A Novel Approach of Area Optimized and pipelined FPGA Implementation of AES Encryption and Decryption

K.Sireesha[*], S.Madhava Rao[**]

[*] M.TECH Student MLEcollge,singaraya konda,prakasam(DT),A.P,INDIA
[**] ASSOC.Professor, Department of E.C.E MLEcollge,singarayakonda, prakasam(DT), A.P,INDIA

Abstract- In this paper we present an architecture to implement Advanced Encryption Standard (AES) Rijndael algorithm in reconfigurable hardware. Rijndael algorithm is the new AES adopted by the National Institute of Standards and Technology (NIST) to replace existing Data Encryption Standard (DES). Compared to software implementation, hardware implementation of Rijndael algorithm provides more physical security as well as higher speed. The first factor to be considered on implementing AES is the application. High-speed designs are not always desired solutions. In some applications, such as mobile computing and wireless communications, smaller throughput is demanded. Architecture presented uses memory modules (i.e., Dual-Port RAMs) of Field-Programmable Gate Array (FPGAs) for storing all the results of the fixed operations (i.e., Look-Up Table), and Digital Clock Manager (DCM) that we used effectively to optimize the execution time, reduce design area and facilitates implementation in FPGA. The architecture consumes only 326 slices plus 3 Block Random Access Memory (BRAMs). The throughput obtained was of 270 Mbits/s. The target hardware used in this paper is Spartan XC3S500E FPGA from Xilinx. Results are presented and compared with other reference implementations, as known from the technical literature. An implementation of high speed AES algorithm based on FPGA is presented in this paper in order to improve the safety of data in transmission. The mathematic principle, encryption process and logic structure of AES algorithm are introduced.. The simulation results show that the high-speed AES encryption algorithm implemented correctly. Using the method of AES encryption the data could be protected effectively. The presented architecture can be used in a wide range of embedded applications.

I. Introduction

Security of data is becoming an important factor for a wide spectrum of applications, including communication systems, wireless devices, and many other embedded applications. Resistance against known attacks is one of the main properties that an encryption algorithm needs to provide. When a new attack is demonstrated as effective (also in term of computation time), the update of the encryption system is a real necessity to guarantee the security of data. In October 2000, National Institute of Standards and Technology (NIST) selected Rijndael [1] as the new Advanced Encryption Standard (AES) [2], in order to replace the old Data Encryption Standard (DES) [3] [4]. It offers a good „„,combination of security, performance, efficiency, implementability and flexibility‟‟‟ [5]. AES specifies a Federal Information Processing Standards (FIPS) approved cryptographic algorithm that is used to safely protect electronic data [6]. The selection process included performance evaluation on both software and hardware platforms and many hardware architectures were proposed. However, most of these architectures simply transcript the algorithm into hardware designs without relevant optimizations and tradeoffs. Moreover, the throughput and area constraints considered are often unrealistic as shown by the recently published results. In this paper, we present an architecture for the AES Rijndael algorithm based on three techniques to improve the implementation of the AES Rijndael Algorithm:

- Look-Up Table to facilitate implementation.
- Digital Clock Manager (DCM) to optimize execution time.
- Memory modules (Dual-Port RAMs) to reduce design area.

The proposed architecture uses only a relatively small area and lower execution time and it can be used for a wide range of applications. However, most of publications on implementations of AES only provide performance and area figures without interfaces and registers.

II. DESCRIPTION OF AES RIJNDAEL ALGORITHM AND PREVIOUS WORK

2.1. Description of AES Rijndael Algorithm:

The Rijndael is a block cipher, which operates on different keys and block lengths: 128 bits, 192 bits, or 256 bits. The input to each round consists of a block of message called the state and the round key. It has to be noted that the round key changes in every round. The state can be represented as a rectangular array of bytes. This array has four rows; the number of columns is denoted by Nb and is equal to the block length divided by 32. The same could be applied to the cipher key. The number of columns of the cipher key is denoted by Nk and is equal to the key length divided by 32. The cipher consists of a number of rounds - that is denoted by Nr - which depends on both block and key lengths. Each round of Rijndael encryption function consists mainly of four different transformations: SubByte, ShiftRow, MixColumn and key addition. On the other hand, each round of

