RESEARCH ARTICLE OPEN ACCESS

FPGA Implementation of an Area Optimized Architecture for 128 bit AES Algorithm

S Ramanathan*, Prof. Prayline Rajabai C**
**(M.Tech 2nd Year, VLSI Design, School of Electronics Engineering (SENSE), VIT University, Vellore)*
*** (Assistant Professor, Micro & Nanoelectronics, School of Electronics Engineering (SENSE), VIT University, Vellore)*

ABSTRACT
This paper aims at FPGA Implementation of an Area Optimized Architecture for 128 bit AES Algorithm. The conventional designs use a separate module for 32 bit byte substitution and 128 bit byte substitution. The 32 bit byte substitution is used in round key generation and the 128 bit byte substitution is used in the rounds. This report presents a modified architecture of 128 bit byte substitution module using a single 32 bit byte substitution module to reduce area.The AES encryption and decryption algorithm were designed using Verilog HDL. The functionality of the modules were checked using ModelSim. The simulations were carried out in ModelSim and Quartus II. The algorithm was implemented in FPGA and achieved a 2% reduction in the total logic element utilization.

Keywords **-** 128 bit AES, Area Optimized Architecture, FPGA, ModelSim, Quartus II

I. INTRODUCTION

These days encryption is coming out as an indispensable part of all data networks and information processing system to protect all forms of data being utilized by the system. Encryption is the process of converting the sensitive information (plaintext) into an incomprehensible string of characters (ciphertext). Decryption is the process of obtaining plaintext back from the ciphertext.

Over the years there were many encryption algorithm that came into picture. The National Institute of Standards and Technology (NIST) called out for nominees for Advanced Encryption Standards in 1997. There were 22 submissions out of which 7 didn't satisfy all the requirements which left 15 submissions in total. Among the 15 submissions, 5 were chosen as finalists, Mars, RC6, Rijndael, Serpent, Twofish. Rijndael by Vincent Rijmen and Joan Daemen was chosen as winner among the 5 finalists. Later in 2001 NIST published the specifications in the Federal Information Processing Standards (FIPS) Publication 197.

AES falls under the category of symmetric cipher [1] i.e. an encryption algorithm where the same key is used for both encryption and decryption. Hence, the encryption key must be kept secret at any cost in symmetric cipher and obtaining plaintext from ciphertext and algorithm information has to be impossible without the encryption key.

Depending on the size of key, there are different versions of AES algorithm available today. These are AES-128, AES-192, AES-256, where the numbers 128, 192 and 256 are the key sizes[2].

In software implementations of encryption algorithms, the secrecy of key is compromised as the OS where the encryption software runs is prone to attacks. Furthermore, software implementation may result in some compatibility issue due to which we may not get required parallelism or speed. On the other hand, these drawbacks are taken care of in hardware implementations which makes it a viable solution.

II. ADVANCED ENCRYPTION STANDARD (AES)

The encryption algorithm converts 128-bit input (plaintext) to 128-bit ciphertext using a cipher key. The cipher key is a string of 128, 192 or 256 bits. 32 bits are considered as 1 word, hence key of 128, 192 or 256 bits are said to be of 4, 6 or 8 words.

Depending on the size of key, AES algorithm is categorized into AES-128, AES-192, AES-256, where the numbers 128, 192 and 256 are the key sizes. The number of rounds of encryption depends on the size of cipher key. There are 10, 12 and 14 rounds in AES-128, AES-192 and AES-256 respectively. The same is tabulated in Table 1.

TABLE I. AES VARIATIONS

AES Version	Key Length (N_k words)	Block Size (N_b words)	Number of Rounds (N_r rounds)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

AES algorithm operations are performed on a 4x4 array of bytes. This two dimensional 4x4 array of bytes is known as state[3]. Initially, the state contains

plaintext. Then some set of permutations and substitutions are performed by the cipher. Once the cipher operations are over the final value is copied to cipher text. The following figure illustrates the same.

