

# Using Advanced Encryption Standard System Services in the SmartFusion2 and IGLOO2 Devices

## Table of Contents

References . . . . .	1
Tools Required . . . . .	1
Introduction . . . . .	2
Requirements and Details . . . . .	2
AES Engine . . . . .	3
SmartFusion2/IGLOO2 Cryptographic Block . . . . .	4
Using AES Services in the SmartFusion2/IGLOO2 Device . . . . .	6
Design Description . . . . .	9
CBC-MAC Example . . . . .	16
Design Example: CBC-MAC . . . . .	17
Conclusion . . . . .	19
Appendix A - Design and Programming Files . . . . .	20

## References

The following list of references is used in this document. The references complement and help in understanding the relevant Microsemi® SmartFusion®2 system-on-chip (SoC) field programmable gate array (FPGA) and IGLOO®2 FPGA device flows and features that are demonstrated in this document.

- [SmartFusion2 Microcontroller Subsystem User Guide](#)
- [SmartFusion2 System Controller User Guide](#)
- [SmartFusion2 Development Kit User Guide](#)
- [IGLOO2 FPGA High Performance Memory Subsystem User Guide](#)
- [IGLOO2 FPGA System Controller User Guide](#)
- [IGLOO2 Evaluation Kit User Guide](#)

## Tools Required

The application design provided with this application note uses Libero® System-on-Chip (SoC) v11.3.

## Introduction

Advanced encryption standard (AES) is an encryption solution that is developed to achieve many rapidly-evolving security concerns that have arisen within the computer and embedded semiconductor industries. Selected devices of the SmartFusion2 SoC FPGA and IGLOO2 FPGA families allow you to access the built-in AES engines and use AES encryption and decryption operation. These devices are marked as S (Data and Design Security) in the device part number. The AES engine in the SmartFusion2/IGLOO2 device is part of the Cryptographic Services block and resides in the system controller. The AES engine in the SmartFusion2/IGLOO2 device can accept 128-bit plaintext input word and generates the corresponding 128-bit ciphertext output word using a supplied 128-/256-bit AES key. It also provides a reverse function by generating plaintext from the supplied ciphertext using the same AES key as used for encryption. The AES engine is accessible through the system services. The system services are system controller actions initiated by asynchronous events from the ARM<sup>®</sup> Cortex<sup>™</sup>-M3 processor in the SmartFusion2 device or a fabric master in the SmartFusion2/IGLOO2 devices. The AES cryptographic services can be used for data security applications and can be disabled through factory or user security settings.

This application note provides the design example for using the AES encryption and decryption in the SmartFusion2/IGLOO2 device. It also shows a cryptography application example for using AES encryption and generating CBC-MAC using the SmartFusion2/IGLOO2 device.

## Requirements and Details

**Table 1 • SmartFusion2 Reference Design Requirements and Details**

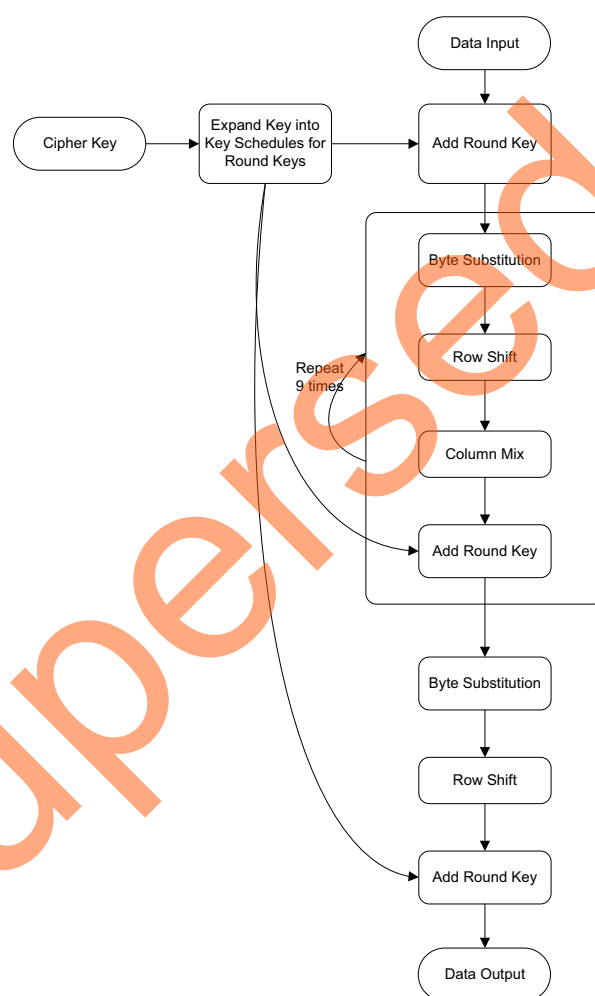
Reference Design Requirements and Details	Description
<b>Hardware Requirements</b>	
SmartFusion2 Development Kit <ul style="list-style-type: none"><li>12 V adapter (provided along with the kit)</li><li>FlashPro4 programmer (provided along with the kit)</li><li>M2S050ST-1FGG896</li></ul>	Rev C or later
Host PC or Laptop	Any 64-bit Windows Operating System
<b>Software Requirements</b>	
Libero SoC	11.3
SoftConsole	3.4

**Table 2 • IGLOO2 Reference Design Requirements and Details**

Reference Design Requirements and Details	Description
<b>Hardware Requirements</b>	
IGLOO2 Evaluation Kit <ul style="list-style-type: none"><li>12 V Wall-Mounted Power Supply (provided along with the kit)</li><li>FlashPro4 programmer (provided along with the kit)</li><li>M2GL010TS-1FGG484</li></ul>	Rev C or later
Host PC or Laptop	Any 64-bit Windows Operating System
<b>Software Requirements</b>	
Libero SoC	11.3

## AES Engine

The AES engine uses the Rijndael algorithm with National Institute of Standards and Technology (NIST) approved parameters as described in Federal Information Processing Standards (FIPS) Publication (PUB) 197. Figure 1 shows the AES encryption algorithm with 128-bit key. The AES encryption algorithm receives 128-bit of plaintext data and 128-/192-/256-bit of a cipher key as input. After several rounds of computation, it produces 128-bit enciphered version of the original plaintext data as output. The key size used for an AES cipher determines the number of transformation rounds to convert the input into the final output, ciphertext. During the rounds of the algorithm, the data bits are subjected to byte substitution, data shift operations, data mixing operations, and additional (that is, XOR) operations with an expanded version of the original 128-/192-/256-bit cipher key. The reverse happens during the decryption operation. The SmartFusion2/IGLOO2 AES engine can be operated in 128-bit key mode or 256-bit key mode and supports both encryption and decryption services.



**Figure 1 • AES Encryption Algorithm (128-bit Cipher Key)**

The SmartFusion2/IGLOO2 AES engine assumes that the input data is in complete 128-bit blocks and provides the complete 128-bit output blocks. You must add any padding bits to incomplete plaintext blocks before calling the AES encryption service and remove any padding bits after receiving the results of the AES decryption service. The input and output data format of the AES services is little-endian type. The first byte of the first block is at the lowest address and there are no word alignment requirements. In other words, consecutive bytes of the plaintext, ciphertext, and keys from the first to last are stored in order in memory from the lowest to the highest bit address.

