Design of a High Throughput 128-bit AES (Rijndael Block Cipher)

Tanzilur Rahman, Shengyi Pan, Qi Zhang

Abstract— **In this paper a hardware implementation of a high throughput 128- bits Advanced Encryption Standard (AES) algorithm on a single chip of Xilinx Spartan III XC3S1000 FPGA has been presented. The bus width of the architecture is 32 bit. Pipelining method has been used in this design in order to achieve a higher speed. SubBytes method has been implemented using both composite field method and fixed Rom for further analysis and comparison of performance. Through a perfect combination of different methods of SBox and key Expansion, a notable speed has been achieved in the range of 1.11 Gbps to 3.22 Gbps. An in depth analysis became possible as the whole architecture was tested in four combination (composite field and Rom for both sub bytes and key expansion). All the methods have been discussed with a proper statistical analysis and performance charts.**

Keyword: **AES, High Throughput, Pipelining, SBox, MixColumn.**

I. Introduction

THE Advanced Encryption Standard (Rijndael Block Cipher) became the new US Federal Information Processing Standard on November 26, 2001[1] in order to replace the Data Encryption Standard (DES) which was used for more than 20 years as a common key block cipher for FIPS. After that, several hardware implementations for FPGA and ASIC have been introduced [2], [3], [4], [5].

The design proposed in this paper is an AES encryption/Decryption core with 128-bit keys. Different techniques of implementing the blocks and pipelining are discussed. The arithmetic block, SubBytes, has been implemented in two ways. One is Look up table and the other is composite field technique. As the Key Expansion block contains the SubBytes as well, there would be two ways for implementing this block. So, there are four ways of implementing the whole encryption technique which will be compared at the end.

The rest of the paper is organized as follows: The second section is a brief introduction of AES encryption and decryption algorithm. The third part explains the design of pipelining level. All the results are presented in section five and the sixth section concludes the paper.

Manuscript was submitted on 7 Jan, 2010.
Authors are the students of Department of Electronic & Electrical Engineering, University Of Sheffield, UK (degree to be conferred).
Tanzilur Rahman's degree major is Electronic Engineering. Contact: tanzil_dhk@yahoo.com
Shengy Pan has major in Data Communication. Mail him at: psyking841@hotmail.com
Qi Zhang also has major in Data Communication and he can be reached at: carbenzq@googlemail.com.

II. An Introduction to AES Encryption/Decryption Algorithm

The principle design of Advanced Encryption Standard (AES) is based on substitution permutation network, which can take a block of the plaintext and the key as inputs. AES consists of four separate blocks which are repeated for 10 rounds by applying the inputs in several alternative layers to produce the cipher text block. For the first nine rounds all four blocks are repeated but for the final round the MixColumns block is excluded.

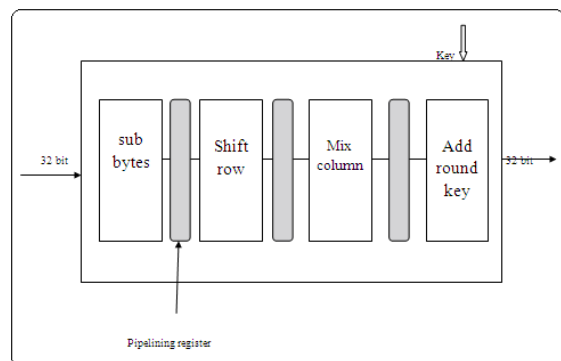

Fig. 1. The basic block of the AES core

Fig.1 shows the basic building block of the AES core which contains four separated blocks, SubBytes, ShiftRows, MixColumns and AddRoundKey. There is a 32-bit pipelining register in between each of these blocks. This full block is repeated ten times in the AES core to get the whole result.

III. Individual Block Architecture

A. SubBytes

There are two approaches to implement the sub byte transform. One is by using look up table (LUT) to get the sub byte value for each input; the other is to calculate the sub byte value by mathematical equations.

Due to all the operations are in finite field GF (2^8), there are 256 different sub byte values in total [7], [8]. All the values can be stored in a ROM as a table. When sub byte is in process, the replacement of original value is achieved by look up this table in rom. Therefore, sub byte with LUT is simple to design.

