

Fully pipelined implementations of AES with speeds exceeding 20 Gbits/s with S-boxes implemented using logic only

Chethan Ananth, Karthick Ramu, Department of ECE, George Mason University

Abstract—This paper discusses the FPGA implementation of AES for high throughput rates of over 20 Gbits/s. The implementation of S-boxes using pure logic instead of Look-Up tables to achieve these throughput rates is shown in the paper. An improvement in area utilization with the reduction of the number of BRAM’s required to implement the algorithm in an FPGA is observed.

Index Terms— S-box, AES, Logic, FPGA,

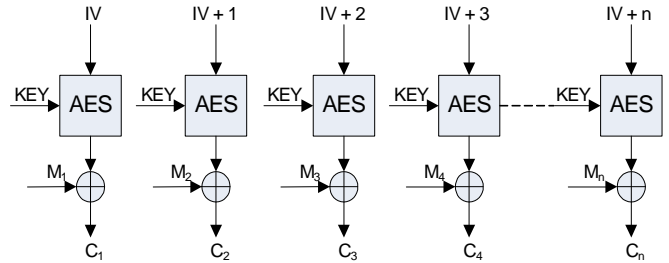


Fig. 1: AES CTR mode

I. INTRODUCTION

ADVANCED encryption standard (AES) is one of the symmetric block encryption algorithms used to encrypt data transmitted over a network. Implementations of AES at throughput rates of 10 Gbits/s have been developed previously in the Counter (CTR) mode. With the advancement in technology, the need for network data rates of 20 Gbits/s is practical. The basic idea behind this paper is to demonstrate the capabilities of AES in CTR mode for high throughput rates of over 20 Gbits/s achieved through the implementation of Subbytes using pure logic as against traditional Look-up tables. FPGA implementations of AES using Look-Up tables for S-boxes utilize a large number of BRAM’s. By implementing S-boxes using logic, the BRAM utilization can be reduced to almost zero.

II. DESIGN DESCRIPTION

A. Functional description of AES-CTR

AES encryption involves variable number of rounds of the basic operations depending on the key size. Each round consists of four basic operations – Subbytes, Shiftrows, Mixcolumns and AddRoundkey. Since our implementation is for a key size of 128 bits, encryption will consist of ten rounds of basic operations. The first step performed before these ten rounds is the XOR of the key with the initialization vector (IV). This is followed by 9 rounds of each of the basic operations in sequence. The tenth round contains all basic operations except

Mixcolumns. AES in counter mode involves the same circuitry for encryption and decryption since all computations including encryption and decryption are performed on the initialization vector (IV). The cipher text is obtained after the logical XOR of the message and the output of the last round. Thus this functional description is relevant to both encryption and decryption. The following pseudo code provides an overview of the entire operation.

```

AddRoundKey (State, RoundKey)
For I=1 to N-1
{
  Subbyte(State);
  ShiftRow(State);
  MixColumn(State);
  AddRoundKey(State,RoundKey);
}
Subbyte(State);
ShiftRow(State);
AddRoundKey(State,RoundKey);

```

1) Subbytes

Each byte in the 128-bit array is substituted with a corresponding byte derived by computing the multiplicative inverse over $GF(2^8)$ and performing an affine transformation. The Subbytes operation provides non-linearity to the cipher due to the multiplicative inverse operation. The affine transformation is performed to avoid attacks based on simple

algebraic properties on the cipher. Since this implementation of AES is in the CTR mode, the same Subbytes transformation can be applied for both encryption and decryption.

2) Shiftrows

This is a transposition step where each row of the state is shifted cyclically a certain number of steps. In every Shiftrows operation, the first row of the 128 bit input is left unchanged. Each byte of the second row is shifted once to the left. Similarly, the third and fourth rows are shifted by an offset of two and three respectively. The general pseudo code for Shiftrows is shown below.

```
ShiftRow(word8 a[4][MAXBC], word8 d, word8 BC)
{
    word8 tmp[MAXBC];
    for(i = 1; i < 4; i++)
    {
        for(j = 0; j < BC; j++)
            tmp[j] = a[i][(j + shifts[SC][i][d]) % BC];
        for(j = 0; j < BC; j++) a[i][j] = tmp[j];
    }
}
```

3) Mixcolumns

Mixcolumns is an operation on the columns of the state, where four bytes of each column are combined using an invertible function. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. MixColumns together with Shiftrows provides diffusion in the cipher. The general pseudo code for Mixcolumns is shown below.

```
MixColumn(word8 a[4][MAXBC], word8 BC)
{
    word8 b[4][MAXBC];
    int i, j;
    for(j = 0; j < BC; j++)
    for(i = 0; i < 4; i++)
        b[i][j] = mul(2, a[i][j])
        ^ mul(3, a[(i + 1) % 4][j])
        ^ a[(i + 2) % 4][j]
        ^ a[(i + 3) % 4][j];
    for(i = 0; i < 4; i++)
    for(j = 0; j < BC; j++) a[i][j] = b[i][j];
}
```

4) AddRoundkey

In the AddRoundKey operation, the subkey is combined with the state in every round. For each round, a subkey is derived from the main key using the AES key schedule where each subkey is 128 bits. The subkey is added by combining each byte of the state with the corresponding byte of the

subkey using bitwise XOR.

