International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Design and Implementation of Galios Field Based AES-256 Algorithm for Optimized Cryptosystem

Veerendra Babu Dara[1], P. Sankara Rao[2]

[1] M.Tech (VLSI & SD) Student, Department of Electronics and Communication Engineering, Sri Vasavi Institute of Engineering and Technology, Nandamuru, Machilipatnam, JNTUK, Kakinada, A.P., India

[2] Associate Professor, Department of Electronics and Communication Engineering, Sri Vasavi Institute of Engineering and Technology, Nandamuru, Machilipatnam, JNTUK, Kakinada, A.P., India

Abstract: ***All of the cryptographic algorithms we have looked at so far have some problem. The earlier ciphers can be broken with ease on modern computation systems. As a result of technology scaling and higher integration densities there may be variations in parameters and noise levels which will lead to larger error rates at various levels of the computations. As far as memory applications are concerned the soft errors and single event upsets are always a matter of problem. In this paper presents an optimized composite field arithmetic based S-Box implemented in four stage pipeline. In traditional look up table (LUT) approaches, the unbreakable delay is longer than the total delay of the rest of operations in each round. LUT approach consumes a large area. It is more efficient to apply composite field arithmetic in the SubBytes transformation of the AES algorithm. It not only reduces the complexity but also enables deep sub pipelining such that higher speed can be achieved. Isomorphic mapping can be employed to convert GF(2^8) to GF($(2^2)^2)^2$), so that multiplicative inverse can be easily obtained. SubBytes and InvSubBytes transformations are merged using composite field arithmetic. It is most important responsible for the implementation of low cost and high throughput AES architecture. As compared to the typical ROM based lookup table, the presented implementation is both capable of higher speeds since it can be pipelined and small in terms of area occupancy (1609/29504 Slices on a Spartan 3 XC3S1600E-4, fg484).***

Keywords: Advanced Encryption Standard (AES) **[4]**, Composite Field Arithmetic [3], Cryptography, Galios Field **[1], [13]**, Memory, Xilinx ISE 12.1 Design suite and Verilog.

1. Introduction

Cryptography is very much important in the field of data transmission with the rapid growing number of Internet and wireless communication users. Advanced Encryption Standard, (AES) is proposed by National Institute of Standards and Technology, (NIST). The AES is a Federal Information Processing Standard, (FIPS). It is a cryptographic algorithm that is used to protect data. The AES algorithm can be used for both encryption and decryption of data. Encryption converts data or plaintext to cipher text. Decryption converts cipher text back to its original form, which is called plaintext. Cryptographic keys of 128, 192, and 256 bits can be used to encrypt and decrypt data in blocks of 128 bits.

The main idea is to employ composite field arithmetic in the computation of the multiplicative inversion in the SubByte/InvSubBytes transformation of the AES algorithm. So that deep sub pipelining is applied, and hardware complexity is reduced. This paper adopts alternative architecture to achieve small area. High throughput can be achieved without using LUT and memory so that no unbreakable delay is introduced in the architecture. In traditional look up table (LUT) approaches, the unbreakable delay is longer than the total delay of the rest of operations in each round. Pipelining and Subpipeling cannot be applied to LUT approaches. The LUT approach is not suitable for resource constrained use as it consumes a large area. Composite field arithmetic can be used to solve the problem.

The process of finding multiplicative inverse in GF(2^8) is very complicated by direct method. But, two fields of the same order are said to be isomorphic. So that we can use an isomorphic transform to convert GF(2^8) to GF((2^4)^2) and further to GF(((2^2)^2)^2).

The algorithm takes a plaintext block size of 128 bits, or 16 bytes as input. The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197 (FIPS 197) on November 26, 2001[4] this block is depicted as a 4x4 square matrix of bytes. This block is copied into the state array, which is transformed at each stage of encryption or decryption. After the final stage, state is copied to an output matrix. Similarly, the key is considered as a square matrix of bytes. This key is then expanded into an array of key schedule words. Each byte in the state matrix is an element in Galois Field GF (2^8) which is constructed with the irreducible polynomial $p(x) = x^8 + x^4 + x^3 + x + 1$.