Rijndael decryption function consists mainly of four different transformations: InvSubByte, InvShiftRow, InvMixColumn, and key addition. The output of the above transformations is called the 'State'. The state consists of the same byte length as each block of the message. The description of the four transformations of the Rijndael cipher and their inverses will be given below.

$$\text{State} = \begin{bmatrix} d_{15} & d_{11} & d_7 & d_3 \\ d_{14} & d_{10} & d_6 & d_2 \\ d_{13} & d_9 & d_5 & d_1 \\ d_{12} & d_8 & d_4 & d_0 \end{bmatrix} \quad \text{----2.1}$$

2.1.1 SubByte Transformation:

The SubByte transformation is a non-linear byte substitution, operating on each of the state bytes independently. The SubByte transformation is done using a once-pre-calculated substitution table called S-box. That S-box table contains 256 numbers (from 0 to 255) and their corresponding resulting values. The SubByte transformation applied to the State can be represented as follows:

$$SB(\text{State}) = \begin{bmatrix} SB(d_{15}) & SB(d_{11}) & SB(d_7) & SB(d_3) \\ SB(d_{14}) & SB(d_{10}) & SB(d_6) & SB(d_2) \\ SB(d_{13}) & SB(d_9) & SB(d_5) & SB(d_1) \\ SB(d_{12}) & SB(d_8) & SB(d_4) & SB(d_0) \end{bmatrix} \quad \text{---2.2}$$

2.1.2 InvSubByte Transformation:

The InvSubByte transformation is done using a once-pre-calculated substitution table called InvS-box. That table (or InvS-box) contains 256 numbers (from 0 to 255) and their corresponding values.

$$SR(SB(\text{State})) = \begin{bmatrix} SB(d_{15}) & SB(d_{11}) & SB(d_7) & SB(d_3) \\ SB(d_{10}) & SB(d_6) & SB(d_2) & SB(d_{14}) \\ SB(d_5) & SB(d_1) & SB(d_{13}) & SB(d_9) \\ SB(d_0) & SB(d_{12}) & SB(d_8) & SB(d_4) \end{bmatrix} \quad \text{---2.3}$$

2.1.3. ShiftRow Transformation

In ShiftRow transformation, the rows of the state are cyclically left shifted over different offsets. Row 0 is not shifted; row 1 is shifted over one byte; row 2 is shifted over two bytes and row 3 is shifted over three bytes. Thus, the ShiftRow transformation proceeds as follows:

2.1.4. InvShiftRow Transformation

In InvShiftRow transformation, the rows of the state are cyclically right shifted over different offsets. Row 0 is not shifted, row 1 is shifted over one byte, row 2 is shifted over two bytes and row 3 is shifted over three bytes.

InvShiftRows() is the inverse of 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. The bottom three rows are cyclically shifted by N_b □□*shift*(*r*, *Nb*) bytes, where the shift value *shift(r,Nb)* depends on the row number, and is given in the section 2.3

Specifically, the **InvShiftRows()** transformation proceeds as follows:

$$s'_{r,(c+shift(r,Nb)) \bmod Nb} = s_{r,c} \quad \text{for } 0 < r < 4 \text{ and } 0 \le c < Nb \quad \text{-------2.4}$$

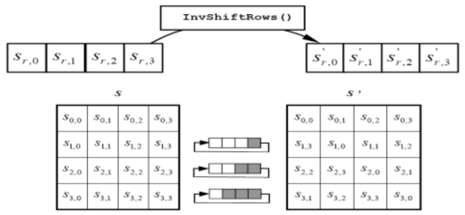


Figure 1 InvShiftRows () transformation

2.1.5. MixColumn Transformation

In Mix-Column, the columns of the state are considered as polynomials multiplied by a fixed polynomial, C(x) given by:

$$c(x) = '03'x^3 + '01'x^2 + '01'x + '02'. \quad \text{--2.5}$$

The MixColumn transformation can be written in a matrix multiplication as follows:

$$R = MC(SR(SB(\text{State}))) = \begin{bmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{bmatrix} \otimes \begin{bmatrix} SB(d_{15}) & SB(d_{11}) & SB(d_7) & SB(d_3) \\ SB(d_{10}) & SB(d_6) & SB(d_2) & SB(d_{14}) \\ SB(d_5) & SB(d_1) & SB(d_{13}) & SB(d_9) \\ SB(d_0) & SB(d_{12}) & SB(d_8) & SB(d_4) \end{bmatrix} \quad \text{--2.6}$$

