Development of Advanced Encryption Standard Architecture with Sbox Parity

Vanitha MOHANRAJ*[*] and Subha SRINIVASAN

School of Information Technology and Engineering, Vellore Institute of Technology University, India

([*]Corresponding author's e-mail: mvanitha@vit.ac.in, ssubha@vit.ac.in)

Received: 21 November 2014, Revised: 21 September 2015, Accepted: 21 October 2015

Abstract

In this paper, an efficient AES (Advanced Encryption Standard) has been designed so that security levels can be increased which is caused due to faults and errors. The AES algorithm includes mainly 4 transformations, which are Sub-byte, Shift row, Mix column, Add round key. The security of Sbox has been increased by using even parity, which is used to detect faults rather than correction. A FIFO (First-In First-Out) is also considered to store the parity bits of Sbox. The expected parity bit of the output is predicted initially with the help of look up table (LUT) and compared with output parity bit. By this we can improve the fault coverage of Sbox. Since the Sbox parity architecture involves more MUX and XOR, their area is reduced by using the Binary Decision Diagram (BDD) approach and a pass transistor implementation of MUX which reduces the area drastically. Verilog HDL language is used to model the architecture and verification was done on Modelsim. Design, synthesized using a Cadence Register Transfer Level (RTL) complier tool. The synthesized result shows that there is an area overhead of 8 % and high fault coverage of 99.23 %.

Keywords: AES, DES, shift row, mix column, FIFO

Introduction

As the development of computer networks and communication systems are growing faster, a great mass of information is being exchanged in communication networks. Security of data is a main concern and plays a very important role in data transmission. NIST (National Institute of Standard and Technology) agreed to adopt the block based cipher system, Data Encryption Standard (DES) in 1970. It solved many security issues, but the security strength of the encryption was weak and it failed for many attacks. Thus, NIST announced a better DES algorithm called Triple DES (3DES). This 3DES uses the same algorithm as DES but the security level was increased in the 3DES as compared to DES. There were 2 major disadvantages in the 3DES algorithm. First, it requires 3 times the execution time of DES. Second, DES and 3DES uses only 64 bit length block data. So the safety of the cipher system was not satisfactory. NIST asked for a new generation cipher, called the Advanced Encryption Standard (AES) in 1997 [1].

NIST choose the Rijndael algorithm as the final AES in 2001. This AES algorithm can resist any kind of attacks. Nowadays this algorithm has become the main source for the security of data. Authors proposed a module for key generation which will be generated on the fly [2]. AES algorithm can resist password attacks of any kind [3]. Sklavos and Koufopavlou [4] proposed an idea for area reduction; they performed encryption and decryption with single core, which differs from the conventional methods. Satoh [5] Authors substituted all the sixteen blocks using sbox during subbyte transformation which has increased the speed but the area got increased.

Related work

Kermani *et al*. [6] addressed the faults that maliciously occur in the hardware implementations of the AES which may cause erroneous output. They proposed to divide the composite field S-box and inverse S-box into blocks and the predicted parities of these blocks are obtained. Through exhaustive searches among all available composite fields, they found the optimum solutions for the least overhead parity-based fault detection structure and through error injection simulations for one S-box (respectively inverse S-box), they proved that the total error coverage of almost 100 % for 16 S-boxes can be achieved.BDD is an alternate method to represent the Boolean functions with nodes as input variables with two decision tree. Beg *et al*. [7] used '0' network and '1' network. Each node selection in a MUX with the select lines as inputs. Yen *at al*. [8] proposed multiple stuck-at-fault detection for mix column, sub-byte but the area overhead of Sbox is very high with low fault coverage. Many authors proposed the fault detection of Sbox which is a part of the sub-byte transformation. Ocheretnij *et al.* [9] proposed parity prediction of Sbox to detect single struck-at-faults with high fault coverage and optimum area overhead. Bertoni *et al.* [10] says that the fault that occurs in the inverter can be detected by using Self-checking invertor. Natale *et al.* [11] proposed Sbox with ROM implementation by using NAND gates. The area overhead of this methodology is low with low fault coverage.