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Sub byte can also be implemented by combinational logic gates. An input in the form of $GF(2^8)$ is a 8 bit value. It costs a lot of hardware resource to transform the value in $GF(2^8)$ straight. The basic idea to simplify the design and reduce the latency is that decomposing one value in $GF(2^8)$ into $GF((2^4)^2)$ and then implementing the transform in $GF(2^4)$. After that, $GF((2^4)^2)$ value is composed into $GF(2^8)$. Finally, sub byte transform is achieved. The procedure is expressed in equation from (1) to (7).

$$map(a) = a_h x + a_l, a \in GF(2^8), a_h, a_l \in GF(2^4) \text{ ------- (1)}$$

$$(a = a_h x + a_l)^{-1} = a_h x + a_l = (a_h \otimes d)x + (a_h \oplus a_l) \otimes d$$
------- (2)

$$d = ((a_h^2 \otimes \{e\}) \oplus (a_h \otimes a_l) \oplus a_l^2)^{-1} \text{ ------- (3)}$$

$$a = map^{-1}(a_h x + a_l), a \in GF(2^8), a_h, a_l \in GF(2^4) \text{ ---- (4)}$$

Where $\otimes$ and $\oplus$ represent finite field multiplication and addition (XOR) respectively.

The finite field multiplication in GF (2^4) can be expressed as (5). Equation (6) and (7) are the square and inverse transform in GF (2^4) respectively.

$$q(x) = a(x) \otimes b(x) = a(x) g b(x) \bmod m_4(x), a(x), b(x), q(x) \in GF(2^4)$$
------ (5)

$$q(x) = a(x)^2 \bmod m_4(x), a(x), q(x) \in GF(2^4) \text{ ----- (6)}$$

$$q(x) = a(x)^{-1} \bmod m_4(x), a(x), q(x) \in GF(2^4) \text{ ----- (7)}$$

Based on these equations the sub byte transform is done by mathematical operation.

If we think of one value only, the calculation method of the transform is slower than the LUT one. However, considering multiple values transform, only one value can be found by LUT at each time which is not suitable for mass data transform. Although multiple tables can be designed in the system, the resource cost is excessive. On the other hand, calculating method is more suitable for mass values transform. Taking vantage of pipeline structure, registers can easily be introduced between logic gates which means as long as the pipeline is full, the transform results can be received continuously at each clock cycle.

The pipeline and system structure of round 1 to 9 are shown in Fig.2. From that figure, it can be seen that two 4 bit registers are introduced for the sub-pipeline of sub byte transform. Another two 8 bit registers refer to pipeline for the round transform. According to the synthesis report, the minimum period is reduced from 28.240ns to 9.703ns and the frequency is three times to that of the original design.

SubBytes block with LUT method is easy to design and faster than the composite field method without pipeline. However, it takes more slices and is difficult to increase speed this way.

Sub byte with composite method takes fewer slices than the LUT one. The frequency can be increased by sub-pipeline structure. The average delay of the logic levels should be considered during project design.

Pipeline and sub-pipeline structure increase the maximum frequency significantly whereas the slices cost almost the same. The average delay in between blocks should be controlled by sub-pipeline to reduce the maximum delay to average latency time. In this project, the maximum frequency increased from 35.411 MHz to 103.061MHz by inserting 4 registers. Therefore taking the vantage of pipelined design, AES can be implemented in FPGA for high throughput purpose. Comparing with a design without sub-pipeline, the sub-pipeline design improves the performance remarkably.

