Hardware Implementation of 128-Bit AES Image Encryption with Low Power Techniques on FPGA

Ali Farmani[1], Hossein Balazadeh Bahar[2]
1- Department Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran.
Email: Ali_Farmani88@ms.tabrizu.ac.ir (Corresponding author)
2- Department Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran.
Email: hbbahar@tabrizu.ac.ir

Received: April 2012 Revised: July 2012 Accepted: August 2012

ABSTRACT:
This paper describes the implementation of a low power and high-speed encryption algorithm with high throughput for encrypting the image. Therefore, we select a highly secured symmetric key encryption algorithm AES(Advanced Encryption Standard), in order to decrease the power using retiming and glitch and operand isolation techniques in four stages, control unit based on logic gates, optimal design of multiplier blocks in mixcolumn phase and simultaneous production keys and rounds. Such procedure makes AES suitable for fast image encryption. Implementation of a 128-bit AES on FPGA of Altera Company has been done, and the results are as follows: throughput, 6.5 Gbps in 441.5 MHz and 130mw power consumption. The time of encrypting in tested image with 32∗32 sizes is 1.25ms.

KEYWORDS: Advanced Encryption Standard (AES), Pipelining, Image Encryption, Decryption Retiming, Galios Field, FPGA, Glitch.

1. INTRODUCTION

Information is significant in every aspect of human life. Like any other property, it needs protection. There are different cryptographic algorithms available to secure information. However, most of them are computationally intensive, either deals with huge numbers and complex mathematics or involves several iterations. Advanced Encryption Standard (AES) is a cryptography algorithm proved to have the best quality between 15 candidates by National Institute of Standards and Technology (NIST). AES has high security with relatively little memory and CPU resource requirements. It is easier to apply cryptographic solutions on computer-based communication systems than on conventional systems like telephone, fax and radios. It is not feasible to dedicate a general computer for each of such systems. Instead, a cheap and portable embedded system can be developed to ensure the communication security. Microcontroller, DSP, or ASIC is used in the construction of embedded systems. Microcontroller based embedded systems have the lowest cost, which is one of the basic criteria of an embedded system design. Variety of microcontrollers available, each has different processor and peripheral devices inside them. ARM7TDMI is a popular embedded processor that has a lion's share on the market. It is reliable, that has low cost, low power consumption and small physical size [1]. AES is implemented in different ways. Many of the implementations are freely available [2] [3]. However, these implementations do not run fast enough for real-time applications, like voice encryption. In such applications, the encryption has to be done in a timely manner. Otherwise, it affects the quality of service of the communication, in a way that it cannot be tolerated by the users. In this case, most developers go for a DSP or ASIC, which can run the available implementations faster so that it can meet the required speed. To encrypt the image in [4] add one key stream generator (A5/1, W7) to AES to ensure improving the encryption performance; mainly, for images characterised by reduced entropy, which has increased the AES security for the image encryption. In [5] used AES 32-bit for encryption of image. AES encryption is an efficient scheme for both hardware and software implementation, and FPGA is used for AES implementation. In most approaches, a RAM/ROM-based lookup table (LUT) is used, such as SubByte [6] [7], and MixColumn [6], which operates on a 4-byte column and corresponds to multiplications and additions in GF (2^8). Addroundkey is simply performed by xoring each state with each key. In [7] MixColumn transformation is based on a chain of xor units. In [8] architecture is used, which speeds up the AES algorithm with no feedback by duplicating hardware for implementing each round unit. These approaches are based on pipelining, subpipelining and loop-unrolling.

In [9] ShiftRow unit is implemented based on a 4-bit counter and two memories (ROMa, ROMb). In [7] and [14] the inner and outer pipelining and loop-unrolling has made it possible to achieve the throughput of 30 to 70 Gbps using 0.18µm CMOS technology. In [11] the implementation of S-BOX is based on Finite Field. In [12] use of only one S-BOX instead of four has made the hardware and area to be reduced but also the speed to be decreased by 4 times. The rest of the paper is structured as follows: Section 2 gives a brief summary of AES algorithm and presents the system architecture adopted in our implementation. Comparison of our implementation with those done is given at section3. Finally, section4 provides the conclusion to this paper.

