An Area Efficient High Speed Fault Detection Scheme For Advanced Encryption Standard Using Composite Fields

D.V.Nageswara Rao[#1]
[#] M.Tech,VLSI System Design,
Pragati Engineering College.

P.Sunitha[*2]
[*] Assoc. Professor,M.Tech,
Pragati Engineering College.

Abstract:

In this paper, we present a lightweight concurrent fault detection scheme for the AES. The selection of appropriate fault detection scheme for the AES makes it robust to internal defects and fault attacks. In the proposed approach, the composite field S-box and inverse S-box are divided into blocks and the predicted parities of these blocks are obtained. By using exhaustive searches We obtained the optimum solutions for the least overhead parity-based fault detection structures. It suggests both the ASIC and FPGA implementations of the fault detection structures using the obtained optimum composite fields, have better hardware and time complexities compared to their counterparts.

Keywords: ***AES** (Advanced Encryption Standard), **ASIC** (application-specific integrated circuit), **FPGA** (field-programmable gate-array).*

I INTRODUCTION

In today's digital world, encryption is emerging as a disintegrable part of all communication networks and information processing systems, for protecting both stored and in transit data.

A. Drawbacks of Software

There are other important drawbacks in software implementation of any encryption algorithm, including lack of CPU instructions operating on very large operands, word size mismatch on different operating systems and less parallelism in software. In addition, software implementation does not fulfill the required speed for time critical encryption applications. Thus, hardware implementation of encryption algorithms is an important alternative, since it provides ultimate secrecy of the encryption key, faster speed and more efficiency through higher levels of parallelism.

II. ENCRYPTION METHODS

A. Key Based Approach

Different versions of AES algorithm exist today (AES128, AES196, and AES256) depending on the size of the encryption key. In this project, a hardware model for implementing the AES128 algorithm was developed using the Verilog hardware description language. A unique feature of the design proposed in this project is that the round keys, which are consumed during different iterations of encryption, are generated in parallel with the encryption process.

B. Language

The hardware model was then completely verified using a test bench, which took advantage of the Verilog, is programming feature, by constructing random test objects and providing them to the model. Then, the verified model was synthesized using the Synopsis Design-Compiler tool to get an estimate of the number of gates, area and timing of the hardware model. Finally, the performances of software and hardware implementations were compared.

C. Finite Fields

In this section, the preliminaries on finite fields (also known as Galois fields) used in the subsequent sections are presented. The detailed description of these fields can be found in a number of publications, see for example [8] and [9]. According to Lin and Costello in [19], the definition of a finite field is as follows. Let F be a set of elements on which two binary operations of addition and multiplication, shown by " + " and " · ", respectively, are defined. Then, the set F and these operations construct a finite field if the following conditions are satisfied:

1. The set F be commutative under addition. The identity element in addition is zero.
2. The non-zero elements of set F be commutative under multiplication. The identity element in multiplication is one.
3. Multiplication be distributive over addition, i.e., for a, b and c in set F we have $a \cdot (b + c) = a \cdot b + a \cdot c$.
4. The number of elements in the field be finite. In the AES, the irreducible polynomial of $P(x) = x^8 + x^4 + x^3 + x + 1$ is used to construct GF(2^8). Each element in GF(2^8) is represented by a polynomial of degree 7, having 8 coefficients in GF(2). Furthermore, all the field operations are carried out using the above mentioned irreducible polynomial.

D. Cryptosystems and Public key cryptography

Cryptography is the process of encrypting the plain text into an incomprehensible cipher text by the

process of Encryption and the conversion back to plain text by process of Decryption.
Most encryption algorithms are based on 2 general principles,

1. Substitution, in which each element in plain text is mapped to some other element to form the cipher text
2. Transposition, in which elements in plaintext are rearranged to form cipher text.
3.

E. Number of keys used

If both the sender and the receiver use a same key then such a system is referred to as Symmetric, single-key, secret-key or conventional encryption. If the sender and receiver use different keys, then such a system is called Asymmetric, Two-key, or public-key encryption.Processing of Plain text: A Block cipher processes the input one block at a time, producing an output block for each input block. A Stream cipher processes the input elements continuously producing output elements on the fly. Most of the cryptographic algorithms are either symmetric or asymmetric key algorithms.

1. Secret Key Cryptography

This type of cryptosystem uses the same key for both encryption and decryption. Some of the advantages of such a system are

- Very fast relative to public key cryptography
- Considered secure, as long as the key is strong

Symmetric key cryptosystems have some disadvantages too. Exchange and administration of the key becomes complicated. Non-repudiation is not possible. Some of the examples of Symmetric key cryptosystems include DES, 3-DES, RC4, RC5 etc.

2. Public Key Cryptography

This type of cryptosystems uses different keys for encryption and decryption. Each user has a public key, which is known to all others, and a private key, which remains a secret. The private key and public key are mathematically linked. Encryption is performed with the public key and the decryption is performed with the private key. Public key cryptosystems are considered to be very secure and supports Non-repudiation. No exchange of keys is required thus reducing key administration to a minimum. But it is much slower than Symmetric key algorithms and the cipher text tend to be much larger than plaintext. Some of the examples of public key cryptosystems include Diffie-Hellman, RSA and Elliptic Curve Cryptography.

