# A Device Independent High Grade Implementation of AES on Xilinx FPGA'S

***Asst. Prof. Dr. Siddeeq Y. Ameen***
***Computers & Information Technology Eng. Dept.***
***University of Technology, Baghdad, Iraq***

***Dr. Dhafer R. Zaghar***
***Computer & Software Eng. Dept., College of Eng.***
***Al-Mustansiriya University, Baghdad, Iraq***

***Asst. Lect. Muayed S. Al-Huseiny***
***Computer & Software Eng. Dept., College of Eng.***
***Al-Mustansiriya University, Baghdad, Iraq***

## Abstract

***The paper proposes a way for the implementation the Advanced Encryption Standard (AES), by matching the algorithm requirements with the hardware (specifically the Xilinx FPGA's) requirements.***

***The aim from the new proposal was an implementation that is not restricted to a particular device. Instead a one guided by the customer requirements, that's to say transforming the AES architecture to general purpose tool.***

***Finally, a comparison of the proposed implementation with other implementation of the AES using FPGA was made and assessed. The results clearly demonstrate the efficiency of the proposed implementation.***

---

الخلاصـــــة

***البحث يقترح طريقه لتنفيذ المشفر القياسي المتقدم من خلال مطابقة متطلبات خوارزمية التشفير مع المتطلبات ألماديه لبناء المنظومة.***

***إن الهدف من المقترح الجديد هو أن التنفيذ يكون غير مقيد بجهاز تنفيذ معين، فالنموذج المقترح موجه من خلال متطلبات المستخدم من خلال تحويل هيكلية ألمشفره ألمتقدمه القياسية تنفذ بمعدات عامة ألأغراض.***

***أخيراً تمت مقارنة التنفيذ المقترح مع أنواع أخرى من التنفيذ للمشفرة وباستخدام نفس ألأجهزة وتقييمه والتي من خلالها أظهرت النتائج كفاءة التنفيذ المقترح.***

## 1. Introduction

Cryptography is an old strategy to guarantee information exchange securely, such that other people have no access to the encrypted information. Historically encryption was used during war; nowadays it is largely used in internet, banking, and other telecommunication applications. In the past, the core of security was due to the assumption that the algorithm should be unknown, an idea no more accepted, since it gives a false confidence of security. The most accepted idea presently is that the algorithm must be public and the security must be in the key [1,2].

Furthermore, their strength should be assessed in terms of cost and speed. For a heavily loaded server, speed is the issue and hence hardware implementation is the choice, while for small designs such as smart card applications, low cost hardware or software implementation is the choice. Standards are essential for cryptography to ensure a bilateral usage of the information being exchanged. In 2001 the national institute of standards and technology (NIST), adopted the Rijndael algorithm as the advanced encryption standard (AES), for encryption purposes in commercial applications [2,3].

It replaces the old DES, due to its best overall scores in security, performance, efficiency, implementation, and flexibility with respect to the other competitors [4].The Rijndael is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits [3].

The main purpose of paper is to implement a high frequency, high throughput, and reduced cost encryption algorithm design, on high volume FPGA devices. This is made by a hyper pipelined architecture dedicated for Xilinx FPGAs.

## 2. Rijndael Structure

This algorithm defines the input block as 128 bits matrix, dividing them as a 4x4 bytes matrix called state. The algorithm has five operations all performed on the state to produce the encrypted data. The inverse of these operations should be applied in order to get the original data back, i.e. to decrypt the cipher text. The operations are SubByte, ShiftRow, MixColumn, AddRoundKey, and Key generation schedule, for encryption or the inverse of each for decryption [3].

The flow chart in **Fig.(1)** shows the Rijndael-128, in which there are 10 rounds (Nr=10). Each of which performs the same operations except the last for encryption has no MixColumn and the first for decryption has no Inverse MixColumn. Each round uses a key generated from the original 128 bits key (W=128) via the key generation schedule. The first operation is the SubByte transformation is a table look-up operation. It is executed byte to byte. It takes the data variable, and deals with it as an address for a specific memory defined by the algorithm (i.e. the content), the data stored in that address is taken as the result of the transformation (lookup table) [3,5].