B. Subbytes using Logic only

SubBytes, as discussed earlier involve multiplicative inverse computation in Galois field $GF(2^8)$ and affine transformation. In most of the previous FPGA implementations of AES, SubBytes has been implemented using Look-Up tables where corresponding byte substitution values of every input byte are stored in memory and read during every round. This memory based implementation of SubBytes can induce an artificial restriction on the maximum clock frequency that can be achieved.

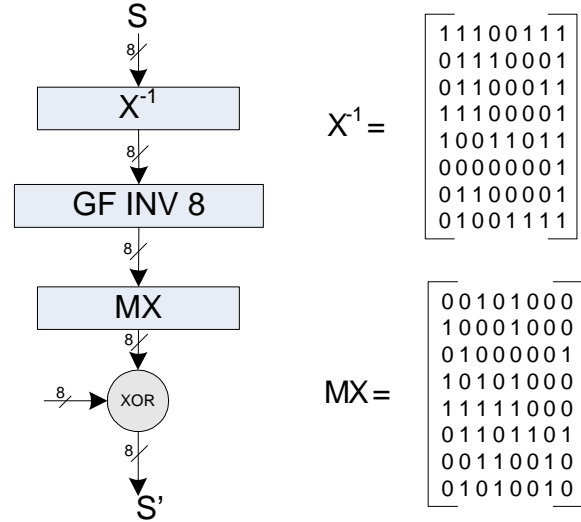


Fig. 2: Operation of S-box, matrix X^{-1} and matrix MX

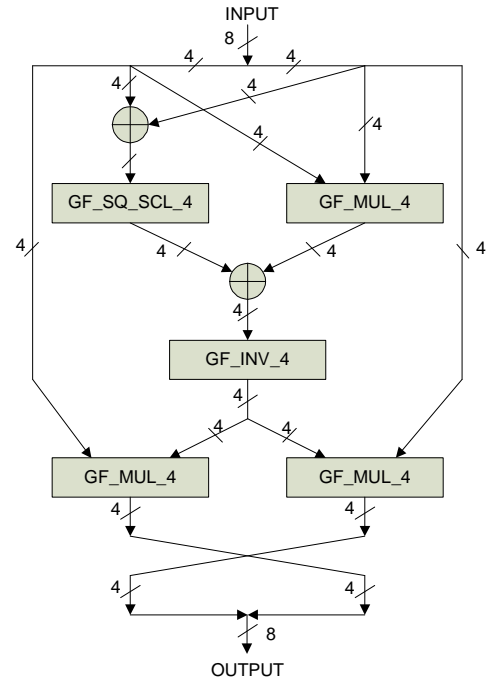


Fig. 3: Block Diagram of GF inverse 8

The datapath for subbytes computation (see Fig. 2) includes conversion from the standard polynomial representation to the internal representation, X^{-1} , inversion in $GF(2^8)$ (see Fig. 3), conversion back to the standard representation, X , combined with the multiplication by matrix M of affine transformation, MX , followed by the addition of the vector b of the affine transformation. Thus, the entire SubBytes transformation can be described by the equation

$$S' = (MX) \cdot (X^{-1}S)^{-1} + b$$

The basic idea of the logic-only implementation is that the inversion in $GF(2^8)$ (see Fig. 3) can be decomposed into a sequence of operations in $GF(2^4)$ (including addition, multiplication, and inversion). Similarly, operations in $GF(2^4)$ can be expressed in terms of operations in $GF(2^2)$, and operations in $GF(2^2)$ in terms of operations in $GF(2)$. The operations in $GF(2)$ can be implemented using simple XOR gate (addition) and AND gate (multiplication). An inverse of 1 in $GF(2)$ is 1, and the inverse of 0 does not exist. Thus, the entire inversion in $GF(2^8)$ can be decomposed into a logic circuit composed of XOR and AND gates only.