2. Cryptography

The branch of cryptology dealing with the design of algorithms for encryption and decryption, intended to ensure the secrecy and/or authenticity of messages. An original message is known as the plain text, while the coded message is called the cipher text. The process of converting the plain text to cipher text is known as enciphering or encryption; restoring the plain text from the cipher text is deciphering or decryption. The many schemes used for enciphering constitute the area of study known as cryptography. Such a scheme is known as a cryptographic system or a cipher. Techniques used for deciphering a message without any

knowledge of the enciphering details fall into the area of cryptanalysis. The cryptanalysis is what the lay person's calls breaking the code. The areas of cryptography and cryptanalyst is together are called cryptology.

2.1 Encryption

The original intelligible message or data that is fed into the algorithm is the input. Encryption algorithm performs varies substitutions and transformations on the plain text. Secret key is also given as input to encryption algorithm. The key is a value independent of the plain text. The algorithm reproduces the different output depending on the specific key being used at the time. The exact substitutions and transformation performed by the algorithm depend on the key. The output depends on the plain text and the secret key. For a given message, two different keys will produce two different cipher texts. The cipher text is an apparently random stream of data, as it stands, is unintelligible.

2.2 Decryption

This is essentially the encryption algorithm run in reverse. It takes the cipher text and the secret key and produces the original plain text. The remaining part of this paper as follow: Section 3 describes the AES Algorithm operation. The S-Box construction method was described in Section 4. Section 5 contains the Galios Field GF (2^8). Section 6 contains Proposed S-Box architecture. The Synthesis, Simulation result & conclusion are drawn from Section 7, 8 & Section 9 respectively.

3. AES Algorithm

The algorithm consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key. The first N-1 rounds consist of four distinct transformation functions:
1. SubByte,
2. ShiftRows,
3. MixColumns and
4. AddRoundKey.

The final round contains only three transformations there is no MixColumns transformation. Initially there is a single transformation (AddRoundKey) before the first round.

Each transformation takes one or more 4x4 matrices as input and produces a 4x4 matrix as output.

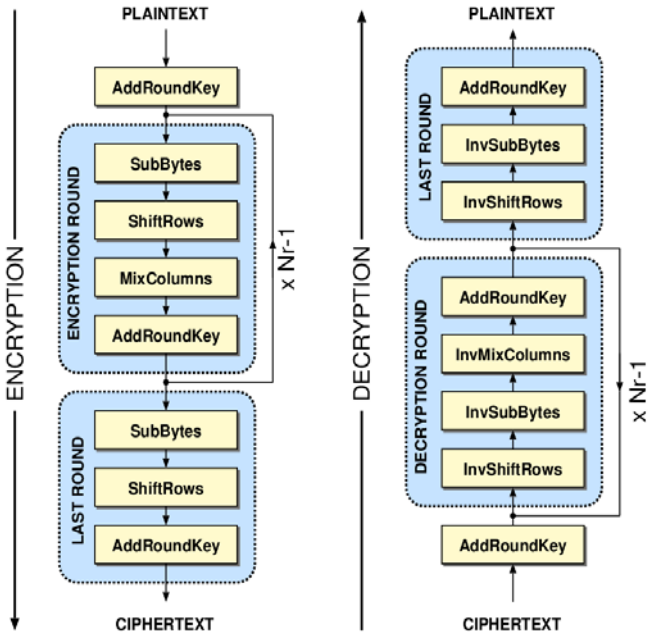


Figure 1: The Basic AES Flow Chart

3.1 SubByte

The SubByte is a non-linear operation where one byte is substituted for another based on the algorithm we have to use fixed 8-bit lookup table, S; $b_i = S(a_i)$.

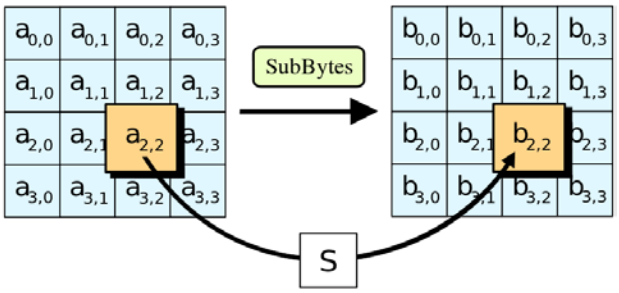


Figure 2: SubBytes Transformation

3.2 ShiftRows

In the ShiftRows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row by a certain offset. Row0 is left unchanged, Row1 is shifted 1 byte. Similarly, the Row2 and Row3 are shifted by offsets of two and three respectively.

