IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676,p-ISSN: 2320-3331, Volume 5, Issue 5 (May. - Jun. 2013), PP 47-56
www.iosrjournals.org

"A High Throughput CFA AES S-Box with Error Correction Capability"

M.Sandhya[1], S.Deepa[2]
[1]PG SCHOLAR, Dept of Electronics and Communication Engineering, Velalar College of Engg and Technology., [2]Asst.Professor., Dept. of Electronics and Communication Engineering., Velalar College of Engg and Technology.

Abstract: *The Advanced Encryption Standard (AES) has been lately accepted as the symmetric cryptography standard for confidential data transmission. The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plain-text into the final output of cipher-text. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform cipher-text back into the original plain-text using the same encryption key. The proposed schemes are independent of the way the S-box and the inverse S-box are constructed. Therefore, they can be used for both the S-boxes and the inverse S-boxes using lookup tables and those utilizing logic gates based on composite fields. Furthermore, for each composite field constructions, there exists eight possible isomorphic mappings. Therefore, after the exploitation of a new common sub expression elimination algorithm, the isomorphic mapping that results in the minimal implementation area cost is chosen. High throughput hardware implementations of the proposed CFA AES S-boxes are reported. In order to avoid data corruption due to SEU's a novel fault tolerant model of AES is presented which is based on the Hamming error correction code. This reduces the data corruption and increases the performance.Thus the data corruption due to Single Event Upset can be avoided and the performance is increased.*

I. Introduction

The Advanced Encryption Standard, in the following referenced as AES, is the winner of the contest, held in 1997 by the US Government, after the Data Encryption Standard(DES) was found too weak because of its small key size and the technological advancements in processor power. The Rijndael, whose name is based on the names of its two Belgian inventors, Joan Daemenand Vincent Rijmen, is a Block cipher, which means that it works on fixed-length group of bits, which are called blocks.

The AES standard specifies the Rijndaelalgorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits.Rijndael was designed to handle additional block sizes and key lengths, however they are not adopted in this standard.

Throughout the remainder of this standard, the algorithm specified herein will be referred to as "the AES algorithm." The algorithm may be used with the three different key lengths indicated above, and therefore these different "flavors" may be referred to as "AES-128", "AES-192", and"AES-256".

II. RELATED WORK

Different construction schemes for composite fields are proposed for the AES algorithm. $GF(2^8)$ is decomposed into $GF((2^4)^2)$ and composite field arithmetic is applied to all the transformations in the AES algorithm. The optimum construction scheme for $GF((2^4)^2)$ is selected based on minimizing the total gate count in the implementation of all transformations. However, it is more efficient to apply composite field arithmetic only in the computation of the multiplicative inversion in the SubBytes and InvSubBytestransformations.CFA is a construction scheme with smaller gate counts and shorter critical paths. Different irreducible polynomials are used to construct the composite fields of the same order which presents 16 ways to construct $GF(((2^2)^2)^2)$. Using composite field arithmetic, the complicated multiplicative inversion $GF(2^8)$ is mapped to operations in subfields. This provides the analytical results of how the coefficients in the irreducible polynomials affect the complexities of the subfield operations. In addition, for each construction scheme, there exist eight isomorphic mappings with various complexities to map the elements between $GF(2^8)$ and $GF(((2^2)^2)^2)$. An efficient algorithm is proposed to find all the isomorphic mappings. Moreover, the lowest mapping complexity is provided for each proposed composite field construction scheme. Based on the complexities of both the subfield operations and the isomorphic mappings, the optimum constructions of the composite field $GF(((2^2)^2)^2)$ for the AES algorithm are selected.

Composite Field Implementations of the Sub-bytes in AES

In the AES algorithm, the message is divided into blocks of 128 bits. Each block is divided into 16 bytes, and each byte is considered as an element of $GF(2^8)$. Although different irreducible polynomials can be used to construct $GF(2^8)$, the one specified for the AES algorithm is $P(x)=x^8+x^4+x^3+x+1$.

