Low Power FPGA Implementation of an Efficient AES- SBOX Realization for Health Care Applications

E. S. Selva Priya (selvapriyaes@ssn.edu.in)  
SSN College of Engineering

L. Suganthi  
SSN College of Engineering

Research Article

Keywords: AES, LUT, OPPRM, Field Programmable gate array, Zynq, Sub pipelining

Posted Date: March 8th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2646986/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: No competing interests reported.
Low Power FPGA Implementation of an Efficient AES-SBOX Realization for Health Care Applications

E.S.Selva Priya,1 Dr.L.Suganthi,2

1 Research Scholar, Department of Biomedical Engineering, SSN College of Engineering, Kalavakkam, Chennai-603110, Tamil Nadu, India

2 Associate Professor, Department of Biomedical Engineering, SSN College of Engineering, Kalavakkam, Chennai-603110, Tamil Nadu, India

Correspondence should be addressed to E.S.Selva Priya selvapriyaes@ssn.edu.in

Abstract

In present digital world, the data transmitted over network has become worldwide. The information compiled, processed and communicated in Bits and bytes of digital format over communication channels. For secure communication an advanced encryption Standard (AES) cryptographic algorithm is employed to avoid the risk of secret information tampering. This Research article predominantly focuses on low power and low area hardware architecture designed using Field Programmable Gate array. The novel architecture is derived by employing composite field arithmetic for substitute by te and inverse substitute bytes operation. By this approach the unwanted delay occurs during the Look up table (LUT) method has been averted. Optimized Positive polarity reed Muller (OPPRM) architecture is introduced in order to minimize the hardware utilization, eventually subpipelining technique has brought in, to enhance the speed of the architecture. The Performance metrics of OPPRM-AES is evaluated with respect to Bonded IPADS, Slice, LUTS, Number of slices, Static as well as Dynamic Power. The performance of a proposed architecture is compared with AES-PPRM, AES-CFA and AES-ECG. The architecture is implemented onto Zynq ZC702 Evaluation Board and the simulation is performed through the vivado 2019.1 software.

Keywords: AES, LUT, OPPRM, Field Programmable gate array, Zynq, Sub pipelining

1. Introduction

Cryptography plays a major role in healthcare industry by securing an electronic health records like insurance details, patient information, medicine dosage summary etc. Security and privacy are the vital factors need to be examined while forwarding the healthcare information. During the course of transmission there is a probability to crack or duplicate sensitive data, which drives to major destruction. Cryptographic algorithm is employed to perform secure communication and to resist against the security threats [1]. Encryption and decryptions are the two major operations involved in the cryptographic algorithm. Encryption is a process of converting the input sensitive information (i.e. plaintext) into an unreadable form (Cipher text) with aid of secret Key. Decryption is totally an inverse process of encryption in which the cipher text is transformed into a plaintext with the aid of cipher key assistance. Depending on the utilization of secret keys, the cryptographic algorithm is categorized as asymmetric algorithm and Symmetric algorithm. In symmetric key cryptography, the transmitter and the recipient works with identical secret keys in order to achieve encrypting and decrypting of information, therefore it also recognised as shared key cryptography or secret key cryptography. The predominantly utilized secret key algorithms are DES, Triple DES, RC2, AES, RC4, RC5, Two fish, and Blowfish, etc[2]. In asymmetric key cryptography the transmitter enciphers the data by means of senders public key and decrypts backs the data using receiver’s private key. The conventionally used algorithms are, RSA, DSA, Diffie-Hellman ECC, Elgamal, etc.

In present circumstances the cryptographic algorithms are extensively applied in many applications namely education, Healthcare, military, government, banking, commercial and network security etc [3]. AES algorithm introduced in 2001 as an alternative to DES[5]. The symmetric block cipher, AES algorithm, also referred to as the rijndael algorithm, provides data blocks of 128 bit and variable cipherkey size of 256,192, and128bits[6].AES algorithm carries out an iterative method instead of Feistel method, here it enacts equal number of rounds for encoding and decoding process. AES implementation could be practised in hardware as well as software. In comparison with software, the hardware realization delivers enormous speed and intense security. FPGA or ASIC board deployed for AES hardware implementation, the foremost advantage of FPGA platform is reconfigurable computing and flexibility, ASIC platform encounters a lack of flexibleness, therefore
by considering each and every attribute, the hardware implementation of AES on FPGA got all the consideration [7].

