Abstract— This paper proposes two approaches for the implementation of AES-128 key expansion. In one approach, S-Box look-up table (LUT) is used for the SubWord transformation. In another approach S-Box values are calculated on-the-fly (OTF). Both implementations have data path of 32 bits. The implementations were targeted on 90nm CMOS technology using standard library cells. Both the designs were clocked at 250MHz achieving throughput of 8gbps. First design consumes 7215 number of gates while the second design requires only 4752 gates.

Index Terms— AES, Galois Field, Key Expansion, S-Box.

I. INTRODUCTION

Advanced Encryption Standard (AES) is a cryptographic algorithm which is accepted as FIPS standard by National Institute of Standards and Technology (NIST) in November 2001. It is a symmetric key block cipher. It can encrypt and decrypt 128 bit data with 128 or 192 or 256 bit key. AES has broad range of applications including smart cards, cellular phones, Automated Teller Machines (ATMs), Smart Phones, Digital Video Recorders, etc. AES can be implemented on hardware using various approaches depending on the need of application. There is always a trade-off between area occupied by hardware, speed of operation and power consumption. Many hardware implementations of AES have been proposed till date. An 8-bit systolic architecture for moderate data rate applications which achieves throughput of 41Mbps for encryption and 37Mbps for decryption has been proposed [2]. It uses only 236 and 280 slices for encryption and decryption respectively. A fully pipelined FPGA implementation for algorithm acceleration has been proposed [3]. It achieves throughput of 21.2Gbps for encryption and 16.6Gbps for decryption. Another fully subpipelined architecture with 6 sub stages in encryption and key expansion achieves very high throughput of 21.6Gbps and 43.71Gbps on Xilinx XCV1000e-8bg560 and XC2VP20-7fg676 devices, respectively [4]. A balanced hardware implementation is proposed considering several existing methods [5]. It achieves throughput of 2.33Gbps on Stratix II family of Stratix family for AES-128 encryption. In this paper, we are presenting two implementations of AES-128 key expansion. One implementation uses look-up table called as S-Box, containing beforehand calculated values of byte substitution transformation (SubByte) in AES. In AES Key Expansion procedure, this transformation is termed as ‘SubWord’. Other implementation uses combinational logic based on Galois Field arithmetic which calculates S-Box values on-the-fly. First method achieves better throughput but occupies more area than the second method.

Rest of the paper is organized as follows. Section II describes AES algorithm. Section III describes our implemented architectures and Section IV shows implementation results.

II. AES ALGORITHM

AES is an iterative cipher meaning that both encryption and decryption consist of multiple iterations of the same basic round functions, as shown in Fig. 1.

A. SubByte

This is a non-linear byte substitution transformation which operates independently on every byte of the state matrix. The AES state is a matrix of size 4*4. Initially, the input to AES, called as plaintext, is copied into state matrix. All subsequent round operations are performed on this data and results are stored in state matrix, which is given as input to next round. Number of rounds (N_r) depends on the key size. It is 10, 12, and 14 for 128, 192 and 256 bit key respectively. SubByte transformation performs following operations on a byte of data [1]:

1) Finding multiplicative inverse in Galois Field (2^8)
2) Applying Affine transformation to the result obtained in the first step

Figure 1: AES Encryption and Decryption
B. Shift Rows

In the ShiftRows transformation a cyclic shift with different constant offsets is applied to all rows except the first row of the state matrix.

C. MixColumn

The MixColumns transformation operates on the State column-by-column. The columns are considered as polynomials over $GF(2^8)$ and multiplied with a fixed polynomial modulo $x^8+1$.

D. AddRoundKey

AddRoundKey is an XOR transformation that adds a round key to the State, in each iteration. The keys are expanded by the Key Expansion Module.

E. AES Encryption and Decryption

AES Encryption begins with addition of initial key with plaintext followed by round function consisting above four transformations. Key expansion procedure generates round keys from the initial key which are added with round data in AddRoundKey transformation. Decryption process of AES basically performs the inverse of each transformation in encryption in reverse order. Last round of encryption does not contain MixColumn transformation and first round of decryption does not contain InverseMixColumn transformation.

