ISSN (e): 2250 – 3005 || Volume, 06 || Issue, 03||March – 2016 ||
International Journal of Computational Engineering Research (IJCER)

An Optimized Parallel Mixcolumn and Subbytes design in Lightweight Advanced Encryption Standard

Mary James[1], Deepa S Kumar[2]
[1] P.G Scholar,
[2] Associate Professor,

ABSTRACT

This paper presents a high speed, FPGA implementation of AES Encryption and Decryption (acronym for Advance Encryption Standard, also known as Rijndael Algorithm) in which the different steps of AES is applied in a parallel manner. This implementation can reduce the latency in which the process of implementation is reduced in a drastic manner. The paper deals with a comparison of the normal implementation of steps of AES and the parallel implementation. Inorder to increase the throughput of the AES encryption process the latency of the AES process should be reduced. Among Add Round Key, Sub Bytes, Shift Rows and Mix Columns, Sub Bytes and Mix Columns produce more latency. The execution delay of Mix Columns results in 60 percent of the total latency. Therefore Parallel Mix Columns is used inorder to reduce the latency. In this the block computes one column at a time such that the four columns are executed at the same time rather than each byte executing at a time. In Parallel Sub Bytes, four columns are executed at the same time rather than each byte executing at a time, this reduces the latency. Encryption is the process of encoding information so it cannot be read by hackers. The information is encrypted using algorithms and is converted into unreadable form, called cipher text. The authorized person will decode the information using decryption algorithms. The cryptography algorithms are of three types -symmetric cryptography (using 1 key for encryption/decryption), asymmetric cryptography (using 2 different keys for encryption/decryption), and cryptographic hash functions using no keys (the key is not a separate input but is mixed with the data).

Keywords: *AES ,Lightweight Cryptography,Parallel Mixcolumn, Parallel Subbytes, Key expansion, Shift rows, FPGA.*

I. Introduction

The CPU cycles needed for symmetric encryption are fewer than for asymmetric encryption. So symmetric algorithms are faster than asymmetric algorithms. Advanced Encryption Standard (AES), Data Encryption Standard (DES), Triple DES, Rivest Cipher (RC2), Rivest Cipher (RC6), and Blowfish are some of the symmetric algorithms. Remote Secure Access is an asymmetric algorithm.The AES algorithm is very difficult to crack and is well suitable to security service applications. It is designed in a way that has better resistance against existing attacks. AES has more elegant mathematical formulas behind it and requires only one pass to encrypt data. It has very low memory requirements, so it is particularly well-suited to embedded applications such as smart cards.

The key size can either be 128 bit, 192 bit, or 256 bit. The actual key size depends on the desired security level. The different versions are most often denoted as AES-128, AES-192 or AES-256.The AES algorithm Add Round Key can be cut in three blocks:

In the Addition of Round Key transformation, a Round Key is added to the State by a simple bitwise XOR operation.

Initial Round: It is the first and simplest of the stages, it only counts one operation of Add Round Key.

N Rounds: N being the number of iterations. This number varies according to the size of the key used. (128 bits need N=9, 192 bits need N=11 need 256 bits. N =13). This second stage is constituted of N iterations including each the four following operations: Sub Bytes, Shift Rows, Mix Columns, Add Round Key.

Final Round: This stage is nearly identical to one of the N iterations of the second stage. The only difference is that it doesn't include the operation Mix Columns

Sub Byte Transformation: In AES algorithm the function of the sub byte is only nonlinear function and that operates independently on each byte of the state using a substitution table (Sbox). It substitutes all bytes of the state array using a LUT which is a 16x16 matrix of bytes, often called S-box.

Shift Row Transformation: In shift rows transformation the last three rows of the State are cyclically shifted over different numbers of bytes in this process the row 0 is not shifted, row1 is shifted one byte to the left, row 2 is shifted two bytes to the left and row 3 is shifted three bytes to the left.

Mix Columns Transformation: This transformation is based on Galois Field multiplication. Each byte of a column is replaced with another value that is a function of all four bytes in the given column. The Mix Columns transformation operates on the State column-by-column, treating each column as a four term polynomial.

II. Design Methodology

2.1 Mix Columns

During the Mix Columns process, each column of the state array is considered as a polynomial. Multiplication using a predefined Matrix is carried out and output is obtained. There are many different architectures of mix column. The method used here is based on the binary calculation. Binary calculation is used because of its easiness in operation.

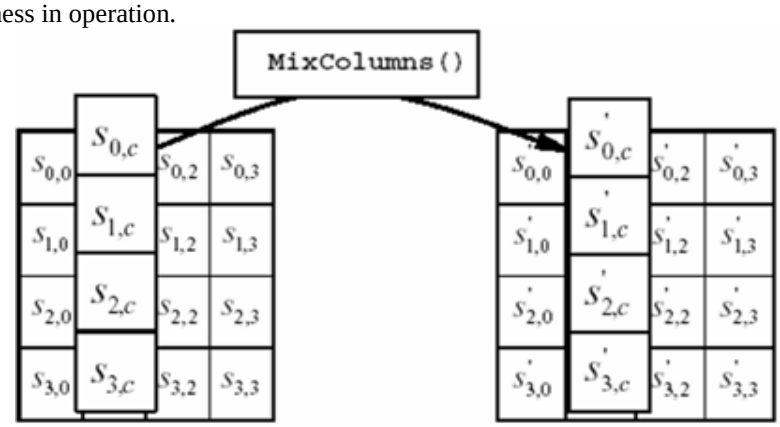

Fig 3.1 Mixing of columns of the state

Each byte of a column is replaced with another value that is a function of all four bytes in the given column. The Mix Columns transformation operates on the State column-by-column. In mix column, it is to multiply a fixed matrix called Mix Column matrix with the output of the shift row and we obtain the result is as shown above. After `Mix Columns`, the 1st column of `State` is turned into the 1st column of the `Mixed` matrix. Multiplying a column by the whole matrix is performed initially. The method of multiplying the matrix is shown in Fig 3.2.

