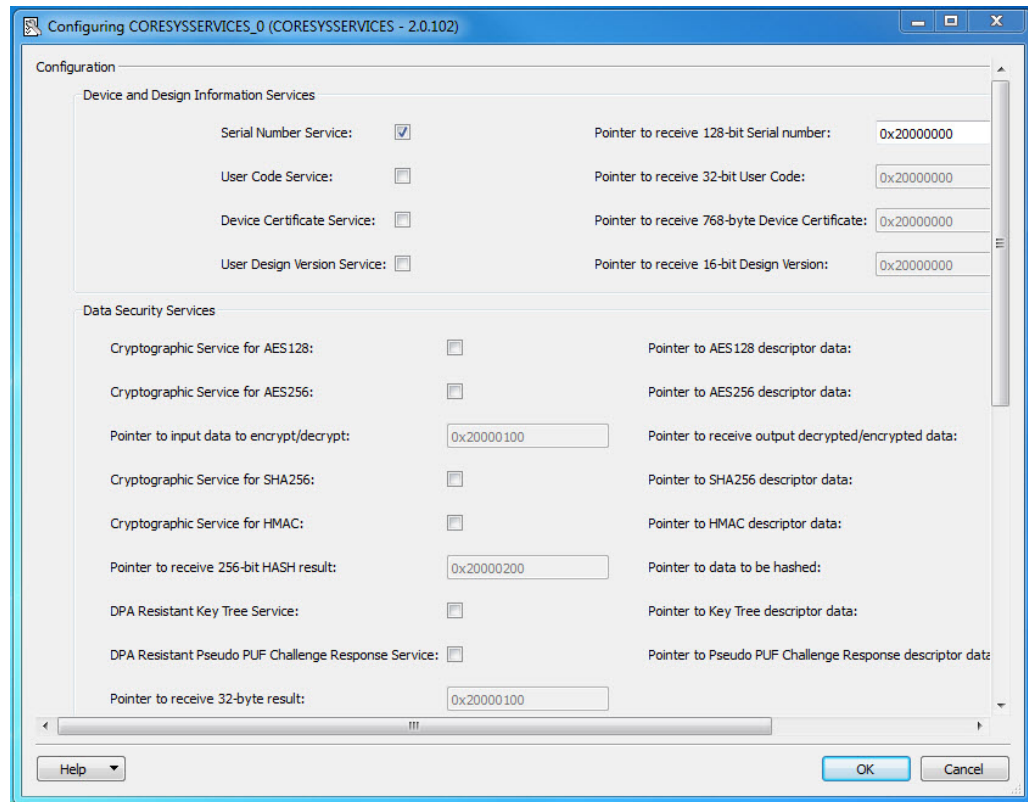
Figure 4 • Three-State FSM



5. In your HDL code for the FSM, use the correct command code ("01" Hex for serial number service) to enter the service request state from the INIT state.
6. Save your HDL file. The FSM appears as a component in the **Design Hierarchy**.
7. Open SmartDesign. Drag and drop your top-level system builder block and your FSM block into the SmartDesign canvas. From the catalog, drag and drop the CoreSysService soft IP core into the SmartDesign canvas.

8. Right-click the CoreSysService soft IP core to open the configurator. Check the **Serial Number Service** checkbox (under the **Device and Design Information Services** group) to enable serial number service.
9. Leave all other checkboxes unchecked. Click **OK** to exit the configurator.

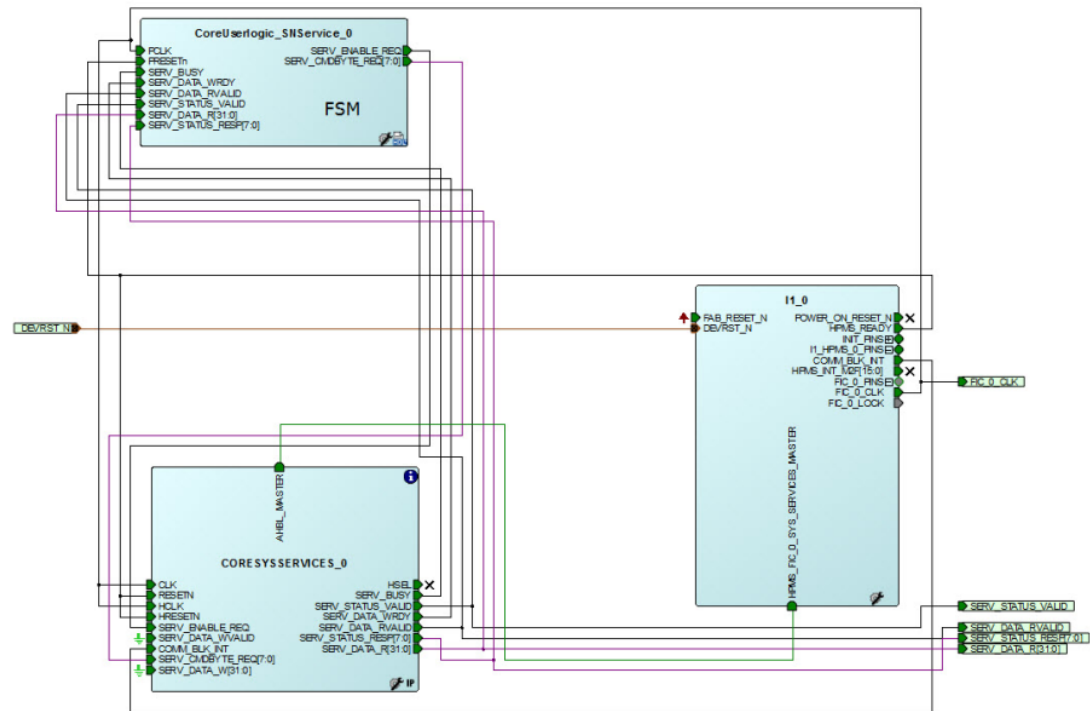
Figure 5 • CoreSysServices soft IP Core Configurator



10. Connect the HPMS_FIC_0 SYS_SERVICES_MASTER BIF of the system builder block to the AHBL_MASTER BIF of the CoreSysService block.

11. Connect the output of your HDL FSM block to the input of the CoreSysService soft IP core. Make all other connections in the SmartDesign canvas as shown in the following figure.

Figure 6 • SmartDesign Canvas with HDL Block, CoreSysServices Soft IP and HPMS Blocks



12. In the SmartDesign canvas, right-click > **Generate Component** to generate the top Level Design.
13. In the **Design Hierarchy** view, right-click the top level design and select **create Testbench > HDL**.
14. Use a text editor to create a text file named "status.txt".
15. Include the command for system service and the 128-bit serial number. For more information, see **Table 1** (System Services Command/Response Values) in the [CoreSysServices v3.1 Handbook](#) for the command codes (Hex) to be used for different system services. For serial number service, the command code is "01" Hex.