Nasir Siddiqui et al.[12] has developed a novel technique for building a computational S-boxes with aid of Galois field arithmetic on a basis of $P^7[2^2]$, the requirement for implementing the matrix on Galois field is to break the Galois structure, as result it is highly challenging for the attackers to produce other elements of S-box. In this novel method it is possible to generate $1.324\times10^{14}$ S-boxes with the nonlinearity of 112. To enhance a hybrid image encryption, a chaos based image encryption method is implemented. Abdelrahman Altigani et.al.[13] has proposed the polymorphic version of an AES algorithm (P-AES), the slight modification of key dependent is made in the substitute byte, shift rows and Mix Column transformations as a result a static nature of an AES algorithm has rebuild with polymorphic cipher. A uniform layer of obscurity is added to P-AES algorithm, which prevents the intruders from identifying the cipher information. Under avalanche effect the key obtains a value of 0.495 and the plaintext obtains 0.504 respectively. Sandhya Koteshwara et al.[14] has worked in various nonce-misuse resistant algorithm using Field Programmable gate arrays and ASIC environment. Serial implementation is highly designed for low area and low power consumption in AES-GCM architecture; it employs one AES module and one Galois Field Multiplier module. AES-GCM architecture, the nonce without deviation operated in CTR mode of AES algorithm [14].

Karim Shahbazi et al.[15] has suggested a Nano-AES accelerator using VHDL coding for both FPGA and ASIC platforms. Initially the proposed design is carried out in a FPGA (Virtex-5), followed by it, a few changes were made to a signals and control unit. By Utilizing 65-nm technology under different timings, this novel technique is synthesized in Synopsys Design Compiler. A clock Frequencies of 1MHZ and 100 KHZ is chosen based on RFID applications. Rash Reyhani-Masole[16] derived a new formulations and designs for new tower field. You can find various types of field constructions, such as composite field over GF($2^{4}$) and tower field upon GF ($((2^{2})^{2})^{2}$) employed for AES S-Box Construction, here the subfield components are described in Normal basis.

The structure of this article is set out below, section 2 gives a detailed explanation about Advanced Encryption Standard algorithm and the sub process involved. Section 3 includes the discussion on subpipeling architecture. Section 4 gives a brief description about Implementation of Composite field arithmetic S-box. Section 5 presents the proposed Optimized Positive Polarity Reed Muller architecture. The result and discussion of OPPRM architecture is presented in section6. Section 7 provides conclusion.

### 2. Method and Methodology

#### 2.1 AES

AES (Advanced Encryption standard) is also proclaimed as a rijndael, with the fixed chunk size of 128 bits. It is a Secret key algorithm in which transmitter and a recipient accept the identical Keys for encrypting as well as decrypting. Based on the size of cipher key, the rijndael algorithm is categorised into three divisions such as i) AES256 bit ii) AES192 bit and iii) AES128bit The Advanced Encryption Standard algorithm enacts in a two dimensional array of byte better Known as state array. Each and every row in a state matrix takes up Nb bytes, where Nb is equal to block length divided by 32. The block length indicates the size of an input, output and the state is of 128bits. It works with help of substitution and permutation, combination of network to attain a ciphertext. The Nr specifies round number, which is completely, reliance on key length specified as Nk as shown in Table1[8]. The cipher key length is specified as Nk ranges from 4, 6, and 8 which depict number of 32bit words or Number of Columns. AES Encryption carries out four sub process in each round namely i) Substitute byte ii) ShiftRows iii) MixColumn and iv) AddRoundkey, whereas in last round the mixcolumn step is excluded. Decryptions is just an inverse process of encryption in which reverse substitute byte, reverse Shift rows, reverse Mixcolumn and addround key were performed, anyhow in final round the reverse mixcolumn technique is not included.

<table>
<thead>
<tr>
<th>AES Cipher Key</th>
<th>Cipher Key Length [Nk words]</th>
<th>Round Number [Nr]</th>
<th>Size of a Block [Nb words]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES256bit-Cipher Key</td>
<td>8</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>AES192bit-Cipher Key</td>
<td>6</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>AES128bit-Cipher Key</td>
<td>4</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

The substitute Byte and inverse Substitute method is the most complicated subdivision in the AES algorithm. It is nonlinear process, which employs individually in a every byte of a state array by use of
substitution box(s-box). It is represented as (16x16) matrix of 256 bytes, the formation of S-box is based upon two vital steps upon GF(2^8). Initially multiplicative inverse, from there on affine transformation is carried out for Substitutebyte transformation. Inverse affine transformation and then multiplicative inverse process is performed for reverse substitute byte method. Result of a substitute Byte process is used to execute the shift row operation. The first row of a shiftrow step has no change in the input. The second row performs one left circular shift, the third row shifts two times to a left side and the fourth row moves three times to a left side of matrix respectively. Inverse shift row for decryption is executed same as encryption except the shifting operation is performed from right side of a row. The mixcolumn operation employs with output of a shift rows on a state matrix, looking upon each and every column as a four term polynomial. Considering each column on a state matrix as polynomial upon GF(2^8) and modulo x^4 +1 is multiplied. Mix column multiplies the row of constant matrix with the column of a state, here the state denotes the output of a shift row, which is an input to a mixcolumn. The Mixcolumn and inverse Mixcolumn process can be realized with the subsequent equation 1 and 2.