Table 1 Relationship between rounds and key length.

Type of AES	Key length (N_k)	Group size	Round number (N_r)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

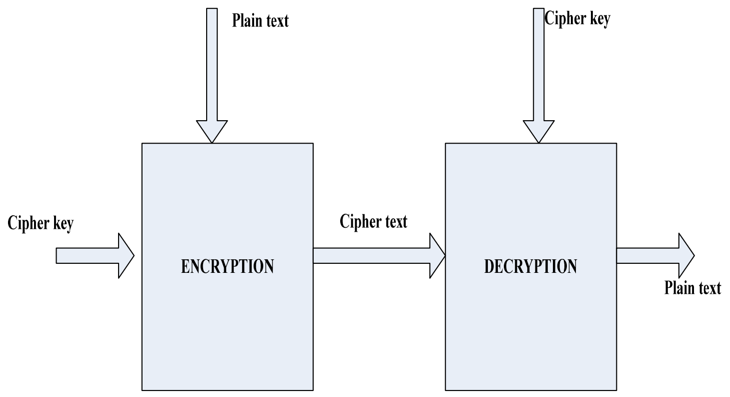


Figure 1 Block diagram of AES algorithm.

Maistri and Leveugle [12] proposed Differential Fault Analysis (DFA) and Double-Data-Rate (DDR) are the most powerful techniques to attack cryptosystems. Satoh *et al.* [13] proposed a concurrent error detection scheme for the AES algorithm. Mozaffari-Kermani *et al*. [14] in another paper presented 3 parity prediction schemes to make the EEA division algorithm more reliable. They have proved that the CED division architecture using dual multiple parity prediction scheme was capable of reaching close to 100 % error coverage for single and multiple stuck-at faults. Hussain and Gondal [15] proposed an algorithm to generate Inverse Sbox value. This gives low convolution architecture and easily achieves high throughput and low latency.

Preliminaries

AES Algorithm

The AES algorithm can be implemented in 3 different ways with key lengths of 128, 256, 192 bits as shown in **Table 1**. Nk represents the key length (number of words in the keys, each word consists of 32 bit or 4 bytes of data). Round number (Nr) is the number of rounds needed by the AES to complete encryption or decryption. It depends on the length and variation of the length of key. As the length of key is increased the complexity of the algorithm also increases and so the security. We generally prefer 128 bits cipher key with block (state), which is always 128 bits.

The algorithm mainly contains 2 parts encryption and decryption. Encryption takes 2 inputs, cipher key and plain text, which are of 128 bits. Cipher keys are of different length 128, 192 and 256 bit. Cipher key converts the plain text into the cipher text which is of encrypted data as shown **Figure 1**. The AES algorithm is performed in 4 different steps. These steps are executed in a sequential manner which form the rounds.

A. Sub-byte transformation B. Shift row transformation
C. Mix column transformation D. Add round key transformation

State is a 4×4 array of bytes, which contain the 128 bits of keys and in another state the 128 bits of plain text. At the initial stage plain text (state) is added (XORed) with the round key. This process is called Add_round_key. The round key is generated by key expansion block. Then the round of AES starts with the initial stage sub-byte operation followed by shift row, mix column and add round key. The number of rounds depends on the key length. In the final rounds the mix column is removed and the process is performed as shown in **Figure 2(a)**, which gives the cipher text output containing 128 bits or 4 words or 16 bytes. The cipher text obtained from the encryption is transmitted through a channel (in communication system) and at the receiver end decryption is used to convert the cipher text to plain text. Decryption is just the inverse of encryption as shown in **Figure 2(b)**. The key expansion block in decryption is same as that of the encryption.