III. AES ROUNDS AND TRANSFORMATIONS

Here we briefly explain the four transformations of each round of the encryption. Each transformation in every round of encryption/decryption acts on its 128-bit input which is considered as a four by four matrix, called state, whose entries are eight bits. The transformations in each round of encryption except for the last round are as follows:

i. SubBytes: The first transformation in each round is the bytes substitution,called SubBytes, which is implemented by 16 S-boxes. These S-boxes are nonlinear transformations which substitute the 128-bit input state with a 128-bit output state. In the S-box, each byte of the state (Ii in Figure 2.1) is substi-10 tuted by a new byte (Bi in Figure 2.1). S-box will be explained in detail in the next section.

ii.ShiftRows: ShiftRows is the second transformation in which the four bytes of the rows of the input state are cyclically shifted to the left and the first row is left unchanged as shown in the leftmost part of Figure 2.1. The number of left shifts for each row is equal to the number of that row. Let us denote rows as rowi where, i, $0 \leq i \leq 3$, is the row number. Then, for row0 no shift, for row1 one shift, for row2 two shifts and for row3 three shifts are required.

iii. MixColumns*:* The third transformation is Mixcolumns in which each entry in the output state is constructed by the multiplication of a column in the input state with a fixed polynomial over GF(2^8). The output state is obtained by multiplying the columns of the input state modulo x^4 + 1 with the fixed polynomial of a(x) = (03)x^3+(01)x^2+(01)x+(02), where the coefficients are inhexadecimal form. The matrix representation of Mixcolumns is shown in Figure 2.1. As seen in this figure, the output state is constructed by multiplying the entries of the input state by a fixed matrix whose entries are in the hexadecimal form.

iv.AddRoundKey: The final transformation is AddRoundKey which XORs the input state with the key of that round, i.e., ki, $0 \leq i \leq 10$. The AES key expansion unit in Figure 2.1 takes the 128-bit original key, k0, as input and produces a linear array of expanded keys, k1 to k10. Each key is added to the input by 128 two-input XOR gates. Among the four transformations in the encryption and decryption of AES, only S-box for encryption and inverse S-box for decryption are nonlinear and complex operations. Furthermore, not only is the S-box one of the four round transformations, but it is also used in the key expander unit which generates the keys used in the AES rounds. Therefore, the implementations of these two transformations affects the implementation of the whole AES tremendously. Later in this chapter, the im-11 plementation variations of the S-box and inverse S-box including the composite field implementations are explained in detail.

IV. SECURITY OF AES

Three possible approaches to attacking the AES algorithm are as follows:

- **Brute Force:** This involves trying out all the possible private keys.

- **Mathematical attacks:** There are several approaches, all equivalent in effect to factoring the product of 2 primes.
- **Timing attacks***:* These depend on the running time of the decryption algorithm.

Choosing large p and q values can prevent such attacks. Security of RSA thus lies in choosing the value n, which makes such attacks extremely difficult.

V. PROPOSED SYSTEM

In our proposed approach we introduce fault detection not only in S-box and Inverse S-box and also in all other five levels in order to find the most optimum solutions and we also analyzed all required parameters to proved that the proposed system is not only effective in fault detection and also give proper efficiency in speed, power and area through hardware implementation.

AES Version	Key Length (N_k words)	Block Size (N_b words)	Number of Rounds (N_r rounds)
AES128	4	4	10
AES192	6	4	12
AES256	8	4	14

Table 1 – AES Variations

The basic processing unit for the AES algorithm is a byte. As a result, the plaintext, cipher text and the cipher key are arranged and processed as arrays of bytes:

Block length = 128 bits, 0 <= *n* < 16
Key length = 128 bits, 0 <= *n* < 16
Key length = 192 bits, 0 <= *n* < 24
Key length = 256 bits, 0 <= *n* < 24

All byte values in the AES algorithm are presented as the concatenation of their individual bit values between braces in the order {*b*7, *b*6, *b*5, *b*4, *b*3, *b*2, *b*1, *b*0}. These bytes are interpreted as finite field elements using a polynomial representation:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x + b_1x + b_0x = \sum_{i=0}^{7} b_i x^i$$

All the AES algorithm operations are performed on a two dimensional 4x4 array of bytes which is called the *State*, and any individual byte within the *State* is referred to as $s_{r,c}$, where letter '*r*' represent the row and letter '*c*' denotes the column. At the beginning of the encryption process, the *State* is populated with the plaintext. Then the cipher performs a set of substitutions and permutations on the *State*. After the cipher operations are conducted on the *State*, the final value of the state is copied to the cipher text output as is shown.

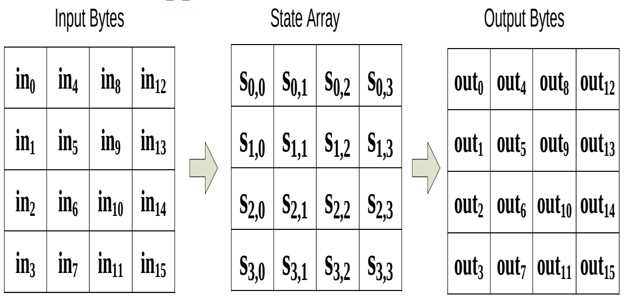


Figure 1 – State Population and Results

At the beginning of the cipher, the input array is copied into the *State* according the following scheme:

s[r,c] = in [r + 4c]for $0 \le r < 4$ and $0 \le c < 4$,

and at the end of the cipher the *State* is copied into the output array as shown below:

out[r+4c] = s[r,c] for $0 \le r < 4$ and $0 \le c < 4$

VI. CIPHER TRANSFORMATIONS

The AES cipher either operates on individual bytes of the State or an entire row/column. At the start of the cipher, the input is copied into the State as described in Section 2.2. Then, an initial Round Key addition is performed on the State. Round keys are derived from the cipher key using the Key Expansion routine. The key expansion routine generates a series of round keys for each round of transformations that are performed on the State.

The transformations performed on the state are similar among all AES versions but the number of transformation rounds depends on the cipher key length.

A. Subbytes ( ) Transformation

The *SubBytes* is a byte substitution operation performed on individual bytes of the *State*, as shown in Figure 3, using a substitution table called S-box.

