AES Implementation on Altera FPGA using Nios-II Soft core Processor

For embedded applications, which typically have a short product life span, time to market is crucial. Field Programmable Gate Arrays (FPGA's) are essential components to obtain a short design time. Compared to a full custom ASIC design, they are cost efficient, easier to manage and can immediately be put into operation. Furthermore, the reprogramability of FPGA's will maximize the product life span and enhance the reliability.

Altera offers a system hardware solution on single chip using FPGA and 32-bit embedded soft core processor, Nios-II at a very short design time. Visit http://www.altera.com/literature/lit-nio2.jsp for more information on Nios-II processor.

This paper demonstrates AES implementation on Altera's Cyclone development board using Nios-II soft processor. We assume that reader is known to Rijndael algorithm and just brief the same below. More information on this algorithm can be found on NIST website, http://www.nist.gov/aes/rijndael.

AES

The US National Institute of Standards and Technology (NIST) selected Rijndael algorithm as a new Advanced Encryption Standard (AES). Rijndael is an iterated block cipher which supports variable block length and key length. Both lengths can be independently specified as 128, 192 or 256 bits. Rijndael has a variable number of iterations: 10, 12 and 14 for key lengths of 128, 192 and 256 respectively.

Transformations in Rijndael treat the AES standard 128-bit data & key block as a 4 column rectangular array of 4-byte vectors.

128-bit Key AES Encryption

The 128-bit key Rijndael encryption algorithm consists of an initial data/key addition, then 9 round transformations followed by a final round. The Key Schedule expands the key entering the cipher so that a different round key is created for each iteration, as shown in Fig.1 below.

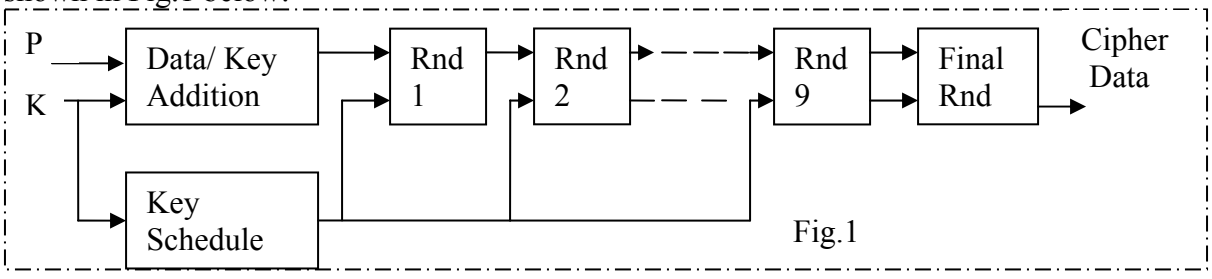

Fig.1

128-bit AES IP core written in Verilog HDL is implemented using Altera Quartus-II design software target to Cyclone device. The design files are listed below,

top_data.v (top-level module), top_key.v, datablock.v, dataschedule.v, keyblock.v, keyschedule.v, mixcolmn.v, shiftreg.v, count.v, 4 sbox.v for key, 16 sbox.v for data.

Verification

Any design before porting on to the actual device, gatelevel netlist should be verified for functional and timing using simulation tool. The timing simulation assures the design functionality on board/ FPGA. Our AES IP core is verified for timing simulation at 50MHz input clock.

IP	Family	Device	Clock (MHz)	Memory	LE
Encryptor	Cyclone	EP1C20F400C7	50	91,904/294,912 (31%)	6,938/20,060 (35%)
Decryptor	Cyclone	EP1C20F400C7	50	83,712/294,912 (28%)	7,042/20,060 (35%)

I/O Requirement

Though the design is verified for timing simulation using simulation tool, finally the design should be ported on target board and verified by passing the inputs. But how???
Because the total required I/O pin's, to pass input data/key, is more than the I/O's available in target board device (EP1C20F400C7 – 400 pin FBGA, with 301 user I/O's).
Minimum I/O requirement for encryption:

Data input	-	128 I/O
Key input	-	128 I/O
Data output	-	128 I/O
Clock	-	1 I/O
Reset	-	1 I/O
Load	-	1 I/O
Datavalid	-	1 I/O

Total number of I/O's required are 388 for encryption module alone!!!! If we port both encryption and decryption on a single device the I/O pin requirement will be more.

SOPC builder

Since Nios-II is a 32-bit processor, the design requires little modification in port declarations in top level module. Instead of having 128-bit ports, declare 4 32-bit ports per 128-bit port and internally append the 4 ports to make 128-bit block as required by the AES as shown in Fig.2 below.