The AES algorithm is carried out in a number of rounds. Each round in the encryption consists of four transformations, namely:

- SubBytes
- ShiftRows
- MixColumns4
- AddRoundKey

The decryption consists of the inverse transformations. Among the four transformations involved in the encryption, the Sub-Bytes is the most complicated. In this transformation, the multiplicative inverse of each byte in $GF(2^8)$ has to be computed, followed by an affine transformation. Denoting each byte by S, the SubBytes can be described by

$$\mathbf{S' = MS^{-1}+C}$$

whereM is an 8 X 8 binary matrix, and C is an 8-bit binary vector. Although two finite fields of the same order are isomorphic, the complexities of the field operations may heavily depend on the representations of the field elements. Composite field arithmetic can be employed to reduce the hardware complexity of the multiplicative inversion in $GF(2^8)$. Two pairs

$$\{GF(2^n), Q(y)=y^n+\sum_{i=0}^{n-1} q_i y^i, q_i \in GF(2)\}$$

$$\{GF(2^n)^m) P(x)=x^m+\sum_{i=0}^{m-1} p_i x^i, p_i \in GF(2^n)\}$$

are called a composite field if

• C is constructed from GF(2) by Q(y) ;
• $GF(2^n)^m)$is constructed from $GF(2^n)$ by P(x).

Composite fields will be denoted by $GF(2^n)^m)$, and a composite Field $GF(2^n)^m)$ is isomorphic to the field $GF(2^k)$ for k=nm. Additionally, composite fields can be built iteratively from lower order fields. For example, the composite field of $GF(2^8)$ can be built iteratively from using GF(2) the irreducible polynomials $GF(2^8)$

GF(2)	$= GF(2^2)$,	$P_0(x) = x^2 + x + 1$
$GF(2^2)$	$= GF((2^2)^2)$,	$P_1(x) = x^2 + x + \Phi$
$GF((2^2)^2)$	$= GF(((2^2)^2)^2)$,	$P_2(x) = x^2 + x + \lambda$

Where $\Phi \in GF(2^2)$, $\lambda \in GF(((2^2)^2)^2)$ and the values of Φ,λ, satisfy that $P_1(x)$ is irreducible over $GF(2^2)$ and $P_2(x)$ is irreducible over $GF(((2^2)^2)^2)$. Moreover, an isomorphic mapping function $f(x) = \delta X x$ and its inverse need to be applied to map the representation of an element in $GF(2^8)$ to its composite field and vice versa. The 8x8 binary matrix is decided by the field polynomials of $GF(2^8)$ and its composite field. In the composite field $GF((2^4)^2)$, an element can be expressed as $S_h x + S_l$, where S_h, S_l, is the $GF(2^4)$ and x is the root of $P_2(x)$. Using the extended Euclidean algorithm, the multiplicative inverse of as $S_h x + S_l$ modulo $P_2(x)$ can be computed as

$$(S_h x + S_l)^{-1} = S_h \Theta x + (S_h + S_l)^{\Theta} \qquad (3.1)$$

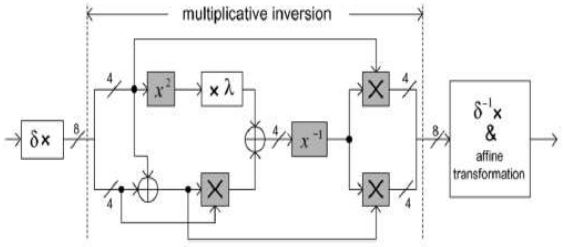

Figure. 3.1 Implementation of the SubBytes transformation.

Where $\Theta = (S_h^2\lambda + S_hS_l + S_l^2)^{-1}$. According to the equation (3.1), the multiplicative inversion in GF (2^8) can be carried out as GF ($(2^4)^2$) in the architecture illustrated in Figure.3.1. The inverse isomorphic mapping is combined with the affine transformation to reduce the hardware complexity.

PROPOSED METHOD

The proposed system uses a new fault tolerant technique based on AES. To address the reliability issues of AES algorithm and to overcome the SEU. Five modes are used in AES. They are Cipher block chaining mode (CBC), Electronic code Book mode (ECB), Cipher Feedback Mode (CFM), Counter mode (CTR) and Output Feedback mode (OFB). Cipher Block Chaining is not suitable for satellite images. data is corrupted due to fault propagations. In Electronic Code Book if a single bit is corrupted the entire block is corrupted. In cipher Feedback mode the fault is propagated to next blocks. No fault is propagated in counter mode. And also satellite image communications are not suitable for counter mode. So to rectify the faults while transmission of data from satellites in noise an On-Board AES OFB based encryption is used. The faults are rectified by using Hamming Error Correction code Algorithm. The proposed approach reduces the SEU while transmission of data from satellites with noise.