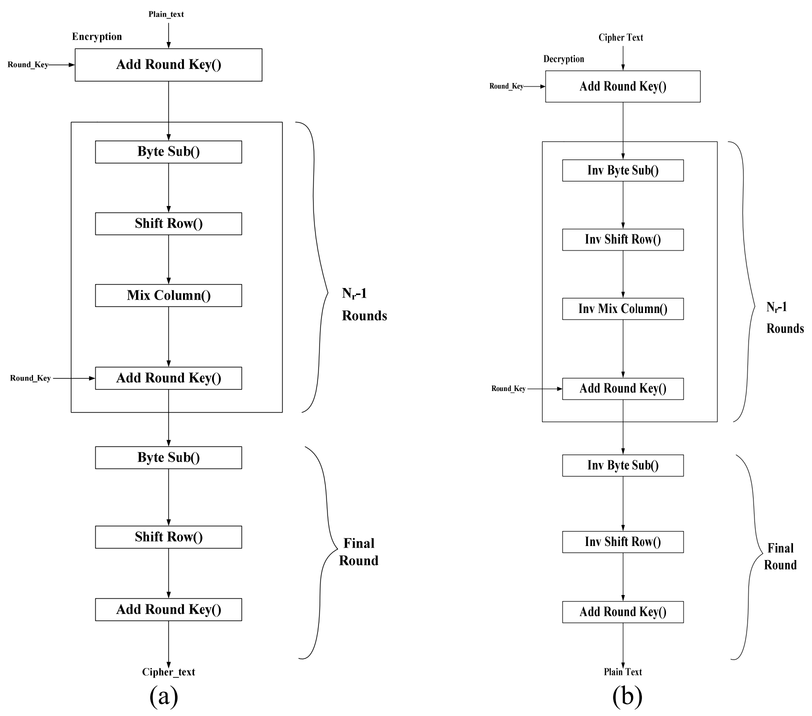


Figure 2 Various steps involved in AES algorithm.

Proposed architecture

Sub-byte transformation

Sub-byte is a nonlinear substitution based on the Sbox matrix as shown in **Figure 3**. The Sbox transformation mainly contains 2 steps.

1. Multiplicative inverse of each and every byte of the state with the irreducible polynomial ($X^8 + X^4 + X^3 + 1$).

2. Apply the affine transformation over $GF(2^8)$.

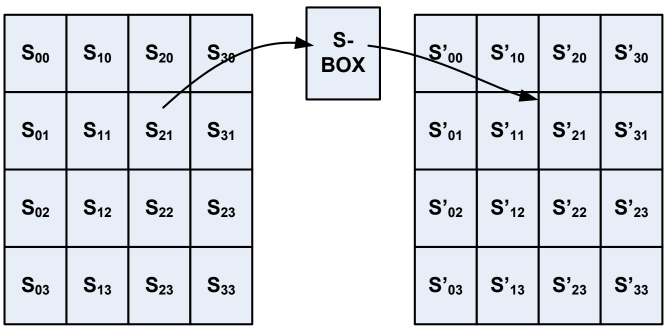

S_{00} S_{10} S_{20} S_{30}
S_{01} S_{11} S_{21} S_{31}
S_{02} S_{12} S_{22} S_{23}
S_{03} S_{13} S_{23} S_{33}
S-BOX
S'_{00} S'_{10} S'_{20} S'_{30}
S'_{01} S'_{11} S'_{21} S'_{31}
S'_{02} S'_{12} S'_{22} S'_{23}
S'_{03} S'_{13} S'_{23} S'_{33}

Figure 3 Transformation of state using Sbox.

Let $\mathrm{I} = (\mathrm{i}_7, \mathrm{i}_6, \mathrm{i}_5, \mathrm{i}_4, \mathrm{i}_3, \mathrm{i}_2, \mathrm{i}_1, \mathrm{i}_0)$ be the input to the Transformation matrix δ and $\mathrm{S} = (\mathrm{s}_7, \mathrm{s}_6, \mathrm{s}_5, \mathrm{s}_4, \mathrm{s}_3, \mathrm{s}_2, \mathrm{s}_1, \mathrm{s}_0)$, then the output of Transformation matrix is;

$$S = i_7 + i_6 + i_1$$
$$S = i_6 + i_3 + i_1\ S = i_6 + i_5 + i_4 + i_3 + i_0$$
$$S = i_6 + i_5 + i_1 + i_0$$
$$S = i_6 + i_5 + i_1 + i_0$$
$$S = i_6 + i_5 + i_4 + i_2 + i_0$$
$$S = i_5 + i_4 + i_2 + i_0$$
$$S = i_6 + i_5 + i_4 + i_3 + i_1 + i_0$$
$$S = i_2 + i_0$$