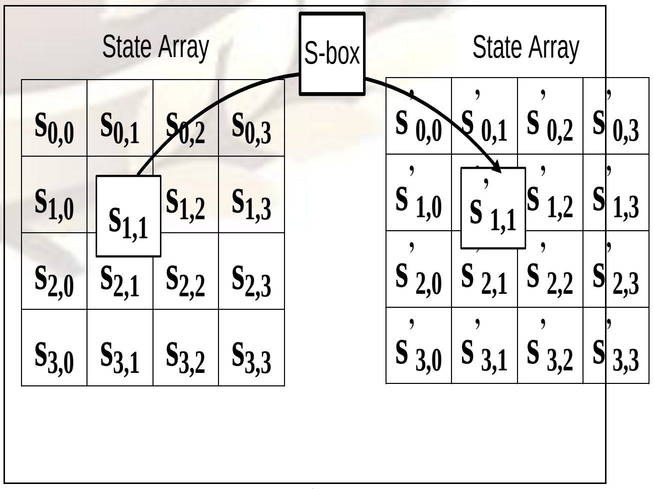

State Array
S-box
State Array
S0,0 S0,1 S0,2 S0,3
S1,0 S1,1 S1,2 S1,3
S2,0 S2,1 S2,2 S2,3
S3,0 S3,1 S3,2 S3,3
S' 0,0 S' 0,1 S' 0,2 S' 0,3
S' 1,0 S' 1,1 S' 1,2 S' 1,3
S' 2,0 S' 2,1 S' 2,2 S' 2,3
S' 3,0 S' 3,1 S' 3,2 S' 3,3

Figure 2 – SubBytes Transformation

The invertible S-box table is constructed by performing the following transformation on each byte of the State. [1]

- Take the multiplicative inverse in the finite field GF(2^8) of the byte.
- Apply the following transformation to the byte:

$$b_i' = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i$$

The b_i is the ith bit of the byte and c_i is the ith bit of a constant byte with the value of {63}. The combination of the two transformations can be expressed in matrix form as shown below:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Table-2 SubBytes Transformation

The S-box table shown in Table 2 is constructed by performing the two transformations described earlier for all possible values of a byte, ranging from {00} to {ff}. For example the substitution value for {53} would be determined by the intersection of the row with index '5' and the column with index '3'.

B. Shiftrows ( ) Transformation

The ShiftRows transformation cyclically shifts the last three rows of the state by different offsets. The first row is left unchanged in this transformation. Each byte of the second row is shifted one position to the left. The third and fourth rows are shifted left by two and three positions, respectively. The ShiftRows transformation is illustrated in Figure 3.

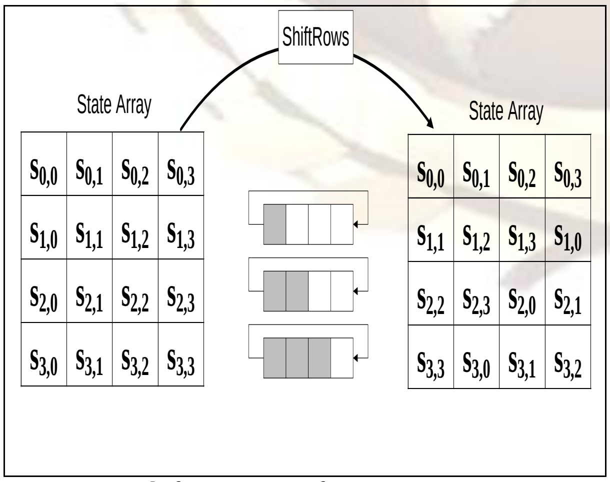


Figure 3 – ShiftRows Transformation

C Mixcolumns ( ) Transformation

This transformation operates on the columns of the *State*, treating each columns as a four term polynomial the finite field GF(2^8). Each columns is multiplied modulo x^4+1 with a fixed four-term polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ over the GF(2^8). The *MixColumns* transformation can be expressed as a matrix multiplication as shown below:

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

The *MixColumns* transformation replaces the four bytes of the processed column with the following values:

$$s_{0,c}' = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s_{1,c}' = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$
$$s_{0,c}' = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s_{1,c}' = (\{03\} \bullet s_{0,c} \oplus s_{1,c}) \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$

For the AES algorithm the irreducible polynomial is:
$m(x) = x^8 + x^4 + x^3 + x + 1$.[1]

The *MixColumns* transformation is illustrated in Figure 4. This transformation together with *ShiftRows*, provide substantial *diffusion* in the cipher meaning that the result of the cipher depends on the cipher inputs in a very complex way. In other words, in a cipher with a good diffusion, a single bit change in the plaintext will completely change the cipher text in an unpredictable manner.

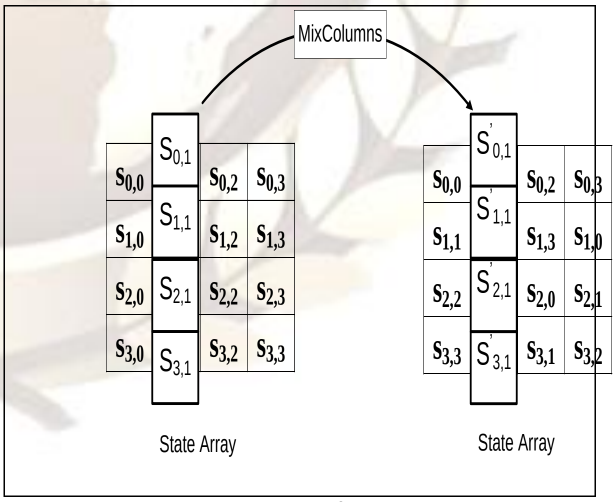


Figure 4 – MixColumns Transformation

D. Addroundkey ( ) Transformation:

During the *AddRoundKey* transformation, the round key values are added to the *State* by means of a simple *Exclusive Or* (XOR) operation. Each round key consists of N_b words that are generated from the KeyExpansion routine. The round key values are added to the columns of the state in the following way:

$$\left[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}\right] = \left[s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}\right] \oplus \left[w_{round*Nb+c}\right]$$

for $0 \le c < N_b$

In the equation above, the *round* value is between $0 \le round \le N_r$. When round=0, the cipher key itself is used as the round key and it corresponds to the initial *AddRoundKey* transformation displayed in the pseudo code in Figure 2.