The format of the status.txt file for serial number service is as follows.

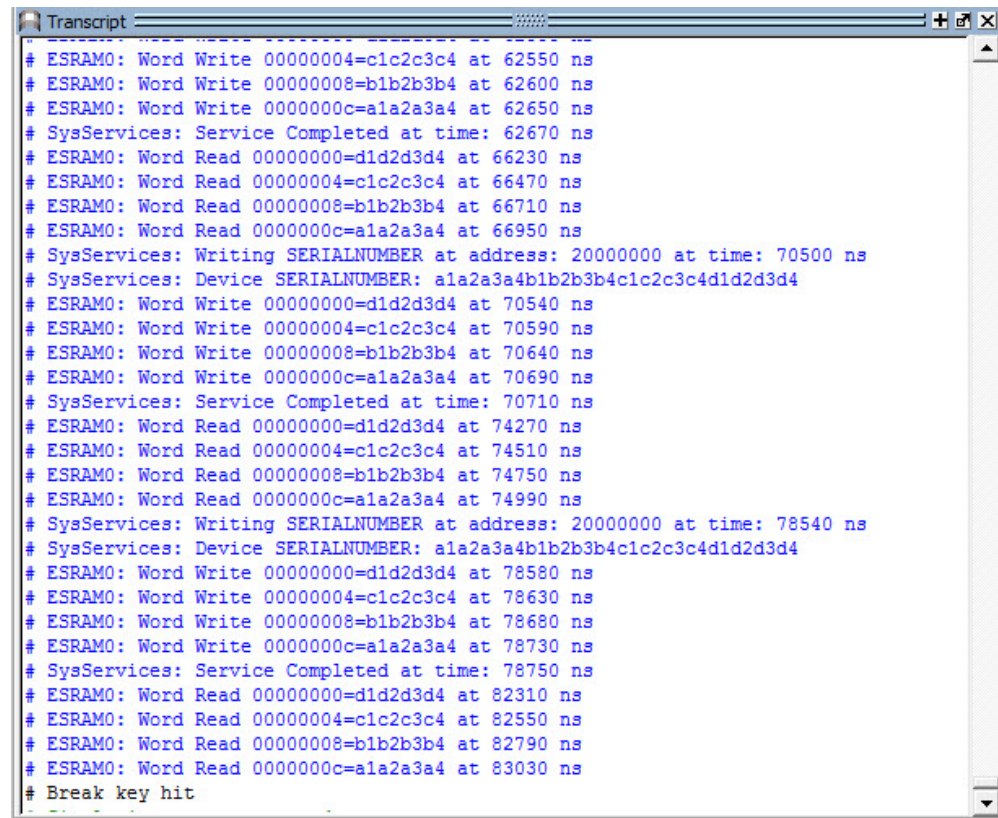
< 2 Hex digit CMD><32 Hex digit Serial Number>

Example: 01A1A2A3A4B1B2B3B4C1C2C3C4D1D2D3D4

Save the status.txt file in the **Simulation** folder of your project. The design is now ready for simulation.

Once the service has begun execution, a message indicating the destination location and serial number is displayed in the ModelSim transcript window, as shown in the following figure.

Figure 7 • ModelSim Simulation Transcript Window



```

# ESRAM0: Word Write 00000004=c1c2c3c4 at 62550 ns
# ESRAM0: Word Write 00000008=b1b2b3b4 at 62600 ns
# ESRAM0: Word Write 0000000c=a1a2a3a4 at 62650 ns
# SysServices: Service Completed at time: 62670 ns
# ESRAM0: Word Read 00000000=d1d2d3d4 at 66230 ns
# ESRAM0: Word Read 00000004=c1c2c3c4 at 66470 ns
# ESRAM0: Word Read 00000008=b1b2b3b4 at 66710 ns
# ESRAM0: Word Read 0000000c=a1a2a3a4 at 66950 ns
# SysServices: Writing SERIALNUMBER at address: 20000000 at time: 70500 ns
# SysServices: Device SERIALNUMBER: a1a2a3a4b1b2b3b4c1c2c3c4d1d2d3d4
# ESRAM0: Word Write 00000000=d1d2d3d4 at 70540 ns
# ESRAM0: Word Write 00000004=c1c2c3c4 at 70590 ns
# ESRAM0: Word Write 00000008=b1b2b3b4 at 70640 ns
# ESRAM0: Word Write 0000000c=a1a2a3a4 at 70690 ns
# SysServices: Service Completed at time: 70710 ns
# ESRAM0: Word Read 00000000=d1d2d3d4 at 74270 ns
# ESRAM0: Word Read 00000004=c1c2c3c4 at 74510 ns
# ESRAM0: Word Read 00000008=b1b2b3b4 at 74750 ns
# ESRAM0: Word Read 0000000c=a1a2a3a4 at 74990 ns
# SysServices: Writing SERIALNUMBER at address: 20000000 at time: 78540 ns
# SysServices: Device SERIALNUMBER: a1a2a3a4b1b2b3b4c1c2c3c4d1d2d3d4
# ESRAM0: Word Write 00000000=d1d2d3d4 at 78580 ns
# ESRAM0: Word Write 00000004=c1c2c3c4 at 78630 ns
# ESRAM0: Word Write 00000008=b1b2b3b4 at 78680 ns
# ESRAM0: Word Write 0000000c=a1a2a3a4 at 78730 ns
# SysServices: Service Completed at time: 78750 ns
# ESRAM0: Word Read 00000000=d1d2d3d4 at 82310 ns
# ESRAM0: Word Read 00000004=c1c2c3c4 at 82550 ns
# ESRAM0: Word Read 00000008=b1b2b3b4 at 82790 ns
# ESRAM0: Word Read 0000000c=a1a2a3a4 at 83030 ns
# Break key hit
  
```

The system controller conducts an AHB write to the address with the serial number. Upon completion of the service, the COMM_BLK's RXFIFO will be loaded with the service response.

Note: For a complete listing of the command codes to be used for different system services, see **Table 1** (System Services Command/Response Values) in [CoreSysServices v3.1 Handbook](#) or [UG0450: SmartFusion2 SoC and IGLOO2 FPGA System Controller User Guide](#).

2.6 SmartFusion2 Serial Number Service Simulation

In this user guide, BFM commands (option 2) are used to access the system controller for system service. BFM commands are used as the Cortex-M3 processor is available on the device for BFM simulation. BFM commands allow you to write directly to and read from the COMM_BLK once you know the memory mapping of the COMM_BLK.

To prepare your design for SmartFusion2 serial number service simulation, perform the following steps.

