



PIPELINED IMPLEMENTATION OF AES – 128 ENCRYPTION MODULE ON RECONFIGURABLE LOGIC

Lokesh Namdeo, Department of Electronics & Communication Engineering, Sagar Institute of Research & Technology, Bhopal

Prof. Himanshu Nautiyal, Department of Electronics & Communication Engineering, Sagar Institute of Research & Technology, Bhopal

Prof. Sangeeta Shukla, Department of Electronics & Communication Engineering, Sagar Institute of Research & Technology, Bhopal

Abstract: *This paper presents the hardware implementation of 128 bit AES encryption. The implementation is optimized in order to reduce delay. In order to reduce delay pipelined architecture is employed. Also sequential shifters in shift_row process are replaced by combinational shifters to increase maximum operating frequency. The target device is Virtex- 5 XC5VLX50-2FF676 speed grade -2.*

Keywords: AES – 128, FPGA, Rijndael Algorithm, FIPS – 197, Pipelined architecture

I. INTRODUCTION

With the development of information technology and widespread used, the security of sensitive data on Internet is especially important. Traditional cryptographic methods have failed to meet the requirements, especially to its security, speed and efficiency. The advanced encryption standard (AES) was adopted to replace the data encryption standard (DES) in 2000. AES specifies a federal information processing standard (FIPS) approved cryptographic algorithm that can be used to protect electronic data. AES is an unclassified publicly disclosed encryption algorithm available royalty free worldwide. This standard specifies the Rijndael algorithm. It is a symmetric block cipher that can encrypt and decrypt information. The data block that AES encrypt/decrypt is of 128 bit using 128, 192 or 256 bit cipher key. The original Rijndael algorithm support variable data block size and cipher block size but it was not taken by AES. So the Rijndael algorithm is taken as 128, 192 and 256 by AES and hence it is called AES 128, AES 192 and AES 256.

AES can be implemented in software or hardware but, hardware implementation is used in real time application. Main goal of AES hardware implementation is to minimize hardware



and lower the power consumption also maintain high throughput at highest operating frequency.

AES hardware implementation is very reliable, fast and conveniently suitable for high speed applications. It does not require system resources used in software during encryption or decryption process. Economically AES hardware implementation has low costs compared to software implementation which requires update. Hardware encrypted drives can easily reset which reduces down time in erasing data which gives better system performance. [1]

II. AES FRAMEWORK

Table 1 shows the structure of Rijndael Algorithm adopted by AES. AES uses the data block of 128 bits and Cipher key of 128, 192 or 256. The number of rounds for AES 128, AES 192 and AES 256 are 10, 12 and 14 respectively. In each round a same set of operations are performed [2].

Table 1: Structure of AES

AES type	Structure of AES		
	Cipher Key Length	Data Block Size	Number of rounds
AES 128	128	128	10
AES 192	192	128	12
AES 256	256	128	14

A. Encryption in AES

The process of encryption begins with the conversion of 128 bit data to a 4 x 4 state matrix of 16 bytes. Similarly the input cipher key is also converted to a 4 x 4 matrix of 16 bytes. For AES – 128, the cipher key matrix size of 16 byte is same the cipher key size 128 bits (16 bytes). For AES 192 and AES 256, the first 128 bits (16 bytes) are used in first round and the remaining bits are used in next round. The set of operations performed in each round are listed below.

1. Add_Round_key: in this operation the state matrix is xored with the cipher key matrix and a new state matrix is formed.
2. Sub_bytes: in this operation each byte of state matrix is replaced by a byte form a 256 byte table called SBOX.
3. Shift_rows: State matrix has 4 rows, in this operation the first row is not shifted, the second row is shifted left cyclically by 1 byte, the third row is shifted left cyclically by 2 bytes and the fourth row is left shifted cyclically by 3 bytes.



Mix_column: A linear transformation is used in this process. The mix column is process is used in 4 columns.