The *AddRoundKey* transformation is illustrated in Figure 5.

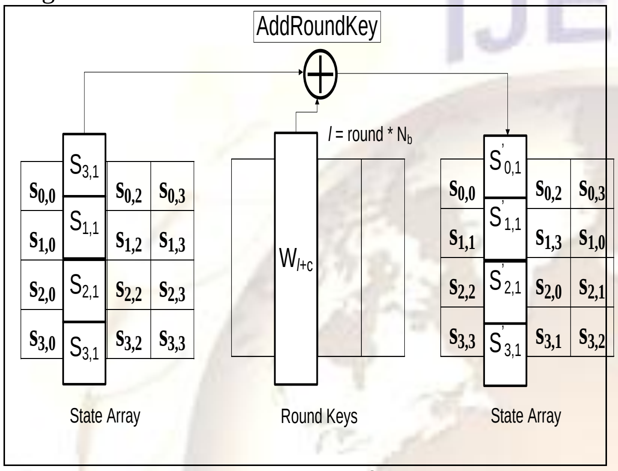

AddRoundKey
l = round * N_b
$S_{0,0}$ $S_{3,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
W_{l+c}
$S_{0,0}$ $S'_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,1}$ $S'_{1,1}$ $S_{1,3}$ $S_{1,0}$
$S_{2,2}$ $S'_{2,1}$ $S_{2,0}$ $S_{2,1}$
$S_{3,3}$ $S'_{3,1}$ $S_{3,1}$ $S_{3,2}$
State Array
Round Keys
State Array

Figure 5 – AddRoundKey Transformation

VII. FAULT DETECTION IN AES

For fault detection of the encryption or decryption in AES one may use redundant units [12], [8]. In [6] algorithm-level, round-level and operation-level concurrent error detection for the AES are used. In the algorithm-level, comparing the plain text with the output of a decryption after an encryption is proposed. The round-level error detection uses similar ideas in the rounds, where, the output of a round in encryption is applied to a round in decryption and is compared with the input. The operation-level (or transformation-level) error detection uses the inversion of a transformation in each round and compares the output with the input. Figure 2.5a shows the operation-level concurrent error detection for S-box and inverse S-box presented in [12]. In this figure, the 8-bit input I of the S-box (8-bit input B of the inverse S-box) is compared with the output of two consecutive transformations, S-box and inverse S-box (inverse S-box and S-box) using an 8-bit comparator to generate the error indication flag.

Fault coverage of the proposed parity code:

In this section we describe the results of extensive simulation experiments which were carried out to evaluate the fault coverage of the proposed parity EDC scheme for the Encryption module. We start with single bit faults injected into the data block at the beginning of the rounds; i.e., faults are not injected between the round transformations. Six types of tests have been performed with data block and key of 128 bits.

1. 5000 data blocks were selected randomly and a single bit error was injected into every position of the data block at each of the 10 rounds. The total number of tests of this type has been 5000 x 10 x 128 = 6.4 x 10^6. All these tests used the same secret key. Our parity bits scheme detected all the faults.
2. 5000 secret keys were randomly selected and used with the same 128-bit data block. The same single bit errors as in (1) were injected for a total of 6.4 x 10^6 tests. Here too, all the faults were detected by our parity scheme.
3. 100 random secret keys and 1000 random data block were selected and every data block was encrypted with each secret key. 1280 single bit errors were injected into every encryption for a total of 1.28 x 10^8 tests. All the faults were detected.

In the above three types of tests the parity check was performed at the end of the tenth round. In the next type of tests the parity check was instead done at the end of the round.

4. 5 x 10^5 random data blocks were selected and a single bit error was injected in each position of the data block. A single round was then performed yielding 100% fault coverage. The total number of tests of this type was 5 x 105 x 128 = 6.4 x 107.

The last two types of tests considered a single simplified round consisting only of SubBytes and MixColumns since these transformations affect the error propagation in the most complex way. The parity check was performed at the end of the (simplified) round. The observed fault coverage has again been 100%.

5. 256 32-bit data words of the type (xOOO)8w ere considered and a single bit error was injected into the first byte (the one that is varying) .The total number of tests of this type was 256 x 8 = 2048. All the faults were detected.
6. 1000 128-bit random data blocks were selected and a single bit error was injected in each position of the data block. The number of tests of this type was 1000 x 128 = 1.28 x 104. Again, all the injected faults were detected.

These six types of tests strongly suggest that the parity-based EDC achieves a 100% fault coverage for single bit faults. In fact, it can be proven that: The proposed parity-based EDC with a single checkpoint scheduled at the end of the last round is capable of detecting every single bit fault injected into the data block in the Encryption module, at the beginning of the rounds or between two round transformations. The proof had to be omitted for the sake of brevity. In this proof, however, we only considered single faults injected at the beginning of an encryption round, at the inputs of one of the four round transformations.

WORKING OF PROJECT

A. Design Hierarchy

The proposed AES128 hardware model is a 3-level hierarchical design as shown in Figure 6. The root module in the hierarchy is the AES128_cipher_top. This module implements the AES128 pseudo code displayed in Figure 2. It has two 128-bit inputs for receiving the cipher key and the plaintext. There is also a single bit input signal, *'Ld'*, which is used to indicate the availability of a new set of plaintext or cipher key on the input ports. The completion of the encryption process is indicated by asserting the 'done' single bit output.

