International Journal Of Engineering Research & Management Technology ISSN: 2348-4039

Email: editor@ijermt.org March 2014 Volume 1, Issue 2 Website: ijermt.org

FPGA Based High Performance Hardware Implementation of AES Using Minimal Resources

Swatantra Kumar
M.TECH (VLSI Design)

Suhaib Abbasi
Assistant Professor

SRM University NCR Campus , Ghaziabad

Abstract —
Increasing need of data protection in computer networks led to the development of several cryptographic algorithms hence sending data securely over a transmission link is critically important in many applications. Hardware implementation of cryptographic algorithms are physically secure than software implementations since outside attackers cannot modify them. In order to achieve higher performance in today's heavily loaded communication networks, hardware implementation is a wise choice in terms of better speed and reliability. This paper presents the hardware implementation of Advanced Encryption Standard (AES) algorithm using Xilinx– virtex5 Field Programmable Gate Array (FPGA). In order to achieve higher speed and lesser area, Sub Byte operation, Inverse Sub Byte operation, Mix Column operation and Inverse Mix Column operations are designed as Look Up Tables (LUTs) and Read Only Memories (ROMs). This approach gives a throughput of 3.74Gbps utilizing only 1% of total slices in xc5vlx110t-3-ff1136 target device.

Keywords— AES; Rijndael; Cryptography; FPGA; Verilog; Encryption; Decryption.

Cryptography allows people to carry over the confidence found in the physical world to the electronic world. The importance of cryptography is constantly increasing since the amount of sensitive data being transmitted over an open environment is also increasing day by day. The more information that is transmitted in computer-readable form, the more vulnerable we become to automated spying. Cryptography is not only important in defense applications but also important in real world applications such as E-commerce, E-mail etc.

Encryption is usually done just before sending data. To utilize the channel resources completely encryption algorithm must have a speed at least equivalent to data transmission speed. Achieving high throughput for encryption algorithm for a communication channel of high data rate is a challenging task. The hardware (FPGAs and Application Specific Integrated Circuits-ASICs) implementation of such algorithm which meets these requirements is done in the present work. FPGAs are chosen considering several advantages over the other counterpart [1].

The AES was published by National Institute of Standards and Technology (NIST) in 2001. Later Rijndael algorithm was selected as AES algorithm. Rijndael algorithm can have key length of 128, 192 and 256 bits while block size must be 128 bit [2].

There are many architecture proposals for AES Rijndael algorithm [3, 4], but many of them are poor in terms of area and speed. This paper proposes a different approach to increase speed by utilizing lesser resources available in FPGA.

This paper is structured as follows: Section II describes the existing AES algorithm and Section III describes the proposed work. The result and conclusion are described in Section IV and V respectively.

The AES is a computer security standard from NIST intended for protecting electronic data. Federal Information Processing Standards (FIPS) Publication 197 gives the specification of AES.

AES use Rijndael algorithm [5] by Joan Daeman and Vicent Rijimen for both encryption and decryption. The AES cryptography algorithm is capable of encrypting and decrypting 128 bit data using cipher keys of 128, 196 or 256 bits (AES128, AES196 and AES256) [6].

Rijndael encryption consist of four operations

1. Substitution
2. Shift Row
3. Mix Column
4. Key Addition

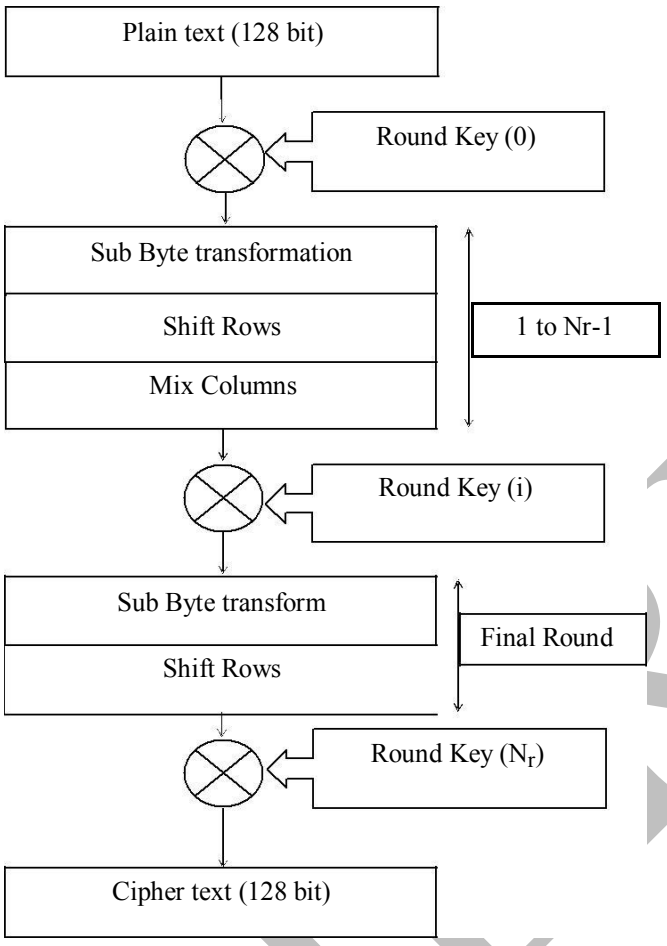


Fig. 1. Algorithm for AES Encryption [2]

The Rijandael decryption consists of four inverse operations of encryption which are compliment functions of encryption. They are

1. Inverse Substitution
2. Inverse Shift Row
3. Inverse Mix Column
4. Key addition

The operations of AES Rijndael algorithm for encryption and decryption is given as follows.