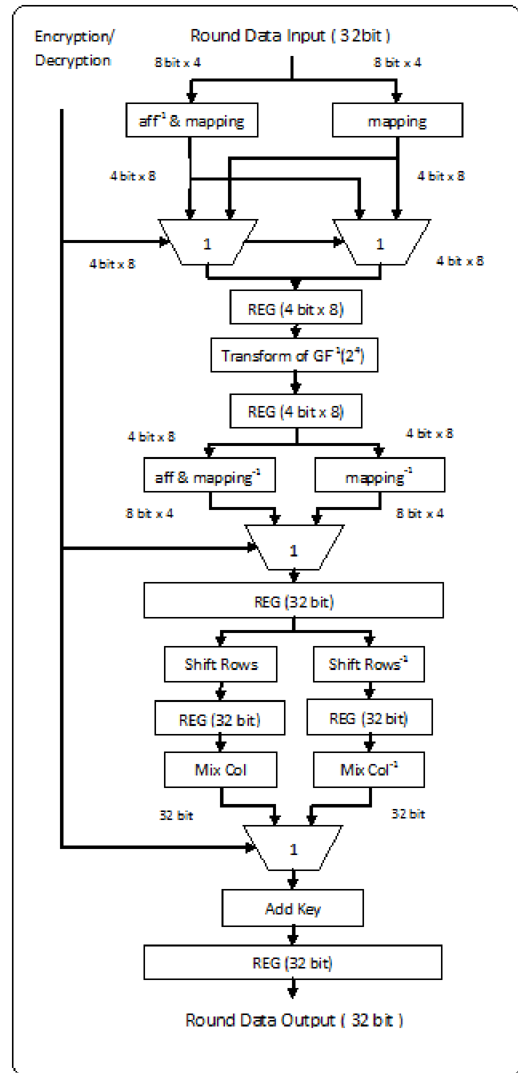

Fig. 2. The pipelined structure of rounds 1 to 9 using composite field method in SubBytes block

B. *ShiftRows*

It is a transposition step on the row of the state where each row of the state is shifted cyclically by certain number of steps. The first row (row 0) is unaltered. The second row (row 1) is shifted by one byte, the third row is shifted by two bytes and final row is shifted three bytes. It also ensures that each byte in each row does not interact solely with their corresponding bytes.

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

There are two schemes to execute the ShiftRows block. The first one is shown in Fig 3 where a mod 4 counter and two 128-bit registers are used. Each of E0 to E15 stands for 8-bit data element. The data comes into the ShiftRows block in the form of 32 bits and thus it takes 4 clock cycles to get one set of data. It requires a mod 4 counter to identify which column is coming into the ShiftRows block so that the first column can be marked as 00, the second one as 01, and so on. The data comes into the Register 1 in 4 clock cycles. In the fifth clock cycle, the elements in register 1 would be put into the corresponding position in register 2 according to the principle of ShiftRows. At the same time (the fifth clock cycle) first 32 bits of next 128-bit data would be read into E0-E3 again. At the sixth clock cycle, first 32 bits of the register 2 can be taken out. In general, there need 4 clock cycles to put data into register 1, 4 clock cycles to get out of data from register 2, 1 cycle for "shifting", and 6 clock cycles latency to get the first 32 bit output. So the counter is not only for identifying the data but also for notifying the registers to get in and output data and shift.

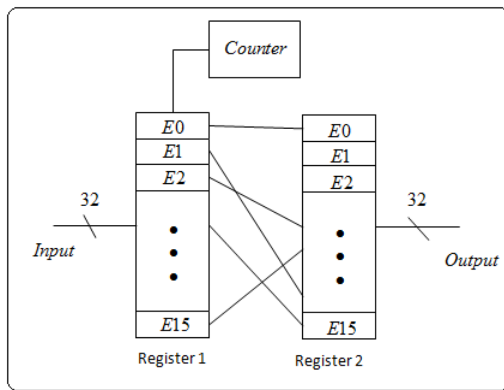

Fig. 3. The ShiftRows block using a counter

Following the second scheme, row shifting is done using shift register. This method is shown in the picture below:

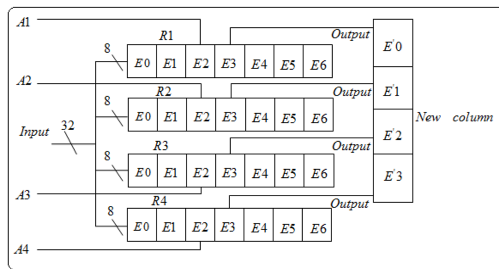

Fig. 4. The ShiftRows block using shift registers

R1, R2, R3 and R4 are shift registers which can shift element to the right hand side at each clock cycle. A1, A2, A3 and A4 are the address lines which can pick up the elements from the position in shift registers according to the address. The data will be transferred to the new column. Data continuously moves into the 4 registers column by column. After 4 clock cycles the first 4 elements of each shift register will be full and at the same time addresses are given to A1-A4. Since the first row is never shifted, A1 will always be 011, which means E3 is always picked up to E0. A2 picks up the elements from addresses "010","010","010","110" at each clock cycle, A3 picks up the elements from "001", "001", "101", "101", A4 picks up the elements from "000", "100", "100", "100".