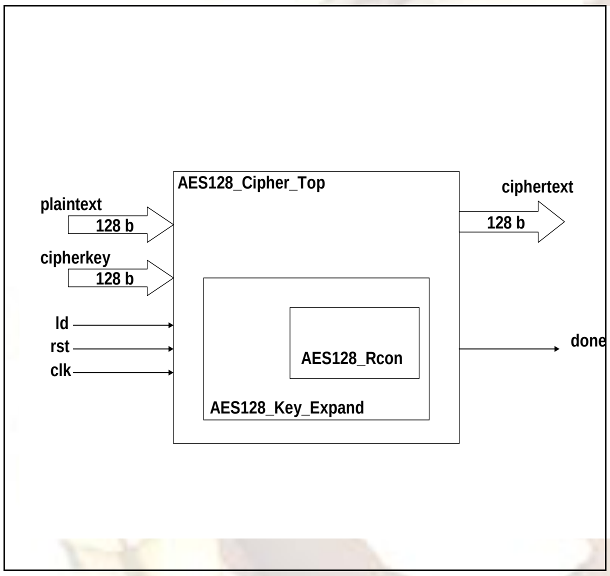


Figure 6 – Design Hierarchy

A unique feature of the proposed design is that the AES128_Key_Expand module is pipelined with the AES128_cipher_top module. While the AES128_cipher_top module is performing an iteration of the encryption transformations on the *State* using the previously generated round keys, the AES128_Key_Expand produces the next round's set of keys to be used by the root module in the next encryption iteration.

B. AES128 Encryption Process

The AES128_cipher_top module state diagram is shown in Figure 7. There are ten rounds of transformations represented by *r1* to *r10* states. The four cipher transformations introduced in section 2.3 are applied to each state. The *r0* state corresponds to the initial *AddRoundKey* transformation in Figure 2.

After leaving the *Reset* state, the AES128_Cipher_Top module waits for assertion of the *'Ld'* signal, which indicates that a valid set of plaintext and cipher key is available on the input ports. After reaching the *r0* state, there is a transition on every clock cycle for the next ten cycles, as ten rounds of encryption is applied to the *State*.

After going through ten rounds of transformations, the *'done'* signal is asserted to indicate the completion of cipher and availability of the ciphertext on the corresponding output port.

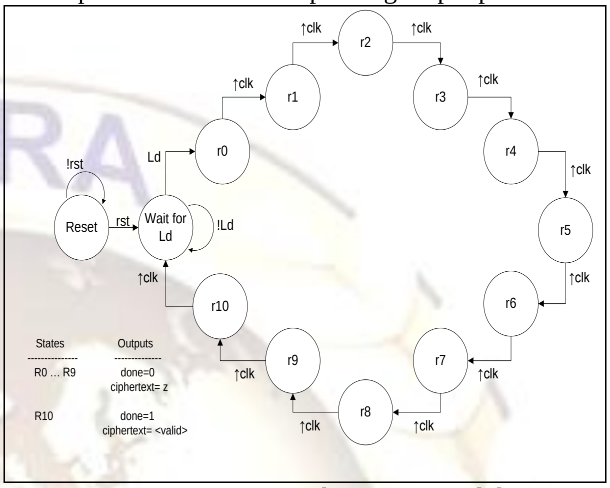


Figure 7 – AES128_Cipher_Top Module State Diagram

C. AES128 Round Key Generation

The round keys used by the AES128_Cipher_Top module are generated based on the state diagram shown in Figure 8. The AES128_Key_Expand and the AES128_RCon modules are responsible for generating the round keys. These two modules operate based on the state diagram shown in Figure 10, which is slightly different than the one used for the encryption process.

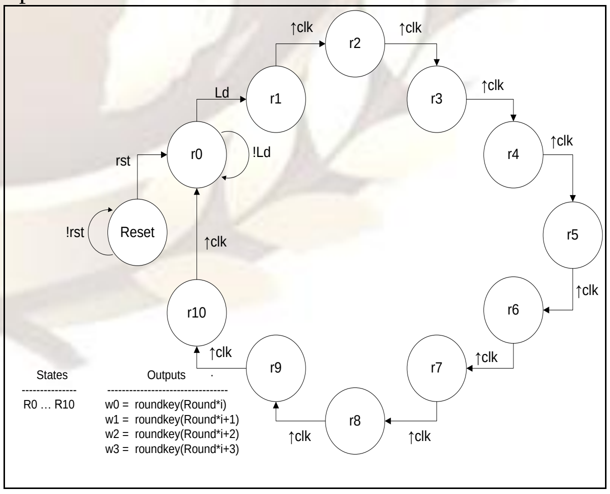


Figure 8 – AES128_Key_Expand Module State Diagram

In the state diagram shown above, the *'Ld'* signal is checked in the *'r0'* state and if asserted, then the cipher key is provided to the AES128_Cipher_Top module to be used for the initial *AddRoundKey* transformation.

The AES128_Key_Expand module generates four 32-bit keys for each round of the encryption process, by using the cipher key. Figure 9 shows the block diagram of the AES128_Key_Expand module. The cipher key is passed to this module through a 128-bit input port, and the round keys are generated on the four output ports.

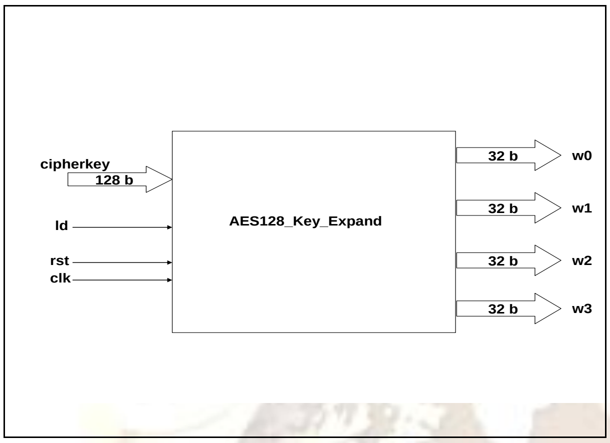


Figure 9 – AES128_Key_Expand Module

There is a 32-bit round constant value, which is used by the key expansion algorithm to generate the round keys. This value varies for each encryption round and for N_r=1 to N_r=10 is given by $[\{02\}^{i-1}$,{00},{00},{00}]. The AES128_RCcon module is used to generate this value as shown in Figure 10. The AES128_RCon module also operates based on the state diagram shown in Figure 10.