A. Sub Byte and Inverse Sub Byte transformation

n the Sub Bytes step, each byte in the state matrix is replaced with a Sub Byte using an 8-bit data from the Rijndael S-Box. In the Inverse Sub Bytes step, each byte in the cipher matrix is replaced with corresponding Inverse Sub Byte. Sub Byte operation provides the non-linearity in the cipher. The S-Box used is derived from the multiplicative inverse over Galois Field (2^8) [7], known to have good non-linearity properties. Much S-Box implementation [7] use combinational circuit consists of an adder, squarer and constant multiplier. Rijndael S-Box is not shown for brevity.

B. Shift Row Transformation

The Shift Rows transformation cyclically shifts the bytes in each row by certain offset to the left. For AES, the first row is left unchanged. Each byte of the second row is shifted by one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. Inverse Shift Row transformation does the same shift operation towards right. Fig.2 shows the Shift Row operation.

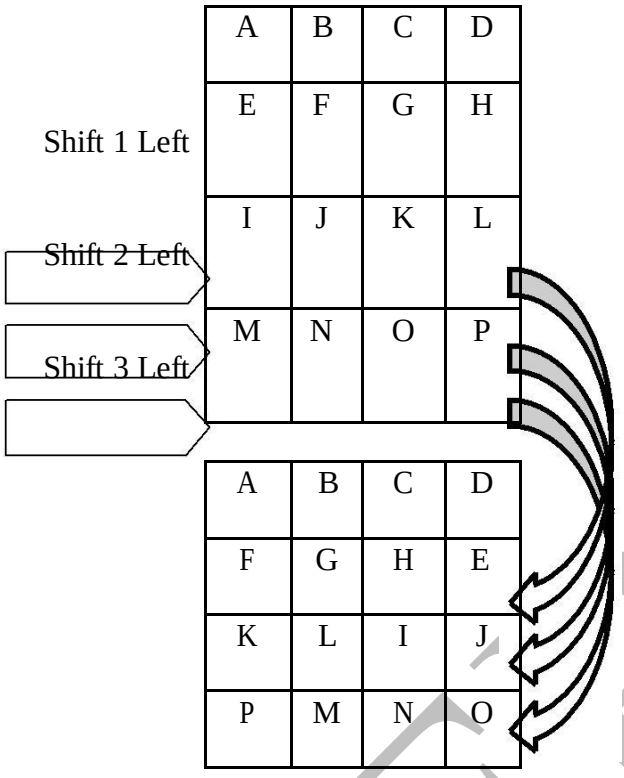


Fig. 2. AES Shift Row Operation

C. Mix Column and Inverse Mix Column operation

In the Mix Column step, the four bytes of each column of the state are combined using an invertible linear transformation. All entries in the state matrix are considered to be a polynomial and it is multiplied by a fixed polynomial. The Mix Column and inverse Mix Column transformation are represented in matrix form as form as equation 1, 2.

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 01 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$$

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 01 & 0E & 0B \\ 0B & 0D & 01 & 0E \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

Where c_{ij} and d_{ij} are Mix Column input and Output respectively, while a_{ij} and b_{ij} are respectively the inputs and outputs of Inverse Mix Column operation.

D. Add Round Key operation

In this operation, bitwise exclusive-or (XOR) operation is performed between outputs from Mix Column and Round Key. For AES-128, 128 bit XOR operations are performed.

The proposed architecture is designed to get maximum speed and lesser area by mapping all the four Logical functions of AES to LUTs, ROMs and Block RAMs. The proposed architecture has three parts

1. Key Generation Module
2. Encryption Module
3. Decryption Module.

The AES encryption and decryption core unit contains key generation module as a common unit. This module gives necessary key expansion for both encryption and decryption functions. Fig.3 presents the block diagram of AES Rijndael encryption and decryption with Key Generation Module as a common unit. The key generation module consists of key register of 128 bits, S-Box and XOR gates for bitwise XOR operation.

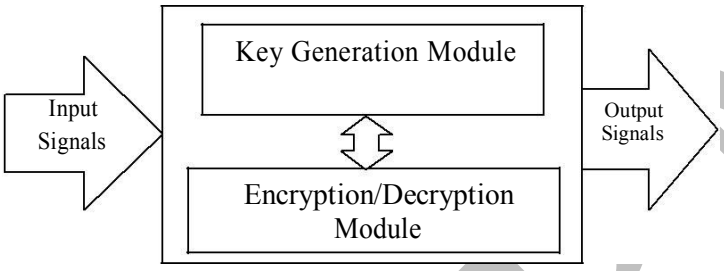


Fig. 3. AES Encryption and Decryption Unit Block Diagram

It is designed to produce round keys on each positive edge of the clock, when it is enabled. However in the proposed work, the key generation architecture does not require any hardware for shift operation and the port mapping between key register and S-Box is done according to the required shift. Hence the proposed work offers the advantage in area. Also in the proposed work the bits are rearranged on data path from register to S-Box and the round constant required for each rounds are stored in ROM and retrieved on each clock. Fig.4 represents proposed architecture of key generation unit.