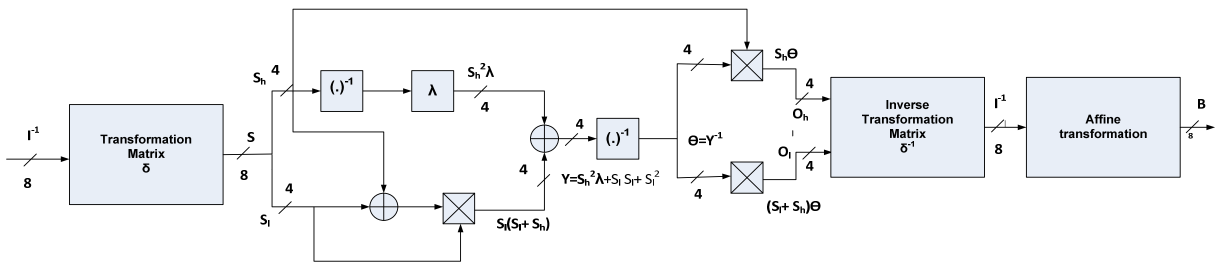


Figure 4 Implementation of Sbox with a combinational circuit.

Sbox can be implemented in 2 ways, first by Look Up Table (LUT) and the second by using a combinational circuit as shown in **Figure 4**. For LUT we need to store the values in the memory which has high access delay as compared to the combinational circuit and it has lower delay and higher speed of operation [6]. Each byte 'I' is transformed to 'S' using Eqs. (1) and (2);

$$I \in GF(2^8) \tag{1}$$

$$S = S_h x + S_l \tag{2}$$

which belongs to $GF(2^8)$, where;

$$X^2 + X + \lambda = 0 \tag{3}$$

Now the multiplicative inverse starts which gives the output;

$$O = O_h x + O_l = S^{-1} \tag{4}$$

where $O_h = S_h\theta$ and $O_l = (S_h + S_l)\theta$ and

$$\theta = S_h^2 + S_h S_l + S_l^2 \tag{5}$$

where $\lambda = (1100)_2$, S_h, S_l are higher and lower bits of S (8 bit) matrix.

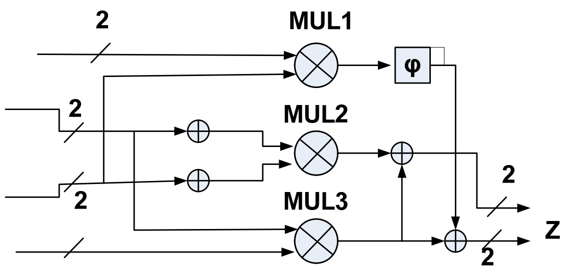
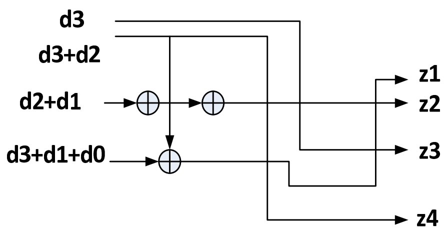


Figure 5 Multiplication in $GF(2^{2^2})$.

Where $\varphi = (01)_2$ in **Figure 5**. Now the multiplication transformation takes place followed by the affine transformation [6]. Each multiplication will take 4 addition (XOR gates) and 3 finite field multiplication. z1, z2, z3, z4 are the outputs of the multiplication with λ in $GF(2^{2^2})$ and they are given as input to the XOR gate as shown in **Figure 4**. **Figures 5 - 7** shows the combinational equivalent circuit to Sbox implementation in **Figure 4**. From these figures the Sbox operation becomes easy to implement and has very little delay when compared with the look up table approach [6].