The second operation is the ShiftRow. It shifts the rows of the state to the left in the following order, none in the first, once in the second, twice in the third, and three times in the

fourth [3,5]. The third operation is the MixColumn. It operates on the state column by column, treating each one as a fourth degree polynomial over GF($2^8$) and multiplied modulo ($X^4 + 1$) with affixed polynomial [3]. The last operation (the AddRoundKey) is a bitwise XOR between the state and the key [3].

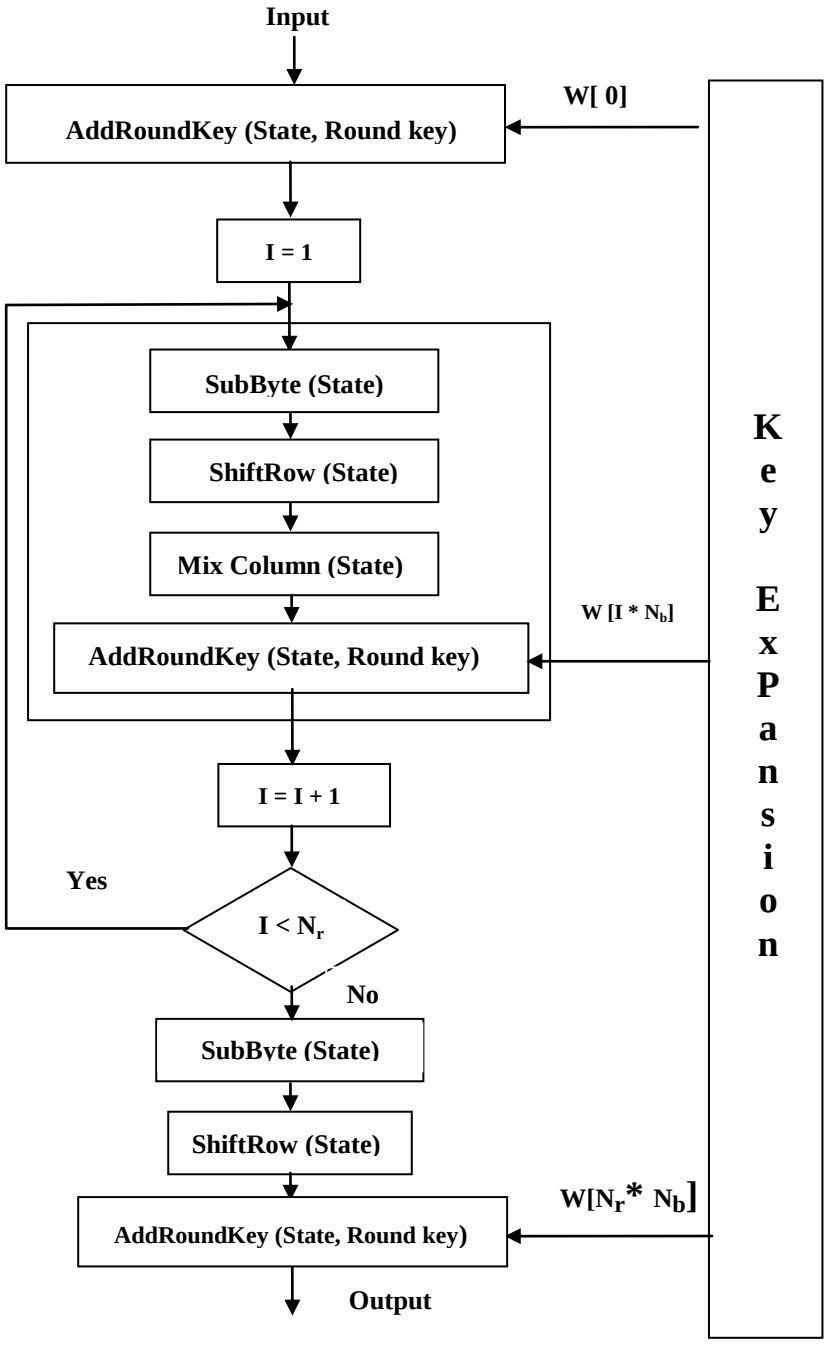


**Figure (1) The AES encryption diagram**

## 3. Xilinx Virtex FPGA

The basic building block of virtex configurable logic block (CLB) is the logic cell (LC). An LC includes a 4-input function generator, carry logic and a storage element. Each virtex CLB contains four LCs organized in two similar slices [6]. Virtex function generators are implemented as 4-input look-up tables (LUT). Each LUT can provide a 16 x 1 bit synchronous RAM. Furthermore, the two LUT's within a slice shown in **Fig.(2)** can be combined to create a 16 x 2 bit, 32 x 1 bit synchronous RAM, or a 16 x 1 bit dual-port synchronous RAM. The storage element in the virtex slice can be configured as edge triggered D-type flip-flops, which can either be driven by the function generators within the slice or directly from slice inputs [6].

The F5 multiplexer in each slice combines the two function generator outputs. This combination provides either function generator that can implement any 5-input function, a 4:1 multiplexer or selected functions of up to nine inputs. Similarly, the F6 multiplexer combines the output of all four function generators in the CLB by selecting one of the F5 multiplexer outputs. This permits the implementation of any 6-input function, 8:1 multiplexer or selected function of up to 19 inputs [6].

## 4. AES-128 Implementation

The AES as mentioned before is implemented with ten rounds. Each has four stages (operations), except the last, which has only three. The proposed implementation is based on using a full pipeline for all of these stages per round with an attempt to have the shortest possible data path per stage, resulting in the capability to have higher operational speed(frequency) with a 128 bit word length ,and hence a higher throughput.

The idea of having shortest data path requires sometimes splitting some fundamental stages into two sub stages separated by a clocked storage as shown in **Fig.(3)**. This is all done with particular characteristics of the Xilinx FPGAs.