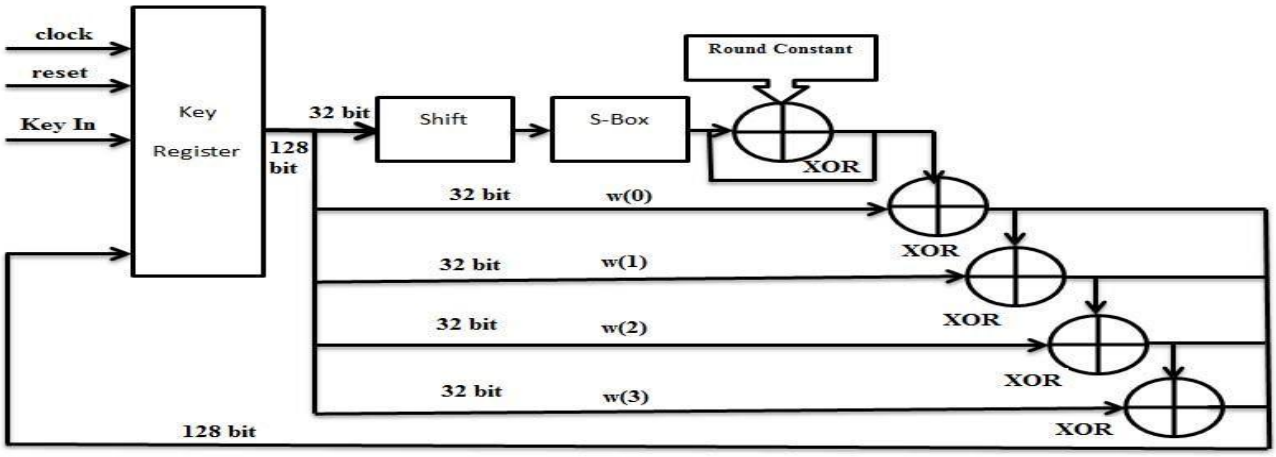


Fig 4 Architecture of Key Genration

The encryption module takes 128 bit text to be encrypted and receives round key from key generation module to do each round of encryption.

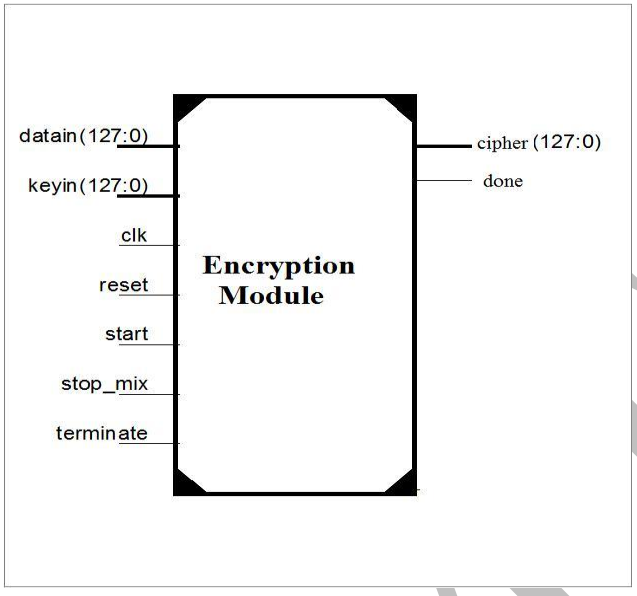


Fig. 5. Encryption Module

Start, stop_mix, terminate are control signal produced by the control unit. The "done' signal is provided to indicate that encryption is done. Architecture is as shown in Fig. 6.

In the proposed work for reducing the hardware of entire architecture, the control unit of encryption module is not designed separately. The control unit of key generation module which is a 4-bit counter is designed to control the entire functioning of encryption module. The sharing of control unit by both encryption and round key generation gives unique advantage of reduction in hardware as compared to other implementations [1, 3].