1. Drag and drop the MSS from the catalog to the design canvas of your project.
2. Disable all MSS peripherals except the MSS_CCC, Reset Controller, Interrupt Management, and FIC_0, FIC_1 and FIC_2.
3. Configure the interrupt management to use MSS to fabric interrupt.
4. Prepare the `serialnum.bfm` file in a text editor or in the Libero's HDL editor. Save the `serialnum.bfm` file in the project's **Simulation** folder. The `serialnum.bfm` should include the following details.
 - Memory mapping to the COMM_BLK (CMBLK)
 - Memory mapping to interrupt management peripheral (FIIC)
 - Command for serial number system service request ("01" Hex)

- Address for the location of the serial number

An example of the serialnum.bfm file is as follows.

```
memmap FIIC 0x40006000; #Memory Mapping to Interrupt Management
```

```
memmap CMBLK 0x40016000; #Memory Mapping to COMM BLK
```

```
memmap DESCRIPTOR_ADDR 0x20000000; #Address location for Serial Num
```

```
#Command Code in Hexadecimal
```

```
constant CMD 0x1 # Comand code for Serial NumberService
```

```
#FIIC Configuration Registers
```

```
constant FICC_INTERRUPT_ENABLE0 0x0
```

```
#COMM_BLK Configuration Registers
```

```
constant CONTROL 0x00
```

```
constant STATUS 0x04
```

```
constant INT_ENABLE 0x08
```

```
constant DATA8 0x10
```

```
constant DATA32 0x14
```

```
constant FRAME_START8 0x18
```

```
constant FRAME_START32 0x1C
```

```
procedure serialnum;
```

```
int x;
```

```
write w FIIC FICC_INTERRUPT_ENABLE0 0x20000000 #Configure
#FICC_INTERRUPT_ENABLE0 # Register to enable COMBLK_INTR #
```

```
#interrupt from COMM_BLK block to fabric
```

```
#Request Phase
```

```
write w CMBLK CONTROL 0x10 # Configure COMM BLK Control #Register to
enable transfers on the COMM BLK Interface
```

```
write w CMBLK INT_ENABLE 0x1 # Configure COMM BLK Interrupt Enable
#Register to enable Interrupt for TXTOKAY (Corresponding bit in the
#Status Register)
```

```

waitint 19 # wait for COMM BLK Interrupt , Here #BFM waits
#till COMBLK_INTR is asserted
readstore w CMBLK STATUS x # Read COMM BLK Status Register for #TXTOKAY
# Interrupt
set x x & 0x1
if x
write w CMBLK FRAME_START8 CMD # Configure COMM BLK FRAME_START8
#Register to request Serial Number service
endif
endif

waitint 19 # wait for COMM BLK Interrupt , Here
#BFM waits till COMBLK_INTR is asserted
readstore w CMBLK STATUS x # Read COMM BLK Status Register for
#TXTOKAY Interrupt
set x x & 0x1
set x x & 0x1
if x
write w CMBLK CONTROL 0x14 #Configure COMM BLK Control
#Register to enable transfers on the COMM BLK Interface
write w CMBLK DATA32 DESCRIPTOR_ADDR
write w CMBLK INT_ENABLE 0x80
write w CMBLK CONTROL 0x10
endif

wait 20

#Response Phase
waitint 19
readstore w CMBLK STATUS x
set x x & 0x80
if x

```

```

readcheck w CMBLK FRAME_START8 CMD
write w CMBLK INT_ENABLE 0x2
endif

```

```

waitint 19

readstore w CMBLK STATUS x
set x x & 0x2
if x
readcheck w CMBLK DATA8 0x0
write w CMBLK CONTROL 0x18
endif

```

```

waitint 19

readcheck w FIIC 0x8 0x20000000
readstore w CMBLK STATUS x
set x x & 0x2
if x
readcheck w CMBLK DATA32 DESCRIPTOR_ADDR
endif

```

```

readcheck w DESCRIPTOR_ADDR 0x0 0xE1E2E3E4; #Readcheck to check S/N
readcheck w DESCRIPTOR_ADDR 0x4 0xC1C2C3C4; #Readcheck to check S/N
readcheck w DESCRIPTOR_ADDR 0x8 0xB1B2B3B4; #Readcheck to check S/N
readcheck w DESCRIPTOR_ADDR 0xC 0xA1A2A3A4; #Readcheck to check S/N

return

```

5. Create the `status.txt` file in Libero's HDL editor or any text editor. Include the serial number system service command ("01" in Hex) and the serial number in the `status.txt` file. See the [CoreSysServices v3.1 Handbook](#) for using the correct command code.

6. The syntax of this file for serial number service is, `<2 Hex digit CMD>< 32 Hex digit Serial Number>`. Example: `01A1A2A3A4B1B2B3B4C1C2C3C4E1E2E3E4`.

7. Save the `status.txt` file in the project's **Simulation** folder.

8. Edit the `user.bfm` (located inside the **Simulation** folder) to include the `serialnum.bfm` file and call the serial number procedure as shown in the following code snippet.

```
include "serialnum.bfm" #include the serialnum.bfm
procedure user_main;
print "INFO:Simulation Starts";
print "INFO:Service Command Code in Decimal:%0d", CMD ;

call serialnum; #call the serialnum procedure
print "INFO:Simulation Ends";
return
```

9. In the **Design Hierarchy** view, generate the **testbench** (Right-click, **Top Level Design > Create Testbench > HDL**) and you are ready to run serial number service simulation.

Once the service has begun execution, a message indicating the destination location and serial number is displayed. The system controller conducts an AHB write to the address with the serial number. Upon completion of the service, the COMM_BLK's RXFIFO will be loaded with the service response. The ModelSim transcript window displays the address and the serial number received as shown in the following figure.

Figure 8 • SmartFusion2 Serial Number Service Simulation in ModelSim Transcript Window

```
# BFM:45:write w 40016000 00000014 at 10945 ns
# BFM:46:write w 40016014 20000000 at 10955 ns
# BFM:47:write w 40016008 00000080 at 10965 ns
# BFM: Data Write 40016000 00000014
# BFM:48:write w 40016000 00000010 at 10995 ns
# BFM: Data Write 40016014 20000000
# BFM:49:endif
# BFM:51:wait 20 starting at 11025 ns
# BFM: Data Write 40016008 00000080
# BFM: Data Write 40016000 00000010
# SysServices: Writing SERIALNUMBER at address: 20000000 at time: 11095 ns
# SysServices: Device SERIALNUMBER: a1a2a3a4b1b2b3b4c1c2c3c4d1d2d3d4
# ESRAM0: Word Write 00000000=d1d2d3d4 at 11135 ns
# ESRAM0: Word Write 00000004=c1c2c3c4 at 11185 ns
# BFM:54:waitint 19 at 11225 ns
# ESRAM0: Word Write 00000008=b1b2b3b4 at 11235 ns
# ESRAM0: Word Write 0000000c=a1a2a3a4 at 11285 ns
# SysServices: Service Completed at time: 11305 ns
# BFM:Interrupt Wait Time 8 cycles
# BFM:55:readstore w 40016004 @3 at 11315 ns
# SFM: Data Read 40016004 00000083 at 11375.000000ns
# BFM:56:set 3= 0x00000080 (128)
# BFM:57:if 00000080 func 00000000
# BFM:58:readcheck w 40016018 00000001 at 11395 ns
# BFM:59:write w 40016008 00000002 at 11405 ns
# BFM:60:endif
# BFM:62:waitint 19 at 11415 ns
# BFM:Interrupt Wait Time 0 cycles
# BFM:63:readstore w 40016004 @3 at 11425 ns
# BFM: Data Read 40016018 00000001 MASK:ffffff at 11455.000000ns
# BFM: Data Write 40016008 00000002
# SFM: Data Read 40016004 00000003 at 11545.000000ns
# BFM:64:set 3= 0x00000002 (2)
# BFM:65:if 00000002 func 00000000
# BFM:66:readcheck w 40016010 00000000 at 11565 ns
# BFM:67:write w 40016000 00000018 at 11575 ns
# BFM:68:endif
# BFM:70:waitint 19 at 11585 ns
# BFM:Interrupt Wait Time 0 cycles
# BFM:71:readcheck w 40006008 20000000 at 11595 ns
# BFM: Data Read 40016010 00000000 MASK:ffffff at 11625.000000ns
# BFM:72:readstore w 40016004 @3 at 11635 ns
```