## AES Mode of Operation

The mode of operation describes how to apply the ciphers single-block operation repeatedly to securely transform the data that is larger than a block. The built-in system services are designed to support the following cipher operating modes as recommended in NIST Special Publication 800-38A, recommendation for Block Cipher Modes of Operation:

1. [Electronic Codebook](#)
2. [Cipher-Block Chaining](#)
3. [Output Feedback](#)
4. [Counter](#)

### **Electronic Codebook**

The electronic codebook (ECB) mode is a confidentiality mode that features, for a given key, the assignment of a fixed ciphertext block to each plaintext block, analogous to the assignment of code words in a codebook. It is the simplest encryption mode. The message is divided into blocks and each block is encrypted separately. Identical plaintext blocks are encrypted into identical ciphertext blocks; thus, it does not hide data patterns well.

### **Cipher-Block Chaining**

The cipher block chaining (CBC) mode features the combination (chaining) of the plaintext blocks with the previous ciphertext blocks. To make each message unique, an initialization vector (IV) must be used in the first block. The IV need not to be secret, but it must be unpredictable.

### **Output Feedback**

The output feedback (OFB) features the iteration of the forward cipher on an IV to generate a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext and vice-versa. The OFB mode requires a nonce IV, that is, the IV must be unique for each execution of the mode under the given key.

### **Counter**

The counter (CTR) mode features the application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext and vice-versa. The sequence of counters must have the property that each block in the sequence is different from every other block, thus the IV should be a nonce and must be unique for each execution of the mode under the given key.

In the SmartFusion2/IGLOO2 device, the OPMODE parameter specifies the cipher operating mode, refer to [Table 3 on page 7](#). The IV parameter used during the AES system service specifies the IV. Refer to ["Using AES Services in the SmartFusion2/IGLOO2 Device" section on page 6](#) for more information.

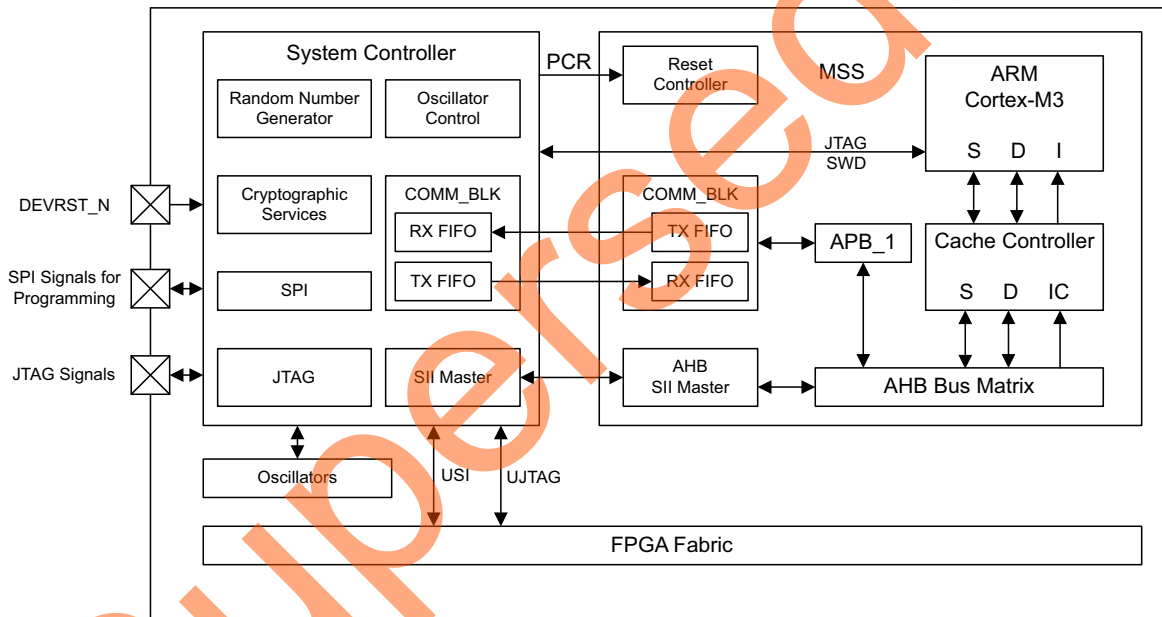
## SmartFusion2/IGLOO2 Cryptographic Block

In the SmartFusion2/IGLOO2 device, the AES engine is part of this Cryptographic Services block that resides in System Controller.

## System Controller Block in the SmartFusion2 Device

The Cryptographic Services block can be accessed through the communication block (COMM\_BLK). There are two communication block (COMM\_BLK) instances: one in the microcontroller subsystem (MSS) that the user interfaces with and the other that communicates with the first block that is located in the system controller. The COMM\_BLK consists of an APB interface, eight byte transmit FIFO, and eight byte receive FIFO. The COMM\_BLK provides a bi-directional message passing facility between the MSS and the system controller. The AES system services are initiated using the COMM\_BLK in the MSS, which can be read or write by any master on the AMBA high performance bus (AHB) matrix; typically either the Cortex-M3 processor or a design in the FPGA fabric (also known as a fabric master). The system controller receives the command through the COMM\_BLK in the system controller. The system controller uses the SII master, an MSS bus master controlled by the system controller, to get the additional details and options of the AES command at an address provided in the original COMM\_BLK command, pointing where this structured data has been stored in the memory before invoking the command. The AES output bytes returned by the system controller are written to a memory address specified in this data structure. On completion of the requested service, the system controller returns a status message through the COMM\_BLK.

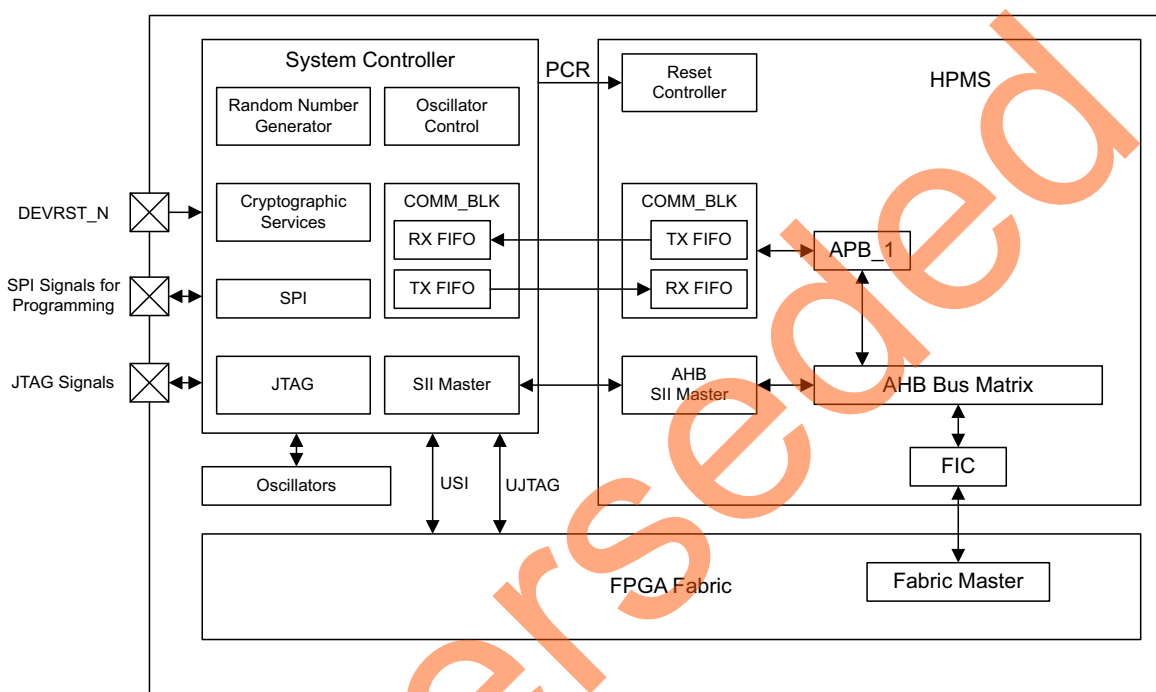
Figure 2 shows the system controller block in the SmartFusion2 device, where the Cryptographic Services block resides.