\[
\begin{bmatrix}
    s_0' \\
    s_1' \\
    s_2' \\
    s_3'
\end{bmatrix} = \begin{bmatrix}
    02 & 03 & 01 & 01 \\
    01 & 02 & 03 & 01 \\
    01 & 01 & 02 & 03 \\
    03 & 01 & 01 & 02
\end{bmatrix} \begin{bmatrix}
    s_0 \\
    s_1 \\
    s_2 \\
    s_3
\end{bmatrix}
\]  -----(1)

\[
\begin{bmatrix}
    s_0' \\
    s_1' \\
    s_2' \\
    s_3'
\end{bmatrix} = \begin{bmatrix}
    0e & 0b & 0d & 09 \\
    09 & 0e & 0b & 0d \\
    0d & 09 & 0e & 0b \\
    0b & 0d & 09 & 0e
\end{bmatrix} \begin{bmatrix}
    s_0 \\
    s_1 \\
    s_2 \\
    s_3
\end{bmatrix}
\]  -----(2)

The cipher key works with the aid of addroundKey. Column wise operation is performed in an addroundkey technique; here the input 128bit plain text is XORed with the keys generated during the course of Key generation architecture. The inverse AddRound Key technique is similar to the addround key technique due to fact that it performs XOR operation in inverse as well.

### 2.2 Key Expansion or Key Schedule

The AES Key expansion module is utilized to generate roundkeys, proceeding from the input cipherkey. The first round key employed during encryption process is made use of an original cipher key. subword is a function that substitutes each and every byte of a word by applying s-box method. The function rot word performs cyclic left shift with the input word \([w_1, w_2, w_3, w_4]\) and attains the output word \([w_2, w_3, w_4, w_1]\). The round constant(Rcon) is achieved by XORing the rotword and the subword. Rcon[i] consist of values \([x^{i-1}, {00}, {00}, {00}]\) along with \(x^{i-1}\) being the power of \(x\) (\(x\) implies \(\{02\}\)) upon Galois field \(2^8\) here the \(i\) value starts from 1 not 0. Rcon value differing in each and every round as it was depicted in the above table 2. The key expansion process generates totally \(4(N_r+1)\) words. The mathematical expression employed in the key expansion is entirely depends on XOR operation among preceding word and the word belonging to the preceding subkey. The key expansion process described in brief with aid of protocol listed below [8]

```
Keyexpansion (byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
Begin
  Word temp
  i = 0
  while (i < Nk)
    w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i = i+1
  end while
  i = Nk
  while (i < Nb * (Nr+1))
    temp = w[i-1]
    if (i mod Nk = 0)
      temp = SubWord (RotWord(temp)) xor Rcon[i/Nk]
    else if (Nk > 6 and i mod Nk = 4)
      temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] xor temp
    i = i + 1
  end while
end
```
2.3 Subpipelined Architecture

Sub pipelined architectural optimization attains high speed and large throughput when analysed with the pipelining and loop unrolling architecture or unfolded architecture. The pipelining architecture is carried out by introducing a row of registers among each and every round. Congruent to pipelining concept, in sub pipelining architecture the registers are included in between each stage of round and also within the round, in an attempt to preserve the intermediary result. The primary advantage of the sub pipelining architecture is that multiple blocks of information can be processed simultaneously. The sub pipelining architecture involved in Advanced Encryption Standard includes registers between each round and also inserts registers between AES sub-stages that imply substitute byte, shiftrows, Mixcolumns and addroundkey as illustrated in the figure 2. The delay obtains in the time of setup time, propagation delay within registers, delay occurs in a combinational logic circuit in each round, multiplexer delay are the total delay occurs in each sub stages are very small. Each and every substages inside the round are separated with equal delay, the registers inserted between the rounds, increase the area of an architecture. It was observed that the number of substages with even delay is increased for every round, due to which an upgrade in speed has been effectuated. But splitting each round into sub stages, no way enhance the speed because of the delay generated while inserting more number of registers.