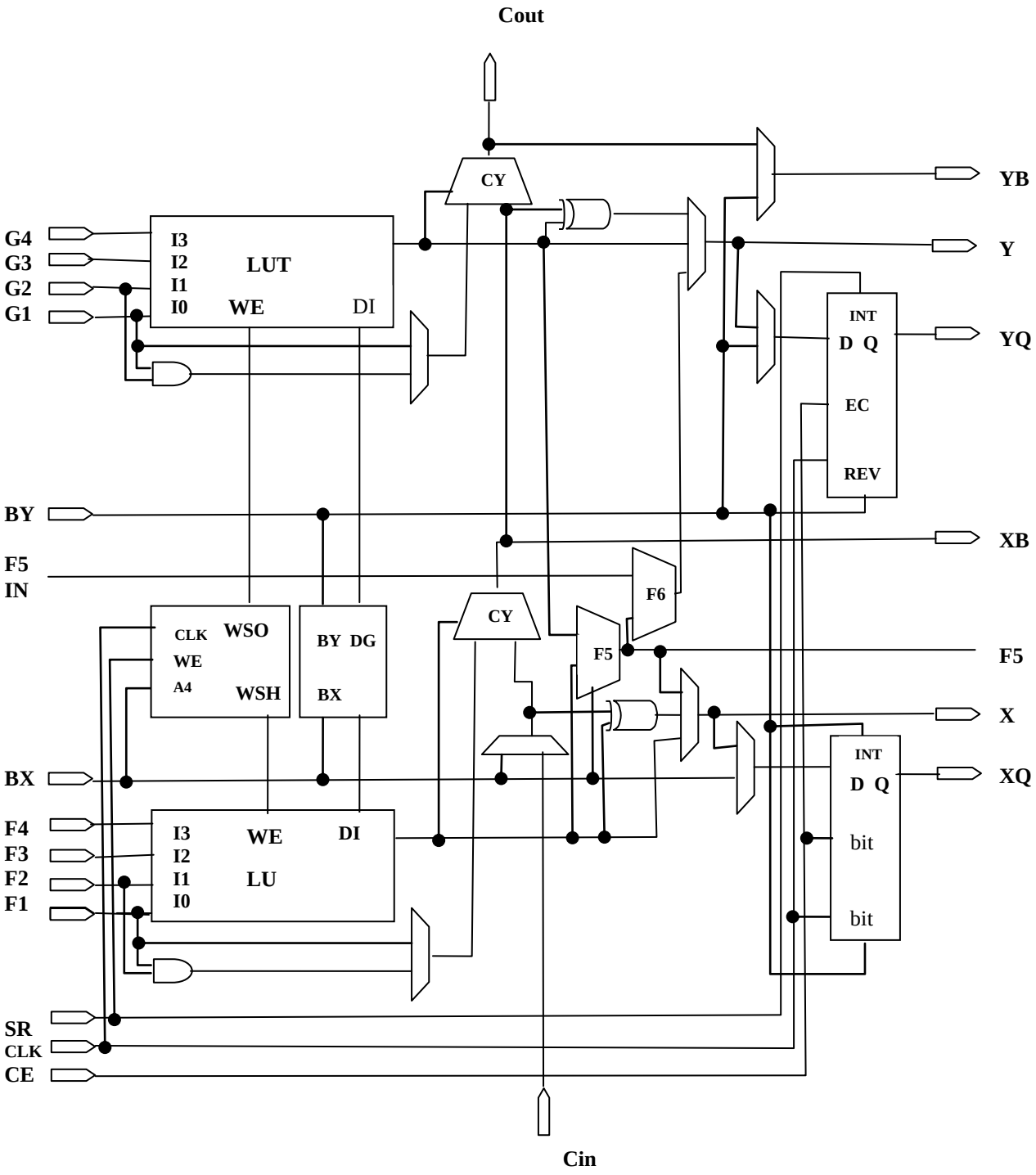


**Figure (2) Detailed view of the virtex slice**

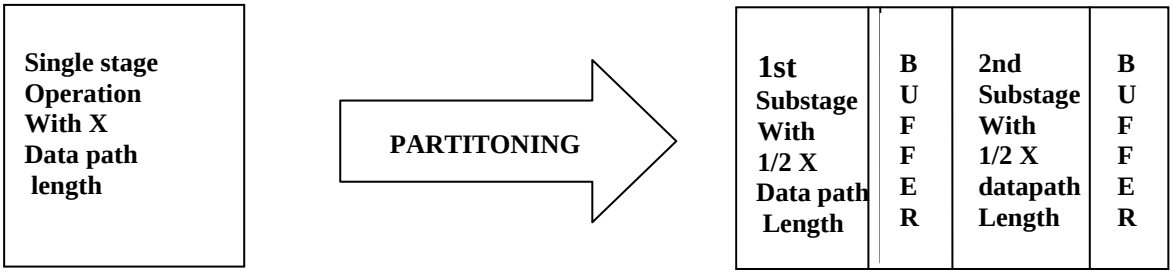


**Figure (3) Stage partitioning**

## 4-1 SubByte

This transformation is mainly a byte oriented table look-up operation, where each byte is used as an address to a pre-stored value. Taking the previous fact together with the way of representing functions in Xilinx FPGAs, one can suggest the way of partitioning the SubByte operation as follows:

**1.** The eight bits entry look-up table shown in **Fig.(3)** is divided into four sub-tables as shown in **Fig.(4)**.

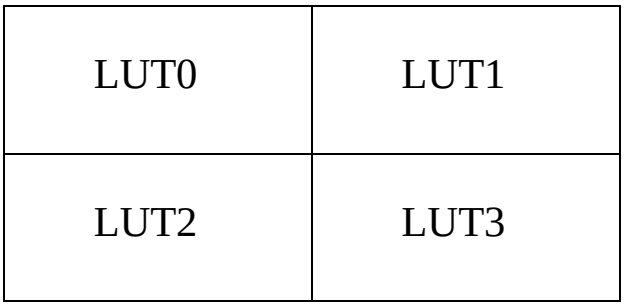


**Figure (4) The four sub-tables**

**2.** Taking the most significant two bits as an index for these sub-tables ,so each sub-table is now of six bits entry. Thus, each resulting bit would use two slices, two F5 Muxs, an F6 Mux and a flip flop. That is to say a single complete LC. This is the first part, which exactly ends at the D-type flip flop (the splitter) as shown in **Fig.(5)**.