2.1.6. InvMixColumn Transformation

In InvMixColumn, the columns of the state are considered as polynomials multiplied by a d(X) fixed polynomial , defined by:

$$c(x) \otimes d(x) = '01' \quad \text{---2.7}$$

$$d(x) = '0B'x^3 + '0D'x^2 + '09'x + '0E' \quad \text{--2.8}$$

2.1.7. AddRoundKey

AddRoundKey performs an addition (bitwise XOR) of the State with the RoundKey:

$$AK(R) = \begin{bmatrix} R_{15} & R_{11} & R_7 & R_3 \\ R_{14} & R_{10} & R_6 & R_2 \\ R_{13} & R_9 & R_5 & R_1 \\ R_{12} & R_8 & R_4 & R_0 \end{bmatrix} \oplus \begin{bmatrix} rk_{15} & rk_{11} & rk_7 & rk_3 \\ rk_{14} & rk_{10} & rk_6 & rk_2 \\ rk_{13} & rk_9 & rk_5 & rk_1 \\ rk_{12} & rk_8 & rk_4 & rk_0 \end{bmatrix} \quad \text{--2.9}$$

The inverse operation (InvAddRoundKey (IAK)) is trivial.RoundKeys are calculated with the key schedule for every AddRoundKey transformation. In AES-128, the original cipher key is the first (rk^0) used in the additional AddRoundKey at the beginning of the first round.rk^i, where $0<i\le 10$, is calculated from the previous rk^{i-1}. Let q(J)($0\le J\le 3$) be the column j of the $rk^{i-1}$and let w(J) be the column j of the rk^i . Then the new rk^i is calculated as follows:

$$W(0)=q(0) \oplus (Rot(SB(q(3))) \oplus rcon^i$$
$$W(1)=q(1) \oplus W(0)$$
$$W(2)=q(2) \oplus W(1)$$
$$W(3)=q(3) \oplus W(2)$$

2.2 THE BASIC STRUCTURE OF THE AES:

A full description of the AES is detailed in FIPS197 [2]. However, for sake of understanding we again outlinethe AES standard structure. AES is a block cipher developed in effort to addressthreatened key size of Data Encryption Standard (DES).It allows the data length of 128, 192 and 256 bits, andsupporting three different key lengths, 128, 192, and 256bits. AES can be divided into four basic operation blocks where data are treated at either byte or bit level. The byte structure seems to be natural for low profile microprocessor (such as 8-bit CPU and microcontrollers).

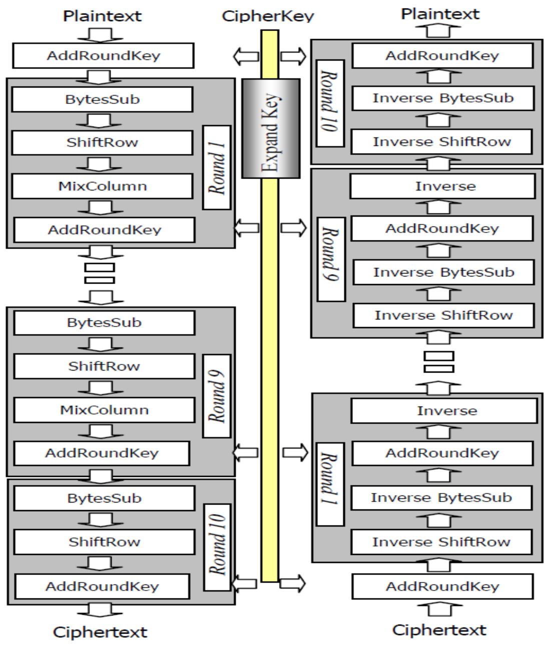


Fig. 2: Rijdael Encryption and Decryption Process

The array of bytes organized as a 4×4 matrix is also called "state" and those four basic steps; BytesSub, ShiftRow, MixColumn, and AddRoundKey are also known as layers. These four layer steps describe one round of the AES. The number of rounds is depended onthe key length, i.e., 10, 12 and 14 rounds for the key length of 128, 192 and 256 bits respectively. The block diagram of the system with 128 bit data is shown above.

