# FPGA Implementation of Fully Pipelined Advanced Encryption Standard

Technical Report on Term Project

Submitted by

**Abdullah Al Mamun**

**Id: g201403680**

Digital System Design and Synthesis (COE – 561)
Teacher: Alaaeldin Amin

19 December 2015

Computer Engineering Department

King Fahd University of Petroleum & Minerals

# Outline

# Abstract

Before sending the information through an electronic medium, we should be aware of data confidentiality, integrity and availability. Encryption is the popular method to protect private data. According to the NIST, AES is the latest secure encryption algorithm. However, data size is growing up dramatically, thus we need high throughput using pipelining. Although, AES can be implemented in both software and hardware. As compared to software implementation hardware implementation of AES has an advantage of increased throughput and more security. In this research, VHDL based FPGA implementation of 128 AES using fully pipeline architecture is proposed.

# 1 Introduction

The National Institute of Standards and Technology, (NIST), solicited proposals for the Advanced Encryption Standard, (AES). The AES is a Federal Information Processing Standard, (FIPS), which is a cryptographic algorithm that is used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt, (encipher), and decrypt, (decipher), information. Encryption converts data to an unintelligible form called cipher-text. Decryption of the cipher-text converts the data back into its original form, which is called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. Many algorithms were originally presented by researchers from twelve different nations. Fifteen, (15), algorithms were selected from the first set of submittals. After a study and selection process five, (5), were chosen as finalists. The five algorithms selected were MARS, RC6, RIJNDAEL, SERPENT and TWOFISH. The conclusion was that the five Competitors showed similar characteristics. On October 2nd 2000, NIST announced that the Rijndael Algorithm was the winner of the contest. The Rijndael Algorithm was chosen since it had the best overall scores in security, performance, efficiency, implementation ability and flexibility, [NIS00b]. The Rijndael algorithm was developed by Joan Daemen of Proton World International and Vincent Fijmen of Katholieke University at Leuven. The Rijndael algorithm is a symmetric block cipher that can process data blocks of 128 bits through the use of cipher keys with lengths of 128, 192, and 256 bits. The Rijndael algorithm was also designed to handle additional block sizes and key lengths. However, the additional features were not adopted in the AES. The hardware implementation of the Rijndael algorithm can provide either high performance or low cost for specific applications. At backbone communication channels or heavily loaded servers it is not possible to lose processing speed, which drops the efficiency of the overall system while running cryptography algorithms in software. On the other side, a low cost and small design can be used in smart card applications, which allows a wide range of equipment to operate securely. [6]

## 2 Problem Statement

In general, a FPGA based device is fully dedicated to run/process a target operation. In case of encryption operation, plaintexts are very large in amount such as MB,GB, even TB. Unfortunately, AES can encrypt only 128 bit per block at a time. But, we need high throughput (encrypted data volume).

# 3 Proposed Solution

AES used simple mathematics but it is a repetitive and multi-stages operation. As, we know pipeline circuit allows high throughput, a fully pipeline architecture is proposed in this research to build FPGA. Also, we have plan to support partially parallelism to reduce the latency.

# 4 Objectives

Main goal of this research is to implement such FPGA based hardware which can achieve high throughput with low latency data encryption. Main objective can be divided into sub items as follows

1. Identify the modules and steps of AES algorithm.
2. VHDL design and implementation for each steps.
3. Implement parallel MixColumn Operation.
4. Implement fully pipeline AES circuit.
5. Develop TestBench for all AES stages and compare output with expected data.
6. Run simulation and measure resource utilization, throughput and latency.
7. Run VHDL on xilinx spartan6 board and observe sample encryption.

# 5 AES Background [11]

## 5.1 Notation and Convention

### 5.1.1 Inputs and Outputs

The input and output for the AES algorithm each consist of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and Cipher Key lengths are not permitted by this standard.

### 5.1.2 Bytes

The basic unit for processing in the AES algorithm is a byte, a sequence of eight bits treated as a single entity. All byte values in the AES algorithm will be presented as the concatenation of its individual bit values (0 or 1) between braces in the order $\{b7,\ b6,\ b5,\ b4,\ b3,\ b2,\ b1,\ b0\}$. These bytes are interpreted as finite field elements using a polynomial representation:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_ix^i \tag{1}$$

For example, {01100011} identifies the specific finite field element $x^6 + x^5 + x + 1$.

### 5.1.3 Arrays of Bytes

The bytes and the bit ordering within bytes are derived from the 128-bit input sequence

$input_0 input_1 input_2 input_1 26 input_1 27$

as follows:

$a_0 = input_0, input_1, , input_7$;

$a_1 = input_8, input_9, , input_1 5$;

.

$a_15 = input_120, input_121, , input_127.$

### 5.1.4 The State

Internally, the AES algorithms operations are performed on a two-dimensional array of bytes called the State. The State consists of four rows of bytes, each containing $N_b$ bytes, where $N_b$ is the block length divided by 32. In the State array denoted by the symbol $s$, each individual byte has two indices, with its row number $r$ in the range $0 \leq r \leq 4$ and its column number $c$ in the range $0 \leq c \leq N_b$. The input the array of bytes copied into the State array as illustrated in Fig. 1.

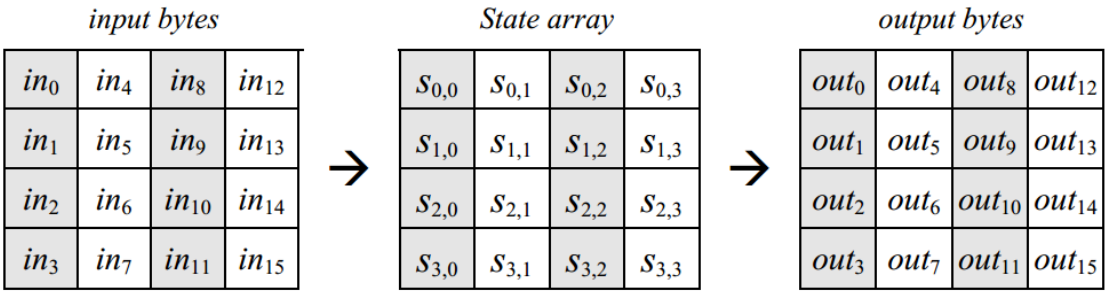



Figure 1: State array input and output.