F. AES Key Expansion Procedure

Key Expansion procedure generates a total of $N_r(N_c+1)$ words from an initial set of $N_h$ words [1]. For AES, $N_h = 4$ and $N_c$ depends on key size. For 128 bit key, $N_c=10$ and $N_c=4$. Hence, 44 words (each word containing 32 bits) are generated during key expansion. The first four words of expanded key are filled with cipher key. Every following word $w[i]$ is equal to the XOR of the previous word $w[i-1]$ and the word $N_c$ positions earlier $w[i-N_c]$. For words in positions that are a multiple of $N_c$, a transformation is applied to $w[i-1]$ prior to the XOR, followed by the XOR with a round constant, $RCon[i]$. This transformation consists of a cyclic shift of the bytes in a word $RotWord()$, followed by the application of $SubWord()$. $SubWord$ transformation substitutes the byte with value in the S-Box.

III. IMPLEMENTED ARCHITECTURES FOR KEY EXPANSION

Key expansion can be carried out in two ways. All round keys can be generated and stored in memory or they can be generated on-the-fly while encryption. The design proposed in [2] uses the same encryption/decryption architecture during key expansion thus completely eliminating the need of a separate hardware unit for key scheduling. In [3], key expansion is executed by purely combinational logic. It is done under the assumption that extended key calculation for consecutive round is faster than single round execution time. In [4], key expansion architecture is divided same as the number of existent substages in encryption unit. Hence, round key generation can be done simultaneously with encryption operation. In [7], two separate registers are used along with block RAM to store first and last round key. This arrangement achieves low latency. Implementation in [8] uses composite field arithmetic in normal bases for SubWord operation. Advantage of using normal bases is, it doesn’t require any hardware for implementation of square operation.

The SubWord operation involved in the key expansion procedure can be implemented as look up table (LUT), which is a traditional method [9]-[11]. But this approach puts a limitation on speed and also increases area. Hence, alternative way is to use combinational logic for on the fly calculation of S-Box values. This approach is exploited in [12]-[13] with 7 subpipelining stages. In this paper, we have presented implementation of Key Expansion procedure in AES using both of these approaches and compared their performance parameters. Fig. 2 shows the block level diagram for our implementation of Key Expansion. Data path is of 32 bits.

A. Key Expansion using LUT for S-Box

In this approach, S-Box look-up table is used for SubWord operation. The data path is of 32 bits and four instances of S-Box are required at a time, after every four clock cycles.

Initially, cipher key is loaded into four 32-bit registers $W_0$, $W_1$, $W_2$ and $W_3$. For generating $W_4$, $W_3$ is applied with transformations $RotWord()$, $SubWord()$ and then XORed with $RCon$ and this transformed word is XORed with $W_0$. Thus, in every clock cycle, a 32-bit word is generated by XOR of two 32-bit words. So, at the end of four clock cycles, we get 128 bit expanded key for a particular round. This key is used as input to the next round of key expansion. Total 40 clock cycles are required to generate all 10 round keys.

B. Key Expansion using on the fly S-Box calculation

This approach uses same architecture as shown in Fig. 2, only the method used for performing SubWord operation is different in this case. The combinational logic based on composite field arithmetic is used to calculate multiplicative inverse in $GF(2^8)$ which is the first step in calculation of S-Box. And later, the affine transformation is applied to this multiplicative inverse to obtain S-Box value. Derivation of this logic is explained in detail in [6]. Fig. 3 shows the blocks of the combinational logic. In this approach, we don’t require separate memory to store S-Box values. The blocks shown in Fig. 3 are implemented using Boolean equations [6] considering value of $\lambda = \{1100\}_2$ and $\phi = \{10\}_2$. 
\[ k = qλ \]