In this work, AES 128 encryption module is implemented, for AES 128, the number of rounds are 10. Figure 1 shows the process of encryption for AES 128. First the input matrix is added with the cipher key. Then in round 1 to round 9, 4 operations are repeated – sub_byte, shift_rows, mix column and add_round_key. In round 10 only sub_byte, shift_rows and add_round_key operation is performed. In each add_round_key operation a new cipher key is needed. This new cipher key is generated in parallel with the encryption process using key expansion logic. The same algorithm can be used for AES 192 and AES 256, just by increasing the number of rounds to 12 and 14 respectively from 10. The key expansion logic for AES 256 is slightly different. [6]

B. Decryption in AES

Figure 2 shoes the process of decryption in AES 128. The decryption process in AES is opposite to encryption process. The set of operation needed for decryption are listed below:

1. Add_round_key: this operation is exactly same to the add_round_key operation used in encryption. The encrypted data is xored with the cipher key.
2. Inv_Sub_bytes: in this operation each byte of state matrix is replaced by a byte form a 256 byte table called inv_SBOX. This SBOX is different from the one used in encryption.
3. Inv_Shift_rows: State matrix has 4 rows, in this operation the first row is not shifted, the second row is shifted right cyclically by 1 byte, the third row is shifted right cyclically by 2 bytes and the fourth row is right shifted cyclically by 3 bytes.
4. Inv_Mix_column: A linear transformation is used in this process. The inv_mix_columns is process is used in 4 columns. This is different from the one used in encryption.

The process of decryption is slightly different from the encryption; here the input is encrypted data. First the cipher key is expanded, this input cipher key is the same input key used in encryption process, the input cipher key is expanded to form 10 new keys K1 to K10 using a process called key_schedule. The process of decryption stars from the bottom, here the K10 key is used first, then K9, K8 and so on. At last the input cipher key is used to generate the decrypted data (plain text). A total of 11 keys are used in the process of



encryption and decryption. The decryption is discussed only for the sake of completion; it is not implemented in this work.