BytesSub Transformation:

This operation is a non-linear byte substitution. It composes of two sub-transformations; multiplicative inverse and affine transformation. In most implementations, these two sub-steps are combined into a single table lookup called S-Box.

ShiftRow Transformation:

This step is a linear diffusion process, operates on individual rows, i.e. each row of the array is rotated by a certain number of byte positions.

MixColumn Transformation:

This is also a linear diffusion process. Column vector is multiplied (in GF(28)) with a fixed matrix where bytes are treated as a polynomial of degree less than 4.

AddRoundKey:

Each byte of the array is added (respect to GF(2)) to a byte of the corresponding array of round subkeys. Excluding the first and the last round, theAES with 128 bit round key proceeds for nine iterations. First round of the encryption performs XOR with the original key and the last round skips MixColumn transform. Round keys are generated by a procedure call round key expansion or key scheduling. Those sub-keys are derived from the original key by XOR of two previous columns.

For columns that are in multiples of four, the process involves round constants addition, S-Box and shift operations. All four layers described above (including key scheduling) have corresponding inverse operations. The deciphering is therefore the reverse order of the ciphering process. However, it should be noted that the MixColumn reverse operation requires matrix elements that are quite complicated compared to {01}, {02} or {03} of the forward one. Thisresults in the more complex deciphering hardware compared with the ciphering hardware.
In the next section we demonstrate how the standard procedure for MixColumn transform is rewritten in order to ease its hardware implementation.

III. IMPLEMENTATION OF AES 256

As we know that we have a plaintext of 128 bits and key of 256 bits size. The number of rounds in AES 256 is 14. The first round consists of all the five operation like Preround operation , subbyte ,shift rows , mix columns and Add round key operations. From 2nd round to 13th round have four operations subbyte ,shift rows , mix columns and Add round key operations. And the last 14th round consists of three operations subbyte , shift rows and Add round key operations.

In AES 256 the process of generating the key is each round key is a 256-bit array generated as,

- Input key of 256 bit is divided into eight parts of 32 bits as columns in a matrix4*8.
- Last column is taken and given as input to S box.
- The output of S box is given shift rows operation.
- The above output MSB side 8 bits are XORed with the round constant(i.e round constant value is different for different rounds)
- The above output is xored with 0th column of input key it gives
- The above output is taken as 0th column of the generated new key.
- 0th column of new key is xored with 1st column of input key gives 1st column of new key.
- 1st column of new key is xored with 2st column of input key gives 2nd column of new key.
- 2nd column of new key is xored with 3rd column of input key gives 3rd column of new key.
- The above obtained 3rd column of new key is given to S box .
- The output of S box is xored with 4th column of input key which gives 4th column of new key.
- 4th column of new key is xored with 5th column of input key which gives 5th column of new key.
- 5th column of new key is xored with 6th column of input key which gives 6th column of new key.
- 6th column of new key is xored with 7th column of input key which gives 7th column of new key.

In this way we generate new keys of 256 bits in AES 256 algorithm by attaching the eight obtained columns of new key.
In the first round this key of 256 bits is divided into two parts each of 128 bits size and this keys of 128 bits are used one in the preround operation (i.e xor operation between plaintext and key) and other is used in Add Round key operation .At the end of round function there will be 128bit output and 256 bit key output obtained from key generation process.

In the second round as we don't have preround operation so the round ouput of 1st round is applied as input to sub byte and the remaining operations are same as round 1. This process of round operation is repeated upto 13th round operation. And the last round is similar to previous round , the only change is it doesn't hav mix columns operation.

AES Decryption process is nothing but the inverse process of encryption AES. The output of the encryption process i.e cipher text is given as input to decryption. The same input key of 256 bits is used as another input. The input key is given to the key generation module to generate new keys as we do in the encryption process. The output keys generated are given as inputs to inverse mix columns which gives keys for the fourteen rounds of decryption.