This is the scheme that we first took into consideration but it's complicated to program and also takes more slices than the first one does. Since the first scheme is a more efficient way to do ShiftRows, it has been used in our architecture.

C. *MixColumns*

MixColumns and inverse MixColumns can be expressed as modular multiplication with constant polynomials and constant matrix multiplication [6]. We merged the two circuits (MixColumns and inverse MixColumns) into one because inverse MixColumns matrix contains a full MixColumns matrix [5]. Through matrix manipulation it is possible to show that the inverse MixColumn is just addition (XOR) of MixColumns and element matrices (Fig. 5). In this merged version, the numbers of XOR logic gates are decreased by 2/3 (from 592 XORs to 195 XORs) with only 2 XOR gates of additional delay [5]. But the question is whether this additional delay is affordable with our high throughput part of fully unrolled version of AES. The comparison shows that the normal MixColumns and inverse MixColumns in separate gives 15% less delay with consuming 30% more area. Therefore it is easy to decide to use this merged version.

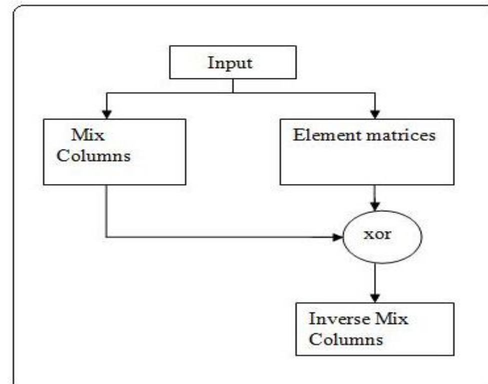

Fig. 5. Merged circuit for MixColumns and inverse MixColumns

In this version, the number of slices used is only 69 which is less than 1% and the XORs used are 60 whereas the sliced flip flops are 64 in number causing 6.109ns delay. Table I shows all the statistics of the MixColumns block.

TABLE I
The statistics of MixColumns Block

option	Gates (merged)	Gates (individual)	Delay (merged)	Delay (individual)
data	*193*	*592*	*6.109ns*	*5 ns*

D. *AddRoundKey*

In this block each byte of states are combined with the subkey where each subkey is derived from the cipher key using key schedule. The subkey and the state are of the same

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

size. The subkey is added by combining each byte of the state with the corresponding byte of the sub key through bitwise XOR manipulation.

E. KeyExpansion

128 bit key is taken as input in this block and expanded for all the rounds and stored .The keys are then used for every round. Key schedule part is dependent on the sub bytes. The sub byte is calculated both using composite filed and LUT (look up table) method. The LUT is definitely not area efficient rather time efficient whereas the composite field sub bytes technique is just opposite.

The area*delay curve comes up with the right solution to be chosen. Without making any decision beforehand on which key expansion should be used, both the key expansion have been used in the core AES in different combination (Table III) with different sub bytes. This makes it easy to analyze the performance of each combination. The key expansion in total takes 12 clock cycles to be completed but data encryption is possible to start right after 4th cycle because of the availability of first few round's keys. All the statistics are shown in table II.