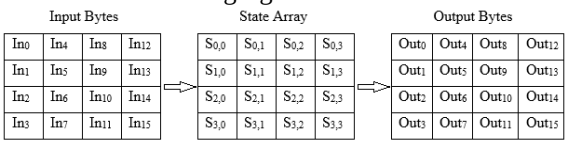

Fig. 1. State Array input and output

A. AES Encryption Algorithm

As shown in Fig.2 the AES encryption contains the following steps

- Key Expantion
- Initial Round
  - Add round key
- Rounds
  - Sub bytes
  - Shift rows
  - Mix columns
  - Add round key
- Final round
  - Sub bytes
  - Shift rows
  - Add round key

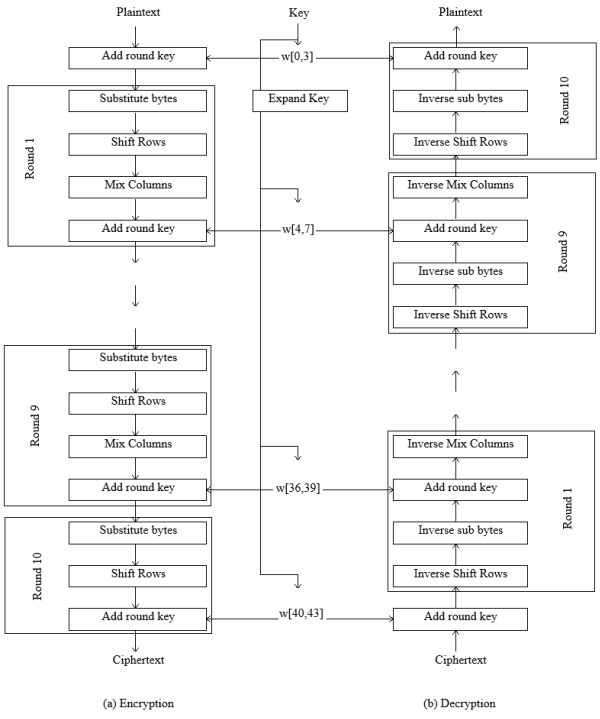

Fig. 2. AES encryption and decryption algorithm

1) Key Expansion

All round keys are obtained from cipher key. For each encryption round 4 words of round keys are required thus total of 44 round keys are required for key size of 128 bit[4]. The first word w0 is obtained from first four bytes of the encryption key, the next word w1 is the next four bytes and so on as shown in Fig.3.

$$\begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}$$

$$\Downarrow$$

$$[\, w_0 \;\; w_1 \;\; w_2 \;\; w_3 \,]$$

Fig. 3. Initial step of key expansion

The algorithm subsequently expands the words [w0,w1,w2,w3] into a 44-word key schedule that can be labelled w0, w1, w2, w3, ..., w43 as shown in Fig.4.

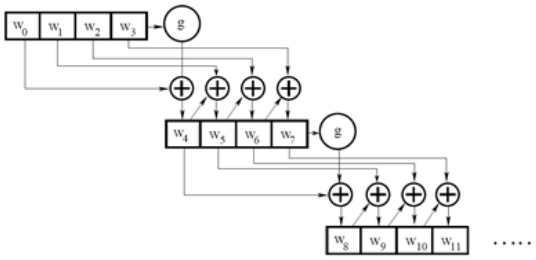

Fig. 4. Key expansion in AES

2) Add Round Key

Every byte of current state is added (bitwise XOR) with the round key values[5]. The round key values and state are added column wise in the following way.

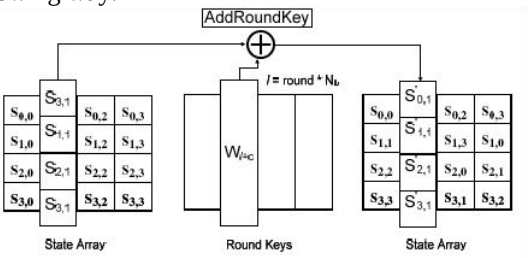

Fig. 5. Add round key transformation

3) SubBytes Transformation

In subbytes, as the name indicates, each byte is replaced by another byte. The byte to be replaced with is obtained from s-box as shown in Fig.6

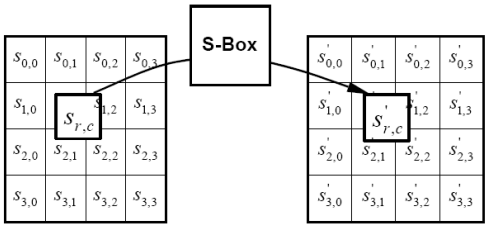

Fig. 6. Subbyte transformation in AES

The s-box is implemented as lookup table (LUT) to minimize complexity[6]. Fig.7 shows the s-box LUT.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Fig. 7. S-box LUT for Subbyte transformation

As we can see, there are 256 entries in S-box LUT shown in Fig.7. The most significant 4 bits are considered as x and least significant 4 bits are considered as y. Fig.8 illustrates how byte substitution 53 is done using s-box.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Fig. 8. Byte substitution 53 is done using s-box

The value corresponding to 5 in x and 3 in y is ED. So 53 is replaced by ED during subbyte transformation.