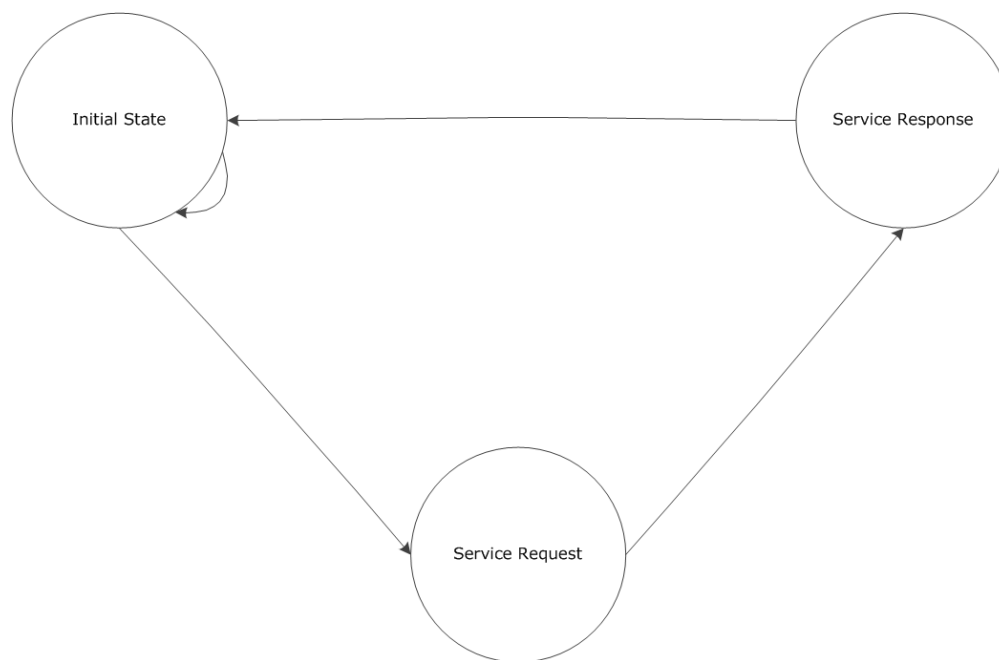
2.7 IGLOO2 Zeroization Service Simulation

To prepare for IGLOO2 zeroization service simulation, perform the steps as follows.

1. Invoke system builder to create the HPMS block. Check the **HPMS System Services** checkbox in the **Device Features** page. This instructs the system builder to expose the HPMS_FIC_0 SYS_SERVICES_MASTER BIF. Leave all other checkboxes unchecked. Accept the default in all other pages and click **Finish** to complete the configuration of the system builder block.
2. In the Libero SoC's HDL editor, write the HDL code for the FSM. In your HDL code for the FSM, include the following three states.
 - INIT state (initial state)
 - SERV_PHASE (service request state)
 - RSP_PHASE (service response state)

The following figure shows the three states of FSM.

