# Improved Mix Column Computation of Cryptographic AES

Aaron Barrera, Chu-Wen Cheng, Dr. Sanjeev Kumar
Department of Electrical and Computer Engineering
The University of Texas Rio Grande Valley
Edinburg, TX, USA
Correspondence Email: sj.kumar@utrgv.edu

*Abstract*— **With today's development and expansion of networks and internet-connected devices, information security is an issue of increasing concern. Confidentiality is one of the focuses in network security for digital communication systems, where large data blocks go through a cryptographic algorithm with a cipher key that increases the security and complexity of the output ciphertext. For the past several years, multiple security algorithms have been developed and utilized in the data encryption process, such as the Data Encryption Standard (DES), Triple Data Encryption Standard (3DES), and the current one, designated by the U.S. National Institute of Standards and Technology (NIST), the Advanced Encryption Standard (AES). AES is a symmetric encryption algorithm that has a minimum input data block size of 128-bits which undergo a series of permutations, substitutions, and digital logic operations over several rounds. Encryption algorithms are always improving on ciphertext complexity, required hardware storage allocation, and execution time. Field Programmable Gate Arrays (FPGA's) are a hardware alternative for encryption algorithm implementation because, although the logic units in it are fixed, the functions and interconnections between them are based on the user's design which allow for improvement. The research presented focuses on the development and analysis of an efficient AES-128 Mix Columns algorithm implementation, utilized in the data block encryption rounds, on an Altera Cyclone IV FPGA using the Intel Quartus II software and Verilog Hardware Description Language.**

***Index Terms*** **— Cryptography, Encryption, Advanced Encryption Standard Algorithm (AES), Rijndael, FPGA, Cyclone IV, VHDL, Intel Quartus, Cyber Security, Parallelization**

## I. INTRODUCTION

Many hardware crypto implementations have been done previously using the Altera Cyclone FPGA series [1-4]. The data block encryption computation, shown in Fig.1, used to obtain the Rijndael ciphertext [5-6] consists of:

A. Initial 'Add Round Key' step to add obscurity
B. 9 rounds of 4 steps to add confusion, diffusion, and non-linearity:

1. Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the data block.
2. Shift Rows: A simple bitwise permutation
3. Mix Columns: A substitution that makes use of polynomial arithmetic over GF ($2^8$)
4. Add Round Key: A bitwise XOR of the current data block with a portion of the expanded key

C. A Final 10th/12th/14th step of Substitute bytes, Shift Rows, and Add Round Key to add obscurity.

### Substitute Bytes

In this layer, each byte in the state will be substituted by values obtained from substitution boxes. This is done to achieve more security according to diffusion-confusion Shannon's principles for cryptographic algorithms design.

### Shift Rows

This layer is to provide diffusion for all the state. It contains two sub-layers to ensure the high-degree diffusion after transformation for many rounds.