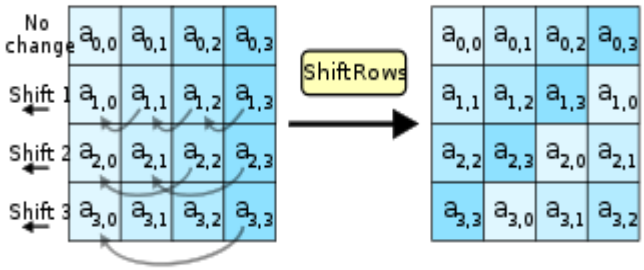


Figure 3: ShiftRows Operation

3.3 MixColumns

In the MixColumns step, the four bytes of each column of the state are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes.

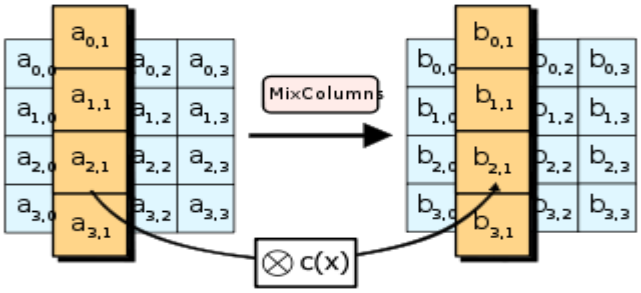


Figure 4: MixColumns Operation

3.4 AddRoundKey

In the AddRoundKey [9] step, the subkey is combined with the state. For each round, a subkey is derived from the main key using Rijndael's key schedule; each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.

$$b'(i,j) = a(i,j) \oplus k(i,j)$$

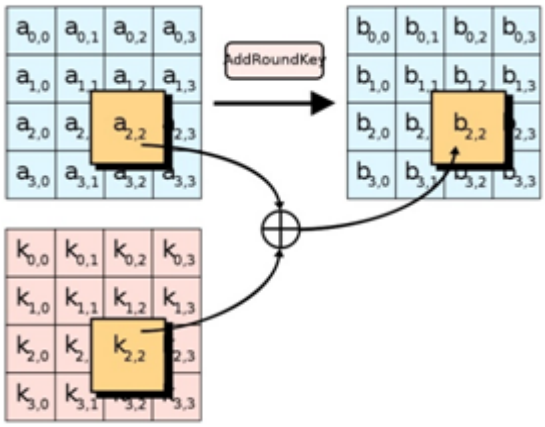


Figure 5: AddRoundKey Operation

The AES process can be defined in three types based on length of the key used for the generating cipher text which are AES-128, AES-192 and AES-256.

In this operation, the AES cipher maintains an internally (4X4) matrix of bytes called states. The state consists of four rows of bytes, each row containing N_b bytes, where N is the number of byte and b is the block length divided by 32 (4 for 128-bit key, 6 for 192-bit key, 8 for 256-bit key). At the same time key length and number of rounds differ from key to key, i.e. we have to use 10 rounds for 128-bit key, 12 rounds for 192-bit key and 14 rounds for 256-bit key.

Table 1: Comparison of AES Algorithm

	Key length (N_k)	*Block size (N_b)*	*Number of rounds (N_r)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

4. S-Box Transformation Using Look Up Table (LUT)

The S-Box by using look up table, all the values is predefined based on the ROM so the area and memory access & latency is high. S-Box stands for Substitution Box; SubBytes transformation is a nonlinear byte substitution that operates independently on each byte of the State using a substitution table (S-box) [4].

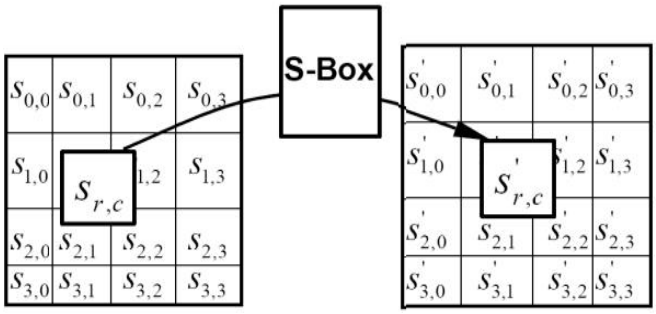


Figure 6: Application of S-box to the Each Byte of the State

Table 2: S-Box Values for all 256 Combinations in Hexadecimal Format

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

For example, if $=S_{1,1}=$ {f0}, then the substitution value would be determined by the intersection of the row with index 'f' and the column with index '0' in figure. This would result in $S'_{1,1}$ having a value of {8c}.