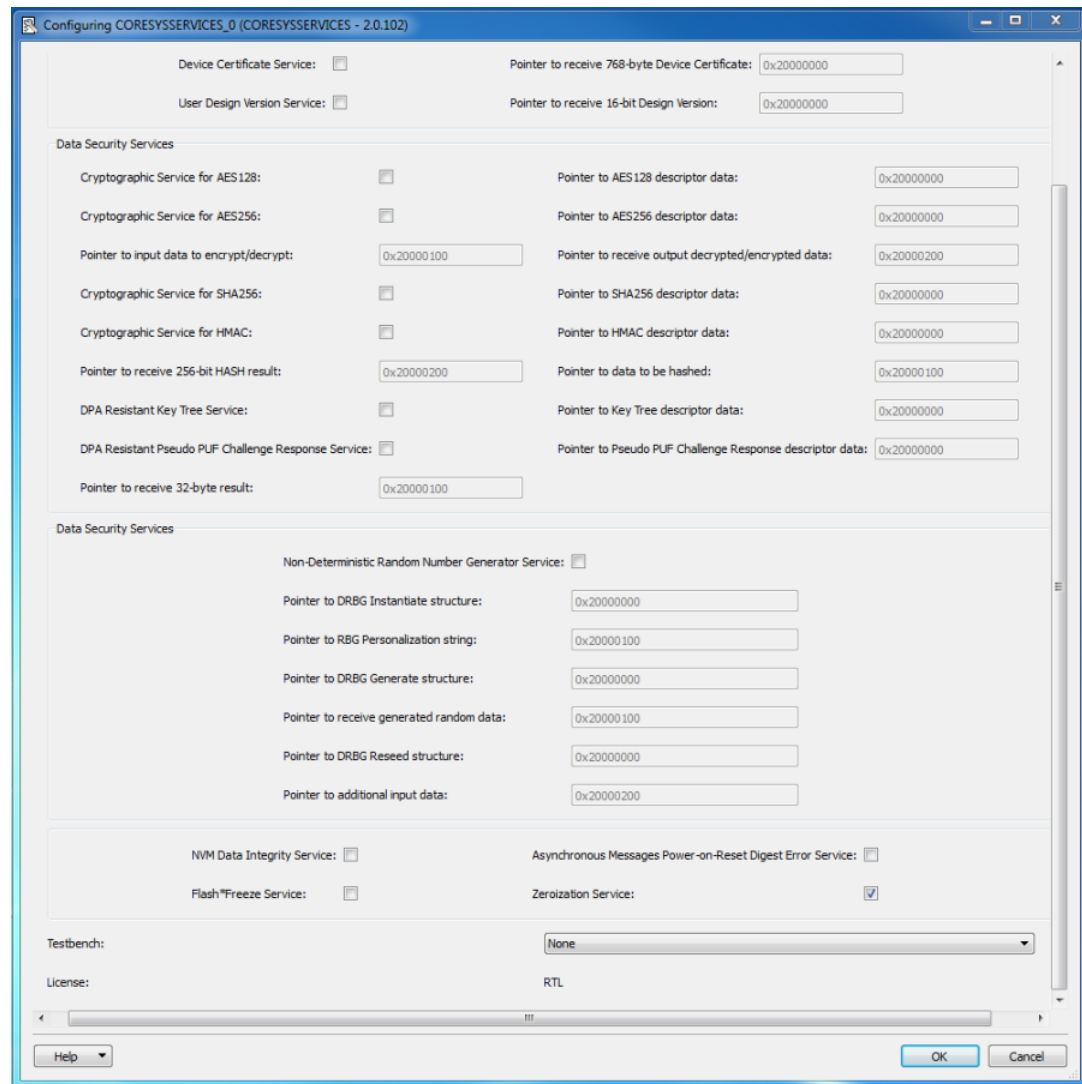
Figure 9 • Three-State FSM



3. In your HDL code, use the command code "F0"(Hex) to enter the service request state from the INIT state.
4. Save your HDL file.
5. Open SmartDesign, drag and drop your top-level system builder block and your HDL FSM block into the SmartDesign canvas. From the catalog, drag and drop the CoreSysService soft IP core into the SmartDesign canvas.

6. Right-click the CoreSysServices soft IP core, to open the configurator and check the **Zeroization Service** checkbox under the **Data Security Services** group. Leave all other checkboxes unchecked. Click **OK** to exit.

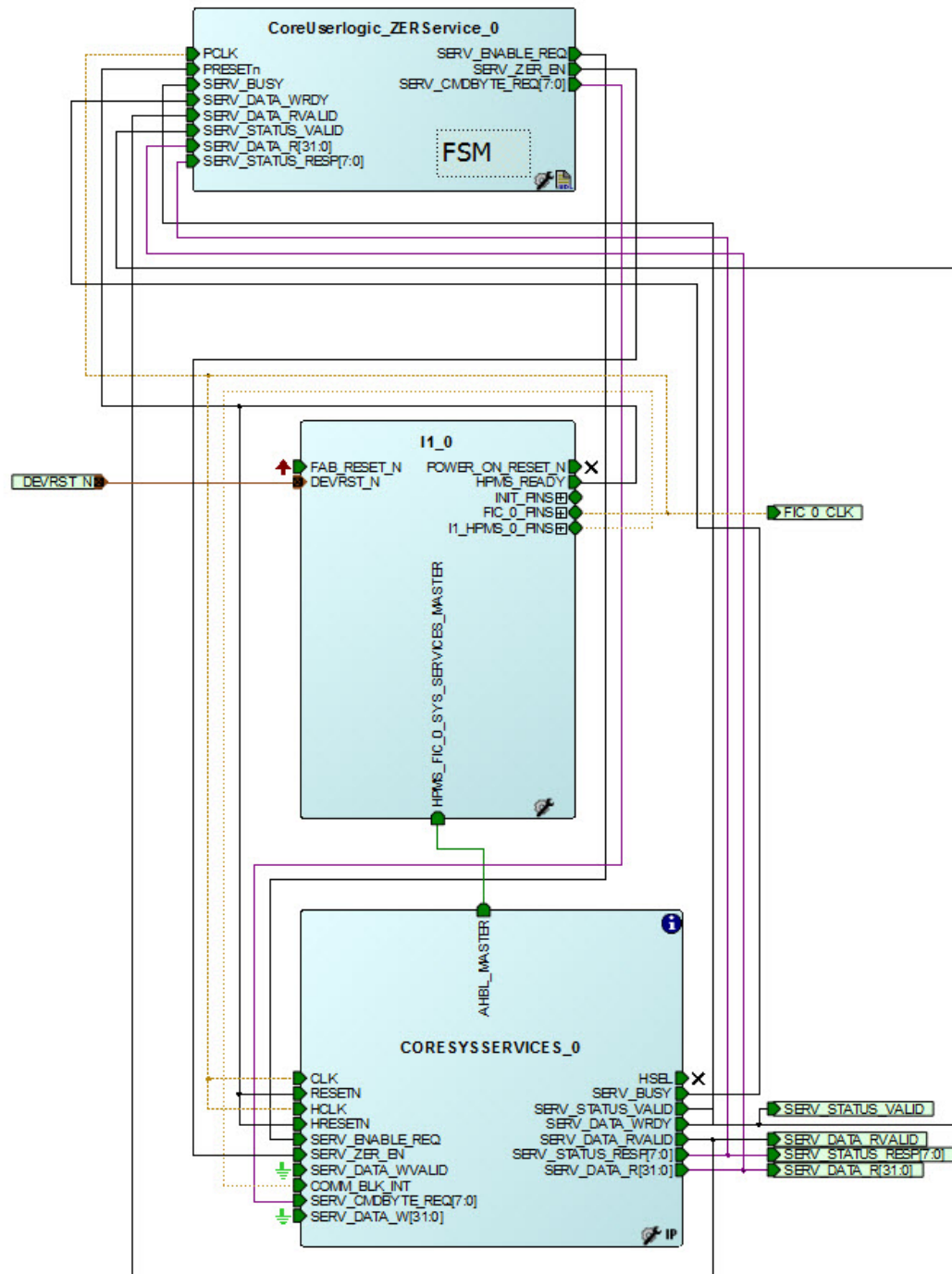
Figure 10 • CoreSysServices Configurator



7. Connect the HPMS_FIC_0 SYS_SERVICES_MASTER BIF of the system builder block to the AHBL_MASTER BIF of the CoreSysService block.

8. Connect the output of your HDL FSM block to the input of the CoreSysService soft IP core. Make all other connections in the SmartDesign canvas.

Figure 11 • SmartDesign Canvas with HDL Block, CoreSysServices Soft IP, and HPMS Blocks



9. In the SmartDesign canvas, generate the top-level design (**Right-click > Generate Component**).

10. In the **Design Hierarchy** view, right-click the top-level design and select **create Testbench > HDL**. You are now ready to run simulation.

Once the service has begun execution, a message indicating that the zeroization has been completed at time x is displayed as shown in the following figure.