### 5.1.5 The State as an Array of Columns

The four bytes in each column of the State array form 32-bit words, where the row number $r$ provides an index for the four bytes within each word. The state can hence be interpreted as a one-dimensional array of 32 bit words (columns), $w_0 ... w_3$, where the column number $c$ provides an index into this array. Hence, for the example in Fig. 1, the State can be considered as an array of four words, as follows:

$w_0 = s_{0,0}s_{1,0}s_{2,0}s_{3,0}$

$w_2 = s_{0,2}s_{1,2}s_{2,2}s_{3,2}$

$w_1 = s_{0,1}s_{1,1}s_{2,1}s_{3,1}$

$w_3 = s_{0,3}s_{1,3}s_{2,3}s_{3,3}$

## 5.2 Mathematical Preliminaries

Required mathematical operations are finite field addition and multiplication.

### 5.2.1 Addition

Finite field addition is noting but bitwise XOR operation. For example

Binary notation - $\{01010111\} \oplus \{10000011\} = \{11010100\}$

Hex notation - $\{57\} \oplus \{83\} = \{d4\}$

### 5.2.2 Multiplication

In the polynomial representation, multiplication in $GF(2^8)$ (denoted by $\bullet$) corresponds with the multiplication of polynomials modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. For the AES algorithm, this irreducible polynomial is

$$m(x) = x^8 + x^4 + x^3 + x + 1, \tag{2}$$

or 01 1b in hexadecimal notation. For example, $\{57\} \bullet \{83\} = \{c1\}$, because

$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1)$

$= x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x^1 + x^6 + x^4 + x^2 + x + 1$

$= x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 and x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 modulo (x^8+x^4+x^3+x+1)$

$= x^7 + x^6 + 1.$

## 5.3 Algorithm Specification

For the AES algorithm, the length of the input block, the output block and the State is 128 bits. This is represented by $N_b = 4$, which reflects the number of 32-bit words (number of columns) in the State. For the AES algorithm, the length of the Cipher Key, K, is 128, 192, or 256 bits. The key length is represented by $N_k = 4, 6, or 8$, which reflects the number of 32-bit words (number of columns) in the Cipher Key. The only

Key-Block-Round combinations that conform to this standard are given in Fig. 2.

| | Key Length *(Nk words)* | Block Size *(Nb words)* | Number of Rounds *(Nr)* |
|---|---|---|---|
| **AES-128** | 4 | 4 | 10 |
| **AES-192** | 6 | 4 | 12 |
| **AES-256** | 8 | 4 | 14 |

Figure 2: Key-Block-Round Combinations.

### 5.3.1 Cipher

Encrypted plaintext is called cipher, the input is copied to the State array and perform four operation SubBytes, ShiftRows, MixColumns, and AddRoundKey repeatedly then cipher can be ready to transmit. Performing the encryption of the given plaintext using four different transformations in the initial round, the nine main rounds and a final round where mix column operation is absent. Whole encryption process is shown in Fig 3.

**SubBytes Transformation** Its a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box). This S-box (Fig. 5), which is invertible, is constructed by composing two transformations:

1. Take the item from state array, spit into two hex number.

2. Match row and column according to the first and second number. Replace the state content with matched s-box value which is shown in Fig 4.

**ShiftRows Transformation** In the ShiftRows transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, r = 0, is not shifted. Specifically, the ShiftRows transformation proceeds as follows:

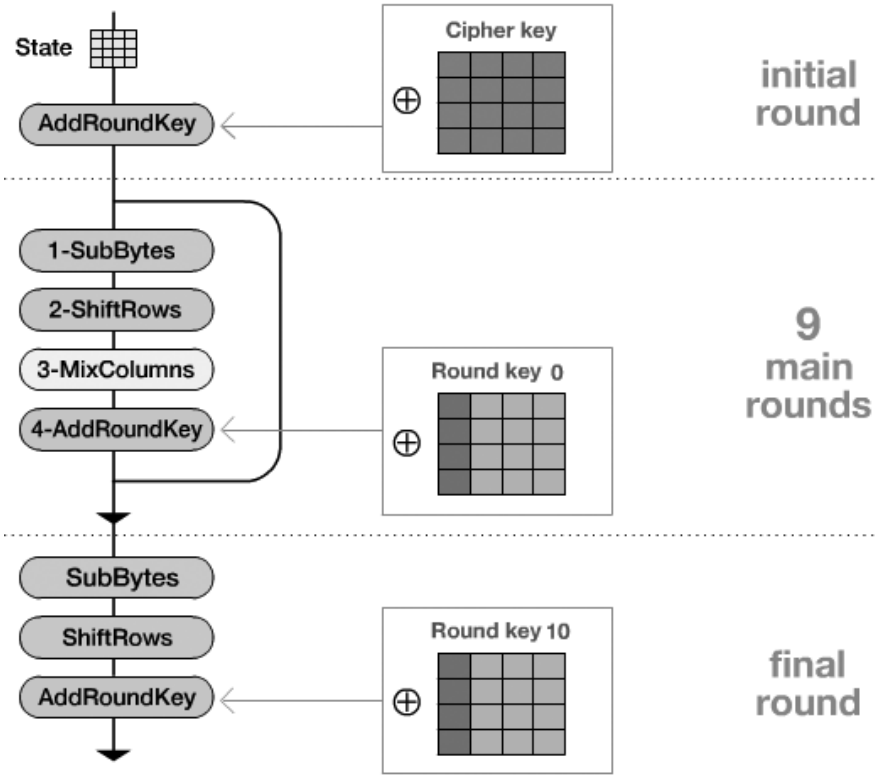



Figure 3: Encryption Process

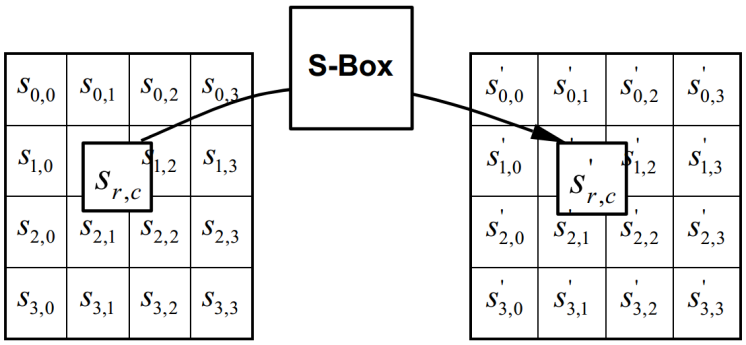



Figure 4: SubBytes applies the S-box to each byte of the State.

$$s'_{rc} = s_{r,(c+shift(r,Nb))modNb} \quad for \quad 0 \leq r \leq 4 \quad and \quad 0 \leq c \leq Nb, \tag{3}$$

| | | y | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| x | 0 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| | 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| | 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| | 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| | 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| | 5 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| | 6 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| | 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| | 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| | 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| | a | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| | b | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| | c | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| | d | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| | e | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| | f | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

Figure 5: S-box: substitution values for the byte xy (in hexadecimal format).

where the shift value shift(r,Nb) depends on the row number, r, as follows (recall that Nb = 4):

$$shift(1,4) = 1; \quad shift(2,4) = 2; \quad shift(3,4) = 3 \tag{4}$$

Figure 5 illustrates the ShiftRows transformation.

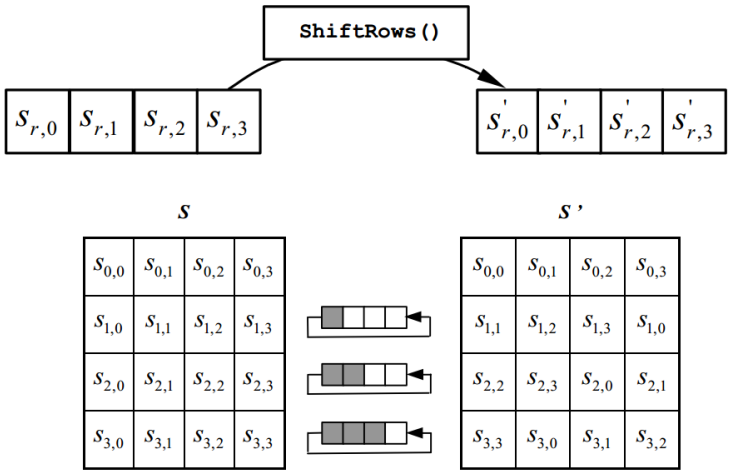



Figure 6: ShiftRows cyclically shifts the last three rows in the State.

**MixColumns Transformation** The MixColumns transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}.$$

$$\begin{aligned} s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}). \end{aligned}$$

GF(28) and multiplied modulo $x^4 + 1$ with a fixed polynomial a(x), given by

$$a(x) = 03x^3 + 01x^2 + 01x + 02. \tag{5}$$

This can be written as a matrix multiplication. Let $s'(x) = a(x) \otimes s(x)$

As a result of this multiplication, the four bytes in a column are replaced by the following

For example,

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} d4 \\ bF \\ 5d \\ 30 \end{bmatrix} = d4 \bullet 02 \oplus bf \bullet 03 \oplus 5d \oplus 30$$

$d4 \bullet 02 =$ xtime(d4) = b3

bf $\bullet$ 02 = xtime(bf) = 65

bf $\bullet$ 03 = bf(01 $\oplus$ 02) = bf $\oplus$ 65 = ba

Thus, $d4 \bullet 02 \oplus bf \bullet 03 \oplus 5d \oplus 30 = 04$

Multiplication is done by a left shift and a subsequent conditional bitwise XOR with 1b. This operation on bytes is denoted by xtime() . By adding intermediate results, multiplication by any constant can be implemented. Figure 7 illustrates the MixColumns transformation.

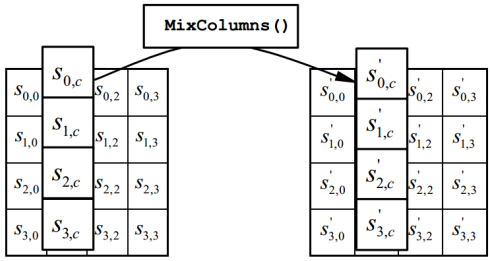



Figure 7: MixColumns operates on the State column-by-column.

**AddRoundKey Transformation** In the AddRoundKey() transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of Nb words from the key schedule shown in Fig 8. Those Nb words are each added into the columns of the State, such that

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c} \oplus [w_{round*Nb+c}] \quad for \quad 0 \leq c \leq Nb, \tag{6}$$

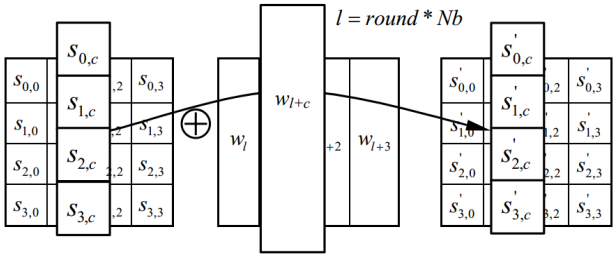



Figure 8: AddRoundKey XORs each column of the State with a word from the key schedule.