$$\begin{bmatrix} S'_{0,c} \\ S'_{0,c} \\ S'_{0,c} \\ S'_{0,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{0,c} \\ S_{0,c} \\ S_{0,c} \end{bmatrix}$$

Fig 3.2 Multiplication Matrix

2.2 Parallel Mix Columns

The above Mix Columns transformation is known as Mix Columns-16, since each Mix Columns-16 operates on a four column data block, and the operation on each column is independent. Therefore, each Mix Columns-16 can be replaced by four Mix Columns-4s and Parallel Mix Columns is shown in Fig 3.3. Each Mix Columns-4 computes only one column at a time rather than a whole data block. . In a single loop, the execution delay of MixColumns-16 results in 60 percent of the total latency. As a result, the throughput of the Parallel-Mix Columns implementation is increased.

2.3 Sub Bytes

The Sub Bytes operation is a nonlinear byte substitution. Each byte from the input state is replaced by another byte according to the substitution box also called the S-box. The S-box is computed based on a multiplicative inverse in the finite field and a bitwise affine transformation. The S-box used in the Sub Bytes transformation is presented in hexadecimal form. Data in the binary format is converted to hexadecimal format and is substituted by some predefined values from a substitution box table. Left most 4 bits of the binary data are used as a row value. Rightmost 4 bits are used as column value. For example hexadecimal value {95} refers to row 9, column 5, the S-box, which contains the value {2A}.

2.4 Parallel Sub Bytes

In the Parallel Sub Bytes implementation, SubBytes-16 contributes the largest execution delay in one loop. In order to increase the throughput further, we parallelize one SubBytes-16 into four SubBytes-4s. In this implementation, each SubBytes-4 processes 4 bytes rather than 16 bytes in one data block. The structure of Parallel Sub Bytes is shown in Fig 3.4 below. The effective execution delay of the Sub Bytes process will be decreased. Therefore, the throughput of the Parallel Sub Bytes model will be increased.

2.5 Shift Rows

In the Shift Rows transformation, the first row of the state array remains unchanged. The bytes in the second, third, and forth rows are cyclically shifted by one, two, and three bytes to the left.This has the effect of moving bytes to "lower" positions in the row, while the "lowest" bytes wrap around into the "top" of the row.

2.6 Add Round Key

The Add Round Key operation is a simple EXOR operation between the State and the Round Key. The Round Key is derived from the Cipher key by means of the key schedule. The State and Round Key are of the same size and to obtain the next State an EXOR operation is done each element.

2.7 Encryption Process

AES encryption goes through 10 rounds of encryption process. In the first 9 rounds the input data is mapped into Sub Bytes, Shift Rows and Mix Columns, and in the 10^{th} round the Mix Columns operation is not included then Add Round Keys is performed.

III. Performance Comparison

From the above table 4.1, the throughput of parallelized blocks are increased and their delay are decreased where as the area is found to be increasing after parallelism.

Table 4.1 Comparison of throughput ,delay and area.

Technique Used	Throughput (Gbps)	Delay (ns)	Area (no. of slices)	Power (mW)
Mix Columns	49.478	2.587	256	385
Parallel Mix Columns	105.26	1.216	800	389
Sub Bytes LUT	90.52	1.414	257	396
Parallel Sub Bytes	126	1.013	512	409

IV.CONCLUSION

The AES which is presented is used to protect the information. To increase the efficiency, the parallelism technique is used. The proposed design requires less area when compared to the previous other techniques and throughput is increased to about percent and power is also optimized using this technique.

References

[1] J. Daemen and V. Rijmen, "The rijndael block cipher: Aes proposal," in First Candidate Conference (AES1), pp. 343–348, 1999.
[2] S. William and W. Stallings, Cryptography and Network Security, 4/E. Pearson Education India, 2006.
[3] A. M. Deshpande, M. S. Deshpande, and D. N. Kayatanavar, "Fpga implementation of aes encryption and decryption," in Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on, pp. 1–6, IEEE, 2009.
[4] P. B. Ghewari, J. Patil, and A. B. Chougule, "Efficient hardware design and implementation of aes cryptosystem," International journal of engineering science and technology,vol. 2, no. 3, pp. 213–219, 2010.
[5] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An fpga-based performance evaluation of the aes block cipher candidate algorithm finalists," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 9, no. 4, pp. 545–557, 2001.
[6] O. S. Gomes, R. L. Moreno, and T. C. Pimenta, "A fast cryptography pipelined hardware developed in fpga with vhdl," in Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2011 3rd International Congress on, pp. 1–6, IEEE, 2011.
[7] T. Hoang et al., "An efficient fpga implementation of the advanced encryption standard algorithm," in Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on, pp. 1–4, IEEE, 2012.
[8] P. N. Khose and V. G. Raut, "Implementation of aes algorithm on fpga for low area consumption," in Pervasive Computing (ICPC), 2015 International Conference on, pp. 1–4, IEEE, 2015.
[9] J. Tay, M. Wong, and I. Hijazin, "Compact and low power aes block cipher using lightweight key expansion mechanism and optimal number of s-boxes," in Intelligent Signal Processing and Communication Systems (ISPACS), 2014 International Symposium on, pp. 108–114, IEEE, 2014.