Inverse Byte Substitution Transformation is the inverse of the byte substitution transformation, in which the inverse S-Box is applied to each byte of the State. This is obtained by first applying the inverse of the affine transformation to the equation and then taking the multiplicative inverse in GF(2^8).

Table 2: Inverse S-Box Values for all 256 Combinations in Hexadecimal Format

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
dx	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

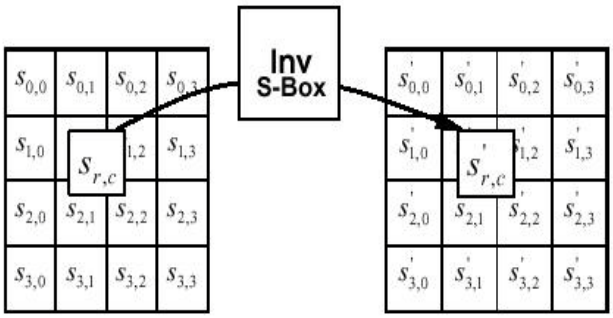


Figure7: Application of the Inverse S-box to Each Byte of the State

Most common method of implementation of the S-Box for the SubByte operation is that the pre-computed values are stored in a ROM as lookup table. All 256 values are stored in a ROM, and the input byte would be wired to the ROM's address bus. However, this method has the disadvantage that the unbreakable delay is very large since ROMs have a fixed access time for its read and write operation. Such implementation is expensive in terms of hardware and consumes large area. So a better way of implementing the S-Box is to use composite field arithmetic. This S Box has the Advantage that it occupies small area and pipelining can also be applied to improve the performance.

5. Proposed S-Box Transformation Using Galios Field (GF)

To optimize the area, our method is based on the Composite Field Arithmetic which involves Galios Field GF (2^8) it contains two main operations as follows:

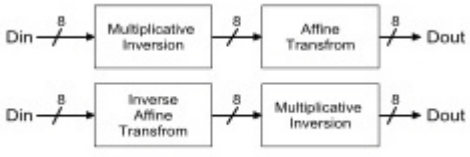


Figure 8: SubByte & Inv SubByte Transformations in S-BOX

This section says that the multiplicative inverse computation will first be covered and the affine transformation will then follow to complete the methodology involved for constructing the S-BOX for the SubByte operation. For the Inverse SubByte operation, that can reuse multiplicative inversion module and combine it with the inverse affine transformation. So the multiplicative inverse can be constructed in GF (2^8).

5.1 Galois Field (GF)

Galois Field, named after Evariste Galois, also known as finite field, which has a finite number of members. Galois fields having 2^m members are used in error-control coding and are denoted GF(2^m). Fields can have 2^m members, where m is an integer between 1 and 16.It is particularly useful in translating computer data as they are represented in binary forms. That is, computer data consist of combination of two numbers, 0 and 1, which are the components in Galois field whose number of elements is two. Representing data as a vector in a Galois Field allows mathematical operations to scramble data easily and effectively.

Computation of the multiplicative inverse in composite fields cannot be directly applied to an element which is based on GF(2^8) .So for that we have to decomposing the more complex GF(2^8) to lower order fields of GF(2^1), GF(2^2), GF($(2^2)^2$). To accomplish this, the following irreducible polynomials are used:

GF (2^2) →GF (2): $x^2 + x + 1$
GF ($(2^2)^2$) →GF (2^2): $x^2 + x + \phi$
GF ($((2^2)^2)^2$) →GF ($(2^2)^2$): $x^2 + x + \lambda$
Where ϕ = {10}2 and λ = {1100}2.