### 5.3.2 Key Expansion

Key expansion process is a combination of three process SubWord is a function that takes a four-byte input word and applies the S-box (see Fig. 5) to each of the four bytes to produce an output word. Words in position that are a multiple of 4(w4,w8,.....w40) are calculated by

1. Applying the RotWord and SubBytes transformation to the previous word $W_{i-1}$.

2. Adding this result to the word 4 position earlier $W_{i-4}$, plus a round constant Rcon which is shown in

Fig 9

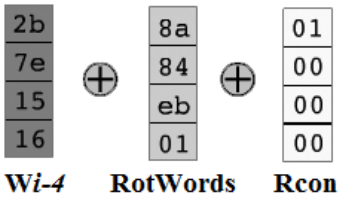



Figure 9: Key generation

# 6 Simulation and Implementation

## 6.1 Target Device, Design Flow and Tools

The FPGA, we select a Xilinx Spartan6 XC6LX16-CS324 for this research. The Xilinx Spartan series of FPGAs, made of many Configurable Logic Blocks (CLBs) and interconnection circuitry covered to form a chip. Each CLB consists two slices, and each slice contains 16 FFs, two 8 inputs Look-Up Tables (LUT), associated carry chain logic. Each LUT can either be used as a 256x1 bit RAM, or 128bits as a 1-16 cycle delay shift register. The Xilinx spartan6 XC6LX16 presents 2278 slices, 136K distributed RAM, 14,579 logic cells and 500K gates. Moreover, it has fully synchronous dual-ported 360K Block SelectRAM. BRAM are organized in columns and every BRAM has 4 CLBs. Regarding design tools, Xilinx XST used for synthesizing VHDL, and Xilinx ISE 14.2 used for place & route and timing analysis. In our AES design experience Simplify appeared to achieve better synthesis results with respect to Xilinx XST and FPGA Express. However, The block diagram of AES encryption is shown in the Fig 10. It takes 128 bits both

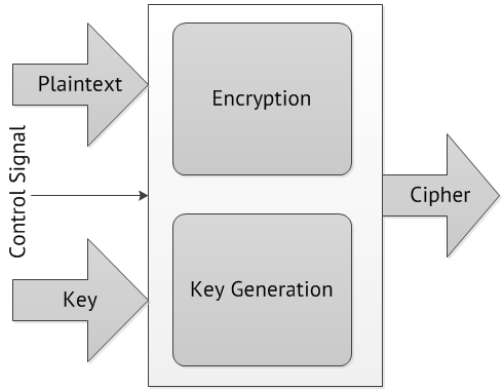



Figure 10: Block Diagram of Encryption

plaintext and key as input and also produce 128 bits cipher.

## 6.2 Why ShiftRows before SubBytes?

Since, the main goal of this research is high throughput that can achieve by using instruction pipelining technology. There are no dependencies found between SubByte and ShiftRows operations that indicates

one can be done before another without any hesitation. All operations of AES algorithm are column based except the ShiftRows. If the ShiftRows is done before SubByte, the position of plaintext blocks will be fixed and remain operation can be pipelined which is shown in Fig 14. However, The S-Box used is derived from the inverse function over GF(28), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box constructed by combining the inverse function with an invertible affine transformation. But in our design to avoid complexity, SBox is simply a Look Up Table (LUT) i.e ROM with capacity $256 \times 8$ bits which is depicted in Fig 11.

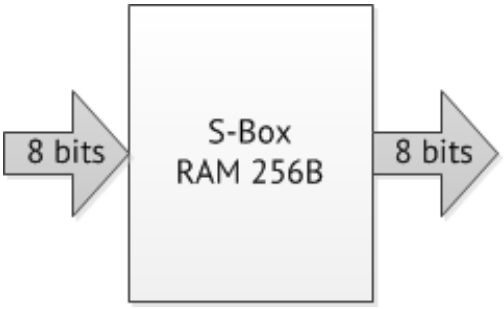



Figure 11: S-Box

### 6.3 Parallelism in Mix-column

In general, mix column function is a combination of shift and xor operation shown in fig 7. This study shows that mix-column operation can be ended in many ways. We discussed two special ways. However, multiplicand 1M, 2M and 3M are requisite to pre-calculation in both cases, and its product are stored in a register. The schematic diagram of the Mix-column architecture is shown in fig 12 where 24*8 multiplexer is used to select multiplicand according to the multiplier. In this case, only one product we can achieve in each cycle. In contrary, if all pre-computation result can be feed to four xor gate in a single cycle, we can get four parallel output that is shown in fig 13. As a result, a increased performance is recorded in second approach.

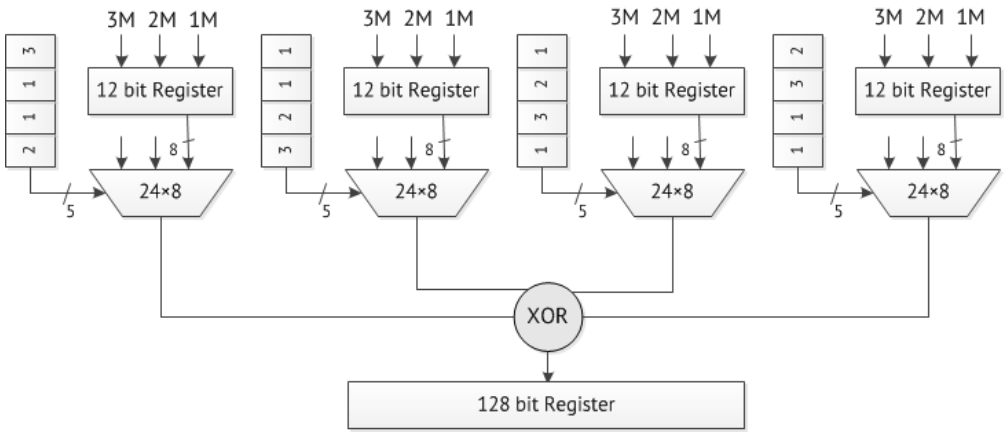



Figure 12: The schematic diagram of the Mix-column architecture.