The output from the four sub-tables together with the two index bits (In6 and In7) of the input byte form a six bits function, which in the same way drives another LC at its D-type flip flop the second part ends as shown in **Fig.(6)**.

$$Out = (\neg In7 \wedge \neg In6 \wedge Sub0) \vee (\neg In7 \wedge In6 \wedge Sub1) \vee (In7 \wedge \neg In6 \wedge Sub2) \vee (In7 \wedge In6 \wedge Sub3) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

where " ¬" (not) ,"∨ " (OR) , and "∧" (AND) are the known logical operators, **Out** is the resultant bit of the SubByte transformation, **Sub0** ,**Sub1** , **Sub2** and **Sub3** are the outputs from the four sub-tables and **In7** and **In6** are the index bits.

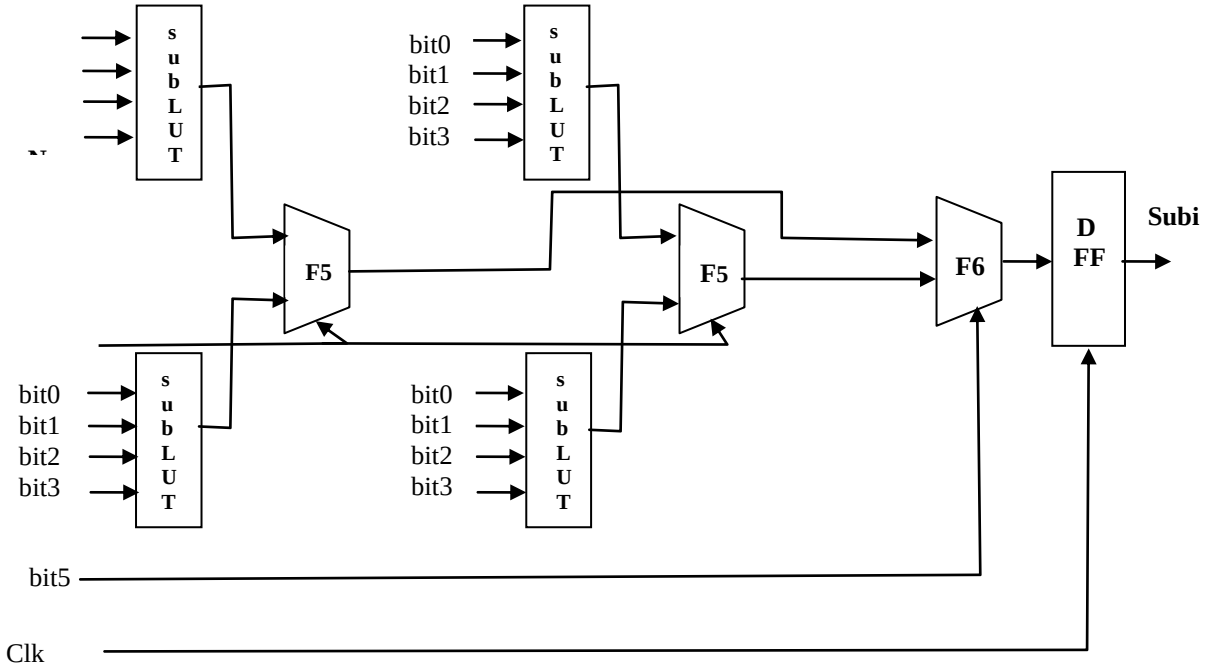


**Figure (5) 1st stage of SubByte transformation**

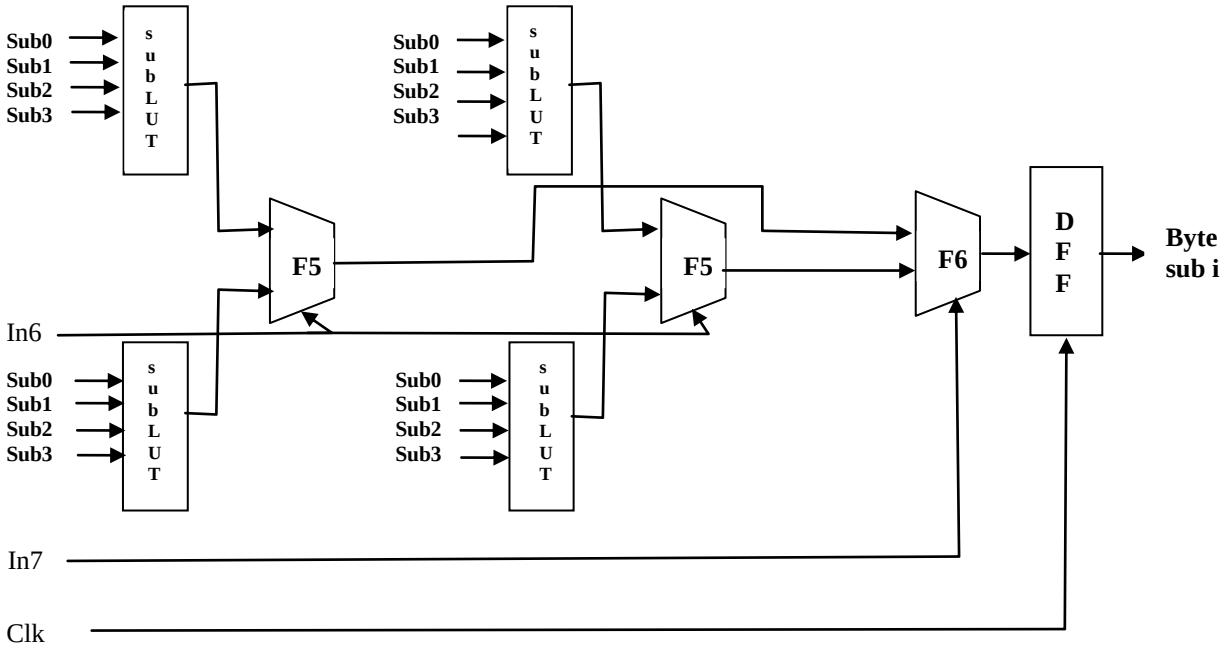


**Figure (6) 2nd stage of SubByte transformation**

The previous organization uses two LC's per bit on two stages, two F5 Muxs, F6 Mux and the delay per stage is the sum of the delays for two LUT's, while the direct placement of the table contents uses three LC's per bit on a single stage, six F5 Muxs, three F6 Muxs and the delay per stage is the sum of the delays for six LUT's. Thus it can be said that the proposed implementation reduces the cost to (2/3) the original and rises the speed three times the original. The above description gives its best results with a pipelined architecture, and gives the highest throughput when the pipe fills with data.