TABLE II
The Statistics of KeyExpansion Block

Method	256x8-bit ROM	XORs	Slices	Minimum period
Memory based	*40*	*1312*	*5092*	*6.279 ns*
Composite field	*no*	*3790*	*2335*	*22ns (combinational delay)*

IV. THE SUB-PIPELINED ARCHITECTURE

Generally there are three ways to optimize the architecture. These three methods are based on pipelining, sub-pipelining and loop-unrolling. Pipelining is actually inserting rows of registers in between each round unit. Sub-pipelining is similarly inserting rows of registers among combinatorial logics, but the difference with the earlier one is that the registers are inserted in both inside and between the round units. These two methods lead multiple blocks of data to be processed simultaneously. On the other hand, just one block of data can be processed at a time, but multiple rounds are processed at one clock cycle in loop-unrolling method. From this brief explanation, it is obvious that the sub-pipelining can achieve the maximum speed among these three methods. The aim of this project is to design a high throughput hardware component and hence we used sub-pipelining method in our architecture.

V. PERFORMANCE ANALYSIS

Performance has been increased through pipelining of the full rounds (pipeline register in between two consecutive rounds) and pipelining the block inside every round (pipeline register in between every single block). Moreover the sub bytes function has been calculated in two different methods (LUT and composite field) giving us the opportunity to measure and compare the performance in a number ways.

TABLE III
The total statistics of the AES core

Combination of different method applied	Slices	Throughput	Through put/Slice (bit/slice)
Composite field SubBytes for both Key Expansion and SubBytes	*5207*	*1.11Gbits*	*213174.6*
Composite field SubBytes for KeyExpansion and look up table for SubBytes	*6385*	*2.32Gbits*	*363351.6*
Composite field for SubBytes and look up table for Key Expansion	*6659*	*1.11Gbits*	*166691.7*
Look up table for both Key Expansion and SubBytes	*7606*	*2.19Gbits*	*287930.6*
Sub-pipelined SubBytes	*6605*	*3.22Gbits*	*484210.5*

Max output 8.856ns

Considering to the statistics above, the bar chart of these five different structures of AES core is shown in Fig. 6. The y-axis determines the "Throughput /Slices" in each structure and it is obvious that the fifth one which is the sub-pipelined structure, has the highest rate of "Throughput/Slices".

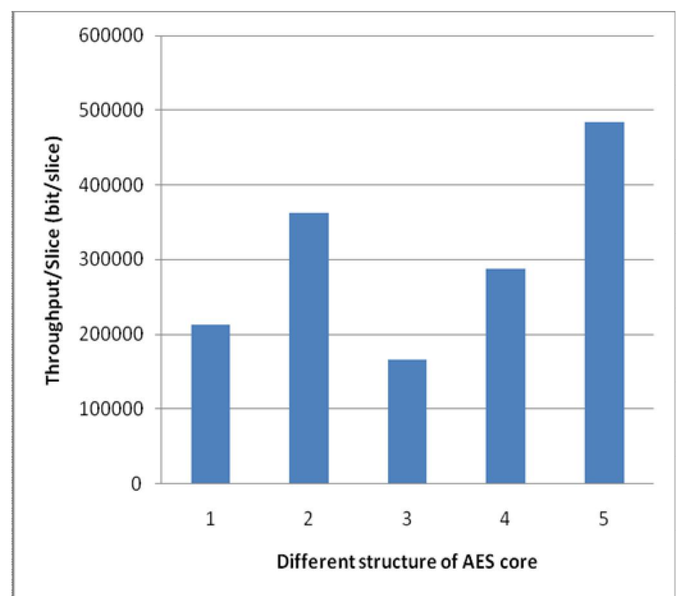

Fig 6. The Bar-chart of five different structure of AES core (Throughput/Slices)

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

VI. CONCLUSION

The detail analysis on using different SBox method has made the decision easy to follow the right one. The speed has been increased substantially through sub pipelining in the SubBytes block.

In the ShiftRows part, comparatively simpler method is followed to implement. However, further research can be done to implement it by using an improved shift-register.

Pipeline and sub-pipeline structure has increased the maximum frequency significantly. The average delay in pipeline should be controlled by sub-pipelining to reduce the maximum delay to average latency time. In this project, the maximum frequency increased from 35.411 MHz to 103.061MHz by inserting 4 registers. Therefore, taking the vantage of pipeline design AES can be implemented in FPGA for high throughput purpose. Comparing with a design without sub-pipeline, the sub-pipeline design improves the performance remarkably.