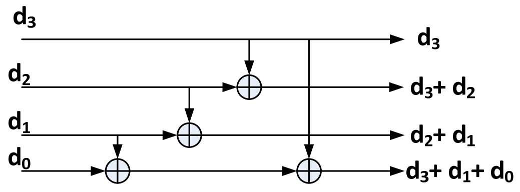


Figure 6 Multiplication of **λ** in $GF(2^{2^2})$.

Figure 7 Squaring in $GF(2^{2^2})$.

Shift row transformation

After sub-byte transformation, shift row follows. It shifts rows to the left by their row number, which mean the zeroth row will be shifted zeroth times and second row will be shifted by 2 times to the left as shown in the **Figure 8**. Parity analysis can also be done to the shift row, mix column and add round key transformation, but the sub-byte transformation which is nonlinear in nature can affect the whole state (4×4 array).

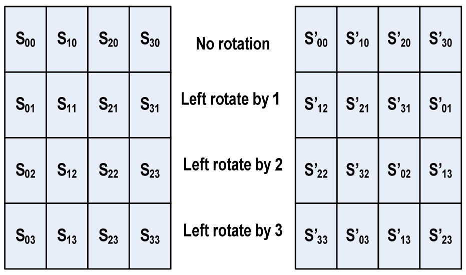


Figure 8 Transformation of shift row.

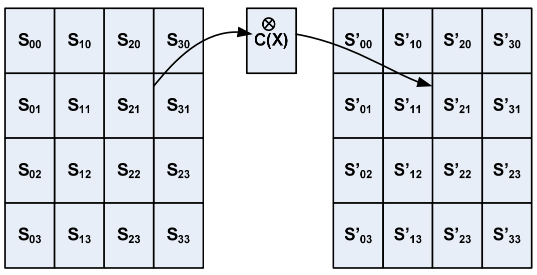


Figure 9 Transformation of Mix column.

Mix column

Mix column operates on the 4 byte or word at a time. Multiplication takes place with each column and is entirely based on the polynomial coefficient $GF(2^8)$. This is shown in **Figure 9**.

Add round key

For each round a 128 bit key is generated from the key expansion process, this key is then given to add round key which is XOR of each column of state with each column of key as shown in **Figure 10**.

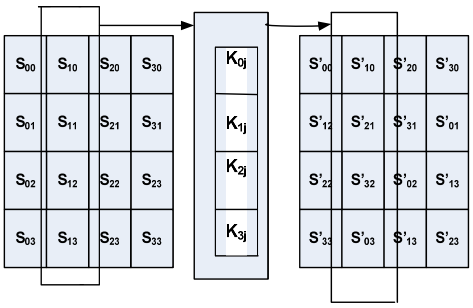


Figure 10 Transformation of add round key.

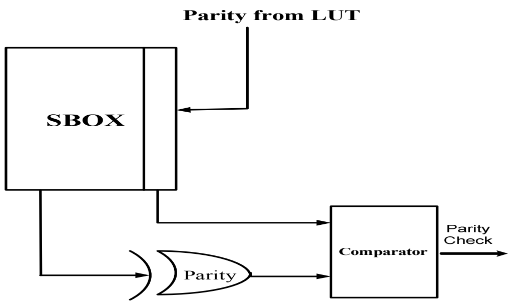


Figure 11 Even Parity implementation.

Fault detection architecture

We concentrate more on the Sbox, as it is non-linear and it will affect all the rounds in AES, if there are any errors or faults in the Sbox (state). Checking these errors is very important as it provides good security. This fault checking is done by parity bit. Prediction of parity bit is easy and provides low cost and high security. The prediction of parity is very easy with respect to the shift row, mix column and add round key. If any fault occurs in these steps, then the fault propagation is not that affective as it is in the Sbox.