4) ShiftRows Transformation

During shift rows, every row except the first one is shifted cylindrically. Each row has different offset i.e. each row is shifted by different amount to its left as shown in Fig.9.

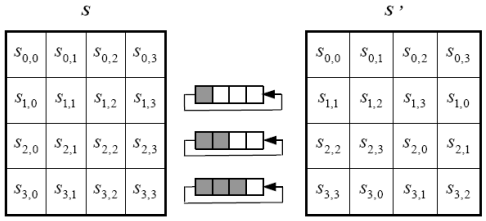

Fig. 9. Shift rows transformation in AES

5) MixColumns Transformation

Every column of state is treated as a 4 term polynomial the finite field GF(2^8)[7]. It can be expressed as a matrix multiplication[8] as shown in Fig.10

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Fig. 10. Mix column transformation matrix

The mix column transformation using the above transformation matrix during encryption is shown in Fig.11

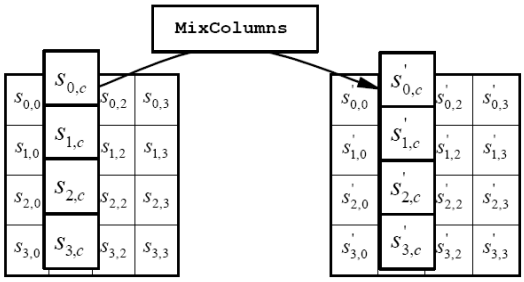

Fig. 11. Mix column transformation during encryption

Mix column and shift rows together provide substantial diffusion i.e. a small change in plaintext has a drastic effect on the cipher text.

B. AES Decryption Algorithm

As shown in Fig.2 AES decryption contains the following steps

- Key expantion
- Initial round
  - Add round key
- Rounds
  - Inv shift rows
  - Inv sub bytes
  - Add round key
  - Inv mix columns
- Final round
  - Inv shift rows
  - Inv sub bytes
  - Add round key

In decryption the form of key schedule remains the same but the transformation is replaced by the respective inverse transformation and the sequence of the transformation also changes.

1) InvShiftRows Transformation

During the inverse shift rows, every row except the first one is shifted cylindrically in the opposite direction of encryption, i.e. the row that was shifted left n times during shift rows will be shifted right n times during inverse shift rows as shown in Fig.12

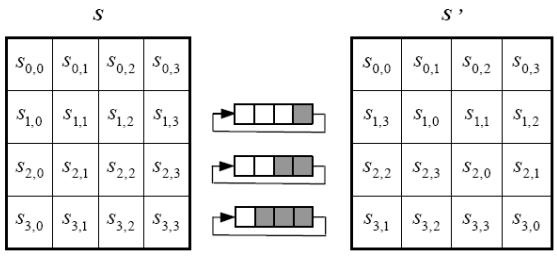

Fig. 12. Inverse shift rows transformation in AES

2) InvSubBytes Transformation

The invsubbytes is similar to subbytes in the sense that each byte is replaced by another byte. The difference is that the byte to be replaced is obtained from inverse s-box as shown in Fig.13 rather than the s-box used during encryption.

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

Fig. 13. Inverse S-box LUT for Inverse subbyte transformation

3) InvMixColumns Transformation

The inverse mix columns is used to perform the inverse operation of the mix columns used during encryption. It can be expressed as a matrix multiplication as shown in Fig.14

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Fig. 14. Inverse mix column transformation matrix

The inverse mix column transformation using the above transformation matrix during decryption is shown in Fig.15

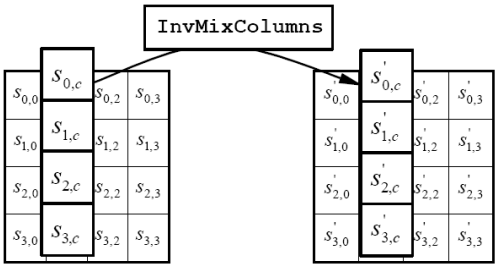

Fig. 15. Inverse mix column transformation during decryption

III. 128BIT AES DESIGN AND IMPLEMENTATION

This section covers the design and implementation aspect of the 128-bit AES algorithm. A synthesizable hardware model was created using bottom-up approach i.e. the lower level modules were designed first then the higher level modules were designed by instantiating the lower level modules. The modules were modelled in behavioural style.