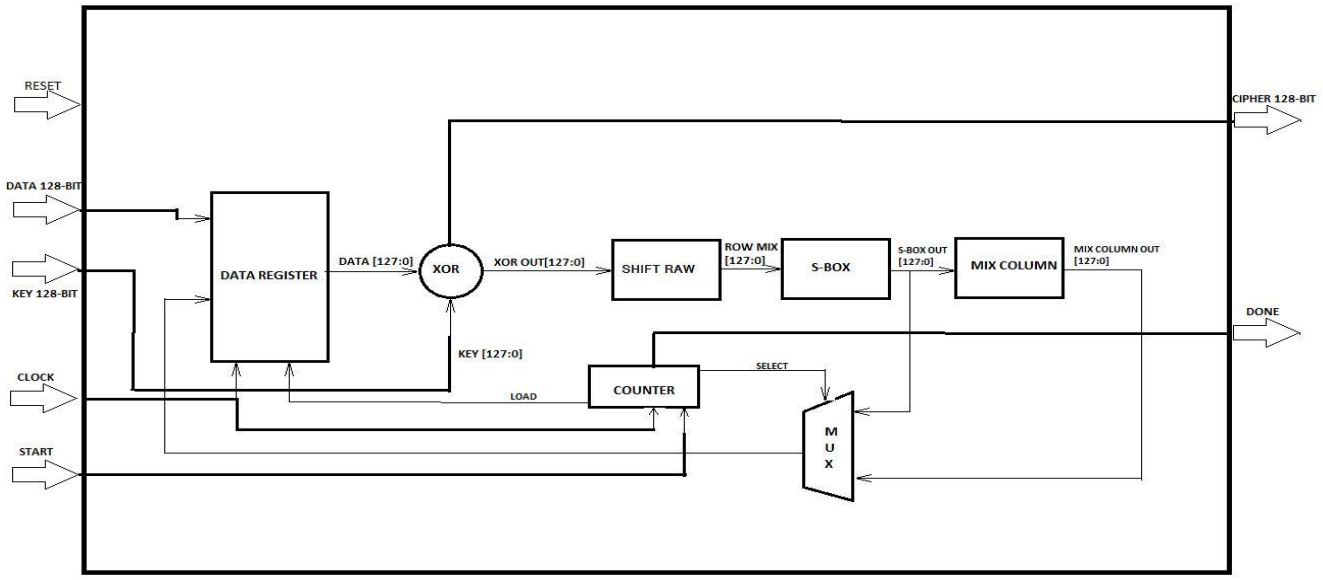


Fig. 6. Proposed architecture of encryption module

In the last round, Rijandael algorithm skips Mix Column Operation. To incorporate this functionality

proposed design use a Multiplexer and NAND gate as shown in Fig. 7.

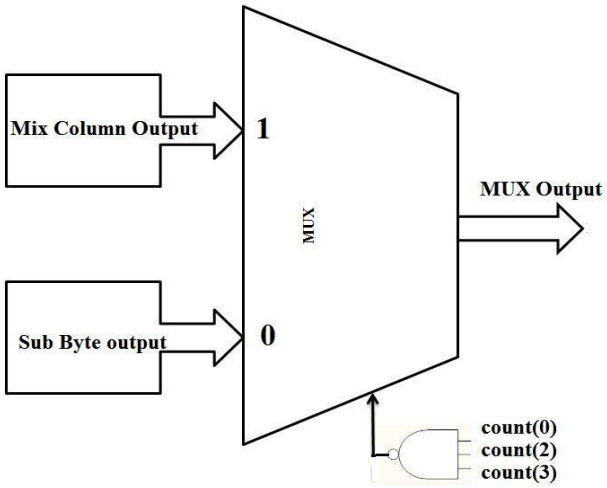


Fig. 7. Hardware to Skip Mix Column Operation for Last Round

NAND gate and the 4-bit counter (Controller) are used to set and reset selection line of Multiplexer. For count one to ten the selection line will be in set condition and multiplexer will pass Mix Column output. However on last round, count will be eleven so selection line will reset and pass Sub Byte output.

hift Row operation is designed in such a way that it does not take any hardware. After Round Key operation data is given to S-Box with required shift by port mapping the signal according to required shift in Verilog HDL description of the design. Since there is no hardware for Shift Row operation design gets the advantage of area, power and speed.

In the proposed work, the S-Box is implemented by a LUT having 8 bit address (256 addresses) and a data width of 8 bit. This implementation gives higher throughput for the design by significantly decreasing delay in data path. As a result the proposed design takes lesser number of slices when compared with other combinational technique proposed in [7].

The Mix Column operation of AES consists of Galois multiplication and four input XOR operation. But unlike combinational implementation [8] of Galois field multiplication, the proposed design uses ROM based implementation of Galois multiplication which makes Galois multiplication significantly faster avoiding combinational delays. For an 8-bit data there are 256 multiplication conditions and all the conditions are stored in (256 x 8) ROM

In the proposed work the Mix Column encryption hardware uses two of such ROM for Galois multiplication of '2' and '3' and for performing 4-Input XOR operation in Mix Column operation, the proposed design use 16 x 1 ROM with the result that Mix Column operation offers higher speed and uses minimum number of slices in the hardware (FPGA).

The decryption unit also uses same design approach for the entire architecture and takes 20 clock cycles to decrypt the given cipher back to original text.