**Figure 2 • System Controller Block in the SmartFusion2 Device**

## System Controller Block in the IGLOO2 Device

The architecture and uses of the AES engine is similar in IGLOO2 device except the COMM\_BLK in the system controller communicated with COMM\_BLK in high performance memory subsystem (HPMS). It is required to use a fabric master to initiate the AES system services. Microsemi provides the CoreSysService Directcore IP that acts as fabric master to use the AES system services. The CoreSysServices soft IP communicates with the COMM\_BLK through one of the fabric interface controllers (FICs), sends the AES system service request, retrieves the cyphertext output, and sends to use the interface. Figure 3 shows the system controller block in the IGLOO2 device.



**Figure 3 • System Controller Block in the IGLOO2 Device**

Refer to *SmartFusion2 System Controller User Guide* and *IGLOO2 FPGA System Controller User Guide* for more details on the system controller. Also, refer to the Communication Block chapter in the *SmartFusion2 Microcontroller Subsystem User Guide*, *IGLOO2 FPGA High Performance Memory Subsystem User Guide* for more details on the COMM\_BLK.

## Using AES Services in the SmartFusion2/IGLOO2 Device

In the SmartFusion2 device, the AES services can be accessed using the `mss_sys_services` driver in the firmware core configurator. In the IGLOO2 device, you must use a master in fabric to initiate the AES system services in the system controller through the COMM\_BLK. You can create any fabric master block following the steps explained below or use CoreSysServices soft IP for the AES services. CoreSysServices provides a simple user interface in one side and an AHB-Lite master interface on the FIC side to use the system services through the COMM\_BLK. You can use the IGLOO2 approach in the SmartFusion2 device too. Thus, you have two options to use the AES engine in the SmartFusion2 device:

1. Using firmware core through the MSS
2. Using CoreSysService or own state logic as fabric master

The following section describes the steps for using the 128-bit AES encryption system service in the IGLOO2 device:

1. Set up the AES128DATAPTR descriptor in the user memory space as shown in [Table 3](#), containing the following 44 bytes:

**Table 3 • AES128DATAPTR Structure**

Offset	Length (Bytes)	Field	Description
0	16	KEY	Encryption key to be used
16	16	IV	IV (Ignored for ECB mode)
32	2	NBLOCKS	Number of 128-bit blocks to process (maximum 65535)
34	1	MODE	<p>Cipher operating mode.</p> <ul style="list-style-type: none"> <li>• Bit 7: DECRYPT</li> <li>• Bit 6: RESERVED</li> <li>• Bit 1: OPMODE</li> <li>• Bit 0: OPMODE</li> </ul> <p>DECRYPT: If DECRYPT is 0 then the data at SRCADDRPTR field is treated as plaintext for encryption. If DECRYPT is 1 then the data at SRCADDRPTR field is treated as cipher text for decryption.</p> <p>OPMODE: Defines operating mode.</p> <ul style="list-style-type: none"> <li>• 00: ECB mode</li> <li>• 01: CBC mode</li> <li>• 10: OFB mode</li> <li>• 11: CTR mode</li> </ul>
35	1	RESERVED	Reserved
36	4	DSTADDRPTR	Pointer to return data buffer
40	4	SRCADDRPTR	Pointer to data to encrypt/decrypt

2. Enable the COMBLK\_INTR interrupt from the COMM\_BLK block to fabric by enabling COMBLK\_INTR\_ENBL bit (29-bit) in INTERRUPT\_ENABLE0 register at address 0x40006000.
3. Setup the registers in the COMM\_BLK and send the command for 128-bit AES (0x03). [Table 4](#) describes the AES command values.

**Table 4 • AES Command Value**

System Service Name	Command Value
128-bit AES Cryptographic Service	3
256-bit AES Cryptographic Service	6

The system controller receives the command through the COMM\_BLK in the system controller. The system controller reads the key and data from the address pointer and generates the AES ciphertext test. On completion, the service system controller returns a status message through the COMM\_BLK.

Wait for RCVOKAY bit to be set in the COMM\_BLK STATUS register.

4. Read the Word Data register in the COMM\_BLK and check the command, status code, and AES128DATAPTR descriptor pointer as shown in [Table 5 on page 8](#).

**Table 5 • 128-bit AES Service Response**

Offset	Length (Bytes)	Field	Description
0	1	CMD = 3	Command
1	1	STATUS	Command status
1	4	AES128DATAPTR	Pointer to AES128DATA descriptor

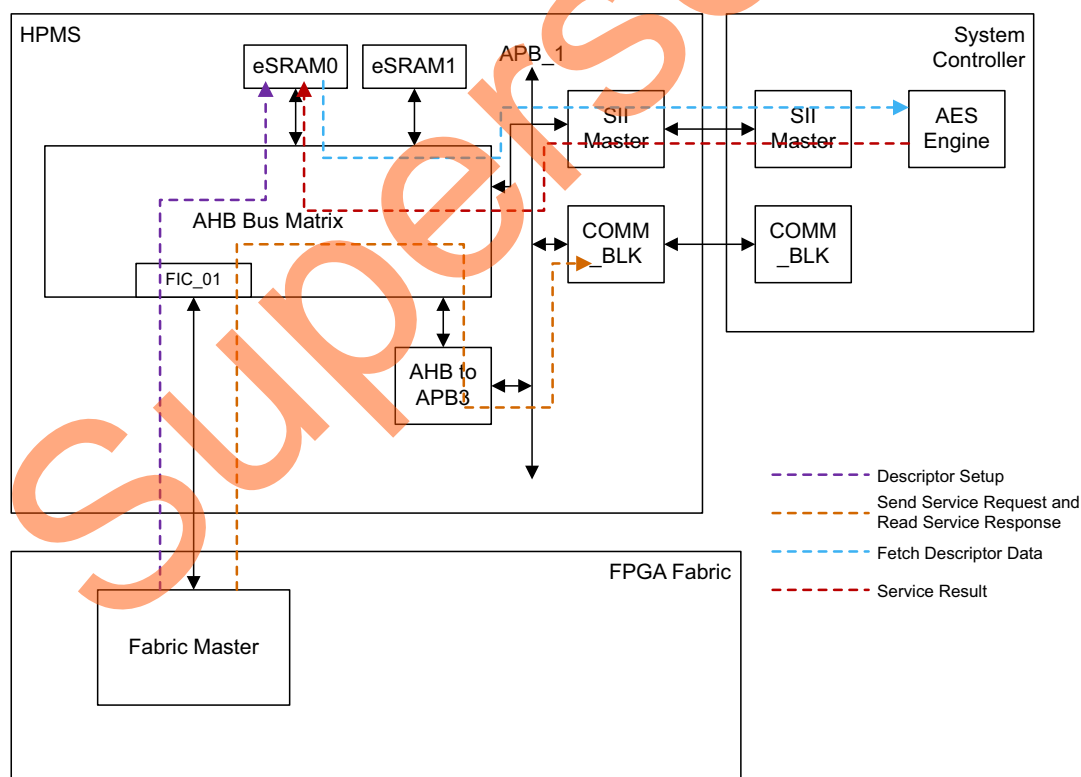
Table 6 shows the service response status codes:

**Table 6 • Service Response Status Codes**

Status	Description
0	Success
127	HRESP error occurred during the MSS transfer
253	Not licensed
254	Service disabled by factory security
255	Service disabled by user security

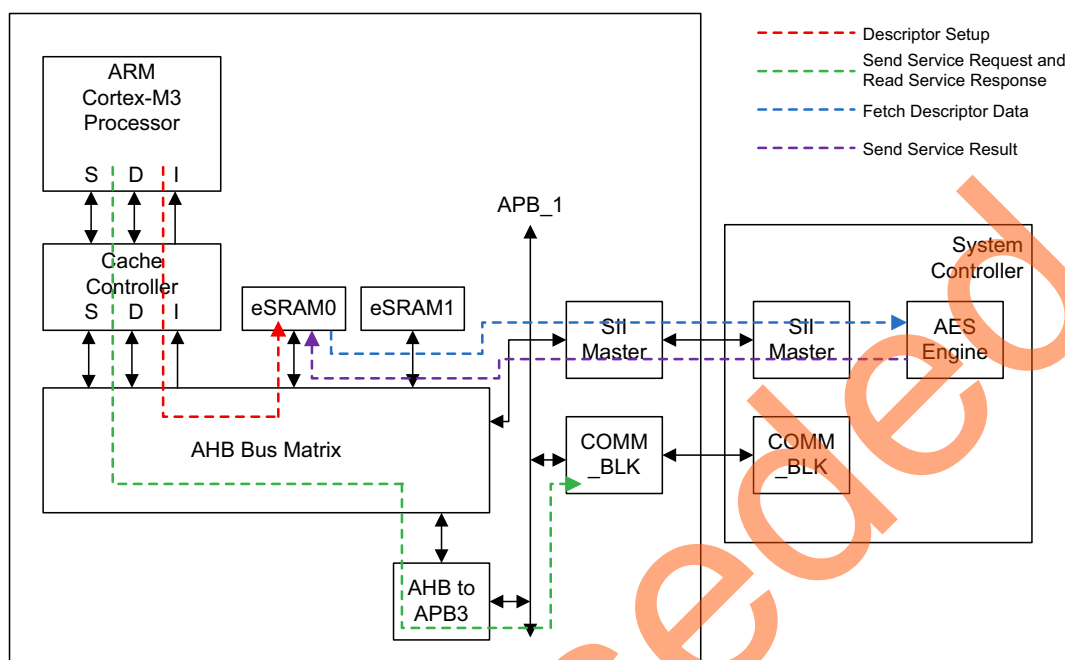
5. Read the AES data from user memory space (at the return, data buffer address is specified in 1).

Figure 4 shows the AES system service data flow diagram in the IGLOO2 device.


**Figure 4 • AES Service Flow Diagram in the IGLOO2 Device**



The steps for using the AES service in 256-bit key mode are similar to SmartFusion2. Figure 5 shows the system service data flow diagram while initiating the AES service from the Cortex-M3 processor.



**Figure 5 • AES Service Flow Diagram in the SmartFusion2 Device**

**Note:** You can use CoreSysServices IP in the SmartFusion2 or IGLOO2 devices and initiate the AES system service using its simple user interface. You just need to send the AES service request with the required data/parameter to CoreSysServices IP. CoreSysServices IP performs the required steps to setup the descriptor, sends the command through the COM\_BLK to the AES service, and reads the data back from the eSRAM. Refer to the CoreSysServices Handbook for more information.

## Design Description

This application note includes two design examples for using the AES system service:

- **AES\_Services\_SF2** design example: Demonstrates 128-bit and 256-bit AES encryption and decryption in the SmartFusion2 device using the system driver firmware code supplied by Microsemi®.
- **AES\_Services\_IGL2** design example: Demonstrates 128-bit AES encryption and decryption in the IGLOO2 device using the CoreSysServices IP core supplied by Microsemi.

The SmartFusion2 device design is implemented on the SmartFusion2 Development Kit Board using an M2S050TS-FG896 device and the IGLOO2 device design is implemented on the IGLOO2 Evaluation Kit Board using an M2S010TS-FG484 device.

### Design Example: Using AES Services in the SmartFusion2 Device

The design example consists of the RC oscillator, a fabric CCC, and the MSS. The fabric PLL is used to provide the base clock for the MSS. The system services are run using various C routine in the MSS, as shown below. In addition, a universal asynchronous receiver/transmitter (UART1) in the MSS is used to display the operation of the AES system service.

## Hardware Implementation

Figure 6 shows a block diagram of the design example. The RC oscillator is used to generate a 50 MHz input clock and the fabric PLL is used to generate a 100 MHz clock from the RC oscillator. This 100 MHz clock is used as the base clock for the MSS. The MMUART\_1 signals are routed through the FPGA fabric for communicating with the serial terminal program. The counter block is used to show that the device is up and running.