Now in the first round all the five operation like Preround operation , subbyte ,shift rows , mix columns and Add round key operations. From 2nd round to 13th round have four operations subbyte ,shift rows , mix columns and Add round key operations. And the last 14th round consists of three operations subbyte, shift rows and Add round key operations.As in the encryption process the output of first round is taken as input to the next round upto the final fourteenth round and the output of fourteenth round is taken as final output.

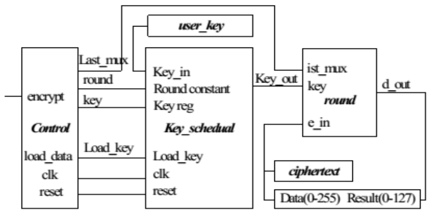


Fig.3 AES Decryption

The design uses a synchronous clock in order to make the circuit works with a unified clock and uses pipeline architecture to improve the working speed. Figure 5 shows the system implementation structure. Round module includes Sub Bytes, Shift Rows, Mix Columns, Add Round Key and an S-box matrix. Sub Bytes is a substituted operation to execute the operation and the affine transformation on finite field. Shift Rows is a cycle shift with bytes for unit. The most important process in Mix Columns is the multiplication on finite field. Add Round Key is a process that makes a 128 bits key to exclusive or the data in state one by one. S-box is a matrix that be defined to make a nonlinear replacement for Sub Bytes.

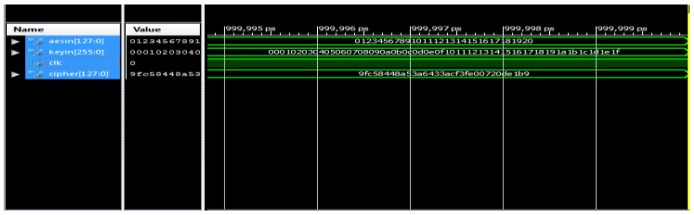
IV. SIMULATION RESULTS

At first each operation like substitution byte, shift row, mix column and key expansion operation are simulated using verilog code and design files. Output is realized using input vectors and key from NIST publication [4].

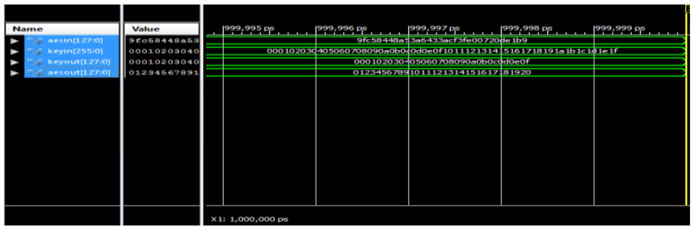
In encryption module each standard round is simulated and output is verified. Figure 4 shows the simulation of one standard round where all basic operations (Substitution bytes shift rows mix columns and add round key) are performed and latency is observed.

The simulation results of full encryption module and decryption is shown in Figure 4.1(a),(b) where input vectors and keys are given from NIST standard publication [4] and output was verified.

Input Plaintext:00112233445566778899aabbcc ddeefff
Input Cipher Key :
000102030405060708090a0b0c0d0e0f1011121314151617
Cipher text:8ea2b7ca516745



```
Name            Value
aesin[127:0]    01234567891    0123456789101112131415161718 1920
keyin[255:0]    00010203040    000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
clk             0
cipher[127:0]   9fc58448a53    9fc58448a53a6433acf3fe00720de1b9
```



```
Name            Value
aesin[127:0]    9fc58448a53    9fc58448a53a6433acf3fe00720de1b9
keyin[255:0]    00010203040    000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
keyout[127:0]   00010203040    000102030405060708090a0b0c0d0e0f
aesout[127:0]   01234567891    0123456789101112131415161718 1920
```

Figure 4.1(a),(b) encryption and decryption module

The output result of the encryption was found accurately after 15 clock cycle from the starting of encryption process.

So the latency of encryption is only 15clock cycle. In the Figure 5 the generated last key(14th) is shown as keyout and latency is observed by keyready function. As the device used is Altera EP2C35F672C6 from Cyclone II family has maximum clock frequency of 50MHz, so the encryption through put will be 6.4Gbps as per clock cycle encrypt 256 bits data samples. If other device having more clock frequency is used then throughput can be increased linearly.