Inverse S-Box architecture uses the same design of S-Box. Entry of LUT is changed according to Inverse Sub Byte transformation. Mix Column operation is implemented using 256X8 ROM. Four such ROMs are designed for the Galois multiplication of 9, 11, 13 and 14. 4-Input XOR operation is designed by 16x1 ROM. Architecture of Decryption module is same as encryption module with all complimentary functions of encryption. Decryption unit contains an extra register for storing Round Keys. Storing key is important since first round decryption use tenth round key and second round use ninth round key and so on. Count register is synthesized as B-Ram to save number of slices. 'Count' input provides the address of key register location to be

accessed. The Architecture of decryption module is shown in Fig. 8. _

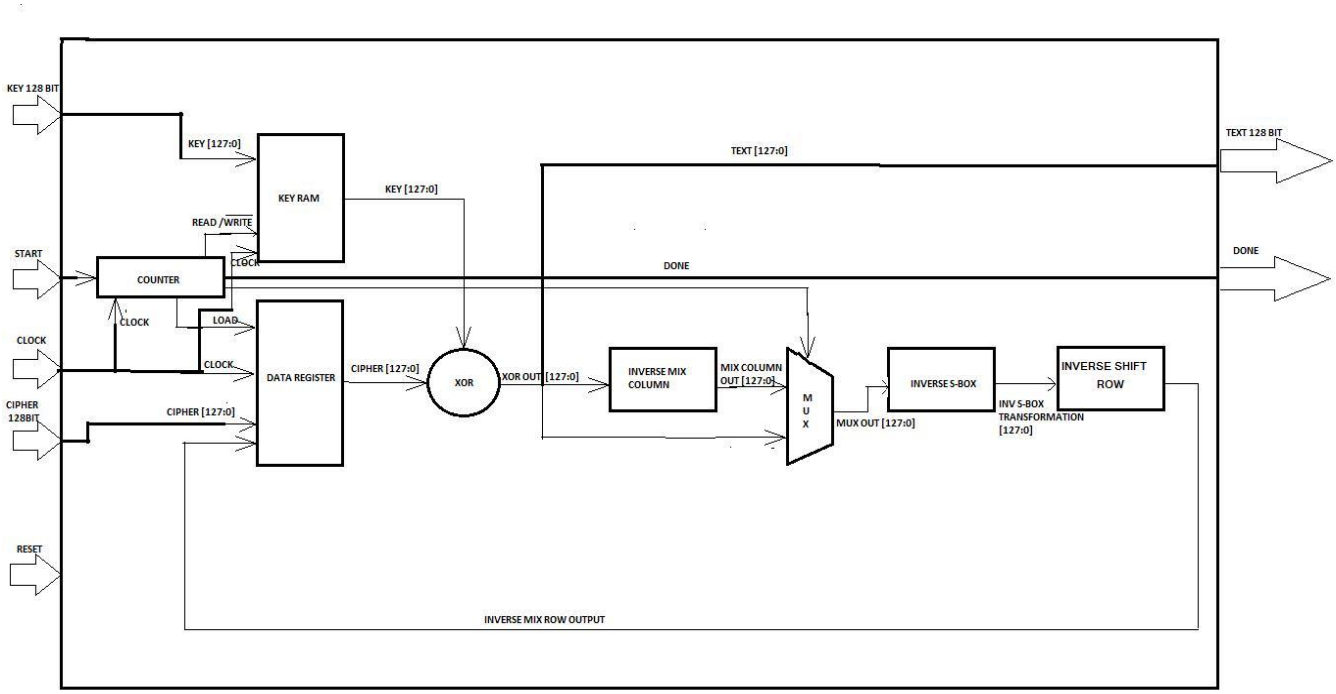


Fig. 8. Proposed architecture of Decryption module

Implementation Tools

Simulation: Modelsim 10.1

Synthesis: Xilinx ISE 12.1

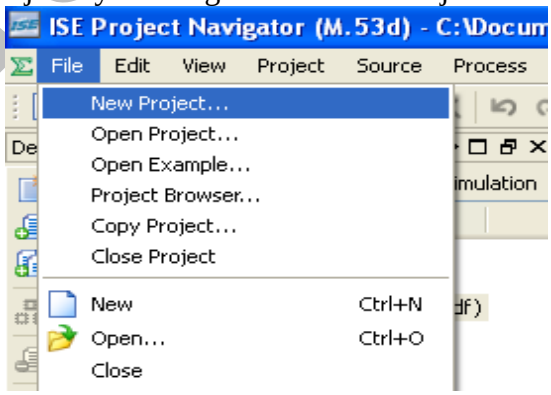
5.3.1 XILINX ISE 12.1