The proposed fault-tolerant model is based on the single error correcting Hamming code (12,8), the simplest of the available error correcting codes. The Hamming code (12,8) detects and corrects a single bit fault in a byte and it is a good choice for satellite applications, as most frequently occurring faults in on-board electronics are bit flips induced by radiation. However, the AES correction model can be extended to correct multiple bit faults by using other error correcting codes such as the modified Hamming code.

Calculation of the Hamming Code

The parity check bits of each byte of the S-Box LUTs are precalculated. These Hamming code bits can be formally expressed as in equation (3.2)

h(SRD[a]) →hRD[a]
h((SRD[a] f{2g}) →h2RD[a]
h((SRD[a] f{03g}) → h3RD[a] (3.2)

where "a" is the state byte and "h" represents the calculation of the Hamming code.

As can be seen from equation (3.2), hRD is given by the parity check bits of the S-Box LUT SRD, h2RD is given by the parity check bits of (SRD − f02g), and h3RD is given by the parity check bits of (SRD − f03g). The procedure to derive the hRD parity bits is described below by taking one state byte a, represented by bits (b7,b6,b5,b4,b3,b2,b1,b0) as an example. The Hamming code of the state byte a is a four-bit parity code, represented by bits (p3,p2,p1,p0), which are derived as follows:

p3 →is parity of bit group b7,b6,b4,b3,b1
p2 →is parity of bit group b7,b5,b4,b2,b1
p1 →is parity of bit group b6,b5,b4,b0
p0 →is parity of bit group b3,b2,b1,b0

Detection and Correction of Fault Using Hamming Code Bits

The Hamming code matrix of the Sub Bytes transformation is predicted by referring to the hRD table. The Hamming code matrix prediction for Shift Rows involves a simple cyclic rotation of the Sub Bytes Hamming code bits. The Hamming code state matrix for Mix Columns is predicted with the help of the hRD, h2RD and h3RD parity bits and it is expressed by the equation

h0,j = h2RD[a0,j] h3RD[a1,j] hRD[a2,j] hRD[a3,j]
h1,j = hRD[a0,j] h2RD[a1,j] h3RD[a2,j] hRD[a3,j]
h2,j = hRD[a0,j] hRD[a1,j] h2RD[a2,j] h3RD[a3,j]
h3,j = h3RD[a0,j] hRD[a1,j] hRD[a2,j] h2RD[a3,j]
0 < j <4 (3.3)

Hamming code is predicted using the input data state to the transformation by referring to the parity check bit tables and also the parity check bits are calculated from the output of the transformation.

The predicted and calculated check bits are compared with detected and corrected faults. Let the predicted check bits of the transformation input be represented by (x3,x2,x1,x0) and the calculated check bits of the transformation output be represented by (y3,y2,y1,y0). Once the faulty bit position is identified, the fault correction is performed by simply flipping that bit. The encryption is then continued without any interruption to

the encryption process. Assumption is that the Hamming code tables will be protected from SEUs by traditional memory protection techniques in satellite applications like memory scrubbing and refreshing.

III. AES DESIGN

The overall structure of AES is shown in Figure.4.1. The input is a single 128 bit block both for encryption and decryption and is known as the inmatrix. This block is copied into a statearray which is modified at each stage of the algorithm and then copied to an output matrix as in Figure 4.1. Both the plaintext and key are depicted as a 128 bit square matrix of bytes. This key is then expanded into an array of key schedule words as the w matrix. It must be noted that the ordering of bytes within the inmatrix is by column. The same applies to the wmatrix.

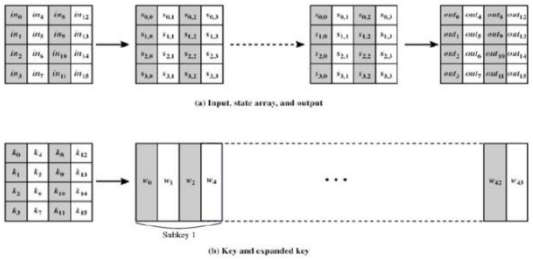

Figure 4.1 Data structures in the AES algorithm

Advanced encryption standard has four stages in a single round of operation.

1. Sub Bytes —a non-linear substitution step where each byte is replaced with another according to a lookup table.
2. Shift Rows —a transposition step where each row of the state is shifted cyclically a certain number of steps.
3. Mix Columns —a mixing operation which operates on the columns of the state, combining the four bytes in each column
4. Add Round Key —each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule.