The output of the algorithm are visualized by the 16 seven segment display of the FPGA board where 256 bit cipher produced from 256 bits plaintext which is the implementation of simulation results in Figure 5.Inputs are given by toggle switch or by input data to the program. 256 bit encryption key are also given to the code directly.

Overall the simulation of XILINX software and implementation results on the FPGA board found accurate with reduced latency of 15 clock cycle.

Conclusion and future work:

From our work we have concluded that the concept of Pipelined AES architecture can be practically implemented. It has been observed that the implementation of AES Encryption onthe FPGA is successful and several data input. The cipher key can be changed with respect to the user requirements. The result shows that the design with the pipelining technology and special data transmission mode can optimize the chip area effectively. Meanwhile, this design reduces power consumption to some extent, for the power consumption is directly related to the chip area. Therefore the encryption device implemented in this method can meet some practical applications. As the S-box is implemented by look-up-table in this design, the chip area and power can still be optimized. So the future work should focus on the implementation mode of S-box. Mathematics in Galois field (28) can accomplish the bytes substitution of the AES algorithm, which could be another idea of further research.

While implementing the AES Algorithm m, the critical aspect was the area utilization. Which was done using implantation of functions for different sub modules in the algorithm the work has approximately reduced around 10% utilization on chip as compared to basic available modules.

We have successfully implemented AES encryption on FPGA.We have achieved the data encryption as per 100% accuracy as compared to data encryption module.

REFERENCES

[1] S. Morioka and A. Satoh "A 10-Gbps Full AES-CryptoDesign with a Twisted BDD S-Box Architecture", IEEE Transactions on VLSI Systems, Vol. 12, No. 7, July 2004, pp.686-691.

[2] V. Fischer and M. Drutarovsky, "Two Methods of Rijndael Implementation in econfigurable Hardware", Proc. CHES, Vol. 2162, 2001, pp.81-96.

[3] C-P. Su, T-F. Lin, C-T. Huang; and C-W. Wu, "A High-Throughput Low-Cost AES Processor", Communications Magazine, IEEE, Vol. 41, Issue: 12, December 2003, pp. 86-91.

[4] NIST, "ADVANCED ENCRYPTION STANDARD (AES,Rijndael)", FIPS-197, November 2001, http://csrc.nist.gov/encryption/aes/.

[5] T. Ichikawa, T. Kasuya, and M. Matsui, "HardwareEvaluation of the AES Finalists," Proc. 3rd AES Candidate Conf., 2000.

[6] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and Performance Testing of a 2.29-Gb/s Rijndael Processor," IEEE J.Solid-State Circuits, Vol. 38, No. 3, Mar. 2003, pp. 569-572.

[7] William Stallings "Cryptography and Network Security – Principl

[8] Zhenzhen Liu. Implementation of AES Encryption based on FPGA. Modern electronic technology, 2007,23(3),pp:103-104.

[9] Jinxiang Shao. The high speed implementation of AES encryption algorithm based on FPGA. Sichuan：Southwest oil college, 2005.

[10] Ma Su,Wang Ji.High speed implementation of AES algorithm based on FPGA.The application of PLD/CPLD/FPGA, 1008-0570(2009)09-2-0127-02,pp:127-128.

[11] Shanxin Qu,Guochu Shou,Yihong Hu,Zhigang Guo,Zongjue Qian. High Throughput Pipelined Implementation of AES on FPGA. International Symposium on Information Engineering and Electronic Commerce.2009

[12] Jianghua Deng, Zhihua Hu, Jiping Niu. The Implementation and rearch of AES Algorithm. Microcomputer Applications,21(7),2005.pp:58-59.

[13] Tian Yun,Xu-Wen-Bo,Hu Bin. Xilinx ISE Design Suite 10.xGuide.Posts&Telecom Press,Bei jing,2008

AUTHORS

First Author – K.Sireesha, M.TECH Student, MLEcollege, Singaraya Konda, Prakasam (DT), A.P, INDIA, Sire20k10@gmail.com
Second Author – S.Madhava Rao, ASSOC.Professor, Department of E.C.E MLEcollge, Singarayakonda, Prakasam (DT), A.P, INDIA, smadhav.r2@gmail.com