2.4 Composite field arithmetic Substitute box

The substitute byte and inverse substitute byte operations were carried by two methods namely LUT based and the composite field based method. The implementation of composite field arithmetic is achieved with the aid of combinational logic circuits instead of predetermined Sbox values. The composite field arithmetic technique is operated to enhance speed as well throughput of an algorithm. The substitution of an s-box undergoes multiplicative inverse and affine transformation process. In inverse substitute byte, initially inverse affine transformation technique is applied soon after Multiplicative inverse operation is performed. CF is represented as GF ((2^m)ⁿ) is isomorphic over GF (2^k) where k =mn. The multiplicative inverse computing is performed by decomposing the higher order fields from GF (2^8) into lower order fields of GF(2) using the following irreducible polynomial equation(3)[9,19]

\begin{align}
\text{GF (2)} & \rightarrow \text{GF (2^2)} & P_0 (x) = x^2 + x + 1 \\
\text{GF (2^2)} & \rightarrow \text{GF ((2^2)^2)} & P_1 (x) = x^3 + x + \phi \\
\text{GF ((2^2)^2)} & \rightarrow \text{GF (((2^2)^2)^2)} & P_2 (x) = x^4 + x + \lambda
\end{align}

![Fig1.KeyExpansion protocol](image)

Table2. Round constant [Rcon] value for each single round

<table>
<thead>
<tr>
<th>Number of Round</th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
<th>r6</th>
<th>r7</th>
<th>r8</th>
<th>r9</th>
<th>r10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rcon Value</td>
<td>01</td>
<td>02</td>
<td>04</td>
<td>08</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td>1B</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
The value of $\phi$ and $\lambda$ were preferred as \{10\} and \{1100\} respectively. It is not possible to directly implement the process of multiplicative inverse GF ($2^5$). The isomorphic mapping $f(x) = \delta \times x$ is done to plot all the components in GF(2) and remapping the calculated results return back to the original field with aid of inverse isomorphic mapping function $f^{-1}(x)$. A $\delta$ matrix related to $p(x) = x^8 + x^4 + x^3 + x + 1$ and the polynomial equation (3) is expressed as.
By considering isomorphic mapping transformation, it is realised that all the computations in the AES algorithm are not appropriate to be implemented in a composite field. For the sub pipelining design, the Mixcolumn and the inverse Mixcolumn processes need to be refined. By utilizing the $\delta$ matrix defined in the equation (3) the standard matrix of a Mix column elements [02] and [03] are plotted to [5f] and [5e] of composite field. In reverse mix column Transformation [0d], [09], [0b], [0e] elements are plotted to [09], [75], [2a], [57] of composite field. Due to this improvement in the Mixcolumn/Inverse Mixcolumn the hardware complexity increases. Whereas in shift rows/Inverse Shift row operation carries out only by cyclical shifting doesn’t rely on Galalois field.

3. Implementation of the optimized Multistage PPRM Architecture

In the optimized PPRM construction, the substitute Byte and the Inverse Substitute byte blocks were framed with the few variations in the PPRM architecture. Due to this structure the hardware utilization is reduced by minimizing the logical operations initiated in the substitute byte and inverse substitute byte process. Following this shift row operation is performed by the left circular shift and the inverse shift row process is achieved by the right circular shift. Later Mix column procedure is performed with aid of standard matrix for encryption and decryption. Finally the output from mix column is XORed with addround key to acquire output as shown in the Figure 3. The enciphering and deciphering of optimized positive polarity reed Muller architecture is composed with the assistance of Composite field arithmetic in substitute byte Transformation. The S-Box consumes massive power in advanced Encryption Standard has been reasonably decreased by altering few elements of S-Box architecture into two level logic which in turn provides provision for Sub pipelining structure. In an attempt to decrease the power utilization, a substitute byte block it is subdivided into three sub sections namely i) Preinversion module ii) Inversion module and iii) Post-inversion module. These subsections were implemented with aid of AND and XOR gates. With three alternative key sizes of 256,192, and 128bits, the algorithm has a constant block size of 128 bit. It performs 14, 12 and 10 rounds depends on key size.

$$\begin{align*}
\delta &= \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1
\end{bmatrix}
\end{align*}$$
Fig3. Proposed AES Encryption algorithm with 128bit Cipher Key
3.1 OPPRM architecture

AES-OPPRM architecture is framed with help of composite field arithmetic architecture. Figure 4 depicts the composite field arithmetic-based substitute byte transformation. The Conventional PPRM architecture is divided into 3 stages in which Preinversion module denotes stage 1, Inversion module denotes stage 2 and finally postinversion module denotes stage 3. The signals prohibited in an inversion module is connected to delay element, were holdup time is close to an inversion module. OPPRM architecture is equivalent to the PPRM architecture, here the few alterations is made in the stage 3, and as a result power consumption is reduced, the circuit size is made compact when compared to the conventional PPRM, due to the fact that area is minimized.