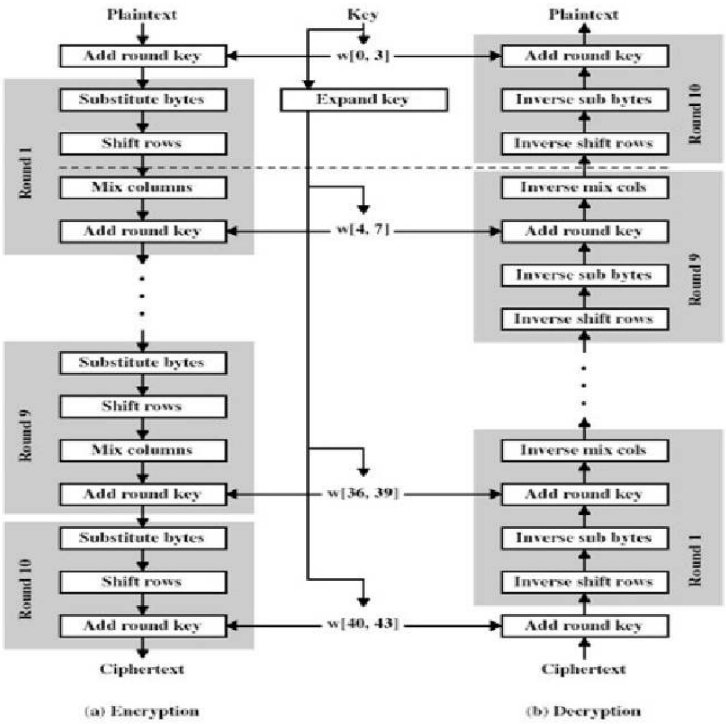

Figure 4.2: Overall structure of the AES algorithm.

Sub Bytes

The Sub-Bytes stage is simply a table lookup using a 16 x 16 matrix of byte values called an S-box. This matrix consists of all the possible combinations of an 8- bit sequence (28 = 16 x 16 = 256). However, the s-box is not just a random permutation of these values and there is a well defined method for creating the S-box tables. The designers of Rijndael showed how this was done unlike the S-boxes in DES for which no rationale was given. The construction of S-box are not concerned rather the table lookups are used simply.

Again the matrix that gets operated upon throughout the encryption is known as state.For each round, the making of matrix is concerned. For this particular round each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the S-box and the rightmost nibble specifies a column. For example, the byte {95}(curly brackets represent hex values in FIPS 197) selects row 9 column 5 which turns out to contain the value {2A}. This is then used to update the statematrix which is depicted in the Figure 4.3.

The Inverse Substitute byte transformation (known as InvSubBytes) makes use of an inverse S-box. In this case what is desired is to select the value {2A} and get the value {95}. Table 4.1 shows the two s-boxes and it can be verified that this is in fact the case. The S-box is designed to be resistant to known cryptanalytic attacks.

Table 4.1 AES s-boxes both forward and inverse.

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

Inverse S-box

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

Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits, and the property that the output cannot be described as a simple mathematical function of the input. In addition, the s-box has no fixed points (S-box(a) = a) and no opposite fixed points (S-box(a) =(-a)) where -a is the bitwise compliment of a. The S-box must be invertible if decryption is to be possible (Is-box[S-box(a)]= a) however it should not be its self inverse i.e. S-box(a) 6= Is-box(a)

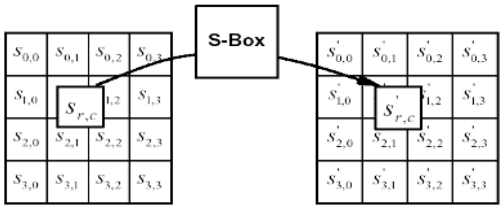

Figure 4.3 Substitute Bytes Stage.

Shift Row Transformation

The ShiftRow stage is shown in figure 4.4. This is a simple permutation and nothing more. It works as follow:

- The first row of stateis notaltered.
- The second row is shifted 1 bytes to the left in a circular manner.
- The third row is shifted 2 bytes to the left in a circular manner.
- The fourth row is shifted 3 bytes to the left in a circular manner.