APPENDIX

MixColumns matrix:

$$\begin{pmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$
$$= \begin{pmatrix} 02\ 02\ 00\ 00 \\ 00\ 02\ 02\ 00 \\ 00\ 00\ 02\ 02 \\ 02\ 00\ 00\ 02 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 00\ 01\ 01\ 01 \\ 01\ 00\ 01\ 01 \\ 01\ 01\ 00\ 01 \\ 01\ 01\ 01\ 00 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

$$\begin{pmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} = \begin{pmatrix} 0E\ 0B\ 0D\ 09 \\ 09\ 0E\ 0B\ 0D \\ 0D\ 09\ 0E\ 0B \\ 0B\ 0D\ 09\ 0E \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$
$$= \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$
$$+ \begin{pmatrix} 08\ 08\ 08\ 08 \\ 08\ 08\ 08\ 08 \\ 08\ 08\ 08\ 08 \\ 08\ 08\ 08\ 08 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 04\ 00\ 04\ 00 \\ 00\ 04\ 00\ 04 \\ 04\ 00\ 04\ 00 \\ 00\ 04\ 00\ 04 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

$$\begin{cases} b_3 &= 02X_3 + X_1 + a_2 \\ b_2 &= 02X_2 + X_1 + a_3 \\ b_1 &= 02X_1 + X_3 + a_0 \\ b_0 &= 02X_0 + X_3 + a_1 \end{cases} \qquad \begin{cases} X_3 &= a_3 + a_2 \\ X_2 &= a_2 + a_1 \\ X_1 &= a_1 + a_0 \\ X_0 &= a_0 + a_3 \end{cases}$$

$$\begin{cases} c_3 &= b_3 + Z_1 \\ c_2 &= b_2 + Z_0 \\ c_1 &= b_1 + Z_1 \\ c_0 &= b_0 + Z_0 \end{cases} \qquad \begin{cases} Z_1 &= Y_2 + Y_1 \\ Z_0 &= Y_2 + Y_0 \end{cases} \qquad \begin{cases} Y_2 &= 02(Y_1 + Y_0) \\ Y_1 &= 04(a_3 + a_1) \\ Y_0 &= 04(a_2 + a_0) \end{cases}$$

ACKNOWLEDGMENT

We are grateful to Dr. Benaissa and Dr. Luke Seed who supervised the project and helped us with proper guidelines and information. We would also like to thank Arefeh Taghi Khani, Murali Krishna and Chu who helped us a lot during the project.

REFERENCES

[1] National Institute of Standards and Technology (U.S.), Advanced Encryption Standard. Available at: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] Xinmiao Zhang,and Keshab K. Parhi, "High Speed VLSI Architecture for the AES Algorithm", *IEEE Transactions on very Large Scale Integration (VLSI) Systems,* vol. 12, NO. 9, September 2004.

[3] A. Hodjat and I. Verbauwhede, "A 21.54 Gbit/s Fully Pipelined AES Processor on FPGA", *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines(FCCM 2004),* pp. 308- 309,April 2004.

[4] I. Verbauwhede, P. Schaumont, H. Kuo, "Design and Performance testing of a 2.29 Gb/s Rijndael Processor", *IEEE Journal of Solid-State Circuits (JSSC)*, March 2003.

[5] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh, "A Compact Rijndael Hardware Architecture with SBox Optimization", *ASIACRYPT 2001*, LNCS 2248, pp.239-254.

[6] J. Daemen and V. Rijmen, "AES Proposal: Rijndael", *NIST AES Proposal*, June 1998.

[7] Atri Rudra1, Pradeep K. Dubey1, Charanjit S. Jutla2, Vijay Kumar_,1,Josyula R. Rao2, and Pankaj Rohatgi2, "Efficient Rijndael Encryption Implementation with Composite Field Arithmetic", *Cryptographic Hardware and Embedded Systems — CHES 2001*,vol. 2162, Jan. 2001, pp. 171-184

[8] R. W. Ward, Dr. T. C. A. Molteno, "Efficient Hardware Calculation of Inverses in GF (2^8)", *Proceedings of* the 10th *Electronics New Zealand Conference* (*ENZCon*'03), September, 2003.