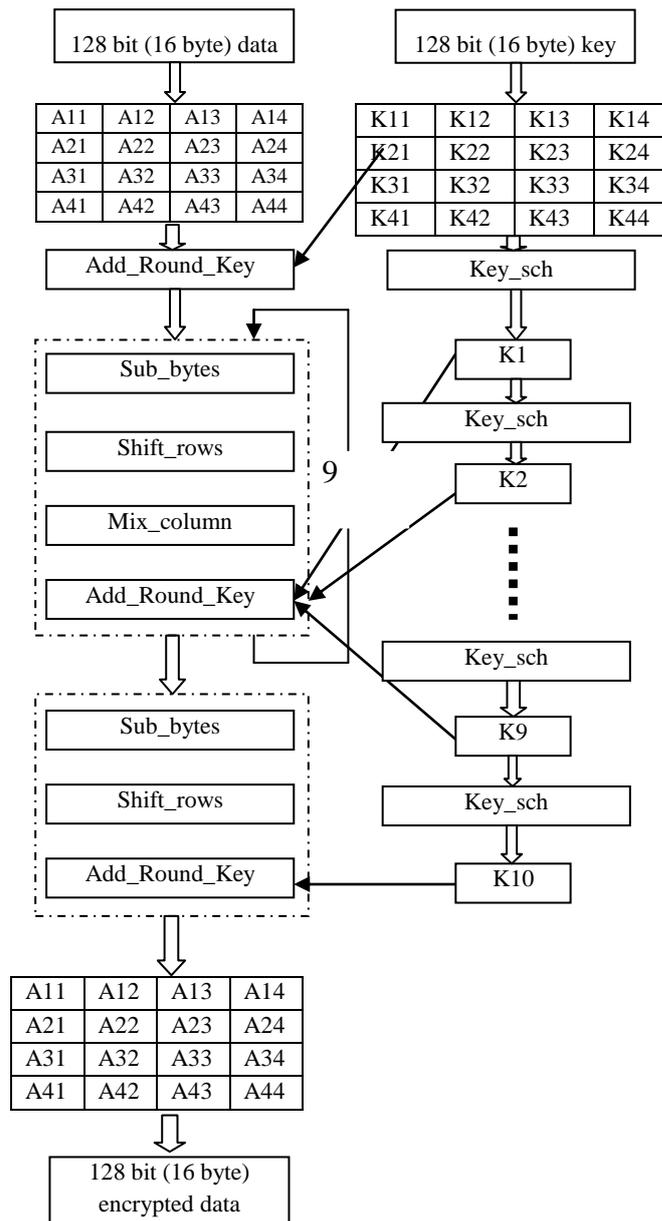


Figure 1: AES 128 encryption algorithm

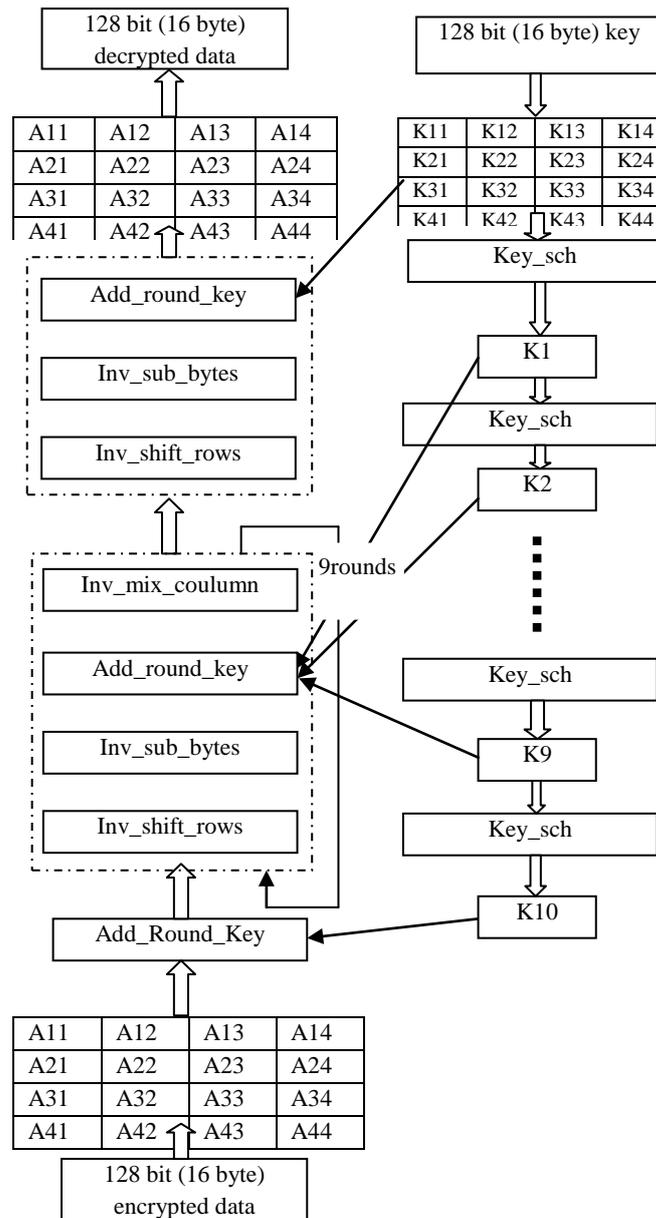


Figure 2: AES 128 decryption algorithm

III. AES – 128 ENCRYPTION MODULE PIPELINED IMPLEMENTATION

In this section the pipelined hardware implementation of AES – 128 Encryption is discussed. The high level block diagram is shown in figure 3.

AES round 1 to round 9 has four processes, 1. Sub_Bytes, 2. Shift_Rows, 3. Mix_Column and 4. Add_round_Key. AES round 10 has only sub_bytes, Shift_rows and Add_round_key process running in it. Each of these processes are discussed in following sections. The decryption process is not implemented in this work.

1. **Add_round_key unit:** This unit has an array of XOR gates, which performs bit by bit XOR operation. This operation is used in Add_round unit (both encryption and decryption), AES_encryption unit (round 1 to 9), AES_encryption unit (round 10), AES_decryption unit (round 1 to 9) and AES_decryption unit (round 10).
2. **AES_encryption unit (round 1 to 9):** This unit has four internal units: add_round unit, Sub_bytes unit, Shift_rows unit, Mix_Column unit. add_round unit has already been discussed, remaining three units are discussed.

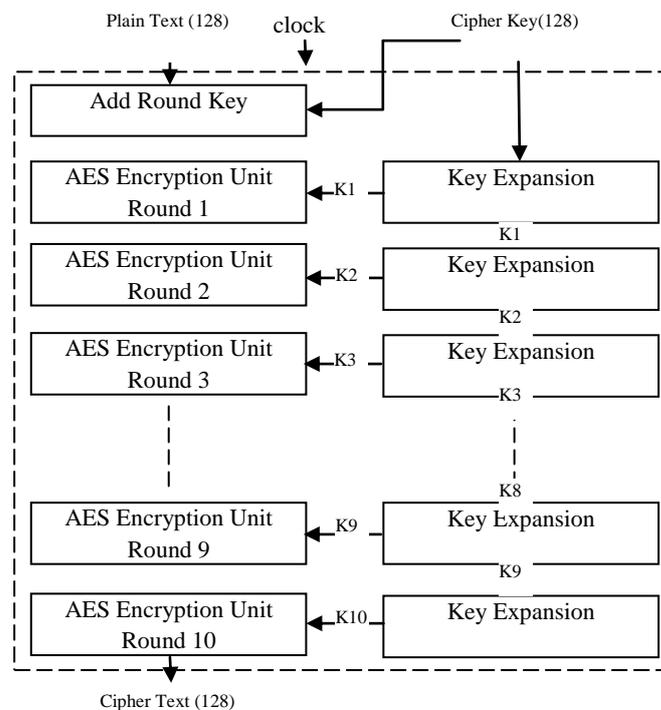


Figure 3: High Level Block Diagram AES-128-Encryption Module – Pipelined Architecture