## 4-2 ShiftRow

From the point of view of implementation on FPGAs, the ShiftRow costs no resources more than the (almost) freely provided wiring, since it involves no mathematical and no logical operations, just rerouting of the input bytes. Hence, it is not considered as a one of the pipeline stages.

## 4-3 MixColumn

This operation is to some extent complex and requires knowledge in finite fields. However, it operates on state column by column, treating each as a four term polynomial over GF($2^8$) and multiplied modulo ($X^4 + 1$) with a fixed polynomial a(X) given by [3]:

$a(X) = \{03\}X^3 + \{01\}X^2 + \{01\}X + \{02\}$

The resultant four bytes can be defined as follows:

$$S'_{0,c} = (\{02\} \bullet S_{0,c}) \text{ xor } (\{03\} \bullet S_{1,c}) \text{ xor } S_{2,c} \text{ xor } S_{3,c} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

$$S'_{1,c} = S_{0,c} \text{ xor } (\{02\} \bullet S_{1,c}) \text{ xor } (\{03\} \bullet S_{2,c}) \text{ xor } S_{3,c} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

$$S'_{2,c} = S_{0,c} \text{ xor } S_{1,c} \text{ xor } (\{02\} \bullet S_{2,c}) \text{ xor } (\{03\} \bullet S_{3,c}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

$$S'_{3,c} = (\{03\} \bullet S_{0,c}) \text{ xor } S_{1,c} \text{ xor } S_{2,c} \text{ xor } (\{02\} \bullet S_{3,c}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

From the above equations, there are an xor operation and a multiplication in GF($2^8$), which can actually be further partitioned into left shift ,and if needs reduction (in case of overflow) by xoring with the constant polynomial $m(X) = X^8 + X^4 + X^3 + X + 1$.