Sbox is implemented either by combinational circuit or by the 256×9 bits memory with address decoding technique. A solution was provided to generate the outgoing parity bit in [7] and it is shown in the **Figure 11**.

The width of memory is 9 bits which contains each byte of Sbox along with the parity bit. The parity here was considered as even parity. To increase the dependability, a new architecture is proposed as shown in **Figure 12**. In this paper we mainly focus on the Sbox, as it is non-linear in nature, which has dependency on fault propagation. This was described in [9]. If the fault occurs in the early stage of the rounds then the fault propagation is very high up to 70 to 80 bits, which is quite very high, but if the fault occurs in the later stage (say 8th or 9th rounds) then the fault propagation is low and may be zero some times. So the detection of a fault at the early stages (round) makes an efficient AES algorithm.

Figure 12 shows the architecture for single stuck at fault occurrence. We consider the even parity in the architecture. It mainly contains the combinational circuit Sbox module and LUT and some equalizers (comparators). In the Sbox transformation, each byte of state is transformed to another state. Each byte coming into the combinational circuit as shown in **Figure 12**, the even parity bit of each byte is predicted with the XOR gate and output parity bit is predicted by taking from the output look up table of Sbox. After execution of Sbox the parity of this executed byte is found through the XOR gate and the corresponding input parity prediction is taken from the input LUT. These values are compared to each with an equalizer. If they are all equal then we can conclude that there is no error or fault in the AES Sbox.

BDD based MUX design

Since the parity implementation of the SBox requires more number of XOR gates and MUX, this requires a huge area for implementation and it increase the area by 8 % and thereby its power and its speed. So an alternate implementation is followed in this Sbox building i.e. Binary Decision Diagram (BDD) using the Shannon's decomposition theorem as shown in Eq. (6) is applied for the construction of the MUX and XOR which reduces the area exponentially.

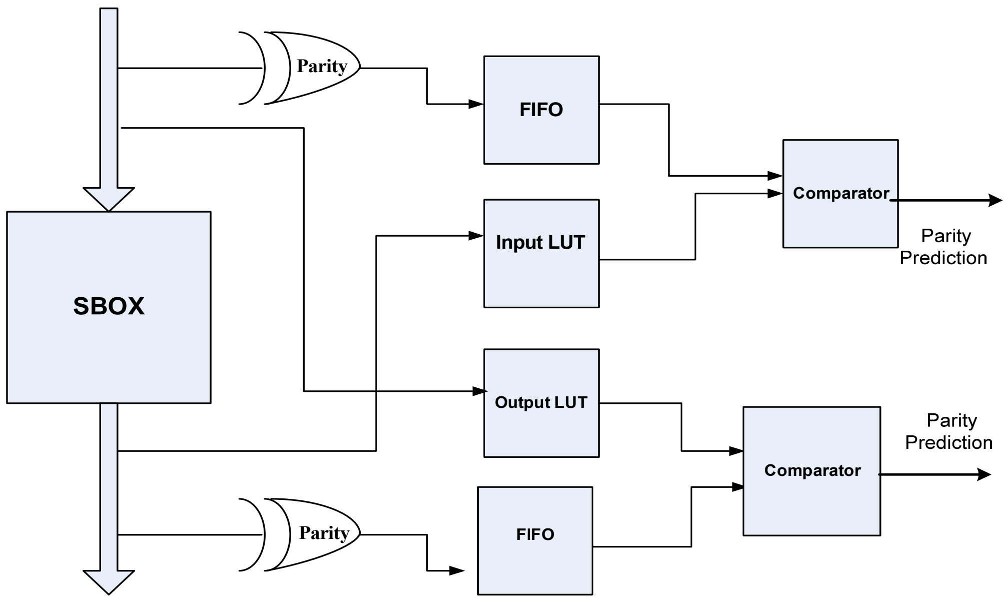


Figure 12 Efficient Parity implementation of Sbox.