- A. **Sub_byte unit:** In this unit a memory table called SBOX is used. This SBOX table has 256 entries arranged in a matrix form of size 16 x 16. [6]. The input state matrix to the sub_byte unit has 16 (4 x 4) bytes, these 16 bytes are replaced by the values stored in SBOX. To replace all these 16 bytes simultaneously 16 such SBOX are used. This will reduce delay.
- B. **Shift_rows unit:** In this unit three shifters are used to shift last three rows. The first shifter is set to shift by left by one byte, the second shifter is set to shift row by 2 bytes and the third shifter is set to shift the row by 3 bytes. Employing three parallel shifters will reduce delay. Figure 4 shows the arrangement of state matrix after shift rows.

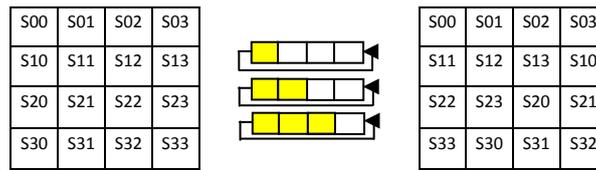


Figure 4: Arrangement of state after shift_rows

C. *Mix_column unit*: In mix column a linear transformation is applied to the state matrix. The input state matrix to the mix_unit and the output matrix are depicted in figure 5.

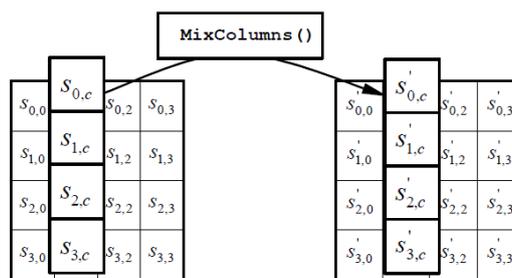


Figure 5: Arrangement of state after Mix_column

Each column of the state matrix is multiplied (galiios multiplication) by constant 4 x 4 matrix and a new column matrix is formed, the new column replaces the old column.

$$\begin{bmatrix} s0c' \\ s1c' \\ s2c' \\ s3c' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s0c \\ s1c \\ s2c \\ s3c \end{bmatrix}$$

$$S0c' = (\{02\}.S0c) \text{ xor } (\{03\}.S1c) \text{ xor } S2c \text{ xor } s3c$$

$$S0c' = (\{02\}.S0c) \text{ xor } (S1c \text{ xor } (\{02\}.S1c)) \text{ xor } S2c \text{ xor } s3c$$

The original equation requires x_time(2) and x_time(3) operation, whereas in this work we have replaced all x_time(3) operation by a x_time(2) and XOR gate. X_time(3) is comparatively complex to x_time(2), this results in the reduction of FPGA resources.

The remaining three equations are listed below along with their reduced form.

$$S1c' = s0c \text{ xor } (\{02\}. S1c) \text{ xor } (\{03\}.S2c) \text{ xor } s3c$$

$$S1c' = S0c \text{ xor } (\{02\}. S1c) \text{ xor } (\{02\}.s20) \text{ xor } s20 \text{ xor } s3c$$

$$S2c' = S0c \text{ xor } S1c \text{ xor } (\{02\}.S2c) \text{ xor } (\{03\}.s3c)$$

$$S2c' = S0c \text{ xor } S1c \text{ xor } (\{02\}.S2c) \text{ xor } (\{02\}.S3c) \text{ xor } s3c$$

$$S3c' = (\{03\}.S0c) \text{ xor } S1c \text{ xor } .S2c \text{ xor } (\{02\}.s3c)$$



$$S3c' = (\{02\}.S0c) \text{ xor } S0c \text{ xor } S1c \text{ xor } S2c \text{ xor } (\{02\}.S3c)$$

Figure 6 shows the logic diagram for mix_column operation for a single column.