2. AES ALGORITHM

The AES algorithm is a symmetric block cipher that processes data blocks of 128-bits using a cipher key of length 128,192 or 256 bits each data-block consists of a 4∗4 array of bytes called the state, on which the basic operations of the AES algorithm are performed. The AES encryption procedure is shown in Fig.1.

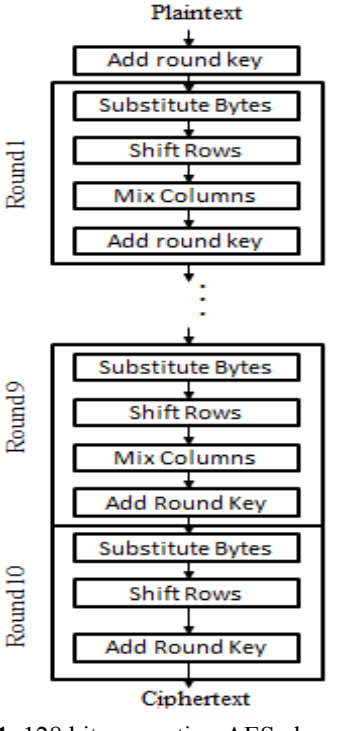

Fig. 1. 128 bit encryption AES algorithm.

The AES decryption procedure is shown in Fig. 2. Round function consists of different transformations: SubBytes, Shiftrows, MixColumns and Addroundkey. The four transformations are described briefly as follows[21]:

1. SubByte: every byte in the state is replaced by another, using the Rijndael S-Box. It is a non-linear substitution that operates independently on each byte of the state using a substitution table (S-Box). The S-Box is invertible and is constructed by composition of two transformations. For example, multiplicative inverse in a finite field $GF(2^8)$ is followed by the affine transformation [16].

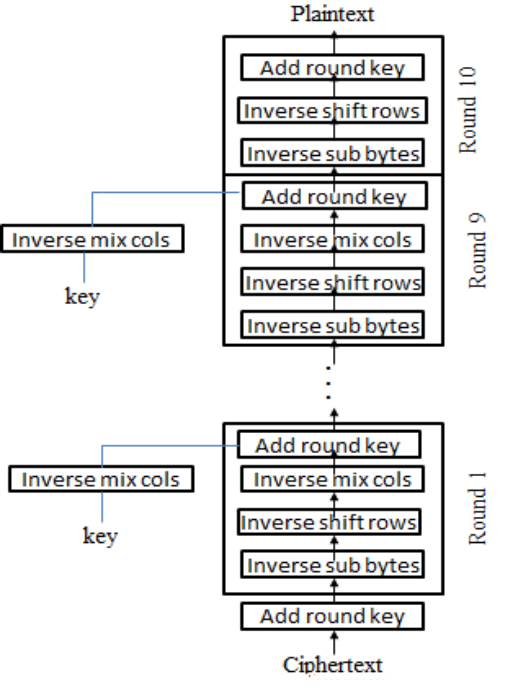

Fig. 2. 128 bit decryption AES algorithm.

Calculating entries of the S-Box is computationally expensive, and its values are independent of the input. For most applications, S-Box values are pre-calculated and stored in a 16∗16 byte (256 byte) memory. Each byte of state is mapped into a new byte in the following way: The left most 4 bits used as a row value and the right most 4 bits are used as a column value. These row and column values serve as indexes into the S-Box to select a unique 8-bit output value as shown in the Fig.3. In our implementation, S-Box is based LUT as a way of increasing the speed. This implementation is shown in Fig. 3.