BDD is an alternative method to represent the Boolean functions with nodes as input variables with 2 decision trees as '0' network and '1' network [7,14]. Each node selection is a MUX with select lines as inputs.

$$F(x_1,\ldots x_i,\ldots x_n) = x_i f(x_1,\ldots 1,\ldots x_n) + x_i' f(x_1,\ldots 0,\ldots x_n) \quad (6)$$

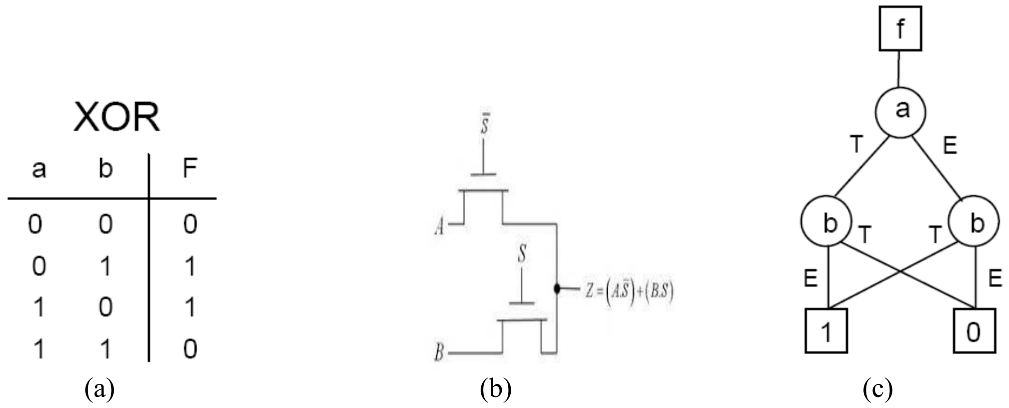


Figure 13 (a) Truth Table of an XOR gate, (b) BDD representation of XOR gate and (c) Pass Transistor implementation of MUX.

Figure 13(a) represents the XOR gate implementation and **Figure 13(b)** XOR gate using the BDD representation. Variables 'a' and 'b' are input and Edge 'T' represents the input variable representing the '1' value and 'E' represent the '0' value of the node. The node can be implemented with a MUX as shown in **Figure 13(c)** and thereby any Boolean function could be represented using only MUX. This drastically reduces the transistor count against the CMOS implementation because the 2:1 MUX implementation requires 2 AND gates and 1 OR gate with 1 INV which counts to 20 transistors compared to only 4 transistors in pass transistors implementation. The only problem with this pass transistor logic is the logic degradation which could also be resolved by using a buffer at the end of the MUX which would consume 2 extra transistors. So it is still more efficient in terms of area and power consumption.

Experimental results

In this section area and fault coverage calculation has been done using 0.35 and 0.18 μm. The simulation has been performed in Modelsim and Cadence Nclaunch. The synthesis has been performed in the Cadence Register Transfer Level (RTL) compiler.

Table 2 ASIC Synthesis results showing area and power of Sbox FIFO.

Sbox with FIFO (μm)	Area (μm^2)	Power (μW)
0.18	26202	86240
0.35	37562	11776

Table 2 shows the area and power required by Sbox with FIFO in both 0.18 and 0.35 μm. These values are compared with the architectures proposed in the [9-11]. This architecture has been synthesized using the same technology library. In all the above mentioned architectures Sbox have been implemented as a combinational logic.

Table 3 shows the comparison of fault coverage and area overhead. The solution proposed in [9] allows an area overhead of 27.62 % which is optimum and fault coverage of 91.95 %. The solution proposed in [10] has an optimum fault coverage of 93.43 % with a high area overhead. In [11] the fault coverage is very high with a lower overhead. Our proposed architecture has an area overhead of 8 % and a fault coverage of 99.23 % compared with [11].

In this experiment we considered only a single stuck-at-fault and fault coverage has been measured using Cadence-Nclaunch. By introducing the single error (fault) in Sbox we tried to determine the fault coverage by providing effective test vectors.

Table 3 Comparison with proposed architecture.