![Diagram of OPPRM architecture](image)

Figure 4 AES Substitute byte transformation using Composite Field Arithmetic

3.3 Mathematical interpretation of PPRM Architecture

In stage 1, an input 8bit \((x_0 \cdots x_7)\) produces three different outputs namely \(a_0\), \(b_0\), \(c_0\) by utilizing AND and XOR array, each output carries 4bit value, as derived in the following equation (5) (6) (7)[19].

\[
\begin{align*}
    a_0 &= x_7 \text{ XOR } x_5 \text{ XOR } x_3 \text{ XOR } x_2 \text{ XOR } x_1 \\
    a_1 &= x_7 \text{ XOR } x_5 \text{ XOR } x_3 \text{ XOR } x_2 \\
    a_2 &= x_7 \text{ XOR } x_6 \text{ XOR } x_4 \text{ XOR } x_3 \text{ XOR } x_2 \text{ XOR } x_1 \\
    a_3 &= x_7 \text{ XOR } x_5 \\
\end{align*}
\]

---

8
A 4bit input of (a0-a3), (b0- b3), (c0-c3) of equation (5) (6) (7)[19] reserved as input to stage2 and produces 4bit output of (d0-d3) as it is depicted in the equation(8)[19]

\[
\begin{align*}
\text{b}_0 &= x_7 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_6 \oplus x_0 \\
\text{b}_1 &= x_7 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_6 \oplus x_4 \oplus x_1 \\
\text{b}_2 &= x_6 \\
\text{b}_3 &= x_5 \oplus x_6 \oplus x_2 \oplus x_1 \\
\text{c}_0 &= (x_1 \oplus x_0) \oplus (x_2 \oplus x_0) \oplus (x_3 \oplus x_0) \oplus (x_5 \oplus x_0) \oplus (x_7 \oplus x_0) \oplus (x_3 \oplus x_2) \oplus (x_4 \oplus x_0) \oplus (x_7 \oplus x_2) \oplus (x_3 \oplus x_0) \\
\text{c}_1 &= (x_2 \oplus x_1) \oplus (x_2 \oplus x_4) \oplus (x_3 \oplus x_2) \oplus (x_3 \oplus x_0) \oplus (x_3 \oplus x_6) \oplus (x_5 \oplus x_0) \oplus (x_5 \oplus x_6) \oplus (x_5 \oplus x_2) \oplus (x_5 \oplus x_3) \oplus (x_3 \oplus x_0) \oplus (x_7 \oplus x_3) \oplus (x_7 \oplus x_2) \oplus (x_3 \oplus x_2) \oplus (x_3 \oplus x_4) \oplus (x_4 \oplus x_3) \\
\text{c}_2 &= (x_6 \oplus x_1) \oplus (x_2 \oplus x_6) \oplus (x_3 \oplus x_4) \oplus (x_5 \oplus x_0) \oplus (x_5 \oplus x_6) \oplus (x_5 \oplus x_2) \oplus (x_5 \oplus x_3) \oplus (x_3 \oplus x_4) \oplus (x_5 \oplus x_4) \oplus (x_6 \oplus x_0) \oplus (x_6 \oplus x_3) \oplus (x_3 \oplus x_2) \oplus (x_7 \oplus x_3) \oplus (x_7 \oplus x_2) \oplus (x_3 \oplus x_2) \oplus (x_3 \oplus x_6) \oplus (x_6 \oplus x_0) \oplus (x_6 \oplus x_3) \\
\text{c}_3 &= (x_5 \oplus x_1) \oplus (x_7 \oplus x_1) \oplus (x_5 \oplus x_2) \oplus (x_5 \oplus x_6) \oplus (x_5 \oplus x_7) \oplus (x_5 \oplus x_4) \oplus (x_7 \oplus x_4) \oplus (x_5 \oplus x_0) \oplus (x_7 \oplus x_0) \oplus (x_5 \oplus x_1) \oplus (x_4 \oplus x_1) \oplus (x_3 \oplus x_2) \oplus (x_2 \oplus x_4) \oplus (x_4 \oplus x_6) \oplus (x_2 \oplus x_1) \oplus (x_2 \oplus x_6) \oplus (x_6 \oplus x_0) \\
\text{d}_0 &= (x_3 \oplus x_2) \oplus (x_3 \oplus x_1) \oplus (x_5 \oplus x_0) \oplus (x_3 \oplus x_2) \oplus (x_5 \oplus x_0) \oplus (x_3 \oplus x_1) \oplus (x_5 \oplus x_0) \\
\text{d}_1 &= (x_3 \oplus x_2) \oplus (x_3 \oplus x_1) \oplus (x_5 \oplus x_0) \oplus (x_3 \oplus x_2) \oplus (x_5 \oplus x_0) \oplus (x_3 \oplus x_1) \oplus (x_5 \oplus x_0) \\
\text{d}_2 &= (x_3 \oplus x_2) \oplus (x_3 \oplus x_1) \oplus (x_5 \oplus x_0) \oplus (x_3 \oplus x_2) \oplus (x_5 \oplus x_0) \oplus (x_3 \oplus x_1) \oplus (x_5 \oplus x_0) \\
\text{d}_3 &= (x_3 \oplus x_2) \oplus (x_3 \oplus x_1) \oplus (x_5 \oplus x_0) \oplus (x_3 \oplus x_2) \oplus (x_5 \oplus x_0) \oplus (x_3 \oplus x_1) \oplus (x_5 \oplus x_0) \\
\end{align*}
\]