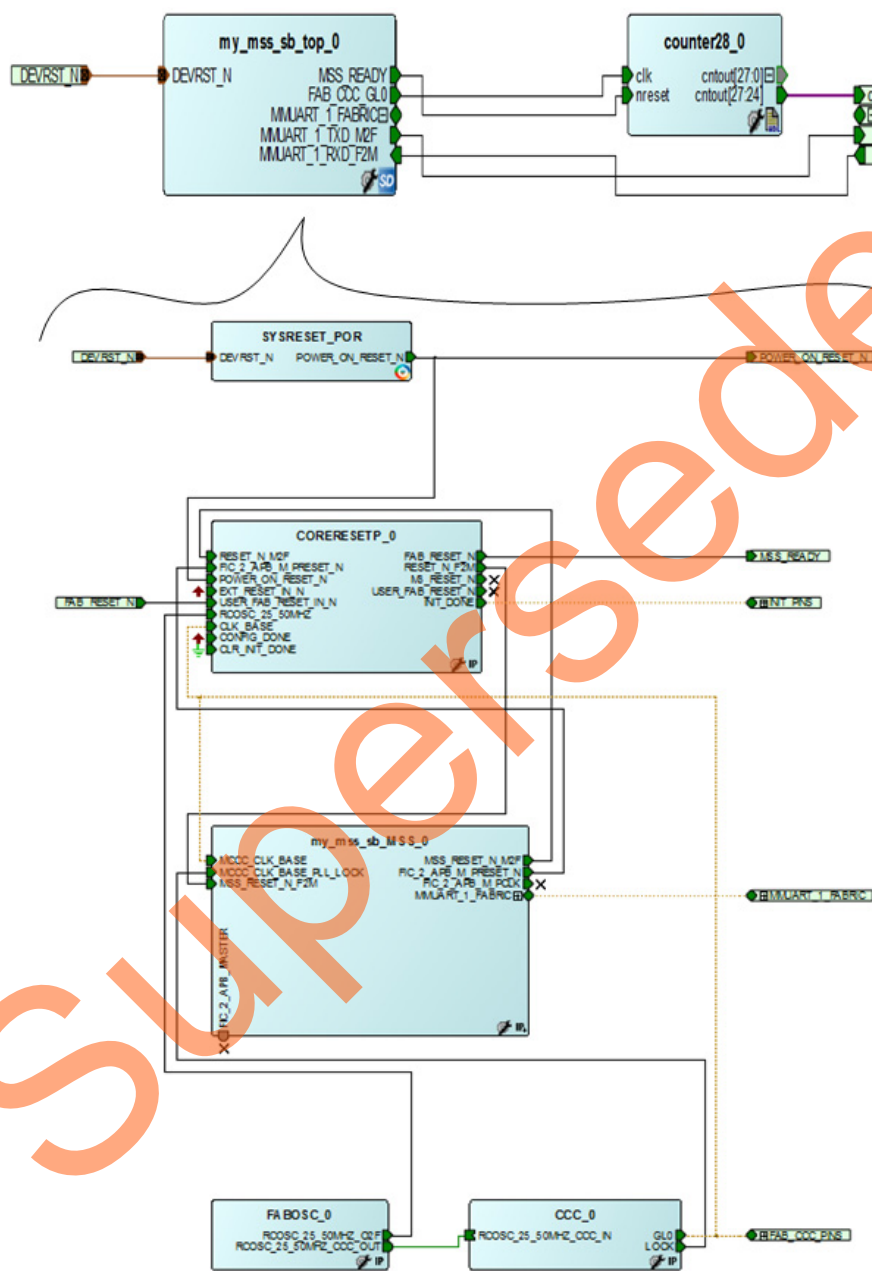


Figure 6 • SmartFusion2 AES System Service Design Example

## Software Implementation

The software design example performs the following operations:

1. Initialize the System Controller Enable
2. Initialize MMUART\_1
3. Perform various cryptography services:
  - Encrypt with 128-bit AES cryptography service
  - Decrypt with 128-bit AES cryptography service
  - Encrypt with 256-bit AES cryptography service
  - Decrypt with 256-bit AES cryptography service

### **aes128\_encryption ();**

The `aes128_encryption()` function provides access to the SmartFusion2 AES-128 encryption cryptography service. It allows you to perform AES encryption and choose the mode of operation: ECB, CBC, OFB, or CTR mode. It allows you to specify the number of 128-bit blocks of plaintext to be processed by the AES-128 system service.

### **aes128\_decryption ();**

The `aes128_decryption()` function provides access to the SmartFusion2 AES-128 decryption cryptography service. It allows you to perform AES decryption and choose the mode of operation: ECB, CBC, OFB, or CTR mode. It allows you to specify the number of 128-bit blocks of ciphertext to be processed by the AES-128 system service.

### **aes256\_encryption ();**

This function is similar to `aes128_encryption()` and provides access to the SmartFusion2 AES-256 encryption cryptography service function using the 256-bit key.

### **aes256\_decryption ();**

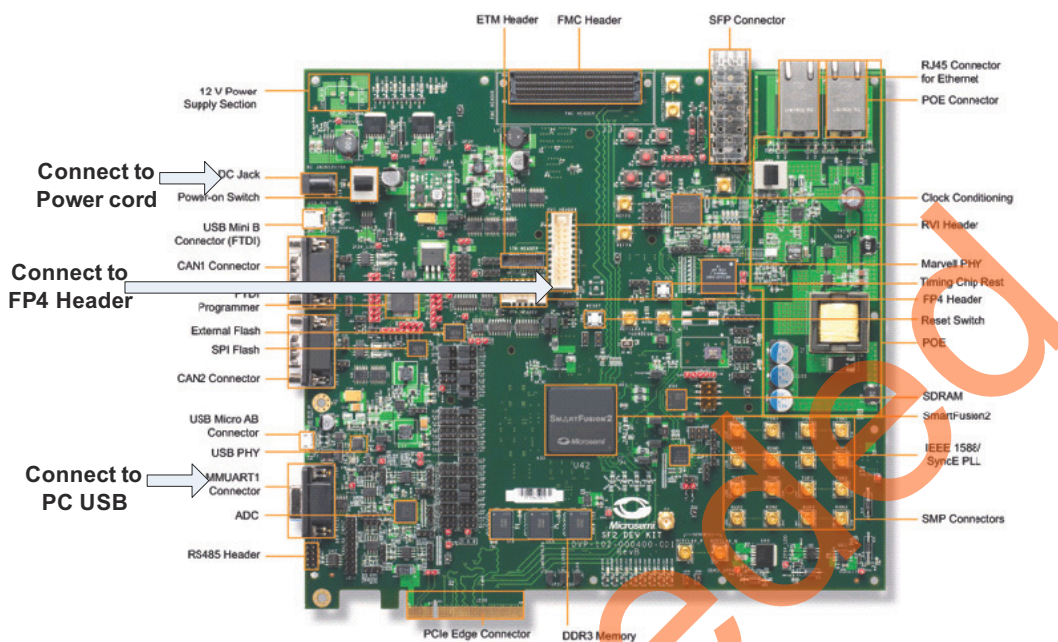
This function is similar to `aes128_decryption()` and provides access to the SmartFusion2 AES-256 decryption cryptography service function using the 256-bit key.

## Running the Design

This section describes running the design example on the SmartFusion2 Development Kit board using the M2S050TS-FG896 device:

1. Connect the power supply to the SmartFusion2 Development Kit board.
2. Plug the FlashPro4 ribbon cable into JTAG Programming Header on the SmartFusion2 Development Kit board.
3. Program the SmartFusion2 Development Kit board with the provided STAPL file (refer to ["Appendix A - Design and Programming Files" on page 20](#)) using FlashPro4.

4. Connect one end of the USB Mini-B cable to the J24 connector provided in the SmartFusion2 Development Kit board. Connect the other end of the USB cable to the host PC.



**Figure 7 • SmartFusion2 Development Kit Board**

5. Invoke the SoftConsole Integrated Design Environment (IDE) from the Libero SoC software project and launch the debugger.
6. Start a HyperTerminal session with 57600 baud rate, 8 data bits, 1 stop bit, no parity, and no flow control. If the computer does not have the HyperTerminal program, any free serial terminal emulation program such as PuTTY or Tera Term can be used. Refer to [Configuring Serial Terminal Emulation Programs Tutorial](#) for configuring HyperTerminal, TeraTerm, or PuTTY.