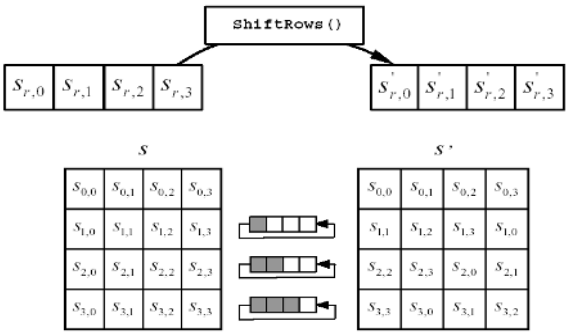

Figure 4.4 ShiftRows stage.

The Inverse Shift Rows transformation, known as InvShiftRows performs these circular shifts in the opposite direction for each of the last three rows and the first row is unaltered to begin.

This operation may not appear to do much but if you think about how the bytes are ordered within statethen it can be seen to have far more of an impact. Remember that stateis treated as an array of four byte columns, i.e. the first column actually represents bytes 1,2,3 and 4. A one byte shift is therefore a linear distance of four bytes. The transformation also ensures that the four bytes of one column are spread out to four different columns.

Mix Column Transformation

The MixColumn stage is basically a substitution but it makes use of arithmetic of GF(2^8). Each column is operated on individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column. The transformation can be determined by the following matrix multiplication on stateas shown in figure 4.5.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \tag{4.1}$$

Each element of the product matrix is the sum of products of elements of one row and one column. In this case the individual additions and multiplications are performed in GF(2^8). The MixColumns transformation of a single column j (0 _ j _ 3) of statecan be expressed as in equation (4.2)

$$\begin{aligned} S'_{0,j} &= (2 \bullet S_{0,j}) \oplus (3 \bullet S_{1,j}) \oplus S_{2,j} \oplus S_{3,j} \\ S'_{1,j} &= S_{0,j} \oplus (2 \bullet S_{2,j}) \oplus (3 \bullet S_{2,j}) \oplus (3 \bullet S_{3,j}) \\ S'_{2,j} &= S_{0,j} \oplus S_{1,j} \oplus (2 \bullet S_{2,j}) \oplus (3 \bullet S_{3,j}) \\ S'_{3,j} &= (3 \bullet S_{0,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus (2 \bullet S_{3,j}) \end{aligned} \tag{4.2}$$

where• denotes multiplication over the finite field GF(2^8).

As an example, from the first column of a matrix to be s0,0 = {87}, s1,0 ={6E}, s2,0 = {46}, s3,0 = {A6}. This would mean that s0,0 = {87} gets mapped to the value s00,0 = {47} which can be seen by working out the first line of equation (4.2) with j = 0. Therefore

$$(02 \bullet 87) \oplus (03 \bullet 6E) \oplus 46 \oplus A6 = 47 \tag{4.3}$$

So to show the equation (4.3) each Hex number is represented by a polynomial

$\{02\} = x$
$\{87\} = x^7 + x^2 + x + 1$

Multiplying these two together

$$x \bullet (x^7 + x^2 + x + 1) = x^8 + x^2 + x$$

The degree of this result is greater than 7 so it has to be reduced to modulo an irreducible polynomial m(x). The designers of AES chose m(x) = x8 + x4 + x3 + x + 1. So it can be seen that

$$(x8 + x3 + x2 + x) \bmod (x8 + x4 + x3 + x + 1) = x4 + x2 + 1$$

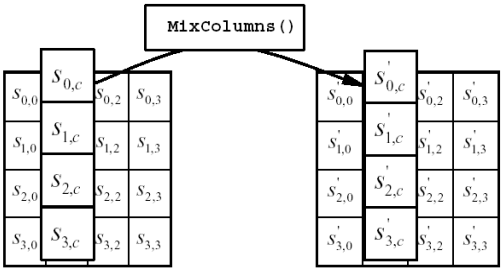

Figure 4.5MixColumns stage.

This is equal to 0001 0101 in binary. This method can be used to work out the other terms. The result is therefore:

$$\begin{array}{rl} & 0001\ 0101 \\ & 1011\ 0010 \\ & 0100\ 0110 \\ \oplus & 1010\ 0110 \\ \hline & 0100\ 0111 = \{47\} \end{array}$$

There is infact an easier way to do multiplication modulo m(x). If {02} has to be multiplied, then a 1-bit left shift followed by a conditional bitwise XOR with (00011011) has to be done, if the leftmost bit of the original value (prior to the shift) is 1.Multiplication by other numbers can be seen to be repeated application of this method. What is important to note however is that a multiplication operation has been reduced to a shift and an XOR operation. This is one of the reasons why AES is a very efficient algorithm to implement.