C. Pipelining:

As the main objective of this project was to achieve very high throughput AES encryption/decryption system, it was imperative for the design to be fully pipelined. The entire AES design was fully unrolled in order to implement a very deep level of pipelining. All 10 cipher rounds were unrolled as shown in the Fig. 4.

However, key scheduling in AES was not unrolled or pipelined as the throughput of the system does not depend on it. Instead it was decided to complete the key scheduling operation before starting the AES round operations and store the keys in memory which could then be accessed during the different AES rounds as shown below. It should also be noted that because of the above factor, key scheduling was not included to compute the throughput of the system.

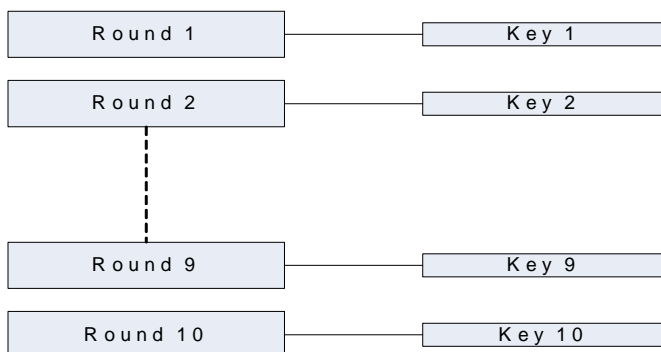


Fig 4 AES rounds unrolled

A design is said to have a balanced pipeline when the number of gate delays between pipeline stages are equal. Thus instead of adding pipeline registers between every basic operation and every round (Shiftrows, Subbytes, MixColumns, AddRoundkey) as shown in traditional pipelining methods, it

was decided to calculate the time delay for every basic operation and every round, flatten the entire design and apply the pipeline registers at appropriate locations to balance the time between each pipeline register.

As a first step towards pipelining, one round operation of the design was flattened and a pipeline register was applied to divide the total time required to complete one round operation into half and the throughput was measured. Furthermore, the design was divided into multiple slots of equal time periods and pipeline registers were introduced between every slot as shown in Fig. 5. This approach lead to a gradual increase in the throughput as the entire design was divided into chunks of smaller time periods.

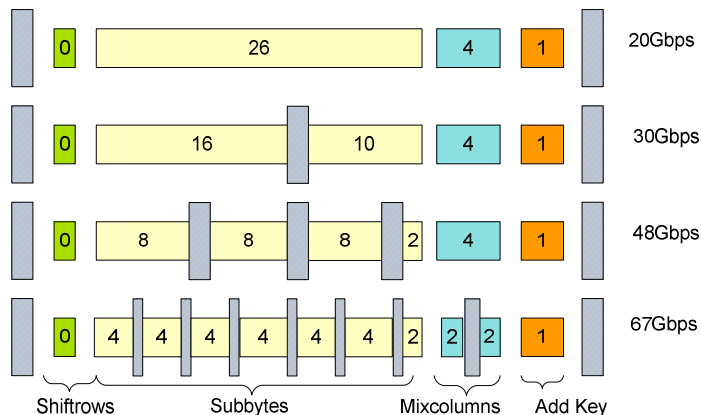


Fig. 5: AES rounds fully pipelined

Fig.5 shows measurements of the number of gate delays involved in MixColumns, Shiftrows, Subbytes and AddRoundkeys in a single round and how the pipeline registers are placed in the design to achieve high throughput. As it can be seen, the logic only implementation of Subbytes contains high gate delays and thus it was critical to pipeline Subbytes in a balanced manner.

III. RESULTS:

With the completion of the initial design, the entire design was pipelined in several stages as discussed to compare the different performance parameters like throughput, area and latency. The following results with respect to throughput and area were obtained from AES in ECB mode rather than in CTR mode due to certain issues with IV incrementing in CTR mode as detailed in the current issues section.

The following table shows the results obtained for different pipeline stages. All implementations were done on the Xilinx high-performance Virtex 5 FPGA device which is the latest of the virtex family of devices. The device used was XC5VLX110t-2ff1136 which has a speed grade of -2.

Pipeline stages	Throughput (Gbps)	Area (slice)	BRAM (%)	Latency (ns)
1	20	8,800	0	6.25
2	27	9,086	0	4.67
4	48	10,561	0	2.777
8	67	13,649	0	1.905

A. Throughput Vs Pipeline Stages.

It is a well known fact that the throughput of the system increases as the number of pipeline stages increase. The following graph shows the increase in the throughput of AES round process (excluding key scheduling) with increase in the number of pipeline stages (Fig. 5).