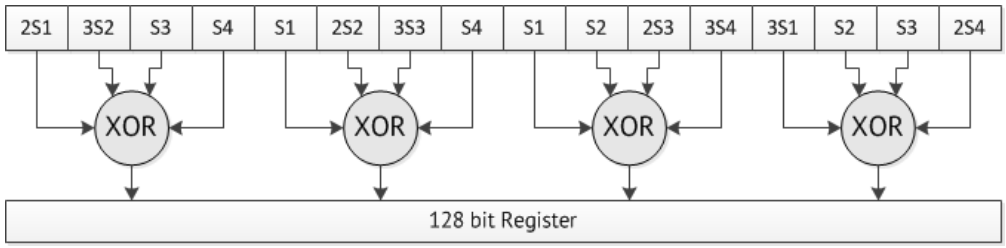



Figure 13: The schematic diagram of the Mix-column parallel architecture.

## 6.4 Key Scheduler

The operation of the key ex-pander is to take the input 128-bit original key and scheduled it into ten 128-bit sub round keys by feeding back after first round using a $2 \times 1$ mux. As shown in Fig. 14, the fist action is Rot-Word, which makes one-byte circular left shift onword. The second operation is byte substituted (subwords) through S-box. Finally, it XORed with the input key and the round constants (Rcon). The key scheduler is recurring ten times to achieve ten sub round keys [3]. Rcon is nothing but a LUT in a fussy pattern. The sub round keys altered only with the input key change.

## 6.5 Fully Pipelined Encryption

The aim of AES implementation using fully pipelined architecture is to optimize the delay and reach the peak throughput. The schematic diagram of fully pipelined of architecture encryption is shown in Fig. 14. In general, there are two types of pipelining are feasible for AES: inner round pipelining and outer round

pipelining. In this research, pipelining is made after each round that is called outer round pipelining, and it is a non-feedback pipeline mode, so it is called fully pipelined. In this kind, pipeline registers are located after each round. High performance can be achieved by placing pipelining in between each round. Although iterative pipelining is one of the accepted approaches, in this research the fully expanded implementation have used for alliteration for lucidity. The output generated in every iteration is fed back as input data to next round. Pipeline registers are fundamentally used for a transitional data operation. Here insertion the

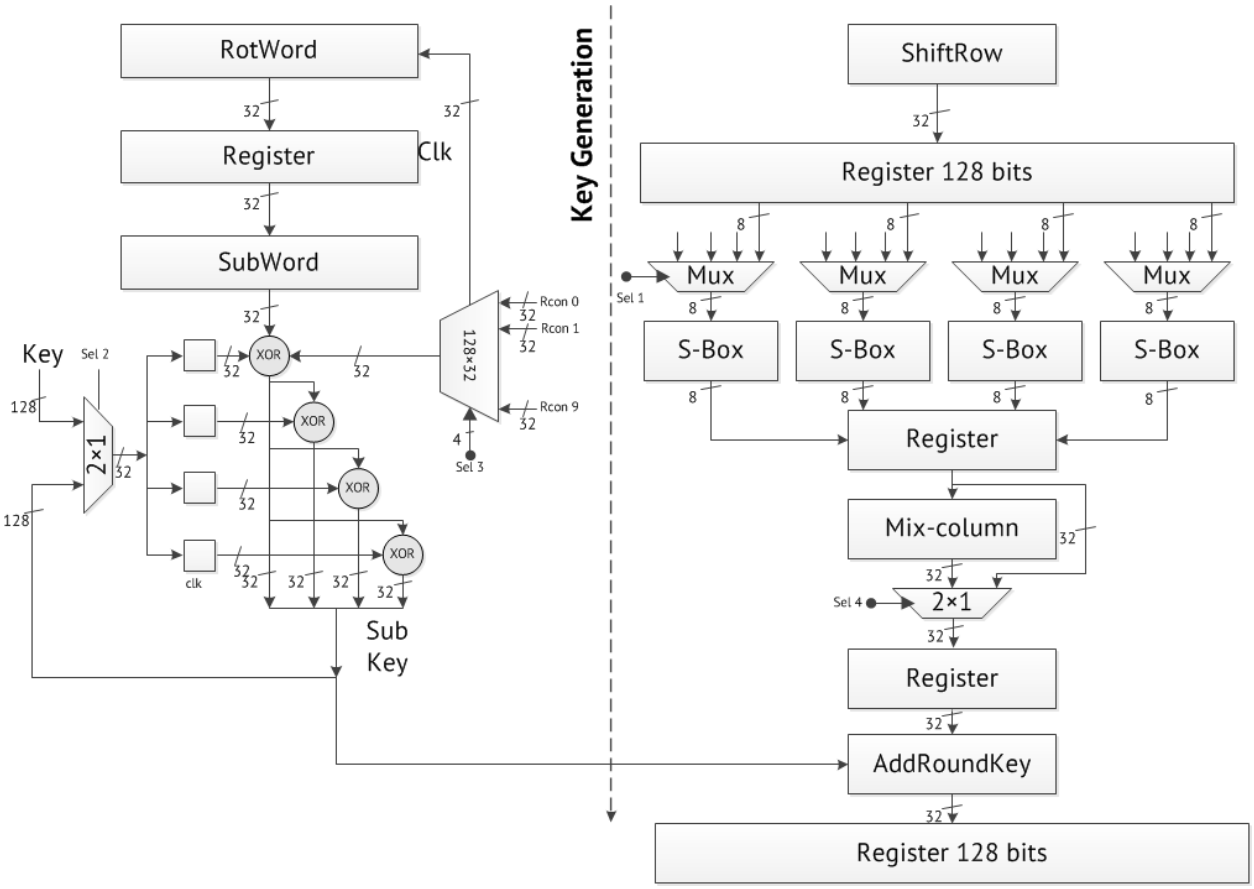



Figure 14: The schematic of the pipelined AES architecture of Encryption.

pipeline registers are the solution to achieving improved performance. Also, overall process completion cycle can be reduced using pipeline register.