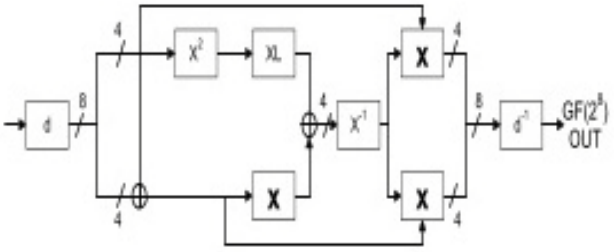
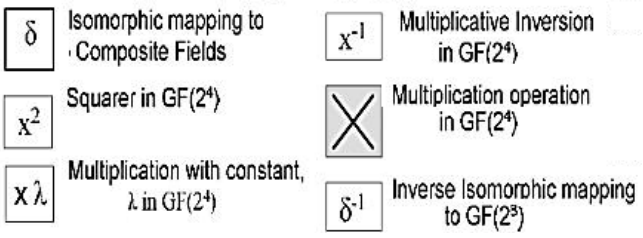


Figure 9: Multiplicative Inverse Module

The notations for the modules within the multiplicative inversion module are below [4].



5.2 Affine Transform

The affine transform is normally should improve our result. It's the second building for the composite field arithmetic

based S-Box. Our proposed affine transform & Inverse affine transform as follows Equation 1 & 2:

$$\delta = b_i \oplus b_{((i+4) mod\ 8)} \oplus b_{((i+5) mod\ 8)} \oplus b_{((i+6) mod\ 8)} \oplus b_{((i+7) mod\ 8)} \oplus d_i \quad (1)$$

Where d = {01100011} & i = 0 to 7

$$\delta^{-1} = b_{((i+2) mod 8)} \oplus b_{((i+5) mod 8)} \oplus b_{((i+7) mod 8)} \oplus di \quad (2)$$

Where d = {00000101} & i = 0 to 7.

5.3 Isomorphic and Inverse Isomorphic mapping

The element in GF (2^8) has to be mapped to its composite field representation via an isomorphic function, δ. After performing the multiplicative inversion, the result will also have to be mapped to its equivalent in GF (2^8) via the inverse isomorphic function δ -1.

Let q be the element in GF (2^8), in that δ & δ^{-1} can be represented as 8x8 matrixes, where q7 is the most significant bit, q0 is the least significant bit. The equation is given as below,

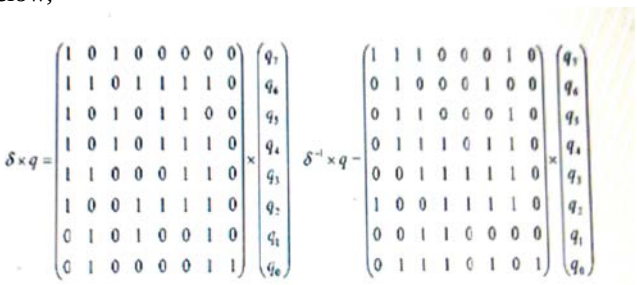


Figure 10: Implementation of Isomorphic and Inverse Isomorphic mapping

5.4 Addition in GF(2^4)

Addition of two elements in Galois Field can be translated to simple bitwise XOR operation between the two elements.

5.5 Squaring in GF(2^4)

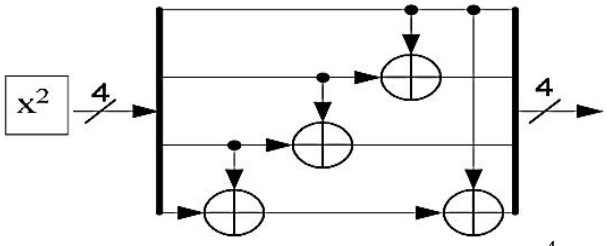


Figure11: Implementation of Squarer in GF (2^4)

5.6 Multiplication with constant λ

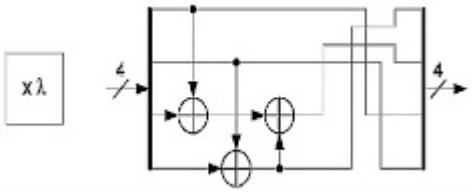


Figure 12: Implementation of multiplication with constant

In that we take k=qλ

Where k= {k3 k2 k1 k0}2,

q= {q3 q2 q1 q0}2 and

λ= {1100}2 are element in GF(2^4)