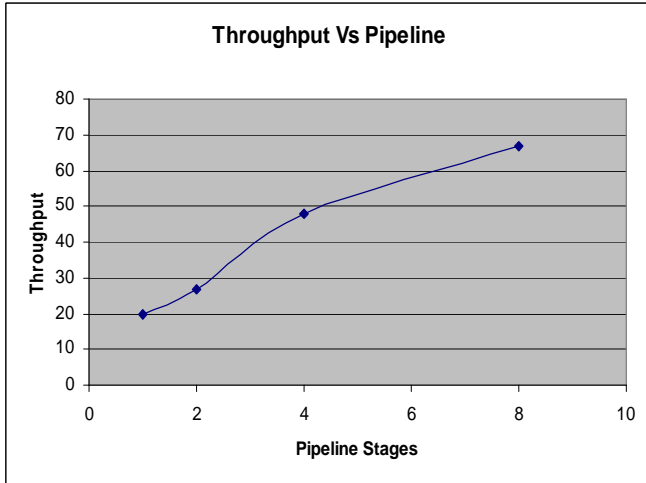


Fig. 5: Throughput Vs Pipeline stages

B. Area Vs Pipeline Stages

The graph below shows the increase in area of the design with the increase in the number of pipeline stages. It should be noted that the increase in area is also balanced by replacing S-box look-up table which occupies a large number of BRAM's.

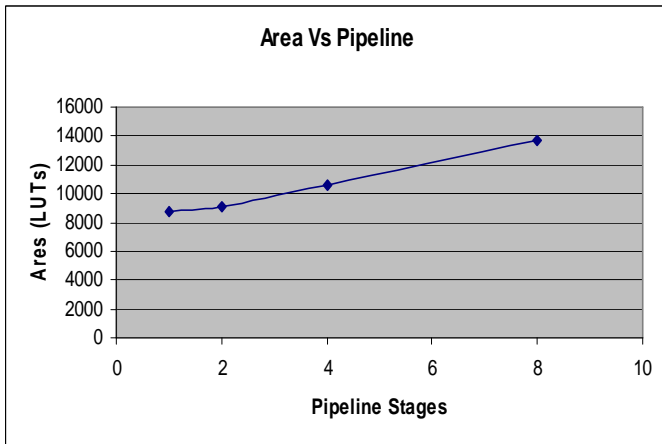


Fig. 6 Area Vs Pipeline Stages

IV. CURRENT CHALLENGES:

Even though high throughput rates for AES in ECB mode were achieved, similar throughput rates have not been achieved in counter mode (CTR) since counter mode requires

a 128 bit IV increment operation for every 128 bit block of plaintext. This increment needs lower clock frequency than the rest of the operations in AES and acts as a show stopper for implementing AES in CTR mode with high throughput. However, this aspect of the design needs to be worked and a dedicated fast-adder will be implemented to increment the IV in every clock cycle.

V. FUTURE WORK:

The traditional IV increment operation in AES Counter mode (CTR) needs to be implemented using high performance pipelined adders like Brent-Kung, Kogge stone to meet high throughput requirements in CTR mode.

Compare and analyze the performance of AES in all modes by replacing traditional LUT based S-box with logic only SubBytes implementation.

VI. CONCLUSION:

A fully pipelined AES encryption/decryption system built with logic only Subbyte instead of table lookup s-box was developed to achieve very high throughput rate. It has been observed that by pipelining the entire design in a balanced manner, throughput rates of more than 20 Gbits/s can be easily achieved. Our results show that by adding pipeline registers for every 4 gate delays in a balanced manner, a throughput of more than 70 Gbps could be achieved.

It should also be noted from the results that AES with logic only implementation of Subbytes uses absolutely zero BRAMs in traditional FPGAs which adds up as the biggest advantage of this design approach. Replacing look up table type S-box with logic only S-box gives enough space and chip area to unroll all 10 rounds of the AES encryption process to achieve high throughput with almost the same area utilization.

REFERENCES

- [1] FPGA and ASIC Implementation of AES by Kris Gaj and Pawel Chodowiec. http://teal.gmu.edu/courses/ECE746/project/S08_Project_resources/AES.pdf
- [2] FPGA and ASIC implementation of AES, Section 6, Implementation of basic operation of AES in hardware. http://teal.gmu.edu/courses/ECE746/project/S08_Project_resources/Canright_NPS_report.pdf
- [3] Using AES in Counter Mode. <http://www.ietf.org/rfc/rfc3686.txt>
- [4] D.Canright, "A very compact S-box for AES," CHES 2005. http://teal.gmu.edu/courses/ECE746/project/S08_Project_resources/Canright_CHES.pdf
- [5] Counter Mode Security Analysis <http://www.mindspring.com/~dmcgrew/ctr-security.pdf>