Most of the existing designs use a separate subbyte module of 32 bits and 128 bit size. 32 bit subbyte is used during key expansion and 128 bit subbyte is used during the rounds.

The unique feature of the proposed design is that it uses the same 32bit subbyte to implement 128 bit subbyte thereby reducing hardware or resource utilization. The proposed subbyte module is shown in the figure below.

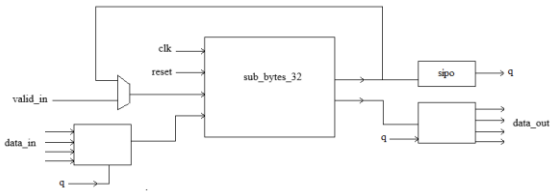

Fig. 16. Modified 128 bit subbyte using a single 32 bit subbyte

The 128 bit subbyte operation is performed by using the 32 bit subbyte sequentially 4 times. In the first iteration, the subbyte of first 32 bits is performed, in the next iteration, the subbyte of next 32 bit is performed and so on. The mapping of all the 4 sets of 32 bit data_in and 32 bit data_out to 128 bit data_in and 128 bit data_out is done using individual bits of sipo.

The scheduling of data_out is as follows, if q[0] is high, data_out[127:96] is received, if q[1] is high, data_out[95:64] is received, if q[2] is high, data_out[63:32] is received, if q[3] is high, data_out[31:0] is received.

The scheduling of data_in is as follows, if q[1] is high, data_in [127:96] has to be sent, if q[2] is high, data_in[95:64] has to be sent, if q[3] is high, data_in[63:32] has to be sent, otherwise data_in[31:0] has to be sent.

In this way, only the hardware corresponding to 32bit subbyte is used and the hardware corresponding to 128 bit subbyte is saved as the same existing hardware is used 4 times sequentially to perform the 128 bit subbyte.

IV. RESULTS

The RTL coding of the Encryption and Decryption Module has been completed. AES Encryption and Decryption modules were designed using Verilog HDL. Simulation were be carried out in ModelSim and Quartus II. The functionality of the Encryption and Decryption module has been checked using ModelSim. The Encryption algorithm was implemented in FPGA.

Total logic elements used is observed to be 10,784/33,216 (32%) in the device EP2C35F672C7 which 2% less than 11,187/33,216 (34%) as in [2] as shown in the table II. The frequency of operation is found out to be 160 MHz.

TABLE II. RESULTS

Architecture	**Total Logic Elements**	**Percentage of logic elements used**
Luanlan [2]	11,187	34%
Our design	10,784	32%

REFERENCES

Definitive standard:

[1] FIPS 197, *"Advanced Encryption Standard (AES)"*, November 26, 2001.

Journal Papers:

[2] Luanlan, *"The AES Encryption And Decryption Realization Based On FPGA"*, Seventh International Conference on Computational Intelligence and Security, 2011

[3] Hoang Trang, Nguyen Van Loi, *"An efficient FPGA implementation of the Advanced Encryption Standard algorithm"*, IEEE 2012

[4] Douglas Selent, *"Advanced Encryption Standard"*, Rivier Academic Journal, *Volume 6, Number 2*, Fall 2010

[5] Sujatha Hiremath, M.S.Suma, *"Advanced Encryption Standard Implemented on FPGA"*, Second International Conference on Computer and Electrical Engineering, 2009

[6] Tuan Anh Pham, Mohammad S. Hasan and Hongnian Yu, *"Area and Power optimisation for AES encryption module implementation on FPGA"*, 18th International Conference on Automation & Computing, Loughborough University, Leicestershire, UK, September 2012

[7] Dong Chen, Guochu Shou, Yihong Hu, Zhigang Guo, *"Efficient Architecture and Implementations of AES"*, 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), 2010

[8] Xin Cai, Rong Sun, Jingwei Liu, *"An ultrahigh speed AES processor method based on FPGA"*, 5th International Conference on Intelligent Networking and Collaborative Systems, 2013.

Thesis:

[9] Avinash Kak, *"AES: The Advanced Encryption Standard"*, Purdue University