An input of three 4bit values is given as an input to stage3 and produces a output of 8bit (y0- y7) with help of AND and XOR array as shown in the equation(9)[19]

\[
\begin{align*}
\text{y}_0 &= 1 \oplus (x_3 \oplus x_0) \oplus (x_2 \oplus x_1) \oplus (x_1 \oplus x_2) \oplus (x_0 \oplus x_3) \oplus (x_2 \oplus x_0) \oplus (x_3 \oplus x_2) \oplus (x_2 \oplus x_0) \oplus (x_2 \oplus x_0) \oplus (x_0 \oplus x_3) \oplus (x_2 \oplus x_0) \oplus (x_3 \oplus x_2) \\
\text{y}_1 &= 1 \oplus (x_3 \oplus x_0) \oplus (x_2 \oplus x_1) \oplus (x_1 \oplus x_2) \oplus (x_0 \oplus x_3) \oplus (x_2 \oplus x_0) \oplus (x_3 \oplus x_2) \oplus (x_2 \oplus x_0) \oplus (x_2 \oplus x_0) \oplus (x_0 \oplus x_3) \oplus (x_2 \oplus x_0) \oplus (x_3 \oplus x_2) \\
\end{align*}
\]
The mathematical derivation for the OPRM architecture; however, modifications were made only in a stage 3 of the architecture. In the Optimized Positivity Polarity Reed Muller Architecture, stage 1 and stage 2 are identical.

### 3.4 Mathematical interpretation of OPRM Architecture

In the Optimized Positivity Polarity Reed Muller Architecture, stage 1 and stage 2 are identical to PPRM architecture; however, modifications were made only in a stage 3 of the architecture. The mathematical derivation for \(y_0 \text{ to } y_7\) is expressed below in equation (10).

\[
y_0 = 1 \oplus ((a_0 \oplus b_1) \& (d_3 \oplus d_2)) \oplus ((d_1 \oplus d_0) \& (a_2 \oplus d_0) \oplus ((a_0 \oplus b_0) \& (d_2 \oplus d_1)) \oplus (b_1 \oplus b_2) \& (d_0 \oplus a)
\]

\[
y_1 = 1 \oplus ((a_0 \oplus b_2) \& (d_3 \oplus d_2)) \oplus ((d_2 \oplus d_1) \& (a_3 \oplus d_0) \oplus ((b_1 \oplus a_2) \& (d_1 \oplus d_3)) \oplus (d_3 \oplus a_3)
\]

\[
y_2 = ((d_2 \oplus d_0) \& (a_0 \oplus d_0)) \oplus ((a_0 \oplus b_1) \& (d_1 \oplus d_2) \oplus ((a_1 \oplus b_3) \& (d_0 \oplus d_1)) \oplus ((b_2 \oplus b_3) \& (d_0 \oplus b_0)) \oplus (b_1 \& d_2)
\]

\[
y_3 = ((b_0 \oplus b_3) \& (d_1 \oplus b_1)) \oplus (b_0 \& (d_2 \oplus d_0) \oplus (b_2 \& d_1))
\]

\[
y_4 = ((d_3 \oplus d_1) \& (a_1 \oplus d_1)) \oplus ((a_0 \oplus a_3) \& (d_0 \oplus d_3) \oplus ((d_1 \oplus d_0) \& (b_0 \oplus d_2)) \oplus (b_1 \& d_0)
\]

\[
y_5 = 1 \oplus ((a_1 \oplus a_1) \& (d_3 \oplus d_3) \oplus ((a_2 \oplus d_3) \& (d_3 \oplus d_2) \oplus ((b_2 \oplus a_3) \& (d_0 \oplus d_3)) \oplus (d_2 \& d_1))
\]