The InvMixColumns is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \tag{4.4}$$

This first matrix of equation (4.1) can be shown to be the inverse of the first matrix in the equation (4.4). If these**A** and $\mathbf{A^{-1}}$ respectively are labeled and the mix columns operation are labeled as **S** and after as **S**0,

$$\mathbf{AS = S'}$$

Therefore

$$\mathbf{A^{-1}S' = A^{-1}AS = S}$$

Add Round Key Transformation

In the AddRoundKey stage, the 128 bits of stateare bitwise XORed with the 128 bits of the round key. The operation is viewed as a columnwise operation between the 4 bytes of a statecolumn and one word of the round key. This transformation is as simple as possible which helps in efficiency but it also effects every bit of state.

AES Key Expansion

The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words as shown in figure 4.6. Each word contains 32 bytes which means each subkey is 128 bits long. The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word **w**[i] depends on the immediately preceding word, **w**[i - 1], and the word four positions back **w**[i - 4]. In three out of four cases, a simple XOR is used. For a word whose position in the **w** array is a multiple of 4, a more complex function is used. Figure 4.7 illustrates the generation of the first eight words of the expanded key using the symbol g to represent that complex function. The function g consists of the following subfunctions:

1. **RotWord**performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
2. **SubWord**performs a byte substitution on each byte of its input word, using the s-box described earlier.
3. The result of steps 1 and 2 is XORed with round constant, Rcon[j].

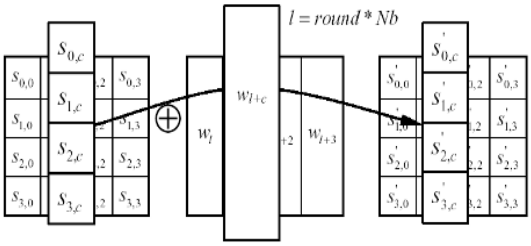

Figure 4.6 AES key expansion.

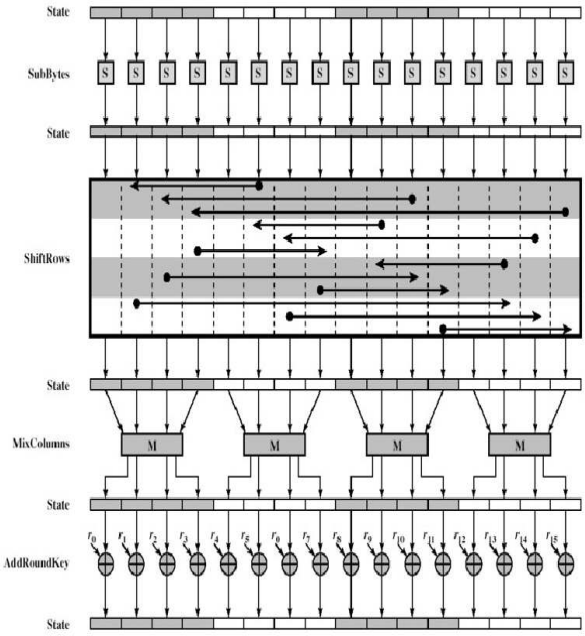

Figure 4.7 AES encryption round.

The round constant is a word in which the three rightmost bytes are always 0. Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as Rcon[j] =(RC[J], 0,0,0), with RC[1]= 1, RC[j]= 2• RC[j -1] and with

multiplication defined over the field GF(28).The key expansion was designed to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity,between the way in which round keys are generated in different rounds.

Figure 4.7 gives a summary of each of the rounds. The ShiftRows column is depicted here as a linear shift which gives a better idea how this section helps in the encryption.

5.1 Software description

AES can be implemented in software. VHDL is used as the hardware descrption language because of the flexibility to exchange among environments. The software used for simulation is "Modelsim 6.3f". This is used for writing, debugging and optimizing efforts and also for fitting, simulating and checking the performance results. All the individual transformations of both encryption and decryption are simulated using Modelsim 6.3f. The encryption and decryption time can be calculated for different data size and different key length. Usually, software implementations are very inexpensive.