Partitioning the operation in the above way simplifies it and makes it suitable for FPGA implementation as follows:

**1.** The most significant bit of the bytes to be multiplied (with two and three) are both xored and the result is stored as a flag, and this is the end of the first stage.

**2.** If the flag is one, then one of the shifted bytes needs to be reduced. Thus one of the variables in the next stage must be m(X)

**3.** If the flag is zero so either none or both of needs a reduction. Since xoring a polynomial with itself gives an all zero result. Hence in either case the constant should be replaced by all zeros polynomial.

**4.** As a result the first stage needs half a slice. The second stage has two un-multiplied variables plus the shift of the one multiplied by two, one multiplied by three plus its shift and the constant polynomial (m(X) or all zeros). Thus it is a six variable function implemented in the same optimized way of SubByte transformation. Hence for each bit there are two stages with five LUT's, two F5 Muxs, F6 Mux and two Flip Flops.

## 4-4 AddRoundKey

The round key addition is performed by the bitwise xoring of the state with the round key. Hence for each bit we need a single LUT and a D-type Flip Flop. Thus the design is actually a five stages pipeline, two for SubByte, and two for MixColumn and one for AddRoundKey as shown in **Fig.(7)**.

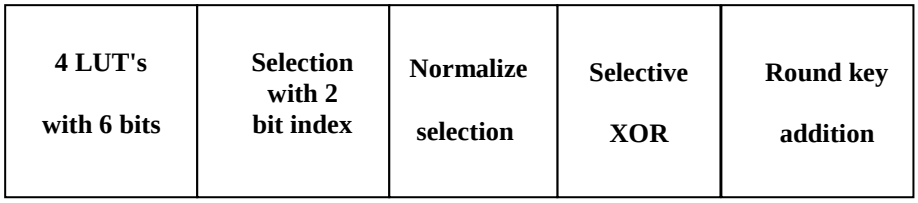


**Figure (7) The five stage pipeline AES round**

All the above procedures are for a single round. Having completed, there is freedom to choose:

**1.** The multiplicity of this round to give a highest throughput with up to one cycle architecture, in the expense of cost.

**2.** Feed the output of this round back to its input ten times before the result becomes available.

After passing the baseline the costumer requirements and nothing else would cause the choice of a particular FPGA device, with the same ratio of speed to cost for almost all devices. The proposed architecture gives the designer the highest degree of flexibility to support the required systems, such as if the costumer requires the encryption and decryption processes to be separate, the xcv1000 for each of the separate processes will be recommended. On the other hand, if the request was for both processes on the same system, the virtex device will be recommended. Furthermore, if in addition to the previous case the key generation is required, then the virtex E is the choice and so on. Finally, a comparison was made with other implementations as shown in **Table (1)**.

The following evaluation criteria were made in the comparison, the number of slices, the clock frequency, the throughput, and throughput per slice (TPS), most but not all of the

implementers use the same criteria, so we will as possible get the benefit of their criteria. The best and most expressing criteria as we think is the TPS*, since it gives a view for connecting frequency, size and throughput.

*TPS = Encryption rate / # CLB slices used

The implementation of the others that used in the comparison with the proposed has been abbreviated in Table (1) by their author names. A brief description of these implementations in their order in Table (1) is:

**Table (1) A Comparison between the proposed implementation and others**

| | *Prop.* | *Weaver Virex-E* | *Weaver Spartrn* | *Amphion* | *Chodowiec1* | *Kown* | *Chodowiec2* | *Elbirit* |
|---|---|---|---|---|---|---|---|---|
| **# Slices** | 1504 | 770 | 770 | 570 | 222 | 2256 | 1228 | 4871 |
| **$f_{max}$ (MHz)** | 310.6 | 155 | 115 | / | 50 | 30 | 47 | / |
| **Throughput (Gb/s)** | 3.98 | 1.75 | 1.3 | 1.06 | 0.147 | 0.167 | 0.521 | 1.94 |
| **TPS (Mb/s)** | 2.64 | 2.3 | 1.7 | 1.9 | 0.63 | 0.074 | 0.424 | 0.195 |

**i.** Weaver and Wawrzynek worked on the throughput optimization, making use of manual design (hand mapping) and C-low pipelining. Their implementation was made on virtex E-8 [7].