Implementing Verilog HDL Designs Using Xilinx ISE

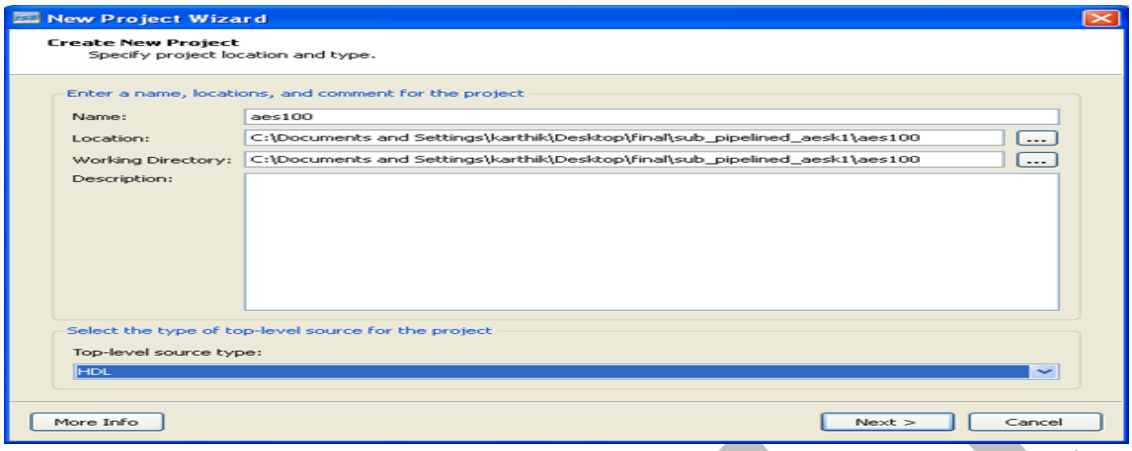
This following shows how to create, implement and synthesize Verilog HDL designs for implementation in FPGA chips using Xilinx ISE 12.1

1. Launch Xilinx ISE from either the shortcut on your desktop or from your start menu under Programs → Xilinx ISE 12.1 → Project Navigator.

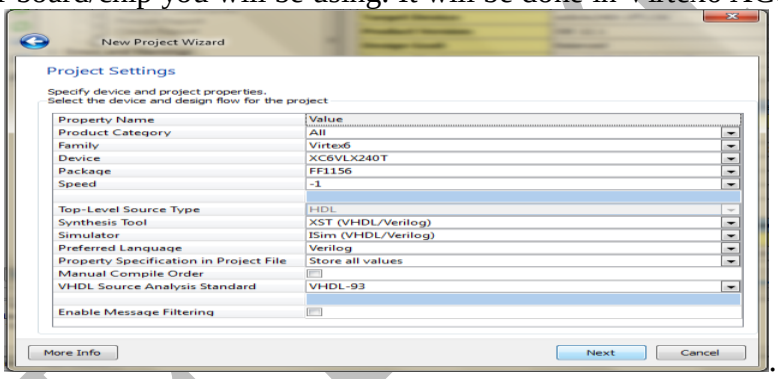
2. Start a new project by clicking File → New Project



3. In the resulting window, verify the "Top-Level Source Type" is VHDL. Change the "Project Location" to a suitable directory and give it whatever name you choose

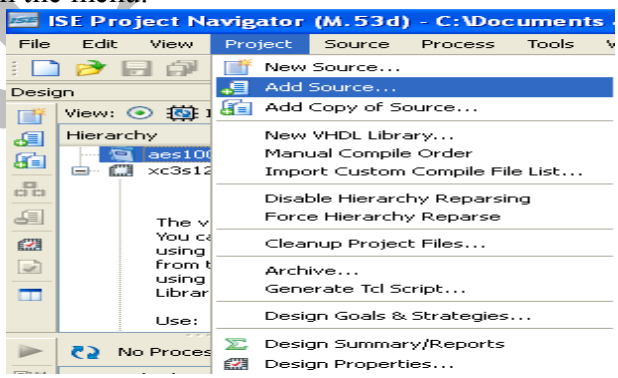


4. The next window shows the details of the project and the target FPGA. We will be synthesizing designs into FPGA. so it important to match the target with the particular board/chip you will be using. It will be done in Virtex6 XC6VCX240T

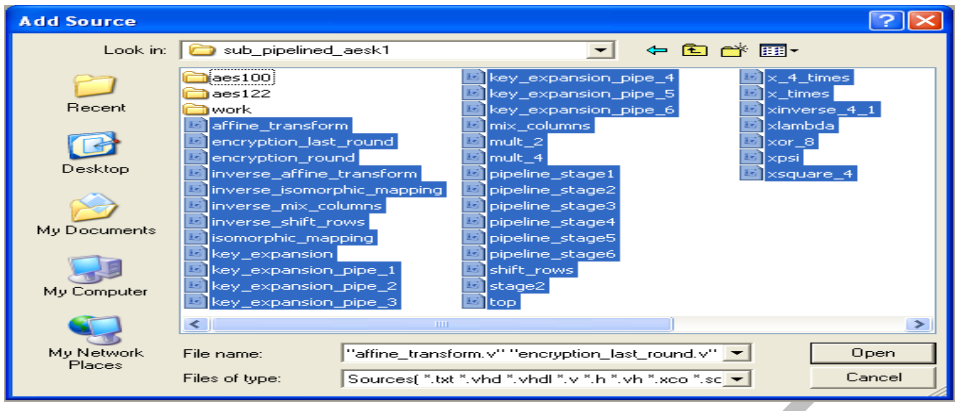


5. Since we are starting a new design the next couple of pop-up windows aren't relevant, just click next and next and Finish.