\[
y_6 = 1 \oplus ((a_3 \oplus a_2) \& (d_3 \oplus d_0)) \oplus ((a_2 \oplus a_1) \& (d_0 \oplus d_1) \oplus ((d_2 \oplus d_0) \& (a_3 \oplus a_0) \oplus ((a_1 \oplus a_0) \& (d_2)
\]

\[
y_7 = ((a_0 \oplus b_2) \& (d_3 \oplus d_2)) \oplus ((d_0 \oplus d_3) \& (a_3 \oplus d_1) \oplus ((a_3 \oplus b_2) \& (d_1 \oplus d_2)) \oplus ((d_0 \oplus d_2) \& (b_2 \oplus d_3)) \oplus ((a_3 \oplus b_1) \& d_2 \oplus (b_2 \& d_0))
\]
The architecture of the AES-OPPRM architecture is made with the slight changes in the stage 3 ($y_0$-$y_7$) of the PPRM architecture as depicted in the diagram 5(b). The PPRM architecture is constructed using 18 EXOR gates and 19 AND gates, however the OPPRM architecture is developed using 14 EXOR gates and 6 AND gates. Due to this conversion, the hardware utilization is reduced and the speed of the proposed architecture is enhanced by minimizing the operating frequency.

Figure 5 (a) Existing PPRM architecture (b) Proposed OPPRM architecture

4. Result and Discussion

The proposed Optimized PPRM construction is developed with Verilog Hardware Description Language (VHDL) and implemented onto Zynq ZC702 Evaluation Board. The simulation is performed through the vivado 2019.1 software. The logical optimization is initiated by reducing the gate count. The Proposed architecture requires only 14 EXOR Gate and 6 AND gates to implement the architecture and it is compared with the existing one as it is illustrated in the Table 3.
Table 3. Gate count comparison

<table>
<thead>
<tr>
<th>Architecture</th>
<th>EXOR Gate Utilized</th>
<th>AND Gate Utilized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing Positive polarity reed Muller (PPRM) architecture</td>
<td>18 Gates</td>
<td>19 Gates</td>
</tr>
<tr>
<td>Optimized Positive polarity reed Muller (OPPRM) architecture</td>
<td>14 Gates</td>
<td>6 Gates</td>
</tr>
</tbody>
</table>

The high level synthesis of proposed system in VHDL is carried out using Register Transfer Level (RTL) synthesis. The RTL is designed to be implemented in Zynq7020 Evaluation board. The RTL description of an optimized Positive Polarity Reed Muller Architecture includes preinversion module, inversion module and post inversion module as it is displayed in the figure 6. The Figure communicates the resources being utilized by the architecture.

Fig 6. RTL Schematic OPPRM Architecture

The post implementation Outcomes of OPPRM architecture is analysed with respect to Look UP Table (LUT), Slices, Input Output Pins (IO) and its Utilization is presented in the table 4. The slice has been reduced from 0.23% to 0.17% which results in area minimization as compared to the Conventional Lookup table method.

Table 4. Post Implementation result with the Existing architecture

<table>
<thead>
<tr>
<th>FPGA Performances</th>
<th>Total Available Resources</th>
<th>Occupied resources</th>
<th>% of Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice LUTS</td>
<td>53200</td>
<td>108</td>
<td>74</td>
</tr>
<tr>
<td>Slice</td>
<td>13300</td>
<td>30</td>
<td>22</td>
</tr>
<tr>
<td>LUT as Logic</td>
<td>53200</td>
<td>108</td>
<td>74</td>
</tr>
<tr>
<td>IO</td>
<td>200</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

The Net list has been created for OPPRM architecture module, when the VHDL code is synthesized and implemented successfully in Vivado is depicted in the Figure 7. The implementation stage proceeding in Vivado will place and interconnect all the flat net-list constructed in the synthesis stage. The connections are made among the sub modules like, adders, mux, flip flops that are developed. The implementation stage joins up with the physical components from the the logic constructed.
The synthesis design approach connects the gates and flip flops to generate a flat net list. The implementation phase undergoes 3 stages namely translate, map, place and route. The translate stage combines all the input net list to a logic design file[18]. Mapping subdivides the circuit into sublocks so that it can be easily adapted to an FPGA resources. It places and routes the components in the programmable logic. The Figure8 denotes synthesis and implementation stage completed successfully and consumes total power of 11.142 Watts to implement the architecture.