	Area (µm^2)	**Area overhead**	**Fault coverage (Area optimization)**
V Ocherentnij [9]	29432	27.62 %	91.95 %
G Bertoni [10]	23614	59.06 %	93.43 %
G Di Natale [11]	34780	8 %	99.20 %
Proposed Architecture (350 nm)	37562	8 % (Excess)	99.23 %
Proposed Architecture with BDD	10390	70 % (Less with [11])	99.23 %

Conclusions

In this paper, we have successfully designed an efficient AES algorithm with parity of Sbox. We have used Verilog HDL language to describe the model and verification was done on Modelsim and Cadence Nclaunch. The area and power were calculated and tabulated in the results using Cadence RTL compiler. There is an area overhead of 8 % due to the usage of FIFO, but the security of Sbox is improved with a fault coverage of 99.23 % with a slight increase in power, which is negligible.

References

[1] National Institute of Standards and Technologies (NIST). Advanced encryption standard (AES). *Fed. Inform. Process. Stand.* 2000; **197**, 1-51.
[2] R Sever, AN Ismailglu, YC Tekmen, M Askar and B Okcan. A high sped FPGA implementation of the Rijndael algorithm. *In*: Proceedings of the Euromicro Symposium on Digital System Design, 2004, p. 358-62.
[3] LIU Zhenzhen. Implementation of AES encryption based on FPGA. *Mod. Electron. Tech.* 2007; **23**, 103-4.
[4] N Sklavos and O Koufopavlou. Architectures and VLSI implementations of the AES-Proposal Rijndael. *IEEE Trans. Comput.* 2002; **51**, 1454-9.
[5] A Satoh, S Morioka, K Takano and S Munetoh. A compact Rijndael hardware architecture with SBox optimization. *Lect. Notes Comput. Sci.* 2000; **2248**, 239-54.
[6] MM Kermani and A Reyhani-Masoleh. Parity prediction of S-box for AES. *In*: Proceedings of the *IEEE Canadian Conference on Electrical and Computer Engineering.* Ottawa, Canada, 2006, p. 2357-60.
[7] B Azam and B Ajmal. Reliability of nano-scaled logic gates based on binary decision diagrams. *In*: Proceedings of the International Conference on Modeling, Simulation and Visualization Methods. 2014, p. 1-5.
[8] CH Hsu and BF Wu. Simple error detection methods for hardware implementation of advanced encryption standard. *IEEE Trans. Comput.* 2006; **55**, 720-31.
[9] V Ocheretnij, G Kouznetsov, R Karri and M Gossel. On-line error detection and BIST for the AES encryption algorithm with different SBox implementations. *In*: Proceedings of the 11th *IEEE International On-Line Testing Symposium.* Saint Raphael, French Riviera, France2005, p. 141-6.
[10] G Bertoni, L Breveglieri, I Koren, P Maisti and V Piuri. Error analysis and detection procedures for a hardware implementation of the advance encryption standard. *IEEE Trans. Comput.* 2003; **52**, 492-505.

[11] GD Natale, ML Flottes and B Rouzeyre. A novel parity bit scheme for SBox in AES circuits. *In*: Proceedings of the Design and Diagnostics of Electronic Circuits and Systems. Kraków, Poland, 2007, p. 11-3.

[12] P Maistri and R Leveugle. Double-data-rate computation as a countermeasure against fault analysis. *IEEE Trans. Comput.* 2008; **57**, 1528-39.

[13] A Satoh, T Sugawara, N Homma and T Aoki. High-performance concurrent error detection scheme for AES hardware. *Lect. Notes Comput. Sci*. 2008; **5154**, 100-12.

[14] M Mozaffari-Kermani, R Azarderakhsh, CY Lee and S Bayat-Sarmadi. Reliable concurrent error detection architectures for extended euclidean-based division over GF(2^m). *IEEE Trans. Very Large Scale Integrat. Syst.* 2014; **22**, 995-1003.

[15] I Hussain and MA Gondal. An algorithm to generating inverse S-box for Rijndael Encryption standard. *3D Res*. 2014; **5**, 1-5.