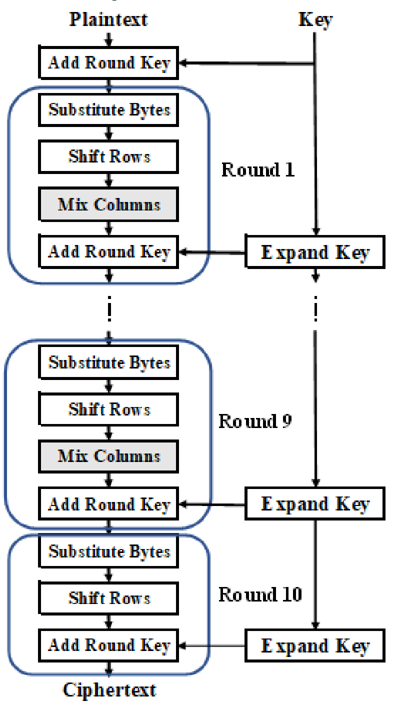

**Fig.1: AES-128 round computation.**

$$\begin{bmatrix} S\_(0,0) & S\_(0,1) & S\_(0,2) & S\_(0,3) \\ S\_(1,0) & S\_(1,1) & S\_(1,2) & S\_(1,3) \\ S\_(2,0) & S\_(2,1) & S\_(2,2) & S\_(2,3) \\ S\_(3,0) & S\_(3,1) & S\_(3,2) & S\_(3,3) \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} S'\_(0,0) & S'\_(0,1) & S'\_(0,2) & S'\_(0,3) \\ S'\_(1,0) & S'\_(1,1) & S'\_(1,2) & S'\_(1,3) \\ S'\_(2,0) & S'\_(2,1) & S'\_(2,2) & S'\_(2,3) \\ S'\_(3,0) & S'\_(3,1) & S'\_(3,2) & S'\_(3,3) \end{bmatrix}$$

**Fig.2: Mix Column matrix multiplication.**

Mix Columns

The forward mix column transformation, called Mix Columns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on State, shown as Fig.2.

This layer is to provide diffusion for all the state. It contains two sub-layers to ensure the high-degree diffusion after transformation for many rounds.

Add Round Key

In this layer, the operation is to conduct XOR operation on round key (round key is obtained from the extension of secret key operation) and state. This layer is to establish the relationship between the key and the cipher-text more complicated and to satisfy the confusion principle.

Key expansion of AES-128

The AES key extension algorithm takes a four-word (16-byte) key as input and produces a linear array of 44 words (176 bytes). This is enough to provide a four-character round key for the initial Add Round Key phase and each field, 10 rounds of password.

## II. Rijndael Mix Columns Computation

Intensive computation of AES takes place in the Rijndael Mix Column segment. The Mix Column transformation operates on each column of the 4-byte by 4-byte matrix formed from the input 128-bit data block. Each byte of the column is mapped into a new value that is a function of all four bytes in that column. The implementation of Mix Columns is based on the mathematical analysis in the Galois field, GF ($2^8$). For the AES algorithm this irreducible polynomial is:

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad \text{(Eq.1)}$$

The columns of the matrix are multiplied by modulo $x^4 + 1$ with a fixed polynomial c (x), given by:

$$c(x) = [03]x^3 + [01]x^2 + [01]x + [02] \quad \text{(Eq.2)}$$

This polynomial is coprime to $x^4 + 1$ and therefore invertible. Only the multiplication module and the 32-bit XOR module of each processing unit (one column) are needed for the design because the elements of the multiplication and addition in the Galois field are commutative and associative. In Fig.3 shows a Mix Columns computation example. With this approach, the function of Mix Columns can be achieved.

For example:

1. {02.BF}
{BF} . {02} = 1011 1111 << 1
=0111 1110 ⊕ 0001 1011
=0110 0101

2. {03.5D}
{5D} . {03} = {0101 1101 . 02} ⊕ { 0101 1101}
= 1011 1010 ⊕ 0101 1101
= 1110 0111

∴ r1 = {01.D4} + {02.BF} + {03.5D} + {01.30}
= (1101 0100) ⊕ (0110 0101) ⊕ (1110 0111) ⊕ (0011 0000)
= 0110 0110
= $(66)_{16}$

**Fig.3: Rijndael Mix Columns computation example.**

Many researchers have published comparative analysis and optimized the speed of hardware implementations on the performance of different modes of operation. The results, from [7], reveal that the scheme has low hardware resource consumption, high throughput and an excellent overall performance ratio. Here, we focus on the development and analysis of an efficient AES-128 Mix Columns algorithm implementation in two different approaches.