We can apply the same procedure as seen in Addition we get,

k3 = q3

k2= q3⊕q2

k1=q2⊕q1

k0=q3⊕q1⊕q0

5.7 Multiplicative Inversion in GF(2^4)

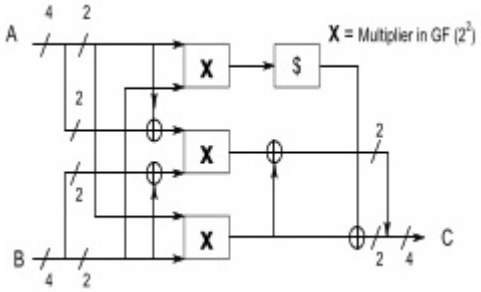


Figure 13: Implementation of Multiplicative Inversion

The multiplicative inverse of q (where q is an element of GF (2^4)) such that q^{-1} = {q_3^{-1}, $q2^{-1}$, $q1^{-1}$, $q0^{-1}$} [10].The inverse of the individual bits can be computed as below,

$$q_3^{-1} = q_3 \oplus q_3q_2q_1 \oplus q_3q_0 \oplus q_2$$

$$q_2^{-1} = q_3q_2q_1 \oplus q_3q_2q_0 \oplus q_3q_0 \oplus q_2 \oplus q_2q_1$$

$$q_1^{-1} = q_3 \oplus q_3q_2q_1 \oplus q_3q_1q_0 \oplus q_2 \oplus q_2q_0 \oplus q_1$$

$$q_0^{-1} = q_3q_2q_1 \oplus q_3q_2q_0 \oplus q_3q_1 \oplus q_3q_1q_0 \oplus q_3q_0 \oplus q_2 \oplus q_2q_1 \oplus q_2q_1q_0 \oplus q_1 \oplus q_0$$

From the above discussion is the operation for the composite field arithmetic based S-Box .Our proposed method is the implementation of this S-Box in the four stage pipeline. So that the area, delay, power will be reduced. The diagram will show below for proposed pipelined implemented S-Box.

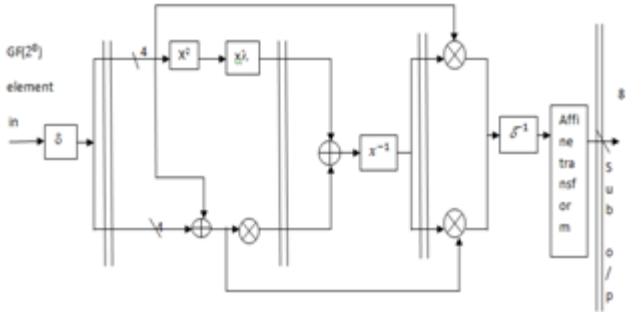


Figure 14: Proposed Pipelined implemented S-Box

6. Comparison Result

We design the S-Box is based on composite field arithmetic method. In this paper proposed method coding can be written using verilog language. The XC5S1600E device of Xilinx FPGA is used to validate the power with Verilog code for the proposed architecture also the power is analyzed using Xilinx ISE 12.1 Xpower analyzer. Table IV shows the comparison of power, delay and slices for conventional & proposed

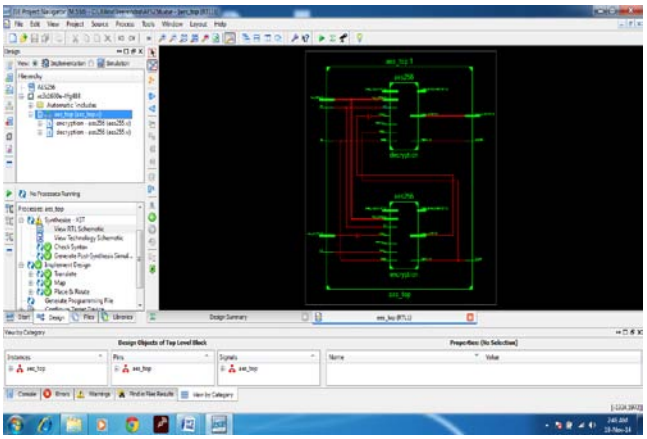
method. Waveform3 shows power report for the proposed method.