Figure 12 • IGLOO2 Zeroization System Service Simulation Transcript Window

```
# Loading IGLOO2.phy_data_dqs_out
# Loading IGLOO2.phy_data_local_odt_dqs_load_wrapper
# Loading IGLOO2.phy_data_fifo_we_out
# Loading IGLOO2.phy_data_and2_wrapper
# Loading IGLOO2.phy_data_fifoweout_load_wrapper
# Loading IGLOO2.phy_data_dqin_dly
# Loading IGLOO2.phy_data_in_bit
# Loading IGLOO2.phy_data_dqin_nor2_load_wrapper
# Loading IGLOO2.phy_data_dqin_load_wrapper
# Loading IGLOO2.phy_data_pos_captrff_wrapper
# Loading IGLOO2.phy_data_rd_fifo
# Loading IGLOO2.phy_data_nor2_wrapper
# Loading IGLOO2.dfi_phy_bist
# Loading IGLOO2.dfi_prtbs
# Loading IGLOO2.ingot_sync_fifo_rst
# Loading IGLOO2.phy_data_slice
# Loading IGLOO2.phy_ctrl_cmd_bit
# Loading IGLOO2.phy_ctrl_posff_wrapper
# Loading IGLOO2.phy_ctrl_cmd_bit_load_wrapper
# Loading IGLOO2.phy_ctrl_buf1_wrapper
# Loading IGLOO2.phy_ctrl_clkout
# Loading IGLOO2.phy_ctrl_invl_wrapper
# Loading IGLOO2.phy_ctrl_negff_set_wrapper
# Loading IGLOO2.phy_ctrl_stoy_mux21_wrapper
# Loading IGLOO2.phy_ctrl_clkout_load_wrapper
# Loading IGLOO2.dfi_single_rank_ratio_logic
# Loading IGLOO2.UDF_MUX2
# Loading IGLOO2.UDF_DFF
# Loading IGLOO2.UDF_DL
# ** ENVM_init.mem.
# **      0.
# **      32768.
# SysServices: Device Has Been Zeroized at time: 13040 ns
# ** Note: $stop : nofile(819)
# Time: 13050010 ps Iteration: 1 Instance: /testbench/I1_top_0/I1_0/I1_HPMSS_0/MSS_ADLIB_INST/u_mss/u_mss/u_APB_SLAVES/U_COMBLK
# Break at a source-protected location
# Simulation Breakpoint: Break at a source-protected location
# MACRO ./run.do PAUSED at line 30
# End time: 16:39:35 on Dec 04,2014, Elapsed time: 0:01:58
# Errors: 0, Warnings: 0
```

The system controller conducts an AHB write to the address with the serial number. Upon completion of the service, the COMM_BLK's RXFIFO will be loaded with the service response. It should be noted that the simulation model simulates zeroization by stopping the simulation rather than zeroizing the design itself.

Note: For a complete listing of the command codes to be used for different system services, see **Table 1** (System Services Command/Response Values) in the [CoreSysServices v3.1 Handbook](#) or [UG0450: SmartFusion2 SoC and IGLOO2 FPGA System Controller User Guide](#).

2.8 SmartFusion2 Zeroization Service Simulation

In this guide, BFM commands (option 2) are used to access the system controller for system service. BFM commands are used as the Cortex-M3 processor is available on the device for BFM simulation. BFM commands allow you to write directly to and read from the COMM_BLK once you know the memory mapping of the COMM_BLK. To prepare your design for SmartFusion2 zeroization service simulation, perform the following steps.

1. Drag and drop the MSS from the catalog to the design canvas of your project.
2. Disable all MSS peripherals except the MSS_CCC, Reset Controller, Interrupt Management, and FIC_0, FIC_1 and FIC_2.
3. Configure the interrupt management to use MSS to fabric interrupt.
4. Prepare the zeroization.bfm file in a text editor or in Libero's HDL editor. Your zeroization.bfm should include:
 - Memory mapping to the COMM_BLK (CMBLK)
 - Memory mapping to interrupt management peripheral (FIIC)
 - Command for zeroization service request ("F0" Hex for zeroization)

An example of the `serialnum.bfm` file is shown in the following figure.

Figure 13 • Zeroization.bfm for SmartFusion2 Zeroization System Services Simulation

```
memmap FIIC      0x40006000;
memmap CMBLK     0x40016000;

#Command Code in Hexadecimal
constant CMD      0xF0
constant OPTIONS  0x0

#FIIC Configuration Registers
constant FICC_INTERRUPT_ENABLE0 0x0

#COMM_BLK Configuration Registers
constant CONTROL      0x00
constant STATUS       0x04
constant INT_ENABLE    0x08
constant DATA8        0x10
constant DATA32       0x14
constant FRAME_START8  0x18
constant FRAME_START32 0x1C

procedure zeroization;
int x;

write w FIIC FICC_INTERRUPT_ENABLE0 0x20000000 ; # Configure FICC_INTERRUPT_ENABLE0 Register to enable COMBLK_INTR
# Interrupt from COMM_BLK block to fabric .

#Request Phase
write w CMBLK CONTROL      0x10      # Configure COMM BLK Control Register to enable transfers on the COMM
BLK Interface
write w CMBLK INT_ENABLE    0x1      # Configure COMM BLK Interrupt Enable Register to enable Interrupt for
# TXTOKAY (Corresponding bit in the Status Register)

waitint 19                  # wait for COMM BLK Interrupt , Here BFM waits till COMBLK_INTR is
asserted
readstore w CMBLK STATUS x    # Read COMM BLK Status Register for TXTOKAY Interrupt
set x x & 0x1
if x
write w CMBLK FRAME_START8 CMD # Configure COMM BLK FRAME_START8 Register to request Zeroization
service
endif

waitint 19                  # wait for COMM BLK Interrupt , Here BFM waits till COMBLK_INTR is
asserted
readstore w CMBLK STATUS x    # Read COMM BLK Status Register for TXTOKAY Interrupt
set x x & 0x1
if x
write w CMBLK CONTROL      0x10      # Configure COMM BLK Control Register to enable transfers on the COMM
BLK Interface
write w CMBLK DATA8        OPTIONS  # Configure COMM BLK DATA8 Register to send data
endif

wait 20

return
```

5. Save the `zeroization.bfm` file in the project's **Simulation** folder.

6. Edit the `user.bfm` (located in the **Simulation** folder) to include the `zeroization.bfm` using the following code snippet.

```
include "zeroization.bfm" #include zeroization.bfm file
procedure user_main;
print "INFO:Simulation Starts";
print "INFO:Service Command Code in Decimal:%0d", CMD ;
call zeroization; #call zeroization procedure

return
```

7. In the **Design Hierarchy**, generate the **Testbench** (Right click top level > Create Testbench > HDL) and you are ready to run the SmartFusion2 zeroization simulation.

Once the service has begun execution, a message indicating that the device has been zeroized at time `x` is displayed. It should be noted that the simulation model simulates zeroization by stopping the simulation rather than zeroizing the design itself. The ModelSim transcript window in the following figure shows that the device has been zeroized.