**ii.** Extra implementation of Weaver is the same as that of (i) but on Spartan II [8].

**iii.** Amphin implementation is a commercially available design. Its core uses a highly optimized HDL implementation [9].

**iv.** Chodowiec and Gaj designed a compact implementation directed for low-end products. This is achieved by folding the iterative architecture to minimize circuit area [10].

**v.** Kwon and et al employed basic loop architecture. It supports all three key sizes 128, 192 and 256 bits required by the AES standard. The switching from one key to another is instantaneous and triggered by the control signals [11].

**vi.** Chodowiec and et al employed the pipelined architecture and implemented their design on a PCI-based FPGA board named SLAAC-IC equipped with virtex xcv 1000BG5606 [12].

**vii.** Elbirt and et al employed a 5-stage partial pipeline with one subpipe-line optimized for speed [13].

## 5. Conclusions

An implementation oriented architecture design for the Rijndael algorithm was presented. The architecture was chosen to be implemented in different volumes of Xilinx FPGAs; the design is made to be hosted on FPGA layout (i.e. the Xilinx FPGA's). Thus the constraints made on which particular device can be chosen are no more than the total number of pins and slices required. This is the baseline, after which the design will not be guided by the particular device, but fortunately by the costumer requirements. From the results presented in Table 1 that compare the proposed implementation with other implementations, it is clear that the proposed implementation is significantly exceeding the others and the improvement ideas previously mentioned in the design and implementation are verified.

## 6. References

**1.** Schneier, B., ***"Applied Cryptography"***, 2nd Edition, New York, John Willey & Sons, 1996.

**2.** Stallings, W., ***"Cryptography and Network Security"***, Prentice Hall, 2003.

**3.** Federal Information Processing Standards Publication 197, ***"Announcing the Advanced Encryption Standard AES"***, November 26, 2001.

**4.** NIST, ***"Commerce Department Announcing Rijndael as the New AES"***, www.nist.gov/public_affairs/releases/g00-176.html. Aug 25, 2001.

**5.** Alex, P., Marcelo, B., and Ricardo, R., ***"A low Device Occupation IP to Implantent Rijndael Algorithm"***, www.inf.ufrgs.br/~panato/sim02.pdf .

**6.** Xilinx Virtex Manual Preliminary Product Specification, ***"DS003 (v.1.7)"***, October 1, 1999.

**7.** Spartan Throughput, www.cs.berkely.edu/~nweaver/rijndael.

**8.** Nicolas, W., and John, W., ***"High Performance, Compact AES Implementation in Xilinx FPGAs Virtex E"***.

**9.** Amphion, S., CS5210_40, ***"High Performance AES Encryption Cores"***, www.amphion.com/cs5210.html, 2001.

**10.** Pawel, Ch., and Kris, G., ***"Very Compact FPGA Implementation of the AES Algorithm"***, International Symposium on FPGA, 2001

**11.** Ohjun, K. et. al., ***"Implementation of AES and Triple DES Cryptography using a PCI-based FPGA Board"***, National Defense Academy, e-mail 940044@nda.ac.jp.

**12.** Pawel, Ch. et. al., ***"Experimental Testing of the Gigabit IP Sec. Compliant Implementation of Rijndael and Triple DES using SLA'AC.1v FPGA Acceleration Board"***, Proc. Information Security Conference, October 2001.

**13.** A. J., Elbirt, W. Yip, B. Chetwynd, C. Paar, ***"An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists"***, ECE Department, Worcester Polytechnic Institute, e-mail: spunge@alum.wpi.edu