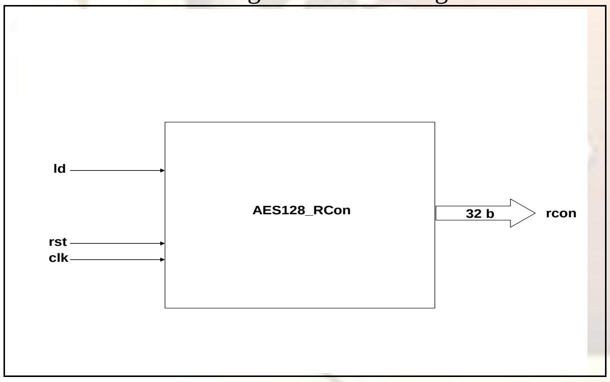


Figure 10 – AES128_Rcon Module

D. S-box

The structure of the S-box is divided into 5 blocks and the parity of each block is predicted as a function of inputs to that block. The formulations for the predicted parities as functions of inputs are obtained in this section. Since there exist three finite field multipliers using GF(((2^2)^2)^2)2.

VIII. IMPLEMENTATION OF ALGORITHM

A. Synthesis Methodology

The first step in the synthesis process is to read all the components in the design hierarchy. There are three components in the 3-level design hierarchy that needs to be synthesized. Since the RTL model utilizes a Verilog *"Package"*, then the synthesis tool needs to enable the semantics of a package. In addition, the synthesis tool needs to know if there are multiple instances of calling an automatic function in the design, to preserve separate values for each instance.

After reading the design files, they are *"Analyzed"* and *"Elaborated"* through which the RTL code is converted into the Synopsys Design Compiler(SDC) internal format. [6] The intermediate results are stored in the defined *"working library"*.

After this step, a 40MHz clock signal is applied to the clock port of the root module, and the synthesis tool is programmed not to modify the clock tree during the optimization phase. In addition, an arbitrary input delay of 5ns with respect to the clock port is applied to all input and output ports (except the clock port itself) to set a safe margin by considering any unintended source of delay such as the delay associated with driving module/modules.

Then, the design is constrained with hypothetical maximum area equal to zero to force the tool to make the gate level netlist as compact as possible.

In the next steps, the tool is programmed to consider a unique design for each cell instance by removing the multiply-instantiated hierarchy in the current design. Then, the synthesis script removes the boundaries from all the components in the design hierarchy and removes all levels of hierarchy. Finally, the tool compiles the design with high effort and reports any warning related the mapping and final optimization step. At the end, the tool generates reports for the optimized gate level netlist area, the worst combinational path timing, and any violated design constraint.

B. Synthesis Timing Result:

The synthesis tool optimizes the combinational paths in a design. In General, four types of combinational paths can exist in any design: [3]

1- Input port of the design under test to input of one internal flip-flip
2- Output of an internal flip-flip to input of another flip-flip
3- Output of an internal flip-flip to output port of the design under test
4- A combinational path connecting the input and output ports of the design under test

The last DC command in the script developed in previous section, instructs the tool to report the path with the worst timing. In this case, the path with the worst timing is a combinational path of type two. The delay associated with this path is the summation of delays of all combinational gates in the path plus the *Clock-To-Q* delay of the originating flip-flop, which was calculated as

24.09ns. By considering the setup time of the destination flip-flop in this path, which is 0.85ns, the 40MHz clock signal satisfies the worst combinational path delay. The delays of combinational gates, setup time of flip-flops and *Clock-To-Q* values are derived from the LSI_10k library file that was used for the mapping step during synthesis. The synthesis timing report is shown below:

C. Synthesis Area Result

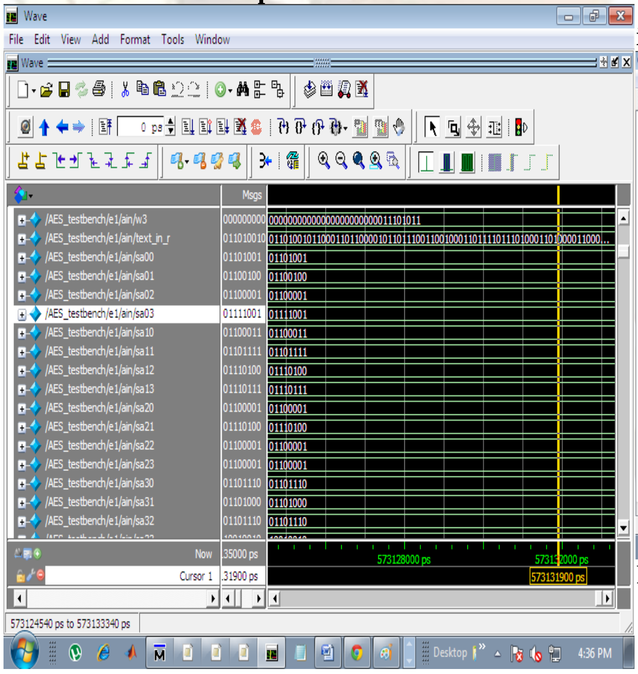
The synthesis area report shows the total number of cells and nets in the netlist. It also uses the area parameter associated with each cell in the LSI_10K library file, to calculate the total combinational and sequential area of the netlist. The total area of the gate level netlist is unknown since it depends on total area of the inter connects, which itself is a function of the wiring load model used in physical design. The total cell area in the netlist is reported as 22978 units, which is the sum of combinational and sequential areas. The synthesis area report is shown below