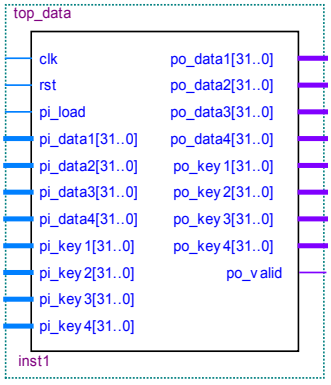

Fig.2.

For faster design cycle, use the available standard project from the Altera installation path, C:\altera\kits\nios2_60\examples\verilog\niosII_cyclone_1c20\standard\standard.qpf.

Open the standard.qpf project in Quartus-II and modify the system using SOPC builder.

- Add 4 32-bit PIO's in Input port only, Rising edge, Generate IRQ, Edge mode for Data input.
- Add 4 PIO's in Input port only, Rising edge, Generate IRQ, Edge mode for Key input
- Add 4 PIO's in Output mode for Data output
- Add 4 PIO's in Output mode for Key output
- Add 1 PIO in output mode for Data valid signal
- Add 1 PIO in Input port only, Rising edge, Generate IRQ, Edge mode for Load signal
- Add 1 PIO in Input port only, Rising edge, Generate IRQ, Edge mode for Reset signal.

(Peripheral names can be renamed according to once convenience. Also, one can disable the components which are not required for the specific application purpose to save the logic resources).
Re-generate the system after auto generating base address and IRQ numbers for the peripherals. When the system generation is successful without any errors, exit from SOPC builder.

Quartus-II software
It is the industry's only design environment that supports IP-based system design-including complete, automated system definition and implementation – without requiring lower level HDL or schematics. This enables you to turn your concepts into working systems in minutes. Quartus-II system design tools include SOPC builder, DSP builder, and off-the-shelf IP cores.

In quartus-II update the std_1c20 system for which it was re-generated using SOPC builder by adding required PIO's. Open the AES top level design file and generate a symbol file for the same and connect the input/ output ports of AES top level symbol file to the required PIO's of standard (std_1c20) system.

Compile the modified standard project without changing any other settings. Power up the target board and program the FPGA using standard.sof file using Altera download cable, USB blaster.

Nios-II IDE
The Nios-II integrated development environment (IDE) is the primary software development tool for the Nios-II embedded processors. One can complete all software development tasks within the Nios-II IDE, including project management, editing and compiling, debugging, and programming flash devices.

Create a new blank project (naming encryptor/ decryptor) and add the application C file to the blank project.

The C code is very simple, write 32-bit data and key to the 4 32-bit respective output PIO's base addresses, similarly for load and reset.

Read the data and key from respective input PIO's base addresses using interrupt service request routine (ISR). Use data valid signal as IRQ.

C code snippet is as shown below.

```
#include <stdio.h>
#include "sys/alt_irq.h"
#include "system.h"
#include <io.h>
#include <stdlib.h>
#include "alt_types.h"
#include <unistd.h>

    IOWR(PO_RESET_BASE,0,0);
    IOWR(PO_LOAD_BASE,0,1);

    IOWR_32DIRECT(PO_KEY4_BASE,0,0x7ab53267);
    IOWR_32DIRECT(PO_KEY3_BASE,0,0x2b1c3abd);
    IOWR_32DIRECT(PO_KEY2_BASE,0,0x6b79abdd);
    IOWR_32DIRECT(PO_KEY1_BASE,0,0xfe127abc);


    IOWR_32DIRECT(PO_DATA4_BASE,0,0x2819dcf9);
    IOWR_32DIRECT(PO_DATA3_BASE,0,0x37ba1c1c);
    IOWR_32DIRECT(PO_DATA2_BASE,0,0xa2b3c4d5);
    IOWR_32DIRECT(PO_DATA1_BASE,0,0x3a4b5c6d);

   IOWR(PO_RESET_BASE,0,1);
   IOWR(PO_LOAD_BASE,0,0);          //-      Active low signal

   L_retValue =  alt_irq_register( PI_KEYVALID_IRQ,
                                   L_keyHolderPtr,
                                   handle_key_valid );
```

For more information on interrupts visit www.altera.com and refer to application note AN284 (Implementing Interrupts Service Routines in Nios systems).

The new project wizard in IDE automates the set-up of C/C++ application projects and system library projects. IDE provides Nios-II run time library, Hardware Abstraction Layer (HAL), which will have the required drivers for the peripherals added in SOPC builder for system generation. Based on the industry standard GNU tool chain, the Nios-II IDE provides a GUI for GNU C compiler (GCC). Nios-II IDE build environment automatically produces a make file based on system configuration.

Once the project is build without any errors, the project can be run on Nios-II hardware (make sure that standard.sof file is downloaded on to FPGA and USB blaster is connected to the target board). The Nios-II debug option and JTAG UART peripheral added in SOPC builder for system generation will help to view the output on screen.