Figure 14 • SmartFusion2 Zeroization System Service Simulation Log

```

*****
# AMBA BFM Model
# Version 2.1 22Dec08
#
# Opening BFM Script file test.vec
# Read 115 Vectors - Compiler Version 26.28
# BFM: Filenames referenced in Vectors
#   test.bfm
#   zeroization.bfm
# BFM: INFO: Simulation Starts
# BFM: INFO: Service Command Code in Decimal: 240
# BFM: 21: call 5
# BFM: 32795: int 1
# BFM: 32797: write w 40006000 20000000 at 10615 ns
# BFM: 32800: write w 40016000 00000010 at 10625 ns
# BFM: 32801: write w 40016008 00000001 at 10635 ns
# BFM: Data Write 40006000 20000000
# BFM: 32803: waitint 19 at 10665 ns
# BFM: Data Write 40016000 00000010
# BFM: Data Write 40016008 00000001
# BFM: Interrupt Wait Time 7 cycles
# BFM: 32804: readstore w 40016004 @2 at 10745 ns
# SFM: Data Read 40016004 00000001 at 10805.000000ns
# BFM: 32805: set 2= 0x00000001 (1)
# BFM: 32806: if 00000001 func 00000000
# BFM: 32807: write w 40016018 000000f0 at 10825 ns
# BFM: 32808: endif
# BFM: 32810: waitint 19 at 10835 ns
# BFM: Interrupt Wait Time 0 cycles
# BFM: 32811: readstore w 40016004 @2 at 10845 ns
# BFM: Data Write 40016018 000000f0
# SysServices: Device Has Been Zeroized at time: 10885 ns
# ** Note: $stop : nofile(819)
#   Time: 10895 ns Iteration: 1 Instance: /testbench/M3_SysService_0/M3_
SysService_MSS_0/MSS_ADLIB_INST/u_mss/u_mss/u_APB_SLAVES/U_COMBLK
# Break at a source-protected location
# Simulation Breakpoint: Break at a source-protected location
# MACRO ./run.do PAUSED at line 20
VSIM(paused)>

```

3 Appendix: Types Of System Services

This chapter describes various types of system services.

3.1 Simulation Message Services

The following sections describe various types of simulation message services.

3.1.1 Flash*Freeze

The simulation will enter the Flash*Freeze state when the proper service request is sent to the COMM_BLK from either the FIC (in the case of IGLOO2 devices) or the Cortex-M3 (in SmartFusion2 devices). Once the service has been detected by the system controller, the simulation will be stopped and a message indicating the system has entered Flash*Freeze (along with the option selected) will be displayed. Upon resumption of the simulation, the RXFIFO of the COMM_BLK will be filled with the service response consisting of the service command and status. It should be noted that there is no simulation support for Flash*Freeze exit.

3.1.2 Zeroization

Zeroization is currently the only high priority service within system services processed by the COMM_BLK. The simulation will enter the zeroization state as soon as the correct service request is detected by the COMM_BLK. Execution of other services will be halted and discarded by the system controller, and the zeroization service will be executed instead. Once the zeroization service request is detected, the simulation stops and a message indicating the system has entered zeroization is displayed. Manual restarts of simulation after zeroization are invalid.

3.2 Data Pointer Services

The following sections describe various types of data pointer services.

3.2.1 Serial Number

The serial number service will write a 128-bit serial number to an address location provided as part of the service request. This 128-bit parameter can be set using a [System Service Simulation Support file \(see page 22\)](#). If the 128-bit serial number parameter is not defined within the file, a default serial number of 0 will be used. Once the service has begun execution, a message indicating the destination location and serial number is displayed. The system controller conducts an AHB write to the address with the serial number. Upon completion of the service, the COMM_BLK's RXFIFO will be loaded with the service response.

3.2.2 Usercode

The usercode service writes a 32-bit usercode parameter to an address location provided as part of the service request. This 32-bit parameter can be set using the [System Service Simulation Support file \(see page 22\)](#). If the 32-bit parameter is not defined within the file, a default value of 0 is used. Once the service has begun execution, a message indicating the target location and usercode is displayed. The system controller conducts an AHB write to the address with the 32-bit parameter. Upon completion of the service, the COMM_BLK's RXFIFO is loaded with the service response, which includes the service command and target address.

3.3 Data Descriptor Services

The following sections describe various types of data descriptor services.

3.3.1 AES

The simulation support for this service is only concerned with moving the original data from the source to the destination, without actually performing any encryption/decryption on the data. The data that needs to be encrypted/decrypted and the data structure should be written before the service request is sent. Once the service has begun execution, a message indicating the execution of the AES service is displayed. The AES service reads both the data structure and data to be encrypted/decrypted. The original data is copied and written to the address provided within the data structure. Once the service is completed, the command, status, and data structure address are pushed into the RXFIFO.

Note: This service is only for 128-bit and 256-bit data, and both 128-bit and 256-bit data have different data structure lengths.

3.3.2 SHA 256

The simulation support for this service is only concerned with moving the data, without actually performing any hashing on the data. The SHA 256 function is designed to generate a 256-bit hash key based on the input data. The data that needs to be hashed and the data structure should be written to their respective addresses before the service request is sent to the COMM_BLK. The length in bits and pointer defined within the SHA 256 data structure must correctly correspond to the length and address of the data to be hashed. Once the service has begun execution, a message indicating the execution of the SHA 256 service is displayed. Rather than executing the actual function, a default hash key will be written to the destination pointer from the data structure. The default hash key is hex "ABCD1234". For setting a custom key, go to the [Parameter Setting \(see page 23\)](#) section. Upon completion of the service, the RXFIFO is loaded with the service response consisting of the service command, status, and SHA 256 data structure pointer.

3.3.3 HMAC

The simulation support for this service is only concerned with moving of data, without actually performing any hashing on the data. The data that needs to be hashed and the data structure should be written to their respective addresses before the service request is sent to the COMM_BLK. The HMAC service requires a 32-byte key in addition to the length in bytes, source pointer, and destination pointer. Once the service has begun execution, a message indicating the execution of the HMAC service is displayed. The key is read and the 256-bit key is copied from the data structure to the destination pointer. Upon completion of the service, the RXFIFO is loaded with the service response consisting of the service command, status, and HMAC data structure pointer.

3.3.4 DRBG Generate

Generation of random bits is performed by this service. It should be noted that the simulation model does not exactly follow the same random number generation methodology used by the silicon. The data structure must be correctly written into its intended location before the service request is sent to the COMM_BLK. The data structure, destination pointer, length and other relevant data are read by the system controller. The DRBG generate service generates a pseudo random set of data of the requested length (0-128). The system controller writes the random data into the destination pointer. A message indicating the execution of DRBG generate service is displayed in simulation. Once the service is completed, the command, status, and data structure address are pushed into the RXFIFO. If the requested data length is not within the range of 0-128, an error code of "4" (Max Generate Request Too Big) will be pushed into the RXFIFO. If the additional data length is not within the range of 0-128, an error code of "5" (Max Length of Additional Data Exceeded) will be pushed into the RXFIFO. If both the requested data length for generate and additional data length are not within their defined range (0-128), an error code of "1" (Catastrophic Error) is pushed into the RXFIFO.