- Run the debugger in the SoftConsole tool. The HyperTerminal window shows various options to run the AES encryption and decryption. Follow the instruction on the screen to run the example. [Figure 8](#) shows the HyperTerminal.

```

COM14:57600baud - Tera Term VT
File Edit Setup Control Window Help

*****
***** SmartFusion2 Cryptography System Services Example *****
*****

This example project demonstrates the use of the SmartFusion2 cryptography
System Services. The following system service are demonstrated:
- AES-128 encryption and decryption.
- AES-256 encryption and decryption.

-----

Select the Cryptographic operation to perform:
Press Key '1' to perform AES-128 encryption
Press Key '2' to perform AES-128 decryption
Press Key '3' to perform AES-256 encryption
Press Key '4' to perform AES-256 decryption

-----

Enter the 128 bit ASCII hex key to be used for AES:
aaaaaaaaaaaaaaaa
1122334455667788

-----

Enter the 128 bit initialization vector(IV) to be used for AES:
aaaaaaaaaaaaaaaa
0000000000000000

-----

Select AES modes:
- ECB: Electronic CodeBook Mode.      Press key 'A'
- CBC: Cipher Block Chaining Mode.    Press key 'B'
- OFB: Output Feedback Mode.          Press key 'C'
- CTR: Counter Mode.                  Press key 'D'

-----

Enter the 32 bytes of input data/text to encrypt:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1111111111111111111111111111

-----

Encrypted data:
8e d6 21 01 52 58 5c 99 46 1f c7 d0 10 b5 32 a2

-----

Press any key to continue.

```

**Figure 8 • AES System Service Design Example using HyperTerminal**

**Note:** The ASCII-Hex notation is used for input by the program so the data is more easily readable.

The data goes from the first byte to the last byte of the multi-byte message, IV, or key, etc. entered or displayed from left to right (and then top to bottom, if multi-line) as shown by the terminal emulator. Each byte is represented by two ASCII characters selected by value from the ordered sixteen character set 0-9 and a-f with the leftmost ASCII character representing the first four bits of the byte (that is, bits 7:4) encoded into a hexadecimal digit having its first binary bit (bit 7) interpreted as the most significant bit, and then the resulting hexadecimal digit encoded into an 8-bit ASCII character; the rightmost ASCII character representing the following four bits (bits 3:0) are encoded with the last binary bit of the byte (bit 0) being interpreted as the least significant of the second hexadecimal digit. The AES output is the Hex data displayed in endian order.



## Design Example: Using AES Services in the IGLOO2 Device

The design consists of the IGLOO2 HPMS, the on-chip 50 MHz RC oscillator, a Fabric CCC, the CoreSysServices IP block, the CoreRESET IP block, a CoreABC IP block, a CoreUART\_apb IP block, a fabric state machine to control the CoreSysServices block, and an APB data block to reformat the AES output so it can be displayed by a terminal emulator.

### Hardware Implementation

Figure 9 shows the block diagram of the design example. The 50 MHz RC oscillator is used as the main clock. It is used with a CCC to provide the 100 MHz reference clock to the HPMS. This 100 MHz clock is also used as main clock for the fabric blocks. The HPMS is configured to use the CoreResetP block to generate reset signals for all the blocks. The CoreSysServices IP is configured to use the AES system services. It sends a command to the system controller through COMM block in the HPMS. The fabric SysService state control logic initiates the AES system service and captures the AES data from CoreSysService. The fabric SysService state block sends the plaintext AES data that (in the example design) is basically a big-endian binary counter that keep increments the AES plaintext after every AES encryption operation. The incremented value is used as the input for the next encryption operation. The fabric SysService state block uses the most recent ciphertext AES data that was calculated as input for the decryption operation. The UART controller block is mainly used to display the AES output to HyperTerminal; it is not required for the AES operation. The APB data block captures the AES data values and converts the binary data to ASCII Hexadata to display in human readable format on the HyperTerminal. The CoreABC program controls initiating fabric state machine and displaying the data through the CoreUartApb interface.

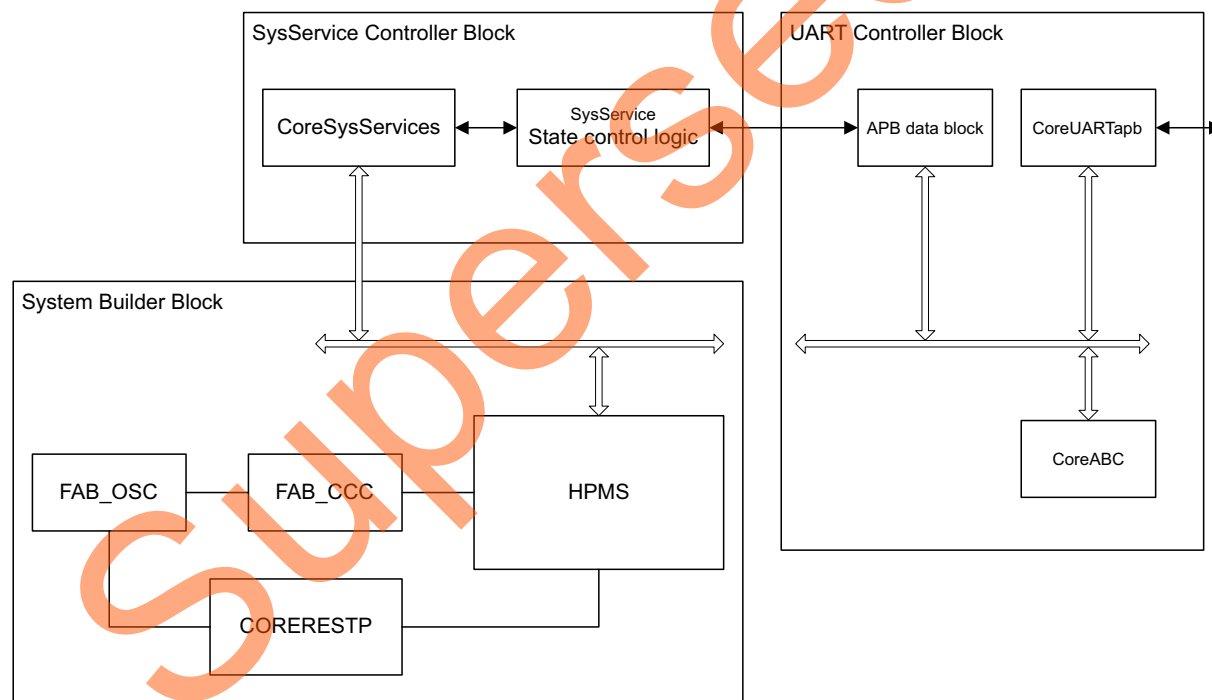


Figure 9 • IGLOO2 AES System Service Design Example

## Running the Design

This section describes running the design example in the IGLOO2 Evaluation Kit board using the M2GL010TS-FG484 device:

1. Connect the power supply to the SmartFusion2 Development Kit board.
2. Plug the FlashPro4 ribbon cable into connector J5 (JTAG Programming Header) on the IGLOO2 Evaluation Kit board.
3. Connect the mini USB cable between the FlashPro4 and the USB port of the PC.
4. Connect one end of the USB mini-B cable to the J18 connector provided on the IGLOO2 Evaluation Kit board. 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).
5. 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](http://www.microsemi.com/soc/documents/CDM_2.08.24_WHQL_Certified.zip).
6. Start a HyperTerminal session with 57600 baud rate, 8 data bits, 1 stop bit, no parity, and no flow control. If the computer does not have the HyperTerminal program, any free serial terminal emulation program such as PuTTY or Tera Term can be used. Refer to [Configuring Serial Terminal Emulation Programs Tutorial](#) for configuring HyperTerminal, Tera Term, or PuTTY.
7. Program the IGLOO2 Evaluation Kit board with the provided STAPL file (refer to "Appendix A - Design and Programming Files" on page 20) using FlashPro4.