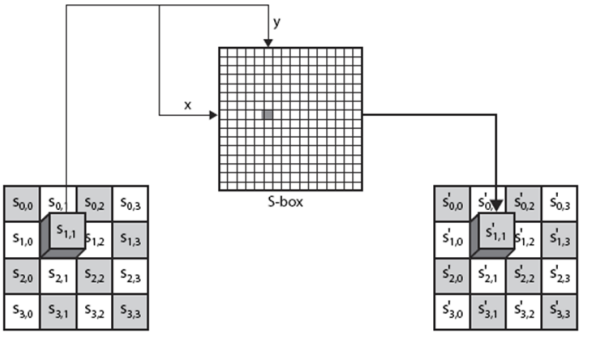

Fig. 3. S-BOX transformation (LUT)

2. ShiftRow: every row in the state is shifted a certain amount to the left. In this operation, each row of the state is cyclically shifted to the left, depending on the row index. The first row is not shifted, the second shifted 1 byte position, the third 2 byte and the fourth 3 byte position. A graphical representation of shiftrows and inverse shiftrows is shown Fig.4 and Fig.5.

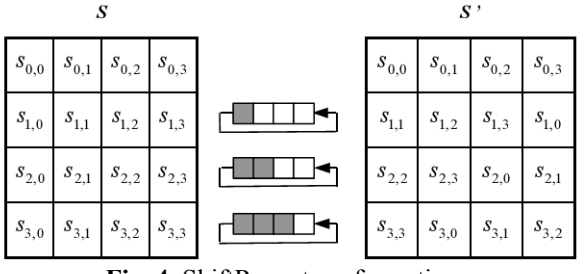

Fig. 4. ShiftRows transformation.

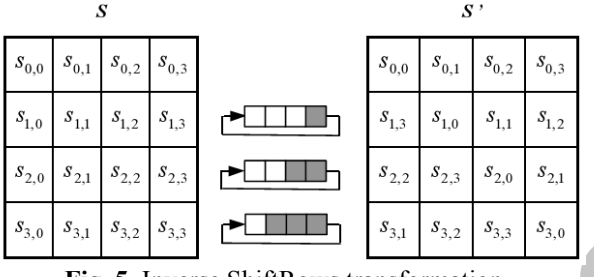

Fig. 5. Inverse ShiftRows transformation.

In our implementation, 16*8-bit registers have been used, and in the Verilog program, each output byte is placed in the position as it has to be after shift operation, making a 128-bit register, which can also be used as one of the pipeline registers. Fig.6 shows the implemented shiftrow.

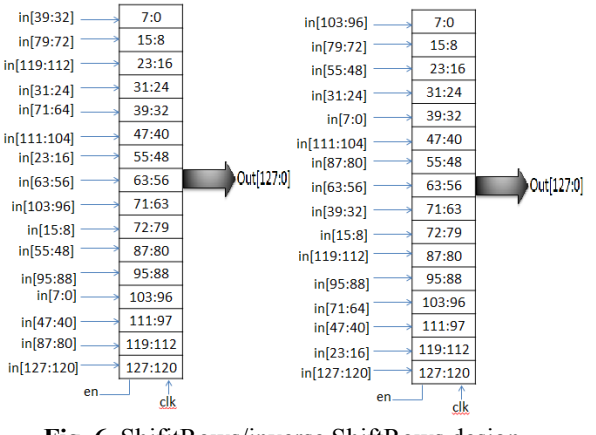

Fig. 6. ShifitRows/inverse ShiftRows design.

3. MixColumns: the data within each column of state are mixed. It operates on the state column wise, treating each column as a four term polynomial over ***GF*(2^8)**. The column polynomial is multiplied module x^4+1 with fixed polynomial, p(x) giving by $p(x) = \{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$.