3.3.5 DRBG Reset

The actual reset function is performed by removing DRBG instantiations and resetting DRBG. Once the service request has been detected, the simulation displays a DRBG Reset service completed message. The response, which includes the service and status, is pushed into the RXFIFO.

3.3.6 DRBG Self Test

The simulation support for the DRBG self-test does not actually execute the self-test function. Once the service request has been detected, the simulation will display a DRBG self-test service execution message. The response, which includes the service and status, will be pushed into the RXFIFO.

3.3.7 DRBG Instantiate

The simulation support for the DRBG instantiate service does not actually perform the instantiate service. The data structure must be correctly written into its intended location before the service request is sent to the COMM_BLK. Once the service request has been detected, the structure and personalization string defined within the MSS address space will be read. The simulation will display a message indicating that the DRBG Instantiate service has begun execution. Once the service is complete, the response, which includes the service command, status, and pointer to the data structure, will be pushed into the RXFIFO. If the data length (PERSONALIZATIONLENGTH) is not within the range of 0-128, an error code of "1" (Catastrophic Error) will be pushed into the RXFIFO for the status.

3.3.8 DRBG Uninstantiate

The simulation support for the DRBG uninstantiate service does not actually perform the uninstantiate service of removing a previously instantiated DRBG, like the silicon does. The service request must include both the command and DRBG handle. Once the service request has been detected, the DRBG handle will be stored. The simulation will display a message indicating that the DRBG uninstantiate service has been initialized. Once the service is complete, the response, which includes the service command, status, and DRBG handle, will be pushed into the RXFIFO.

3.3.9 DRBG Reseed

Due to the simulative nature of the system services block, the DRBG reseed service in simulation is not executed automatically after every 65535 DRBG generate services. The data structure must be correctly written into its intended location before the service request is sent to the COMM_BLK. Once the service request has been detected, the structure and additional input parameter in the MSS address space will be read. A message indicating that the DRBG reseed service has begun executed, will be displayed. The data structure must be correctly written into its intended location before the service request is sent to the COMM_BLK. Once the service is complete, the response, which includes the service command, status, and pointer to the data structure, will be pushed into the RXFIFO.

3.3.10 KeyTree

The actual function is not executed in simulation for the KeyTree service. The KeyTree service data structure consists of a 32-byte key, 7-bit otype data (MSB ignored), and 16-byte path. The data within the data structure should be written to their respective addresses, before the service request is sent to the COMM_BLK. Once the service has begun execution, a message indicating the execution of the KeyTree service will be displayed. The contents of the data structure will be read, the 32-byte key will be stored, and the original key located within the data structure is overwritten. After this AHB write, the value of the key within the data structure should not change, but AHB transactions for the write will occur. Upon completion of the service, the RXFIFO is loaded with the service response, consisting of the service command, status, and the KeyTree data structure pointer.

3.3.11 Challenge Response

The actual function, like authentication of the device, is not executed in simulation for the challenge response service. The data structure for this service requires a pointer to the buffer, to receive a 32-byte result, 7-bit otype, and a 128-bit path. The data within the data structure should be written to their respective addresses before the service request is sent to the COMM_BLK. Once the service has begun execution, a message indicating the execution of the challenge response service will be displayed. A generic 256-bit response will be written into the pointer provided within the data structure. The default key is set as hex "ABCD1234". To get a custom key, check [Parameter Setting \(see page 23\)](#). Upon completion of the service, the RXFIFO will be loaded with the service response, consisting of the service command, status, and challenge response data structure pointer.

3.4 Other Services

The following sections describe various other system services.

3.4.1 Digest Check

The actual function of recalculating and comparing digests of selected components is not executed for the digest check service in simulation. This service request consists of service commands, and service options (5-bit LSB). Once the service has begun execution, a message detailing the execution of the digest check service will be displayed, along with the selected options from the request. Upon completion of the service, the RXFIFO will be loaded with the service response, consisting of the service command, and the digest check pass/fail flags.

3.4.2 Unrecognized Command Response

When an unrecognized service request is sent to the COMM_BLK, the COMM_BLK will automatically reply with a unrecognized command message pushed into the RXFIFO. The message consists of the command sent into the COMM_BLK and the unrecognized command status (252D). A display message indicating an unrecognized service request has been detected will also be displayed. The COMM_BLK will return to an idle state, waiting to accept the next service request.

3.4.3 Unsupported Services

Unsupported services set to the COMM_BLK will trigger a message in simulation indicating that the service request is unsupported. The COMM_BLK will return to an idle state, waiting to accept the next service request. The PINTERRUPT will not be set, indicating that a service has been complete. The current list of unsupported services include: IAP, ISP, Device Certificate, and the DESIGNVER Service.

3.5 System Services Simulation Support File

To support system services simulation, a text file called, "status.txt" can be used to pass instructions about the required behavior of the simulation model to the simulation model. This file should be located in the same folder, that the simulation is run from. The file can be used, among other things, to force certain error responses for the system services supported or even for setting some parameters needed for simulation, (for example, serial number). The maximum number of lines supported in the "status.txt" file is 256. Instructions that appear after line number 256 will not be used in the simulation.

3.5.1 Forcing Error Responses

The user can force a certain error response for a particular service during testing by passing the information to the simulation model using the "status.txt" file, which should be placed in the folder the simulation is run from. In order to force error responses to a certain service, the command and the required response should be typed in the same line in the following format: <Service Command><Response Expected>; where both are two hexadecimal digits. For example, to instruct the simulation model to generate an MSS memory access error response to the serial number service, the command is as follows.

Service: Serial Number: 01

Error message requested: MSS Memory Access Error: 7F

You should have the line 017F entered in "status.txt" file.

3.5.2 Parameter Setting

The "status.txt" file can also be used to set some parameters needed in simulation. As an example, in order to set the 32-bit parameter for the usercode, the format of the line must be in this order: <Service Command><32 Bit USERCODE>; where both values are entered in hexadecimal. In order to set the 128-bit parameter for the serial number, the format of the line must be in this order: <Service Command><128 Bit Serial Number [127:0]>; where both values are entered in hexadecimal. In order to set the 256-bit parameter for the SHA 256 key; the format of the line must be in this order: <Service Command><256 Bit Key [255:0]>; where both values are entered in hexadecimal. In order to set the 256-bit parameter for the challenge response key, the format of the line must be in this order: <Service Command><256 Bit Key [255:0]>; where both values are entered in hexadecimal.

3.5.3 Device Priority

Systems services and the COMM_BLK utilize a high priority system. Currently, the only high priority service is zeroization. In order to perform a high-priority service, while another service is being executed, the current service is halted and the higher priority service will be executed in its place. The COMM_BLK will discard the current service in order to perform the higher priority service. If multiple non-high-priority services are sent before the completion of a current service, these services will be queued within the TXFIFO. Once the current service is complete, the next service in the TXFIFO will be executed.

**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

© 2018 Microsemi. 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.

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.

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. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

50200837