Let \( k = qλ \) where, \( λ = \{1100\} \)

k_0 = q_1

\[ \delta^{-1} × q = \begin{pmatrix} q_7 & q_6 & q_5 & q_4 & q_3 & q_2 & q_1 \\ q_6 & q_5 & q_4 & q_3 & q_2 & q_1 & q_0 \\ q_5 & q_4 & q_3 & q_2 & q_1 & q_0 & q_7 \\ q_4 & q_3 & q_2 & q_1 & q_0 & q_7 & q_6 \\ q_3 & q_2 & q_1 & q_0 & q_7 & q_6 & q_5 \\ q_2 & q_1 & q_0 & q_7 & q_6 & q_5 & q_4 \\ q_1 & q_0 & q_7 & q_6 & q_5 & q_4 & q_3 \\ q_0 & q_7 & q_6 & q_5 & q_4 & q_3 & q_2 \end{pmatrix} \]

Figure 5: Inverse Isomorphic Transformation

Affine transformation, as specified in [1], is applied to the multiplicative inverse to obtain S-Box value of a byte.

IV. IMPLEMENTATION RESULTS AND COMMENTS

The hardware is described using VHDL codes. It is synthesized and implemented on FPGA Virtex 4 device xcvlx40-12ff1148 using Xilinx ISE 8.1 for testing purpose. After verifying the results on FPGA the design is taken for ASIC implementation. It is synthesized using RTL compiler to generate a Verilog netlist file. Using this netlist file, final layout is generated with the Cadence SoC enclosure version 12. The designs are targeted for the implementation on 90nm CMOS technology using TSMC standard library cells. Both the designs are clocked with the clock period of 4ns achieving throughput of 8gbps. Both the designs generate all 10 round keys in 40 clock cycles. The throughput is calculated as follows:

\[ T_p = \frac{(B × F)}{N} \]

Where, \( T_p \) = Throughput (Mbps)
B: Number of bits generated
F: Clock frequency (MHz)
N: Number of clock cycles required to generate B Bits

The implemented hardware generates 128 bit round key in four clock cycles. Hence B = 128 and N = 4. Implementation results from the reports generated by encounter are shown in Table 1. As we can observe from Table 1, LUT based approach consumes more number of gates and acquires larger area than OTF approach. Hence, OTF S-Box calculation approach is highly preferable for the applications which have tight area constraints. Further, we can use subpipelining in the combinational logic for S-Box calculation to increase the throughput.
Table I: Implementation Results

<table>
<thead>
<tr>
<th>Parameter</th>
<th>1st Approach (LUT based S-box)</th>
<th>2nd Approach (On the fly S-box)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Frequency</td>
<td>250MHz</td>
<td>250MHz</td>
</tr>
<tr>
<td>Throughput</td>
<td>8Gbps</td>
<td>8Gbps</td>
</tr>
<tr>
<td>Gate count</td>
<td>7215</td>
<td>4752</td>
</tr>
<tr>
<td>Standard Cells</td>
<td>3590</td>
<td>1499</td>
</tr>
<tr>
<td>Core density</td>
<td>70%</td>
<td>70%</td>
</tr>
<tr>
<td>Core Area</td>
<td>18398.016 um²</td>
<td>15286.207um²</td>
</tr>
<tr>
<td>Chip Area</td>
<td>38347.814um²</td>
<td>26829.596um²</td>
</tr>
<tr>
<td>Power</td>
<td>2.229mW</td>
<td>2.849mW</td>
</tr>
<tr>
<td>CMOS Technology</td>
<td>90nm</td>
<td>90nm</td>
</tr>
</tbody>
</table>

V. CONCLUSION

This paper presented two design approaches for the implementation of AES-128 key expansion procedure. 1st approach uses LUT based S-Box and 2nd approach calculates S-Box values on the fly. The implementation results clearly show that OTF S-Box calculation approach is better in terms of area efficiency than the LUT based approach.

REFERENCES


Amruta U. Page is currently pursuing Masters (M.E.) in Signal Processing from Cummins College of Engg. for Women, Pune which is affiliated to the University of Pune. She has completed Bachelor’s in Electronics and Telecommunication in 2006 from the same institute. Her areas of interest for research are VLSI, FPGA and ASIC implementations, Signal Processing. (E-mail : amrutapage@gmail.com)

P. V. S. Shastry is currently working as Associate Professor in Cummins College of Engineering for Women, Pune.