Table 4: Comparison of S-Box Using LUT and Without LUT

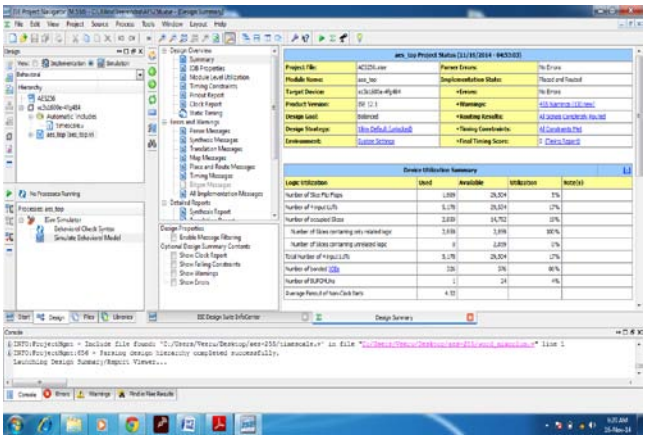
Parameter	*Without LUT*	*Using LUT*
No of slices	2859/14752	5892/14752
No of slice flip flops	1609/29504	3308/29504
Power(W)	0.203w	1.007w
Delay(ns)	4.496ns	6.042ns

7. Synthesis Report From Xilinx Tool

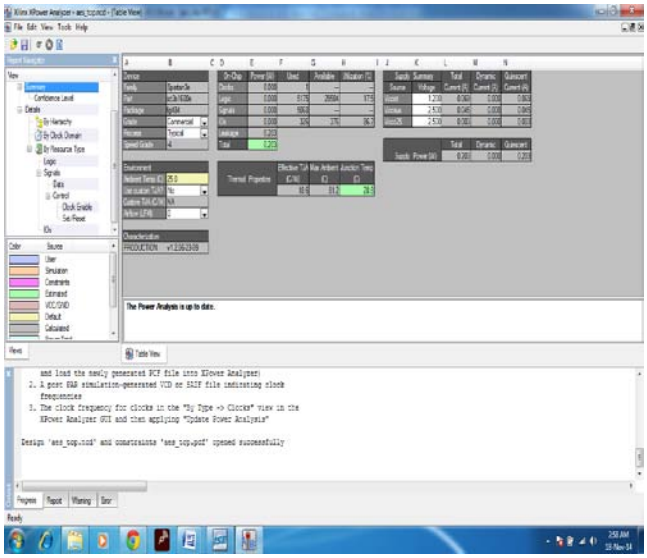
7.1 RTL Schematic of AES-256



7.2 CPLD Report of AES-256

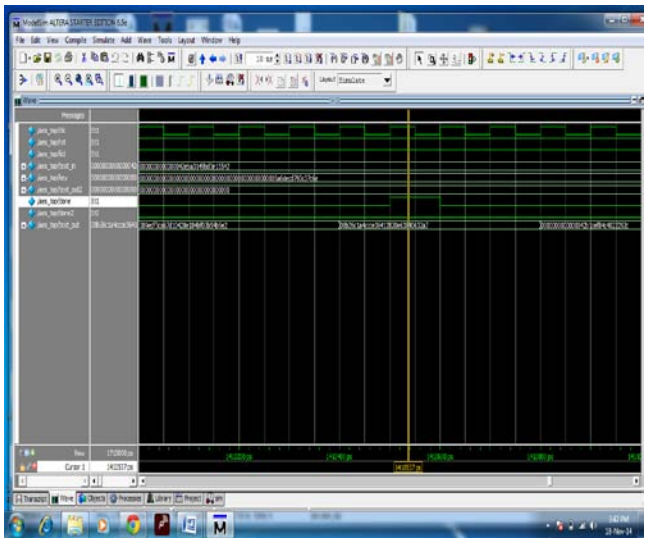


7.3 XPower Analyser Report of AES-256



8. Simulation Result From ModelSIM

8.1 Encryption of AES-256



8.2 Decryption of AES-256

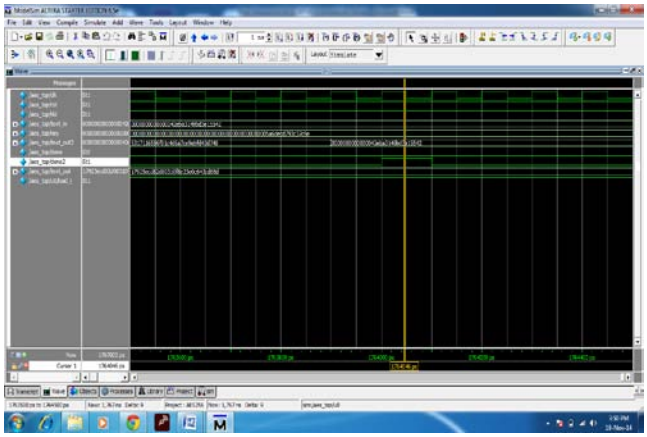


Figure 15: Simulation of S-BOX using Galios Field for encryption & decryption of AES-256.