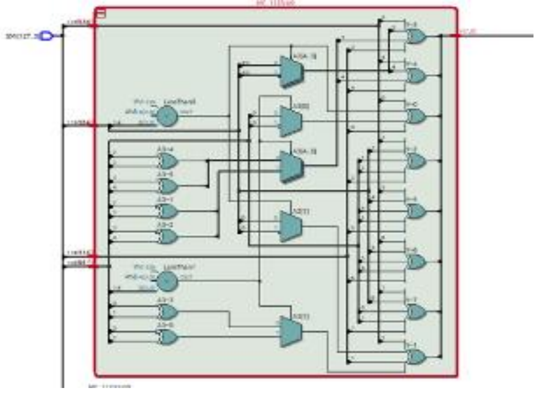

**Fig.4: internal schematic of each submodule.**

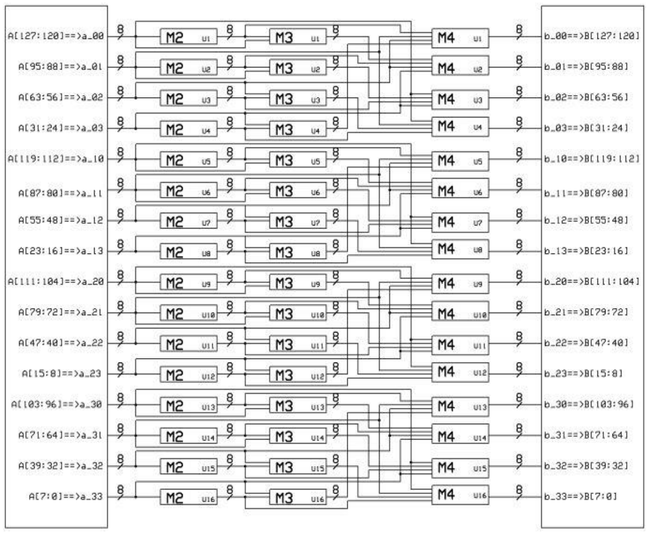

**Fig.5: Module block diagram for parallelized configuration approach.**

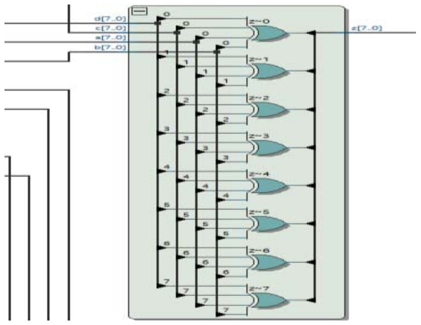

Fig.6: Internal schematic of M4 submodule.

Our first approach involved building the circuit modules around the traditional row-column multiplication method. This was done by creating codes for each row of the Rijndael mix columns matrix producing a total of four distinct sub-modules (shown above in Fig.4). Each module would take 4 bytes (a column of the incoming data block matrix) and produce a single corresponding byte of the output matrix. This approach resulted in a circuit configuration that was simplistic in concept, however based on Fig.5, the circuit reveals only partial bit parallelization which could potentially deter the performance of the circuit because of misaligned clock cycles.

Our second approach focused on forcing a more parallel behavior into the circuit to align the input signals together potentially resulting in decreased delay. As seen in Fig.6, three distinct sub-modules were created that separate the Rijndael multiplication with factors 2 and 3 and adds an additional module 'M4' as a 4-input XOR function. Each of the 16 bytes of the input data block matrix will go through the same sub-modules, aligning them during their clock cycles, and producing the 16-byte output matrix at approximately the same cycle.

## III. Experimental Setup

### AES Design Assessment

We use the EDA tools available on the Altera website to evaluate our designs. These tools, the Quartus II Web Edition and the Altera University Program Simulator [8], allow code to be built, compiled, synthesized, simulated, and finally programmed into DE2 hardware. In this work, we use Altera's Cyclone IV DE2-115 board EP4CE115F29 platform. Cyclone IV technology was released in 2017 [9]. The model EP4CE115F29 has a density of 114,480 LE and it contains an internal 50 MHz clock [10]. The development board is available on the Terasic website [11].

## IV. Results

The following data presented is modeled after previous performance comparison attempts mentioned in [1-4]. According to the AES algorithm we mentioned in Section 2, we can distinguish Rijndael Mix Column into many operations or functions. In addition, we used the Quartus II software to perform the timing simulations for each operation or function in each step, as shown in Fig.7~11. Finally, we developed the following tables to compare the performance.

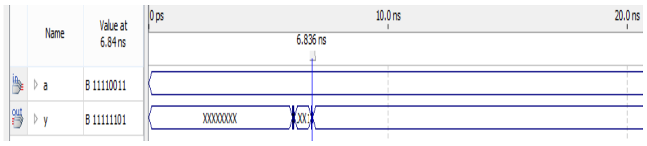

Fig.7: Delay of M2 submodule.