The x_time(2) operation is simple, first the input byte is shifted by 1 bit and the MSB of input byte is checked, if it is 1 then the shifted output is XORED with 1B to produce the output else the shifted byte is the final output of x_time(2).

Figure 6 shows the mix_column operation for one single column, 4 such logic units are required to produce the mix_column output for the complete 4 x 4 matrix.

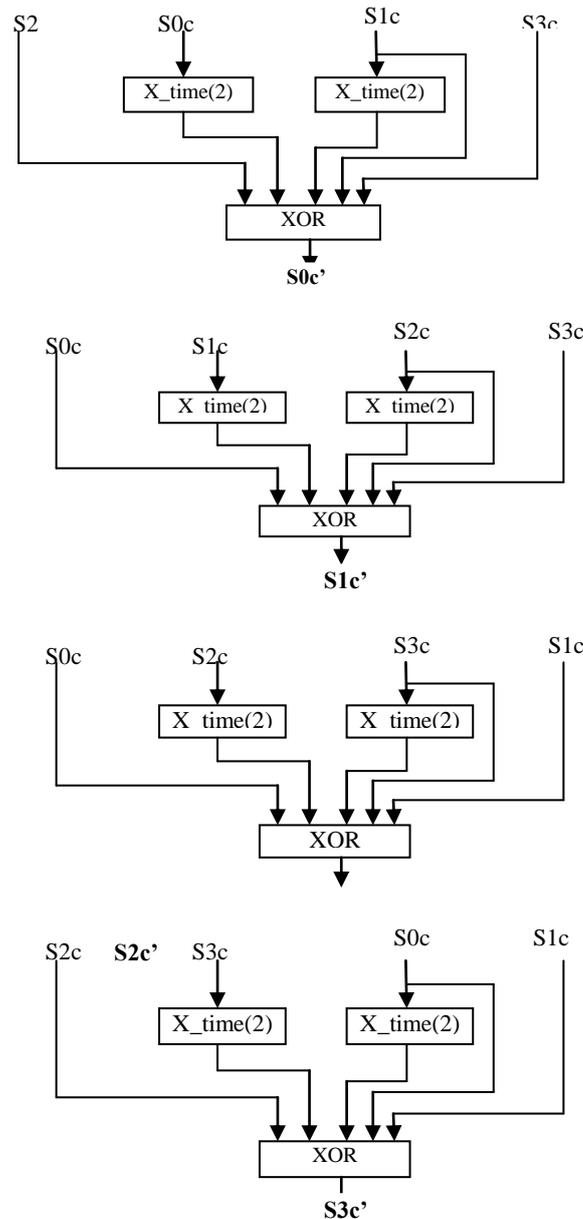


Figure 6: Mix_coulumn – Logic diagram



3. **AES_encryption unit (round 10):** AES last round consists of only three operation sub_bytes, Shift_rows and Add_round_key. The mix column operation is absent in this particular round.
4. **Key_expansion_unit:** As seen from figure 1 and figure 2 11 different cipher keys are needed in every round of encryption and decryption process. One key out of these 11 is the input cipher key and the remaining 10 keys are generated using a key expansion unit. This key expansion unit performs 4 operations rot_word, sub_word, rcon, XOR.[6]
 - A. **Rot_word unit:** The input to this operation is a row of 4 bytes. These 4 bytes are shifted left by one byte using a barrel shifter.
 - B. **Sub_word unit:** The input to this operation is also of 4 bytes, each byte of the input is replaced by a byte from the SBOX table used in encryption.
 - C. **RCON unit:** This is also a memory table which returns a 4 byte value depending upon the current round.

In this work pipelined architecture is implemented. At each clock an input is applied and corresponding output is available after 10 clock cycles. After 10 clocks the output is continuous stream. The advantage of using pipelined architecture is that the system can operate at higher data rates by introducing latency elements (registers) at each block.

IV. SIMULATION & RESULTS

In The target device used for AES – 128 encryption module implementation is XC5VLX50 - 2FF276 speed grade -2. The design is simulated using xilinx 14.1i ISIM tool. Many input vectors are used to test our design and it is found working faithfully. Figure 7 depicts the simulation result of AES-128 during encryption, output will be received after 10 clocks.