# 7 Result and Analysis

## 7.1 Software Simulation

As stated before the xilinx ISE design suit 14.2 is used to mapping, translation, routing and placing. It allows synthesize the design and test each steps using TestBench. In this section, the comparison between simulation output and expected data for each step of AES are presented. A 128-bit stream are feed to SubByte step of AES is shown in Fig 15 where `subbytes_in` variable contains 128 bits as input, `subbytes_out` is the simulation output that is compared to the `correct_output` and it is matched. Similarly, we can see the similarities between output and expected output in both ShiftRow and MixColumn operations in Fig 16 and 17.

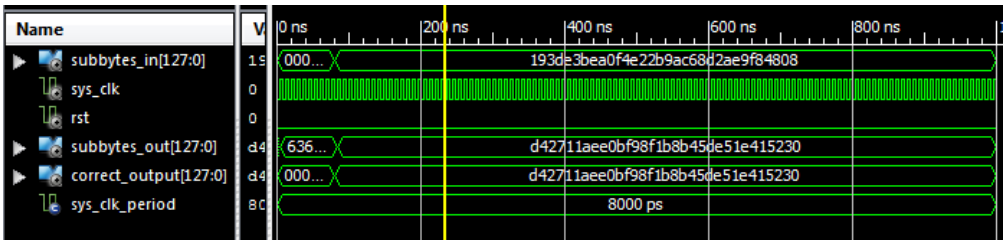



Figure 15: TestBench output of SubByte operation.

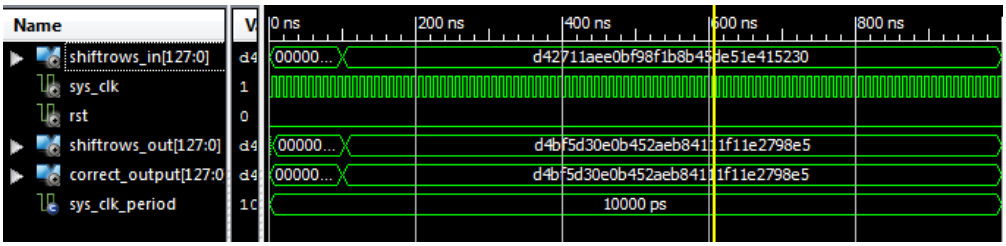



Figure 16: TestBench output of ShiftRow operation.

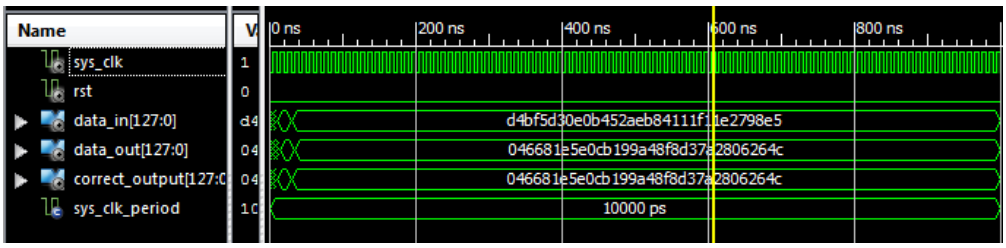



Figure 17: TestBench output of MixColumn operation.

## 7.2 Hardware Simulation

In this research, spartan6 XC6LX16-CS324 development board is used to measure the performance of FPGA implementation. We need to follow few steps to run VHDL codes in this board. First of all, generate programming file (.bit) using ISE deign suit as shown in Fig 18. The second step is connecting Spartan6 Board via USB for Programming the Board and via serial port using UART protocol to encrypt data which is shown in Fig 19. The third step is flash the .bit file to the board using NEXYS3, which is shown in Fig 20.

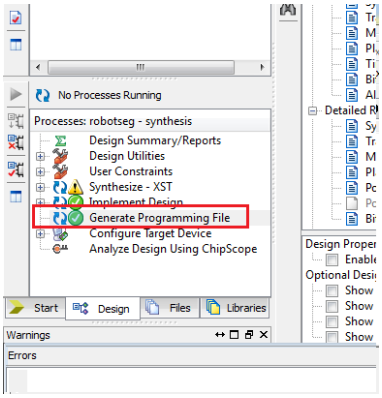



Figure 18: Generating programming file using Xilinx ISE design suit 14.2

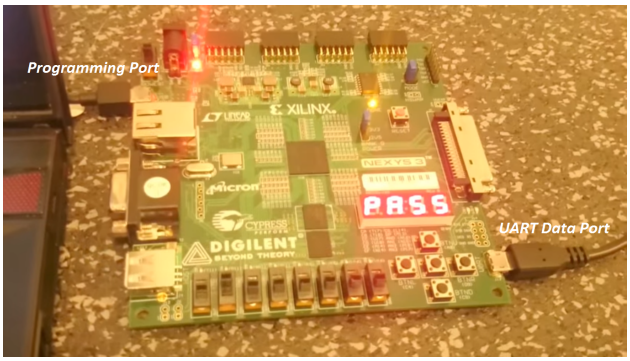



Figure 19: Connection between PC and Spartan6 Board via USB.

Finally, CoolTerm serial terminal is used to send plaintext to the hardware as input is shown in Fig 21 and we can observe the output cipher that is exactly similar to the expected cipher which means that encryption process is successfully done by the FPGA and it is shown in Fig 22.