The transformation can be defined by the following matrix multiplication on state (Fig.7):

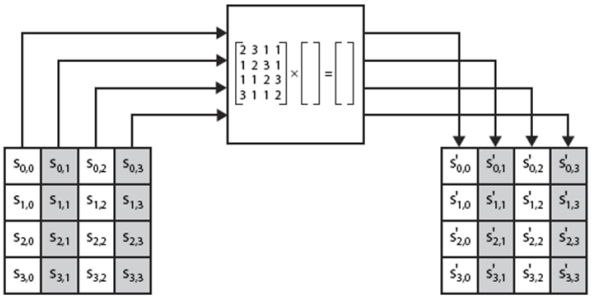

Fig. 7. Mixcolumns transformation

Multiplication of a value by x (i.e, by {02}and{03}) can be implemented as a 1-bit left shift followed by a conditional bitwise xor with (0001 1011) if the leftmost bit of orginal value is 1. (Oprator ^ stands for Exclusive OR) for example, {02}*{87}=(00001110)^(00011011)=(0001 0101) and
{03}*{6E}={6E}^({02}*{6E})=(01101110)^(11011100)= (10110010).

Our implementation is based on multiplication by 2 and 3 (multi2, multi3) and Xor operation, and these two multiplications have been written as a function.

Fig. 8 shows our Mixcolumn implementation. The equation of multiplication of each row by each column has been fully pre-calculated; therefore, the operations are only based on shift and Xor. This has resulted in an increase of speed in this transformation. Implementation of Mixcolumn is shown in Fig. 8.

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{pmatrix} = \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix}$$

InvMixcolumns is the inverse of the Mixcolumns transformation. Our implementation inverse Mixcolumn is shown in Fig.9. InvMixcolumns operates on the State column-by-column. The InvMixcolumne can be written as a matrix multiplication shown below:

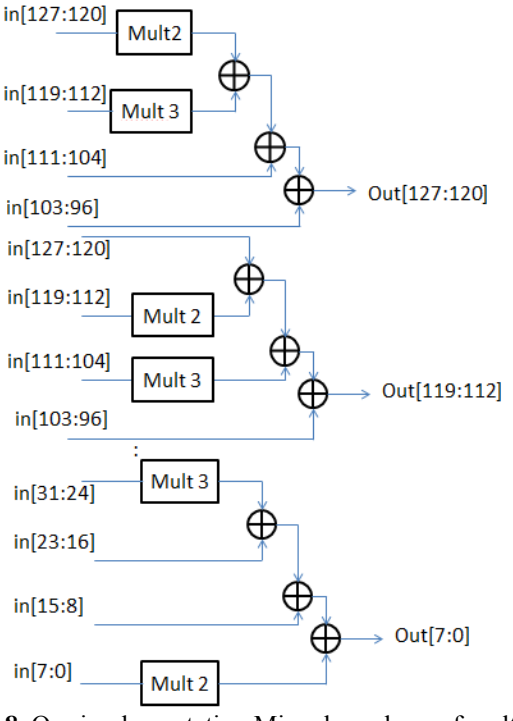

Fig. 8. Our implementation Mixcolumn base of mult3 and mult2.

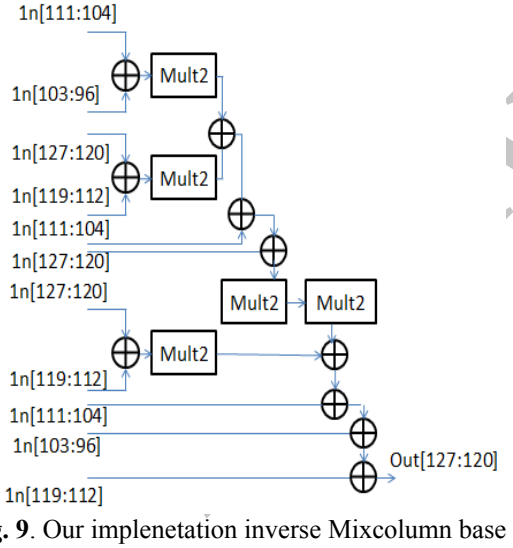

Fig. 9. Our implenetation inverse Mixcolumn base of mult2.