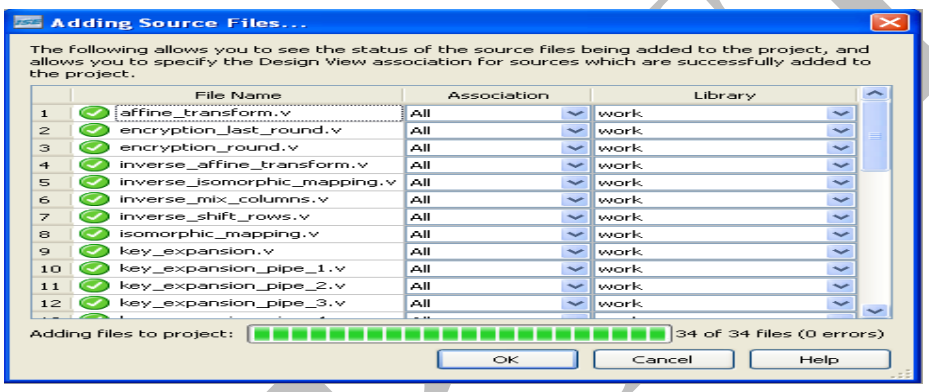
6. You should now be in the main Project Navigator window. Select Project →Add source… from the menu.



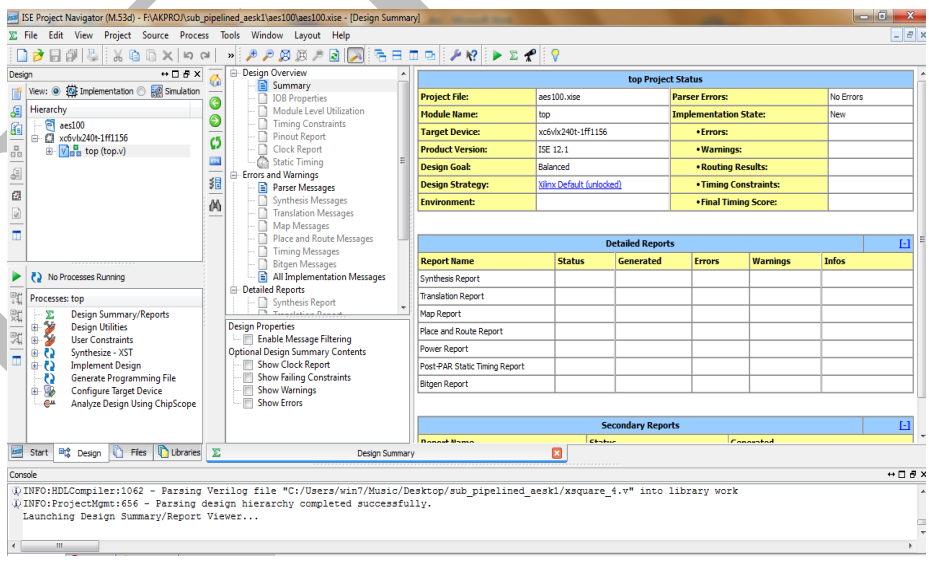
7. The Verilog files to be added from the specified folder.



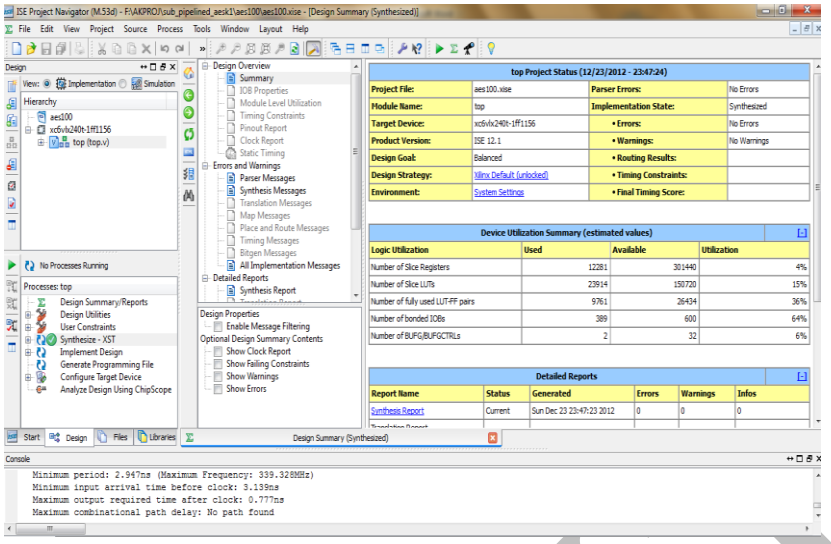
8. Next window shows the successfully added source files.



9. The source files which are added to the Top module are get synthesized by synthesize XST



10. Synthesized report can be obtained on the console and device utilization also estimated



From the console the maximum frequency obtained is 339.328MHz. which is used to calculate the Throughput as mentioned in section 4.2