D. Synthesis Constraint Violators

To enforce the synthesis tool to create the most compact netlist, the area of the gate level netlist was constrained to zero during the synthesis process. As a result, the only constraint violation, which is expected, is related to the area as shown bellow

IX. RESULT AND DISCUSSION

i. Area Utilization Report



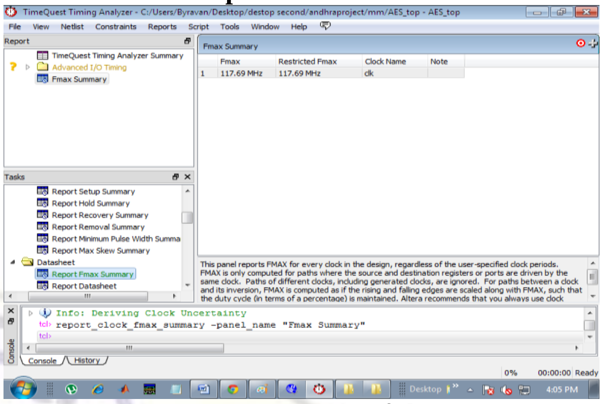
```
/AES_testbench/e1/ain/w3         000000000  00000000000000000000000011101011
/AES_testbench/e1/ain/text_in_r  011010010  0110100101100011011000010110111001100100011011110111010001101000011000...
/AES_testbench/e1/ain/sa00       01101001   01101001
/AES_testbench/e1/ain/sa01       01100100   01100100
/AES_testbench/e1/ain/sa02       01100001   01100001
/AES_testbench/e1/ain/sa03       01111001   01111001
/AES_testbench/e1/ain/sa10       01100011   01100011
/AES_testbench/e1/ain/sa11       01101111   01101111
/AES_testbench/e1/ain/sa12       01110100   01110100
/AES_testbench/e1/ain/sa13       01110111   01110111
/AES_testbench/e1/ain/sa20       01100001   01100001
/AES_testbench/e1/ain/sa21       01110100   01110100
/AES_testbench/e1/ain/sa22       01100001   01100001
/AES_testbench/e1/ain/sa23       01100001   01100001
/AES_testbench/e1/ain/sa30       01101110   01101110
/AES_testbench/e1/ain/sa31       01101000   01101000
/AES_testbench/e1/ain/sa32       01101110   01101110

Now       35000 ps     573128000 ps    573132000 ps
Cursor 1  31900 ps                     573131900 ps

573124540 ps to 573133340 ps
```

Figure 11. Simulated output.

ii. Performance Report



TimeQuest Timing Analyzer - C:/Users/Byravan/Desktop/destop second/andhraproject/mm/AES_top - AES_top

Fmax Summary

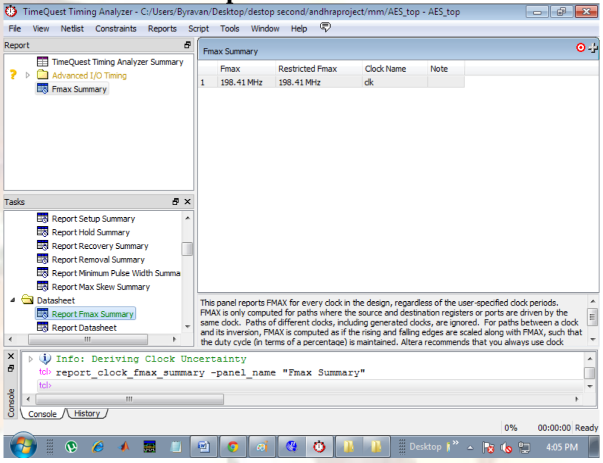
	Fmax	Restricted Fmax	Clock Name	Note
1	117.69 MHz	117.69 MHz	clk	

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock

```
Info: Deriving Clock Uncertainty
tcl> report_clock_fmax_summary -panel_name "Fmax Summary"
tcl>
```

Figure 12.Fmax. Summary report of slow carner.

iii. Performance Report



TimeQuest Timing Analyzer - C:/Users/Byravan/Desktop/destop second/andhraproject/mm/AES_top - AES_top

Fmax Summary

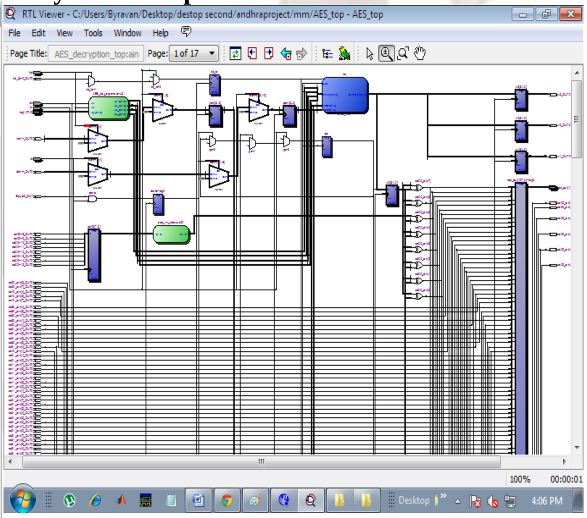
	Fmax	Restricted Fmax	Clock Name	Note
1	198.41 MHz	198.41 MHz	clk	

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock

```
Info: Deriving Clock Uncertainty
tcl> report_clock_fmax_summary -panel_name "Fmax Summary"
tcl>
```

Figure 13.Fmax. summary report of fast carner.

iv. Synthesis Report



RTL Viewer - C:/Users/Byravan/Desktop/destop second/andhraproject/mm/AES_top - AES_top
File Edit View Tools Window Help
Page Title: AES_decryption_top:ain
Page: 1 of 17
100%
00:00:01
Desktop
4:06 PM