IV. Result

AES –Encryption/Decryption

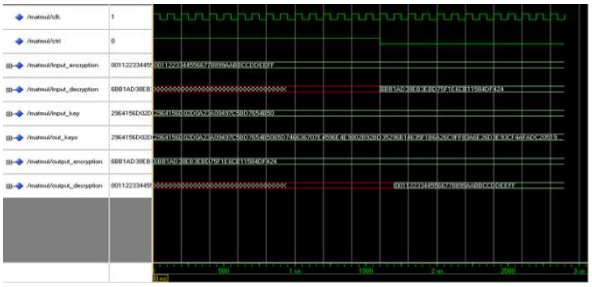

Figure 5.1 Waveform for AES Encryption/Decryption

The Figure 5.1 shown above gives the simulated output of the AES Encryption. For given 128- bits the encrypted and decrypted waveforms are obtained as shown in the Figure 5.1 above.

AES Encryption Error Detection and Correction

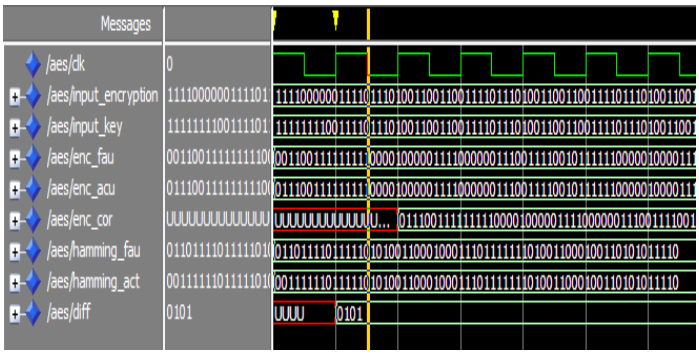

Figure 5.2 Waveform using Hamming Code

The Figure 5.2 shown above gives the simulated output of the AES Encryption with error detection and correction using Hamming Code. Thus from the given bits, single bit error is detected and corrected.

V. Conclusion

Thus the S-box has been replaced with a high throughput CFA which reduces the hardware size. Compared to the existing method, the throughput of this paper has been increased from 3.49Gbps to 4.4 Gbps. This work is the derivation of a new composite field AES S-box that achieves an optimally balanced construction in terms of area of implementation and critical path. Furthermore, all of the possible isomorphic mapping for each of the composite field construction are explored. This paper also has the capability of detecting and correcting the error using hamming code. This finds its application in satellite where Single Event Upset can be neglected.

Future Work

Future work can be done with the aim of detecting and correcting multiple errors.

References

[1] Wong M.M. Wong M.L.D. Nandi A.K. and Hijazin I. (2012) 'Construction of Optimum Composite Field Architecture for Compact High-Throughput AES S-Boxes' *IEEE Trans*. Very Large Scale Integer.(VLSI) systems,vol.20., No.6.

[2] Canright D. (2005) 'A very compact Rijndael S-box' Naval Postgraduate School, Monterey, CA, Tech. Rep. NPS-MA-04-001.

[3] Rijmen V. (2000) 'Efficient implementation of the Rijndael S-box' online available http://ftp.comms.scitech.susx.ac.uk/fft/crypto/rijndael-sbox.pdf.

[4] Rudra A. Dubey P. K. Jutla C. S. Kumar V. Rao J.R. and Rohatgi P. (2001) 'Efficient Rijndael encryption implementation with composite field arithmetic' in *Proc. CHES*, pp. 171–184.

[5] Satoh A. Morioka S. Takano K. and Munetoh S. (2000) 'A compact Rijndael hardware architecture with S-box optimization' in *Proc. ASIACRYPT*, pp. 239–245.

[6] Sivakumar C. and Velmurugan A. (2007) 'High Speed VLSI Design CCMP AES Cipher for WLAN (802.11i)' Proceedings of International Conference on Signal Processing, Communication and Networking(ICSCN'07), pp.398-403.

[7] Wolkerstorfer J. Oswald E. and Lamberger M. (2002) 'An ASIC implementation of the AES S-boxes' in *Proc. RSA Conf.*, pp. 67–78.

[8] Wong M.M. and Wong M. L. D. (2010) 'A new common subexpression elimination algorithm with application in composite field AES S-box' in *Proc. 10th Int. Conf. Inf. Sci. Signal Process. Their Appl. (ISSPA)*, pp. 452–455.

[9] Zhang X. and Parhi K. K. (2006) 'On the optimum constructions of composite field for the AES algorithm' *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 10, pp. 1153–1157,.

[10] Zhang X. and Parhi K. K. (2004) 'High-speed VLSI architectures for the AES Algorithm' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 9, pp. 957–967,.