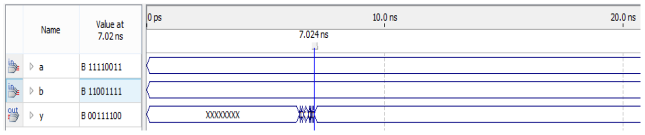

Fig.8: Delay of M3 submodule.

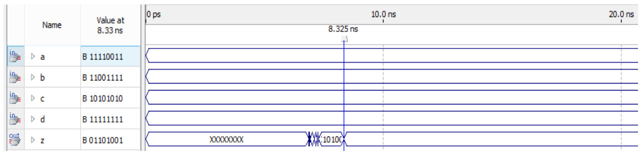

Fig.9: Delay of M4 submodule.

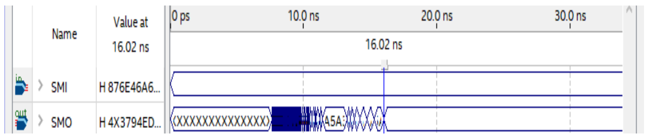

Fig.10: Timing simulation showing delay without parallelism.

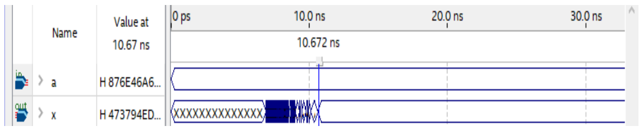

Fig.11: Timing simulation showing delay with parallelism.

| Function | Operation Type | Processing Time(ηs) | | AES 128 |
|---|---|---|---|---|
| Sub Bytes | Substitution (S-Box) | 12 | | 10 |
| Shift Row | Assign | 7.75 | | 10 |
| Mix Columns | M2 (Left Shift + XOR2) | 6.84 | 10.51 | 9 |
| | M3(XOR2) | 7.02 | | 9 |
| | M4(XOR4) | 8.83 | | 9 |
| Add Round Key | XOR2 | 7.02 | | 11 |
| Total Processing Time (ηs) | | | | 369.3 |

Table.1: Delay of Plaintext Encryption.

The results in Table.1 for Mix Columns were obtained for each submodule operation. The final simulation shows that the delay for each operation combines to give a reduced parallel result.

| Prog. | Hardware Consumption | | Delay (ηs) | Delay (ms) | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | TLE | PVM | 128 bits | 1Kbyte | 1Mbyte | 10Mbyte | 100Mbyte | 1Gbyte | 10Gbyte |
| Without parallelism | 224 | 5699MB | 16.01 | 0.009 | 9.0 | 90.1 | 900.6 | 900563 | 9005625 |
| With parallelism | 192 | 5517MB | 10.66 | 0.006 | 6.0 | 60.0 | 599.6 | 599625 | 5996250 |