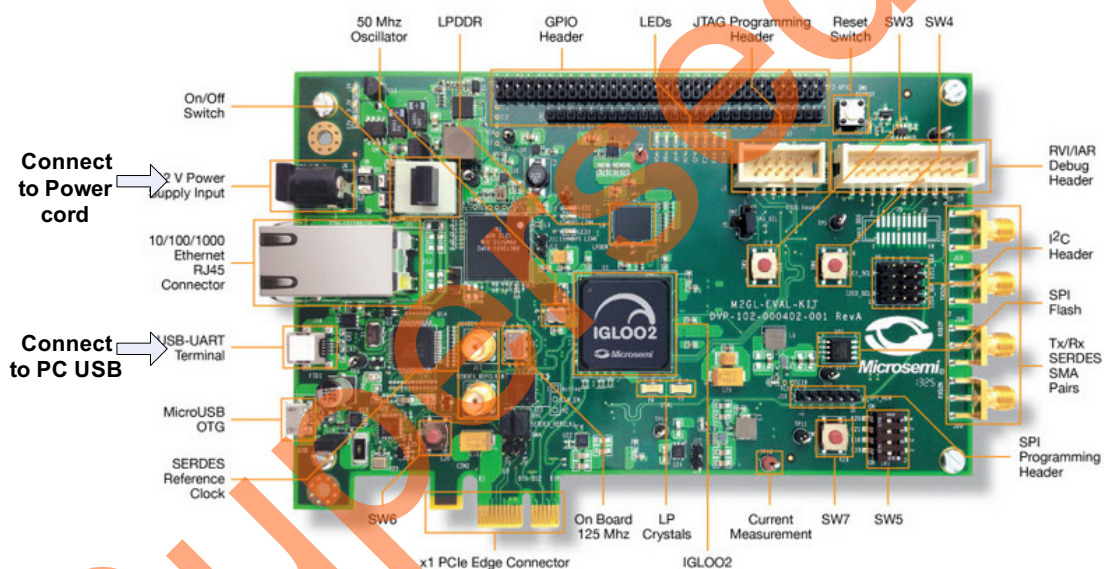
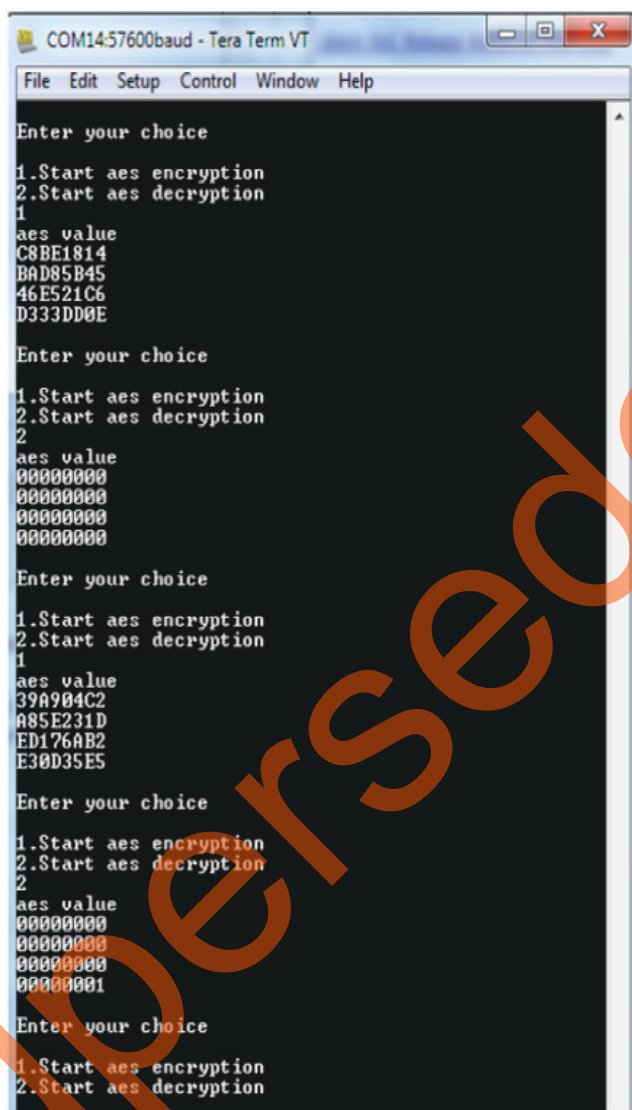


Figure 10 • IGLOO2 Evaluation Kit Board

After programming, HyperTerminal displays a message to run the AES system services as shown in Figure 11.



```
COM14:57600baud - Tera Term VT
File Edit Setup Control Window Help

Enter your choice
1.Start aes encryption
2.Start aes decryption
1
aes value
C8BE1814
BAD85B45
46E521C6
D333DD0E

Enter your choice
1.Start aes encryption
2.Start aes decryption
2
aes value
00000000
00000000
00000000
00000000

Enter your choice
1.Start aes encryption
2.Start aes decryption
1
aes value
39A904C2
A85E231D
ED176AB2
E30D35E5

Enter your choice
1.Start aes encryption
2.Start aes decryption
2
aes value
00000000
00000000
00000000
00000001

Enter your choice
1.Start aes encryption
2.Start aes decryption
```

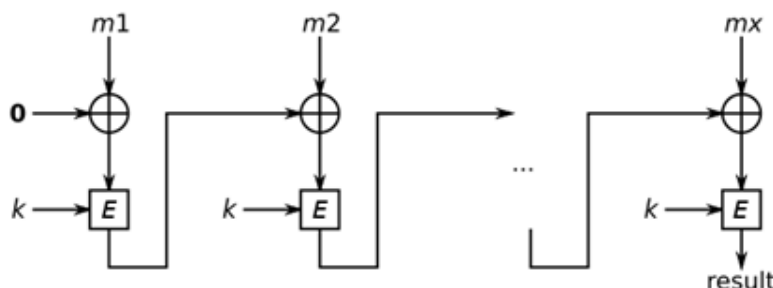
Figure 11 • HyperTerminal Showing CoreSysService Design Output

## CBC-MAC Example

In cryptography, a cipher block chaining message authentication code (CBC-MAC) is a technique for constructing a message authentication code from a block cipher. It uses AES encryption in CBC mode.



Figure 12 shows the technique for CBC-MAC. The IV used in first block is zero. Then a chain of blocks is created as each block depends on the proper encryption of the previous block.



**Figure 12 • CBC-MAC of a Message**

This application note shows the design example of generating CBC-MAC in the SmartFusion2 and IGLOO2 device.

**Note:** The CBC-MAC design example uses message size that is exact multiple of 128 bits, and that it must not be used in a production environment without careful review by a qualified cryptographer.

## Design Example: CBC-MAC

This section describes the CBC-MAC application design example. The CBC-MAC design is implemented in both the SmartFusion2 and IGLOO2 devices.