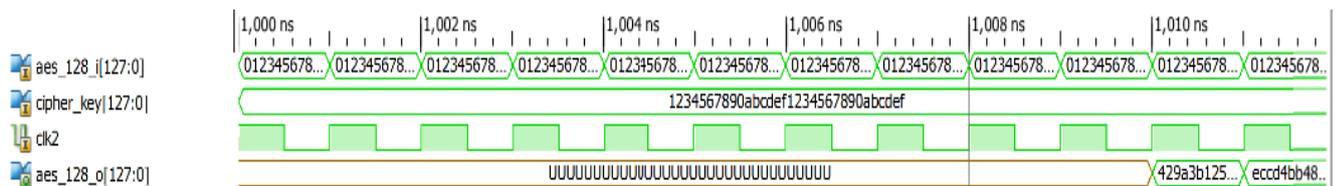


Figure 7: Simulation AES – 128 Encryption Module

The design is synthesized using xilinx XST tool. Table 2 shows the FPGA resources used, these results are compared with the previous design available in literature.



Table 2: Device Utilization Summary

Design	CLB Slices	Clock Mhz	Process
[11]	2162/ 2057	186/ 185	Decryption/ Encryption
[12]	1953/ 1947	195/ 192	Encryption/ Decryption
[13]	NA	60	Encryption
[14]	1005	183Mhz	Encryption
Proposed	1800	279	Encryption

In this work we have optimized the design for delay by introducing latency elements. The maximum operating frequency of this design is 278.827MHz which is much higher than the maximum operating frequency of other designs listed in table.

V. CONCLUSION

In implementation of AES – 128 is presented in this work. The target device is Virtex 5. The design is tested using XILINX 14.1 I and simulated using ISIM. Test vectors of fips document are used to check for any behavioral errors and no behavioral errors are found. We have reduced the number of x_time for the mix column operation this reduces the FPGA resource usage. Also fast shifters are used in this design, it decreases the delay. Also pipelined architecture is used to reduce the delay or in other words to increase the maximum operating frequency of the design.

In future hardware efficiency of AES – 128 can be improved by reusing resources such as using a single encryption and decryption block instead of one for round 1 to round 9 and one for round 10.

REFERENCES

- [1] Hammad I, El-Sankary K, El-Masry E, "High-speed AES encryptor with efficient merging techniques," IEEE Embedded Systems Letters, 2010, pp.67-71.
- [2] I ZHANG Y L, WANG X G, "Pipelined implementation of AES encryption based on FPGA," 2010 IEEE International Conference on Information Theory and Information Security, Piscataway: IEEE, 2010, pp. 170-173.
- [3] FAN C-P, HWANG J-K, "Implementations of high throughput sequential and fully pipelined AES processors on FPGA." ISPACS 2007: Proceeding of 2007 International Symposium on International Signal Processing and Symposium and Communication Systems, Piscataway: IEEE, 2007, pp. 353-356.



- [4] SKLAVOS N, KOUFOPAVLOU O, "Architectures and VLSI implementations of the AES-proposal Rijndael," IEEE Transactions on Computers, 2002, 51(12), pp. 1454-1459.
- [5] BORKAR A M, KSHIRSAGAR R V, VYAWAHARE M V, "FPGA implementation of AES algorithm," The 3rd International Conference on Electronics Computer Technology, Piscataway: IEEE, 2011, 3, pp.401-405.
- [7] Joan Daemen, Vincent Rijmen. AES Proposal: Rijndael. The Rijndael Block Cipher.
- [8] Vincent Rijmen, "Efficient implementation of the of the rijndael SBox," 2000.
- [9] Fischer V, Drutarovsky M, Chodowiec P, "InvMixColumn decomposition and multilevel resource sharing in AES implementations," IEEE Transactions on Very Large Scale Integration Systems, 2005, 13(8), pp. 989-992.
- [10] Chien M Ta, Chee Hong Yong, Wooi Gan Yeoh, "A 2.7mW, 0.064mm² linear-in-dB VGA with 60dB tuning range, 100MHz bandwidth, and two DC offset cancellation loops," IEEE International Workshop on Radio Frequency Integration Technology, Austria: Graz, 2005, pp. 74-77.
- [11] J. Balamurugan, Dr. E. Logashanmugam "High Speed Low Cost Implementation of Advanced Encryption Standard on FPGA" ICCET 2014.
- [12] International Journal of Electronics & Telecommunication and Instrumentation Engineering, "High Speed Low Cost Implementation Of Advanced Encryption Standard On FPGA" March 2010.
- [13] Bing Ji, Liejun Wang and Qinghun Yang "New version of AES-ECC Encryption based on FPGA in WSN" Journal of software engineering, 2015.
- [14] Swierczynski, Pawel, et al. "Protecting against Cryptographic Trojans in FPGAs." Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on. IEEE, 2015.