**Table.2: Time Delay and Memory results of both implementations with and without parallelism.**

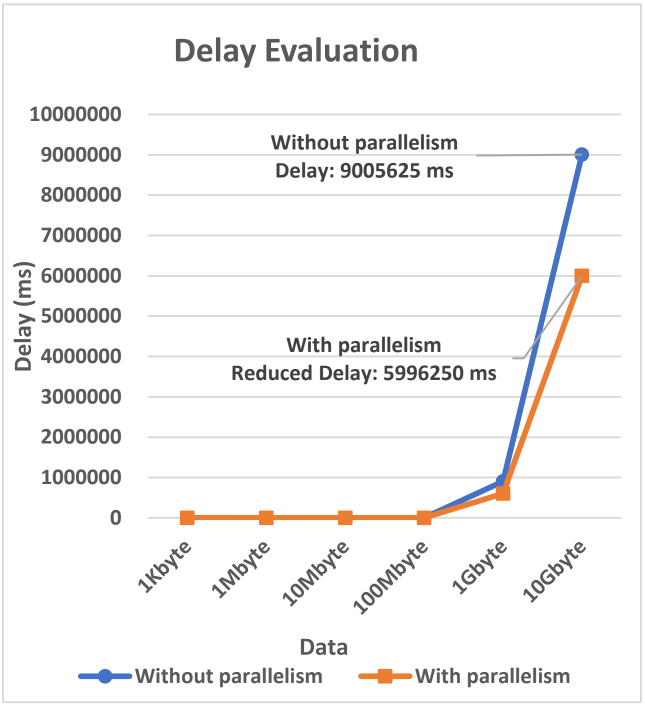

**Fig.10: Time delay analysis of mixed columns.**

The results and calculations shown in Table.2 evaluate the Total Logic Elements (TLE) and the Peak Virtual Memory (PVM) of the non-parallelized program "Without parallelism" and the parallelized one "With parallelism". The graphical evaluation of the gathered data is depicted below.

Compared to the non-parallelized program, the parallelized version showed a significant decrease in needed logic elements and virtual memory for implementation. Based on the data in Table.2, at 10 GB of input data the parallelized implementation reduces the Rijndael Mix Columns operation delay by approximately 33.4%.

Based on encryption time for Mix Columns, we created a Delay Evaluation (Table.2) and made a plot to show the performance of the different approaches in terms of encryption time. We plotted for different file sizes shown in Fig.10 and observed that the 2nd approach, which parallelized the circuit signals more, had less time delay than the 1st approach. It was also noticed that the difference in memory allocation and size corresponded to the difference in delay.

## V. Conclusion

In this paper, we conduct our study on the most popular encryption algorithm AES. We targeted one mode of operation, Cipher Block Chaining (CBC), in terms of encryption time and delay on the Mix columns section. The results in Table.2 reveals that using parallelism in signal processing results in less time delay, logic elements and virtual memory. In the future, we will focus on other sections for parallelization and try to implement AES on the FPGA. Finally, we will be able to obtain optimized area and speed hardware implementations of AES based on the sub-pipelined architecture

## Acknowledgment

The support for this research is provided in part by the US National Science Foundation under Grant No. 041585 Houston Endowment Chair in Science, Math, and Technology Fellowship, and Lloyd Benston Endowment Fellowship.

## References

[1] M.R.M. Rizk, M. Morsy. "Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA." 22 January 2008, https://ieeexplore.ieee.org/document/4437462/

[2] Del Rosal, Edni, and Sanjeev Kumar. "A Fast FPGA Implementation for Triple DES Encryption Scheme." *Circuits and Systems* 8.09 (2017): 237

[3] Diaa, S., E, Hatem M. A. K., & Mohiy M. H. "Evaluating the Performance of Symmetric Encryption Algorithms." May 2010, International Journal of Network Security. http://ijns.jalaxy.com.tw/contents/ijns-v10-n3/ijns-2010-v10-n3-p213-219.pdf

[4] Toa Bi Irie Guy-Cedric, Suchithra. R. "A Comparative Study on AES 128 BIT AND AES 256 BIT", 31 August 2018, http://www.isroset.org/pdf_paper_view.php?paper_id=782&5-IJSRCSE-01186.pdf

[5] William Stallings. "Cryptography and Network Security 5th ed." ISBN-10: 0136097049

[6] "Advanced Encryption Standard (AES)." Federal Information Processing Standards, US National Institute of Standards and Technology, 26 November 2001, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[7] Kit Choy Xintong. "Understanding AES Mix-Columns Transformation Calculation." http://www.angelfire.com/biz7/atleast/mix_columns.pdf

[8] Altera (201) Quartus Prime Design Software, http://fpgasoftware.intel.com/17.0/?edition=lite

[9] John, L. (2016) "Altera Parts History. " University of California, Berkeley, http://www-inst.eecs.berkeley.edu/~cs294-59/fa10/resources/Altera-history/Alterahistory

[10] Altera (2017) Cyclone IV DE2-115 User Manual, Version 2.3, http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502&PartNo=4

[11] Altera DE2 Board, Terasic, N.P. (2016) , https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=502