- CBC\_MAC\_SF2 design example: Demonstrates using CBC-MAC in the SmartFusion2 device. It uses firmware code to generate CBC-MAC.
- CBC\_MAC\_IGL2 design example: Demonstrates using CBC-MAC in the IGLOO2 device. It uses CoreSysServices IP to generate CBC-MAC.

### Design Example: Using CBC\_MAC in SmartFusion2

This design example is similar to AES\_Services\_SF2 design example. It uses same hardware implementation and uses UART1 in the MSS to display the CBC-MAC operation.

#### Software Implementation

The software design example performs the following operations:

1. Initializes the System Controller Enable
2. Initializes MMUART\_1
3. Performs CBC-MAC

**cbc\_mac ();**

The **cbc\_mac()** function allows to run CBC-MAC in the SmartFusion2 AES-128 device. It allows you to enter messages with variable length, perform CBC-MAC operation, and display the result.

## Running the Design

This section describes running the CBC-MAC design example in the SmartFusion2 Development Kit using the M2S050TS-FG896 device. Use AES\_Services\_SF2 design example steps to program the device and open HyperTerminal. Then, invoke the SoftConsole IDE from the Libero SoC software project and launch the debugger. Run the demo design as shown in Figure 13.

```

COM10:57600baud - Tera Term VT
File Edit Setup Control Window Help

*****
***** SmartFusion2 Cryptography System Services Example *****
*****
This example project demonstrates using CBC_MAC SmartFusion2

-----
Select the Cryptographic operation to perform:
Press Key '1' to perform CBC_MAC

Enter the 128 bit ASCII hex key to be used for AES:
00000000000000000000000000000000
123456789abcdef0
Enter the number of messages for CBC-MAC :
4

Enter Messages

00000000000000000000000000000000
11111111111111111111111111111111
-----
00000000000000000000000000000000
22222222222222222222222222222222
-----
00000000000000000000000000000000
33333333333333333333333333333333
-----
00000000000000000000000000000000
44444444444444444444444444444444
-----
Encrypted data output:
c9 62 fc 6a 73 06 50 40 8e 4d 93 92 54 9c b3 cd

Encrypted data output:
c4 af 90 d0 18 8e 88 80 1b 40 a5 d1 da 94 7c 21

Encrypted data output:
60 60 e3 12 c9 a9 6b 8a 75 21 00 89 35 4e 6e f8

Encrypted data output:
15 bc e4 26 ec 62 42 7c 4c 0d 36 5f ee 08 52 9d

*****
CBC-MAC data:
15 bc e4 26 ec 62 42 7c 4c 0d 36 5f ee 08 52 9d
*****
Press any key to continue.

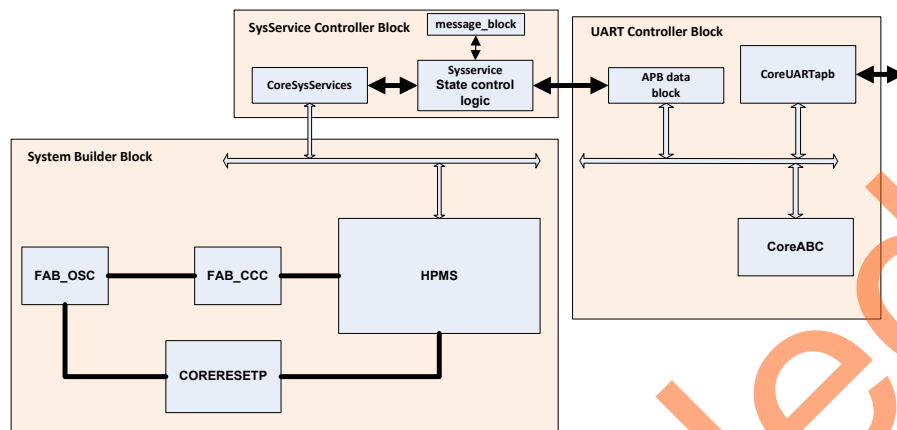
```

Figure 13 • HyperTerminal showing CBC-MAC Design in the SmartFusion2 Device

## Design Example: Using CBC\_MAC in the IGLOO2 Device

This design example is similar to the AES\_Services\_IGL2 design example. Figure 14 on page 19 shows the block diagram of CBC-MAC design. The fabric system service state control logic configures CoreSysService to generate AES CBC mode. It also sends the appropriate IV during each AES services. The design example uses a message block that sends the messages for AES operation. The message block uses four messages in the current implementation. One of the messages is tied to DIP switch in the IGLOO2 Evaluation Kit. You can change the DIP switch and change the message.

**Note:** You can modify the message block, content, and size. However, you need to change the counter in SysService state control logic to match the message length. The other blocks are similar to the AES\_Services\_IGL2 design example.



**Figure 14 • IGLOO2 CBC-MAC Design Example**

### Running the Design

This section describes running the CBC-MAC design example in the IGLOO2 Evaluation Kit using the M2GL010TS-FG484 device. Use AES\_Services\_IGL2 design example steps to program the device and open HyperTerminal. Run the demo design as shown in Figure 15.

```
COM10:57600baud - Tera Term VT
File Edit Setup Control Window Help
Enter your choice
1.Start cb-mac encryption
1
cb-mac value
460D3594
D547FD1D
0C1E165F
584B61B1
Enter your choice
1.Start cb-mac encryption
```

**Figure 15 • CBC-MAC of a Message**

## Conclusion

The SmartFusion2 and IGLOO2 family of FPGAs are the most secure programmable logic devices ever made. In selected SmartFusion2 and IGLOO2 devices, the AES engine can perform encryption or decryption on 128-bit blocks of user data using either 128-bit or 256-bit keys as defined in NIST FIPS 197. Several common modes are provided to encrypt or decrypt arbitrarily sized blocks of data, including ECB, CBC, OFB, and CTR modes as defined in NIST SP800-38a. The AES system services, along with the other cryptographic services offered, allow you to use the SmartFusion2 and IGLOO2 devices in various secure applications.

## Appendix A - Design and Programming Files

You can download the SmartFusion2 and IGLOO2 AES design files from the Microsemi Corporation website: [http://soc.microsemi.com/download/rsc/?f=SF2\\_IGL2\\_AES\\_DF](http://soc.microsemi.com/download/rsc/?f=SF2_IGL2_AES_DF)

The SmartFusion2 design file consists of a Libero Verilog project, SoftConsole software project, and programming files (\*.stp) for the SmartFusion2 Development Kit. The IGLOO2 design file consists of a Libero Verilog project and programming files (\*.stp) for the IGLOO2 Evaluation Kit. Refer to the Readme.txt file included in the design file folder for the directory structure and description.

You can download the SmartFusion2 and IGLOO2 CBC-MAC design files from the Microsemi Corporation website: [http://soc.microsemi.com/download/rsc/?f=SF2\\_IGL2\\_CBC\\_MAC\\_DF](http://soc.microsemi.com/download/rsc/?f=SF2_IGL2_CBC_MAC_DF)

The SmartFusion2 design file consists of a Libero Verilog project, SoftConsole software project, and programming files (\*.stp) for the SmartFusion2 Development Kit. The IGLOO2 design file consists of a Libero Verilog project and programming files (\*.stp) for the IGLOO2 Evaluation Kit. Refer to the Readme.txt file included in the design file folder for the directory structure and description.

Superseded

Superseded



**Microsemi**<sup>®</sup>

**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](mailto: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](http://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.