The above two figures shows the simulation results of S-BOX and inverse S-BOX for encryption and decryption

using composite field arithmetic. There are three input clock, 8 bit input value and '1' or '0' which determines encryption or decryption.'0' stands for encryption and '1' for decryption. FPGA implementation is done for both LUT and non LUT SubByte/inverse SubByte and the synthesis report for both are analyzed and compared.

9. Conclusion

In traditional look up table (LUT) approaches, the unbreakable delay is longer than the total delay of the rest of operations in each round. LUT approach is not suitable for resource constrained use for it costs a large area. Composite field arithmetic has been introduced to solve the problem. The multiplicative inverse in GF (2^8) is very complicated by direct computation. Merging also reduces the area and increases the throughput.

Presented implementation is capable of higher speeds as compared to the typical ROM based lookup table. It can be pipelined and small in terms of area occupancy (2859/14752 slices on a Spartan III XC3S1600E-4FPGA). The design is coded using Verilog language. The design is simulated on ModelSim simulation tool and synthesized on Xilinx ISE 12.1 software.

10. Future Scope

After the implementation of this system we came to notice that this system can be applicable to any type of system where we have regular observation in needed and it makes maintenance so simple and cost effective. In future any new compact and high speed architecture allows the S-Box to be used in both areas limited and demanding throughput AES chips for various applications, ranging from small smart cards to high speed servers can simply increase data security without making a lot of effort.

11. Acknowledgement

I sincerely thank my guide P. Sankara Rao, Associate Professor in ECE Department of Sri Vasavi Institute of Engineering and Technology for his kind advice, support, encouragement as well as guidance for the preparation of research manuscript. I'm really grateful to acknowledge the support provided by my Mom & Dad.

References

[1] Hoang Trang & Nguyen Van Loi "An efficient FPGA Implementation of Advanced Encryption standard Algorithm". IEEE-2012.
[2] Chong Hee Kim "Improved Differential Fault Analysis on AES Key Schedule". IEEE Transactions on Information Forensics and Security, Vol. 7, No. 1, February 2012.
[3] N.Shanthini, P.Rajasekar, Dr. H.Mangalam "Design of low power S-Box in Architecture Level using GF" International Journal of Engineering Research and General Science Volume 2, Issue 3, April-May 2014.
[4] "Announcing the Advanced Encryption Standard (AES)". Federal Information Processing Standards Publication [FIPS]-197. United States National Institute of Standards and Technology (NIST). November 26, 2001. Retrieved October 2, 2012.
[5] Daemen, Joan; Rijmen, Vincent (9/04/2003). "AES Proposal: Rijndael". National Institute of Standards and Technology. p. 1. Retrieved 21 February 2013.
[6] John Schwartz "U.S. Selects a New Encryption Technique". New York Times (October 3, 2000).
[7] Westlund, Harold B. (2002). "NIST reports measurable success of Advanced Encryption Standard". Journal of Research of the National Institute of Standards and Technology.
[8] Daemen J., and Rijmen V, "The Design of Rijndael: AES-the Advanced Encryption Standard", Springer-Verlag, 2002.
[9] "Efficient software implementation of AES on 32-bit platforms". Lecture Notes in Computer Science: 2523. 2003.
[10] "Practical Implementation of Rijndael S-Box Using Combinational Logic" Edwin NC Mui Custom R & D Engineer, Texco Enterprise Ptd. Ltd.
[11] "A High-Throughput Cost-Effective ASIC Implementation of the AES Algorithm"-978-1-4244-3870-9/09/$25.00 ©2009 IEEE.
[12] The Advanced Encryption Standard Algorithm (AES), http://en.wikipedia.org/wiki/Advanced_Encryption_Standard.
[13] Finite Field (or) Galios Field - Wikipedia,Free Encyclopedia, http://en.wikipedia.org/wiki/Finite_field.