The Proposed OPPRM module is analysed with AES-CFA [11], AES-ECG [12] in terms of slice LUTS, Flipflops and Slices. Table5 represents the post implementation utilization summary with the existing architecture. LUT (Look-Up Table) is a small asynchronous SRAM which is utilized to execute a combinational logic, whereas FF (Flip-Flop) is a one-bit memory cell employed to store the data. The Number of LUT input is depends on the architecture. In 7 series FPGA it has 6 input LUTS, Xilinx Virtex 4 has 4 input LUT, Virtex 6 has 5/6 input LUT[17].
The Fig9 denotes the area comparison with the existing and proposed architecture as a result it achieves a reduced number of slice counts. Fig8 depicts the simulation results carried out for S-Box realization of an Encryption Process. Substitute Byte (S-Box) method is an irregular byte substitute method, which performs independently. S-Box and inverse S-Box are the most complicated stage in Encryption and Decryption process. The S-Box is developed using a two-stage process in which the multiplicative inverse of the finite field GF \((2^8)\) is first taken and later subjected to an affine transformation. In Substitute Byte transformation values are represented in a hexadecimal form. When \(\text{enc}_\text{dec}\) is zero, it represent encryption process is performed as shown in the Fig10.

**Table 5. Post Implementation utilization**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice LUTS</td>
<td>2156</td>
<td>13,189</td>
<td>74</td>
</tr>
<tr>
<td>Slices</td>
<td>1132</td>
<td>372</td>
<td>22</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>680</td>
<td>11,797</td>
<td>74</td>
</tr>
</tbody>
</table>
The powers Analysis were performed for the proposed and the existing architecture. Based on the utilization report obtained in post synthesis and post implementation of S-Box realization, the utilization of IO, LUT can be reduced to lower dynamic and static power utilization as it is shown in the table 6. The total on chip power achieved for the proposed architecture is 11.142W. The power consumption is decreased due to the gate reduction technique employed in the architecture. Totally 14 EXOR gates and 10 AND gates were utilized in the proposed circuit.

**Table 6. Power Analysis**

<table>
<thead>
<tr>
<th>Power Analysis</th>
<th>Utilization details</th>
<th>Power (W) [Existing]</th>
<th>Power (W) [Proposed]</th>
<th>Utilization in % [Existing]</th>
<th>Utilization in % [Proposed]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic power Consumption</td>
<td>signals</td>
<td>5.357</td>
<td>2.462</td>
<td>33</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Logic</td>
<td>6.342</td>
<td>3.332</td>
<td>40</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
<td>4.313</td>
<td>4.308</td>
<td>27</td>
<td>43</td>
</tr>
<tr>
<td>Static Power Consumption</td>
<td>Device static</td>
<td>1.039</td>
<td>1.039</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>Total on Chip Power</td>
<td></td>
<td><strong>17.051</strong></td>
<td><strong>11.142</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The inverse S-Box employed during decryption process is been utilized to retrieve the input Byte substituted through encryption technique. The inverse S-Box is developed by employing two stages, initially find inverse affine transformation and later apply Multiplicative inverse over finite field GF(2^8). In Substitutebyte transformation values are represented in a hexadecimal form. When enc_dec is one, it denotes a decryption process is performed as it is represented in the Fig 11. The inverse S-Box realization employed for inverse substitute Byte technique is depicted in the output screenshot.
Composite field Substitute-Box is implemented by utilizing combinational logic circuits which is an alternative to the predefined S-Box values. The SubstituteBox and Inverse SubstituteBox is combined to carry out enciphering and deciphering technique. The Fig12 depicts the S-Box realization for combining Substitute-Box and Inverse Substitute-Box here the Multiplicative inverse module is shared between the substitutebyte method and inverse substitutebyte method.

5. Conclusion

The main aim of this article is to design an optimized low power combinational logic based AES-SBOX Realization on FPGA. The proposed method is based upon combinational logic circuits to avoid unbreakable delay. The substitute byte and inverse substitute transformation of an AES algorithm is implemented by employing OPPRM architecture. By this approach the unwanted delay occurs during the Loo up table (LUT) method has been averted. Efficient subpipelining architecture is developed by utilizing substitute byte and inverse substitute byte module. The hardware utilization of a proposed architecture is significantly reduced to avoid unbreakable delay and latency. This novel architecture is designed and simulated using vivado 2019.1 software and carried out on to Zynq ZC702 Evaluation board. The post implementation results are compared with the existing architecture in view of Bonded IPADS, Number of slices, Slice LUTS, Static and Dynamic Power. The power consumption has been reduced when compared to an existing architecture.

Competing Interest
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

**Funding**

No funds, grants, or other support was received.

**Author Contribution**

E. S. SelvaPriya: Conceptualization, Methodology, Software, Data curation, Writing - Original draft, Visualization, Validation Formal analysis, Investigation. Dr. L. Suganthi: Investigation, Supervision, Writing - Review & Editing, Resources

**Availability of data and materials**

The data used for the findings of this study is available upon request from the corresponding author.

**Ethical Approval**

This declaration is not applicable.

**References**