Figure 14. RTL Schematic report

v. Map Viewer

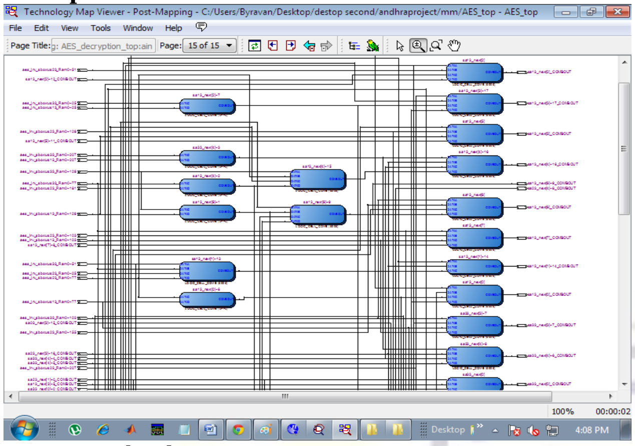
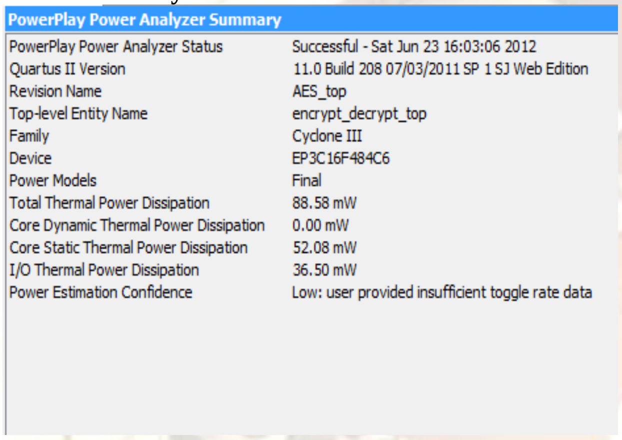

Technology Map Viewer - Post-Mapping - C:/Users/Byravan/Desktop/destop second/andhraproject/mm/AES_top - AES_top
File Edit View Tools Window Help
Page Title: AES_decryption_top:ain
Page: 15 of 15
100%
00:00:02
Desktop
4:08 PM

Fig 15. Technology map viewer

vi. Power Analyzes



PowerPlay Power Analyzer Summary	
PowerPlay Power Analyzer Status	Successful - Sat Jun 23 16:03:06 2012
Quartus II Version	11.0 Build 208 07/03/2011 SP 1 SJ Web Edition
Revision Name	AES_top
Top-level Entity Name	encrypt_decrypt_top
Family	Cyclone III
Device	EP3C16F484C6
Power Models	Final
Total Thermal Power Dissipation	88.58 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	52.08 mW
I/O Thermal Power Dissipation	36.50 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Fig.16 Power dissipation report

CONCLUSION

In our project we perused the concept of Cryptography including the various schemes of system based on the kind of key and a few algorithms such as RSA and AES. We studied in detail the mathematical foundations for AES based systems, basically the concepts of rings, fields, groups, Galois finite fields and their properties. The various algorithms for the computation of the scalar product of a point were studied and their complexity were analyzed.

The advantage of this over the other Fault detection systems are proved by parameters .The key strength of this systems in comparison to other is fault detection is impleted in all levels of algorithm implementation and this will increase reliability.

REFERENCES

[1] M. Akkar and C. Giraud, “An Implementation of DES and AES, Secure against Some Attacks,” In Proc. of the Workshop on Cryptographic Hardware and Embedded Systems (CHES2001), Paris, France, pp. 315-325, May 2001.

[2] R. Anderson, E. Biham, and L. Knudsen, “Serpent: A Proposal for the Advanced Encryption Standard,” AES algorithm submission, June 1998.

[3] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, “Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard,” IEEE Trans. on Computers, vol. 52, no. 4, pp. 492-505, April 2003.

[4] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, “An efficient hardwarebased fault diagnosis scheme for AES: performances and cost,” In Proc. of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT2004), Cannes, France, pp. 130-138, Oct. 2004.

[5] D. Boneh, R. A. DeMillo, and R. J. Lipton, “On the Importance of Eliminating Errors in Cryptographic Computations,” Journal of Cryptology, vol. 14, no. 2, pp. 101-119, 2001.

[6] L. Breveglieri, I. Koren, and P. Maistri, “Incorporating Error Detection and Online Reconfiguration into a Regular Architecture for the Advanced Encryption Standard,” In Proc. of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT2005), Monterey, CA, USA, pp. 72-80, Oct. 2005.

[7] D. Canright, “A Very Compact Rijndael S-box,” Naval Postgraduate School Technical Report: NPS-MA-05-001, May 2005.

[8] G. C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, “Fault localization, error correction, and graceful degradation in radix 2 signed digit-based adders,” IEEE Trans. on Computers, vol. 55, no. 5, pp. 534-540, May 2006.

[9] G. C. Cardarilli, S. Pontarelli, M. Re, and A. Salsano, “A self checking Reed Solomon encoder: design and analysis,” In Proc. of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT2005), Monterey, CA, USA, pp. 111-119, Oct. 2005.

[10] S. Fenn, M. Gossel, M. Benaissa, and D. Taylor, “On-Line Error Detection for Bit-Serial Multipliers in GF(2^m),” Journal of Electronic Testing: Theory and Applications, vol. 13, no. 1, August 1998.

[11] A. Hodjat and I. Verbauwhede, “Area-Throughput Trade-Offs for Fully Pipelined30 to 70 Gbits/s AES Processors,” IEEE Trans. on Computers, vol. 55, no. 4,pp. 366-372, April 2006.

[12] T. Ichikawa et al, “Hardware Evaluation of the AES Finalists,” In Proc. 3th AES Candidate Conference, New York, April 2000.