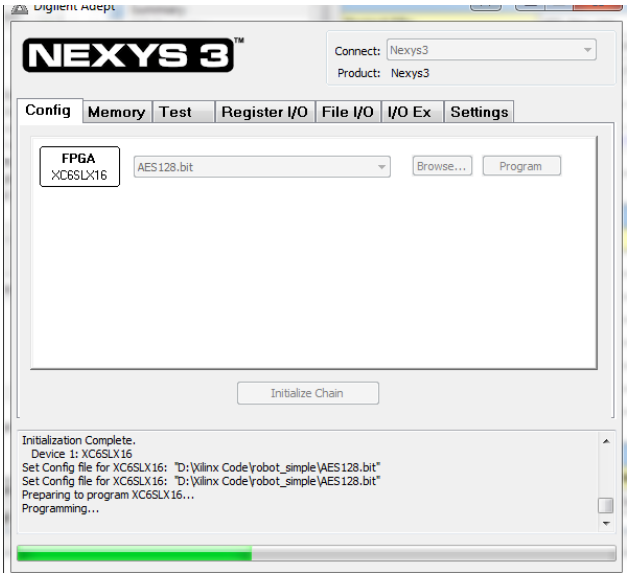



Figure 20: Transferring bit file to Sparta6 board via Digilient Adept (NEXYS3)

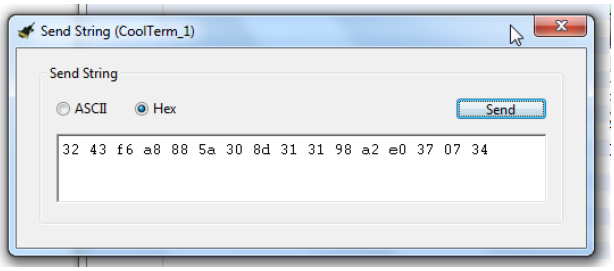



Figure 21: Sending PLAINTEXT via CoolTerm serial terminal to the board.

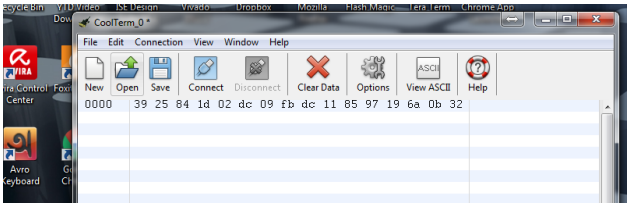



Figure 22: Output CIPHERTEXT from the board.

## 7.3 Performance Analysis

In FPGA, components utilization of spartan6 is shown in table 1 where the number available components versus used components for each type are given. For example, the number of Look Up Table (LUT) used is 1,079 out of 9,112 which is 11% of total LUT. Similarly, 68% of total IOBs is utilized in this experiment. The performance comparison of this FPGA with the previous FPGAs is given sequentially in table 2. However,

Table 1: Device utilization summary.

| Logic | Available | Used | Percentage |
|---|---|---|---|
| Slice Register | 18,224 | 962 | 5% |
| LUTs | 9,112 | 1079 | 11% |
| LUT-FF pairs | 1,340 | 701 | 52% |
| IOBs | 232 | 158 | 68% |
| Block RAM | 32 | 5 | 15% |
| BUFG | 16 | 1 | 6% |

Table 2: Performance comparison of this FPGA with previous FPGAs.

| Design | Target FPGA | Freq. (MHz) | Throughput (Mbps) | Latency (ns) | Area Slices | Data Path |
|---|---|---|---|---|---|---|
| Jarvinen et al [8] | Virtex- E XCV1000 | 129.2 | 16,500 | | 11,719 | Enc |
| Saggesse et al [12] | Virtex-E XCV2000 | 158 | 20,300 | | 5810 + | Enc |
| Standaert et al [13] | Virtex-E XCV3200 | 145 | 18,560 | | 15,112 | Enc |
| Hodjat et al [7] | Virtex-II Pro XC2P20 | 169.1 | 21,640 | 420 | 9,446 | Enc |
| Zhang (r=7), [16] | Virtex- E XCV1000E-8 | 168.4 | 21,556 | 416 | 11,022 | Enc/Dec |
| This work | Spartan6 Xc | 185.2 | 22,832 | 352 | 16,985 | Enc |

the figures quoted for the throughput versus latency. Also, frequency, data path status and area slice are given for each implementation. For example Hodja et al. [7] Virtex-II family for FPGA implementation and get 21,640 Mbps with delay 420 ns. They implemented data path only for the encryption operation. The number of area slices is 9,446. On the other hand, Zhang et al. [16] got approx. The same throughput with the same delay but it's device utilization is more than the previous one. However, this FPGA can encrypt 22,832 Mb data per second with 185.2 MHz frequency. The delay it takes is 352ns. The sum of slices that occupied the area is 16,985.

# 8 Related Work

Zhichao Yu et al [14] main design proposal is to proposed a s-box implementation that is nothing but a implementation of fast read only memory ROM with required size for subbyte and add round key operation. In contrary, Borkar et al [2] and [4] are presented a full hardware implementation of AES without efficiency or area optimization concern whereas Madian et al. [10] changed the fundamentals of AES algorithm a little and did hardware implementation for modified AES algorithm. The implementation performs both operations, encrypt and decrypt, but does not support the on-chip generation of internal keys. The architecture selected is a basic one with maximal resource sharing between encryption and decryption. AES-256 and 512 are implemented in the research paper [9] where they try to prove that their hardware is capable to encrypt 128bit data with the key size 512bit max which is the indication of more security.