Throughput = 339.328MHz × 128

= 43.43Gbp

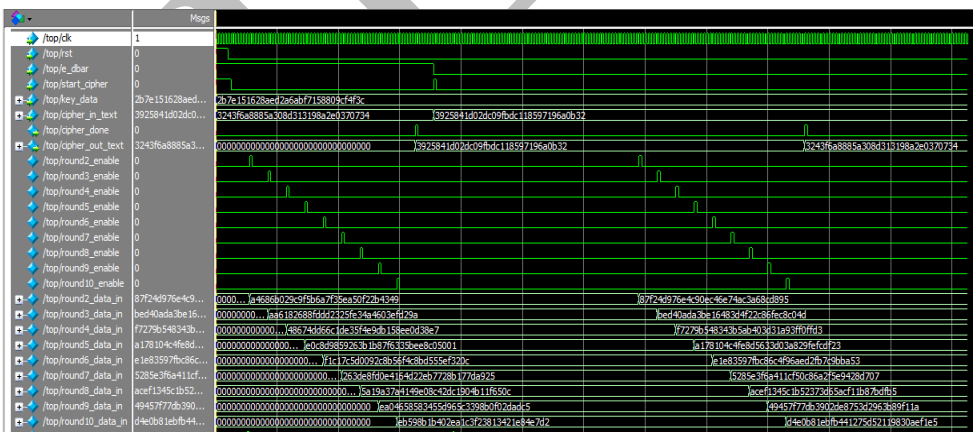
SIMULATION RESULTS

For simulation using the Modelsim simulator of version 10.1c

1.ModelSim should successfully launch and will open several subwindows by default. For now we just need the "Wave" and "Transcript subwindows, so close the other subwindows.

2. To conduct the simulation we basically need to know two commands, "force" and "run". Force is used to set the value of any input variable. Then Run the simulation for a specific amount of time. To use a Force command, in the transcript window simply type Force, space, logic variable you wish to set, space, the value you wish to assign (0 or 1).

Simulation Result For Joint Encryptor And Decryptor



AES Rijndael algorithm is simulated and synthesized using Xilinx 13.1 ISE tool and the targeted FPGA is 5vlx110tff1136-3 which belongs to Virtex-5 family. The design uses only LUTs, ROMs for all the operations of AES encryption and decryption. This approach reduces device utilization and significantly improves the speed compared to other implementation [3,4,9]. The key register in the decryption module is

synthesized as Block-Ram to reduce the number of slices used. The utilization summary for device 5vlx110tff1136-3 is presented in Table I.

Table 1. SLICE LOGIC UTILIZATION

Number of Slice Registers	128 out of 69120 0%
Number of Slice LUTs	1106 out of 69120 1%
Number used as Logic	1106 out of 69120 1%

Table 2. COMPARISON [4]

Design	**Delay (ns)**	**Max.freq uency (MHz)**	**Throughput (Gbps)**	**LUTs**	**Registers**
Proposed design	3.42	292.40	3.74	1%	1%
M. Goswami &S. kannujiya[1]	4.25	235.29	2.73	.7%	1%
W. Wei, C.jie, X.Fei[3]	4.97	201.20	2.57	Not Available	
Kampen *	5.291	189	.016	73%	39%
J.Castillo *	6.711	149	.0375	94%	57%
Four sbox AES *	6.493	154	.241	93%	22%
H.Satyanarayan*	3.690	271	2.166	81%	33%

AES-128 algorithm for encryption and decryption is implemented in Virtex-5 FPGA. With the designing of all the operations as LUTs and ROMs, the proposed architecture achieves a throughput of 3.74 Gbps and thereby utilizing only 1% of slices in the targeted FPGA. Since the speed is higher than the already reported systems, hence the proposed design serves as the best high speed encryption algorithm and is thus suitable for various applications. Moreover with less area utilization, the proposed design can be embedded with other larger designs as we

Refrences:

1. M. Goswami and S. Kannojiya, "High Performance FPGA Implementation of AES Algorithm with 128-Bit Keys," Proc. IEEE Int. Conf. Advances Computing Comm., vol. 1, Himarpur, India, 2011, pp. 281-286.
2. FIPS-197, NIST - National Institute of Standards and Technology, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.
3. W. Wei, C. Jie and X. Fei, "An Implementation of AES Algorithm on FPGA," IEEE 9th Int. Conf. on Fuzzy Systems and Knowledge discover 2012, pp. 1615-1617.
4. U. Kretzschmar, A. Astarloa, J. Lazaro, U. Bidarte and J. Jimenez, "Robustness analysis of different AES implementations on SRAM based FPGAs," Int. Conf. on Reconfigurable Computing and FPGAs 2011, pp. 255-260.
5. J. Daeme and V. Rijmen, "AES proposal: Rijndael," NIST AES Proposal, June 1998.
6. W. Stallings, "Cryptography and network security principles and practice," Pearson edition 2009, pp. 135-160.
7. P.V.S. Shastry, A. Agnihotri, D. Kachhwaha, J. Singh and M.S. Sutaone, "A Combinational Logic Implementation of S-Box of AES," IEEE 54th Int. Midwest Symp. on Circuits and Systems (MWSCAS), Aug. 2011, pp. 1-4.
8. S. Kaur and R. Vig, "Efficient Implementation of AES Algorithm in FPGA Device," Int. Conf. on Computational Intelligence and Multimedia Applications, Dec. 2007, pp. 179 – 187.
9. H. Trang and N.V. Loi, "An efficient FPGA implementation of the Advanced Encryption Standard algorithm," IEEE Int. Conf. on Computing and Communication Technologies, Research, Innovation and Vision for the Future (RIVF), 2012, pp. 1-4.