However, ASIC and HW/SW co-design are presented in consequently ASIC [5] and Co-design [1]. Finally, a pipeline architecture of FPGA for encryption according to AES is presented in Good et al [6] where author try to implement 5 pipeline in instruction level and they got high throughput at the end. It developed a round partially pipeline cipher Idesign resulting in an efficient implementation in terms of area resources. Round keys are stored in internal registers and all keys must be loaded before encryption. Also, it implemented an inner-pipelining cipher/decipher unit with on the fly key scheduler. After completing a long literature review, a list of different types of AES implementation is given in table 3.

Table 3: Related work summary.

| Sl | Work Type | Reference |
|---|---|---|
| 1 | Partial Implementation | Zhichao Yu et al [14] |
| 2 | Full Implementation | Borkar et al [2] and [4] |
| 3 | Modified AES | Madian et al [10] |
| 4 | Different Key Size | AES-512 [9] |
| 5 | Area Optimization | Area/Delay Tradeoffs [15] |
| 6 | ASIC Implementation | ASIC [5] |
| 7 | HW/SW Co-design | Co-design [1] |
| 8 | Pipeline Architecture | Good et al [6] |

# 9 Conclusion and Future Work

This research presents high throughput, less delay Rijndael cipher for encryption using both 5 pipeline and parallel architecture, targeted towards the Spartan family of FPGAs. In this work, a pipeline register is taken place after each operation. Moreover, parallel architecture is introduced in both mix-column and key scheduling operation. The original goal of this work was compute the AES algorithm to generate cipher in high volume at time with low latency which is a big challenge. This is our first implementation of AES and it has 22,832Mbps throughput in 352ns latency which is a clear indication of a better implementation. Also, this architecture needs similar logic blocks and memories that takes in previous works. At the end, we have plan to continue this research to do area optimization.

# Bibliography


[1] Saambhavi Baskaran and Pachamuthu Rajalakshmi. Hardware-software co-design of aes on fpga. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pages 1118–1122. ACM, 2012.

[2] Atul M Borkar, RV Kshirsagar, and MV Vyawahare. Fpga implementation of aes algorithm. In *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, volume 3, pages 401–405. IEEE, 2011.

[3] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1998.

[4] Ashwini M Deshpande, Mangesh S Deshpande, and Devendra N Kayatanavar. Fpga implementation of aes encryption and decryption. In *Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on*, pages 1–6. IEEE, 2009.

[5] Kris Gaj and Pawel Chodowiec. Fpga and asic implementations of aes. In *Cryptographic engineering*, pages 235–294. Springer, 2009.

[6] Tim Good and Mohammed Benaissa. Aes on fpga from the fastest to the smallest. In *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 427–440. Springer, 2005.

[7] Alireza Hodjat and Ingrid Verbauwhede. A 21.54 gbits/s fully pipelined aes processor on fpga. In *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, pages 308–309. IEEE, 2004.

[8] Kimmo U Järvinen, Matti T Tommiska, and Jorma O Skyttä. A fully pipelined memoryless 17.8 gbps aes-128 encryptor. In *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, pages 207–215. ACM, 2003.

[9] Abidalrahman Moh, Yaser Jararweh, and Lo'ai Tawalbeh. Aes-512: 512-bit advanced encryption standard algorithm design and evaluation. In *Information Assurance and Security (IAS), 2011 7th International Conference on*, pages 292–297. IEEE, 2011.

[10] Ahmed Mohamed, Ahmed H Madian, et al. A modified rijndael algorithm and it's implementation using fpga. In *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*, pages 335–338. IEEE, 2010.

[11] NIST FIPS Pub. 197: Advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197:441–0311, 2001.

[12] Giacinto Paolo Saggese, Antonino Mazzeo, Nicola Mazzocca, and Antonio GM Strollo. An fpga-based performance analysis of the unrolling, tiling, and pipelining of the aes algorithm. In *Field Programmable Logic and Application*, pages 292–302. Springer, 2003.

[13] Francois-Xavier Standaert, Gael Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. Efficient implementation of rijndael encryption in reconfigurable hardware: improvements and design tradeoffs. In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 334–350. Springer, 2003.

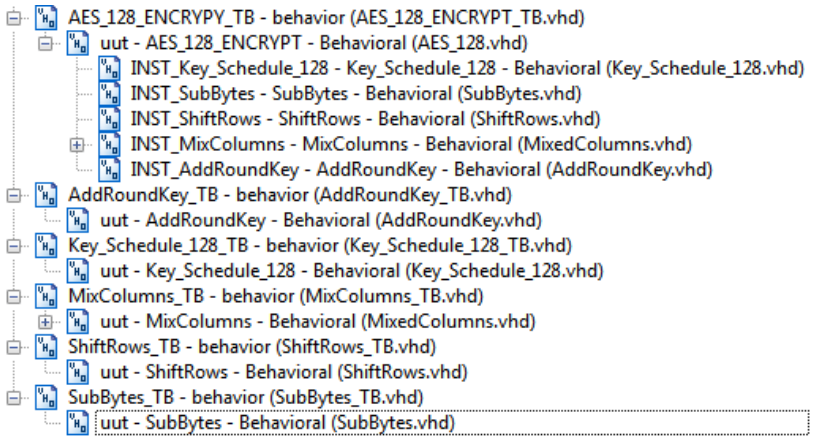
[14] Zhichao Yu. Implementation of aes s-box based on vhdl. In *Innovative Computing and Information*, pages 52–58. Springer, 2011.

[15] Joseph Zambreno, David Nguyen, and Alok Choudhary. Exploring area/delay tradeoffs in an aes fpga implementation. In *Field Programmable Logic and Application*, pages 575–585. Springer, 2004.

[16] Xinmiao Zhang and Keshab K Parhi. High-speed vlsi architectures for the aes algorithm. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(9):957–967, 2004.

# *Appendix*

Source code structure




Attachment

1. Source Code
2. Test Benches
3. Technical Report
4. Presentation
5. Report and presentation hardcopy.