4. AddRoundkey: a round key is added to a state. In this operation, round key is applied to the state by a simple bit wise XOR. The round key is extracted from the cipher key using the mean of key schedule. The operation is viewed as a column wise operation between the 4byte of a state column and word of the round key. It can also be viewed as a byte-level operation. Fig.10 shows AES key expansion.

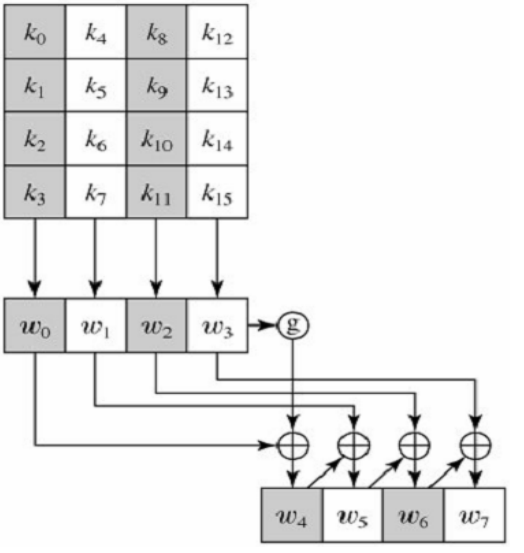

Fig. 10. AES key expansion

The function g consists of the following subfunctions:

1. Rotword performance a one-byte circular left shift on a word. This means that an input word $[b_0,b_1,b_2,b_3]$ is transformed into$[b_1,b_2,b_3,b_0]$.
2. Subword perfoms a byte subtitustion on each byte of its input word using the S-box.
3. The result of steps 1 and 2 is XORed with a round constant shown in table1.

Table 1. The value RC[j] in hexadecimal

J	1	2	3	4	5	6	7	8	9	10
RC[J]	01	02	04	08	10	20	40	80	1B	36

Our implementation of Addroundkey non-pipelining is shown in Fig.11 and Fig.12.

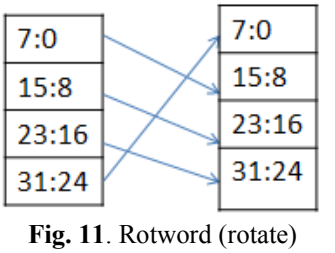

Fig. 11. Rotword (rotate)

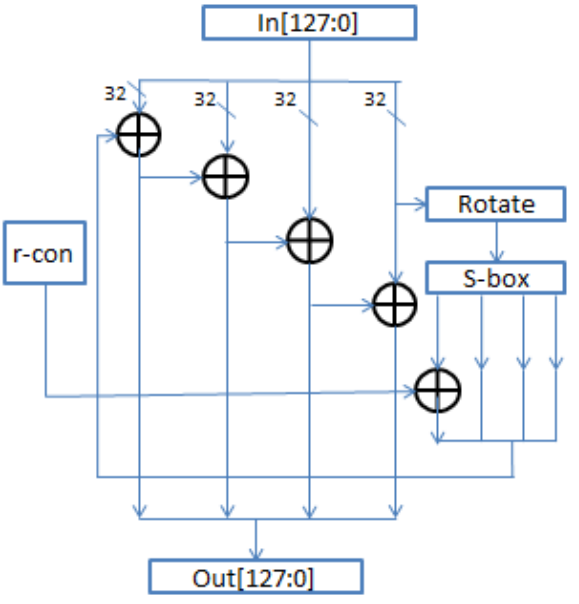

Fig.12. Addroundkey non-pipelining

Inverse Addround key non-pipelining is shown in Fig.13 and inverse R-con is shown in table 2:

Table 2. Inverse R-con

J	1	2	3	4	5	6	7	8	9	10
RC[J]	36	1B	80	40	20	10	08	04	02	01

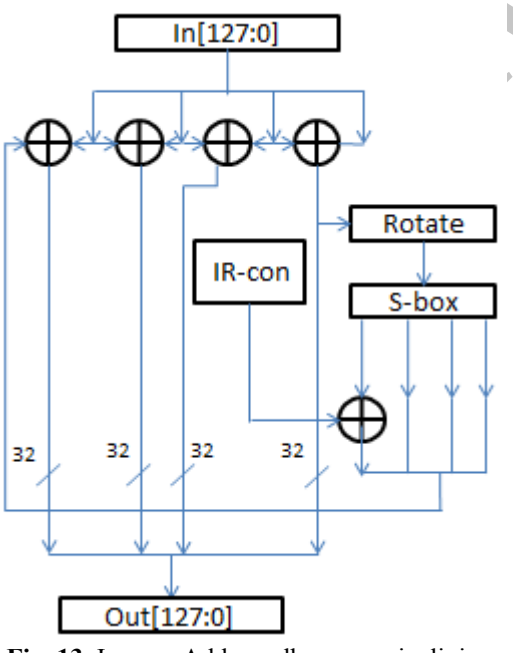

Fig. 13. Inverse Addroundkey non-pipelining

To implement Addroundkey, pipelining technique has been used, and its control unit has been implemented using logic gates. These two factors lead to an increase in speed and throughput of the unit, and it is controlled in the way that in each state, A key is generated. This means that steps of the data shifting in Addroundkey, and round are done simultaneously. Finally, each key is XORed with its corresponding round. The use of 4 stage pipelining, and control unit based on logic gates, design of MixColumn unit based on multiplications by 2 and 3, hardware implementation of multiplication of each row by each column, and simultaneous generation of each key and each round have made this implementation to be high in terms of speed and throughput. Control unit of this system, the result generated in the end of each round and key expansion should be produced simultaneously, in order to prevent to happen a cycle difference and subsequently, no wrong number is produced at this stage. Therefore, the control unit of this design is such that control signals of 4 performed operations in each round and also their multiplexers are simultaneous and done step-by-step with the performed operations in each phase of key production in key expansion, in order to ensure the synchronization and speed increase and that the key production takes place just in the last phase. The number is just produced in the last phase of round. To achieve this aim, control signals have been ordered and arranged accordingly. In other words, because of using 4 stage pipelining both in key expansion unit and in each round, and provided that their control signals are defined correctly, a perfect harmony will be created between these stages. Moreover, this unit is implemented by logic gates, which again cause the production speed of each round and key to be increases. Fig.14 and Fig 15 show the implemented encryption and decryption algorithm.

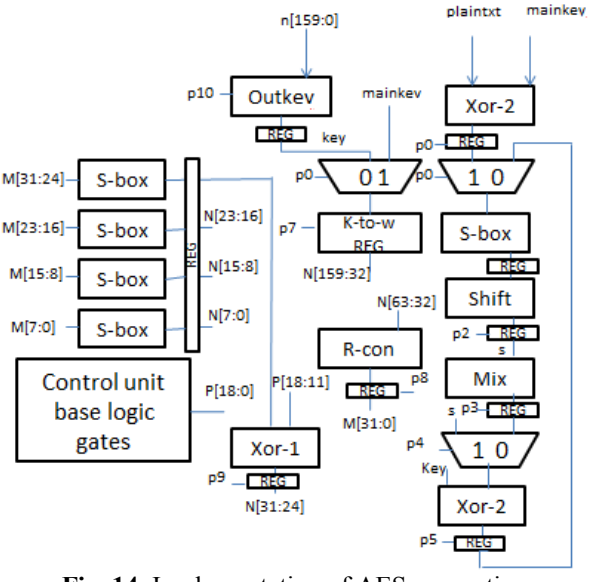

Fig. 14. Implementation of AES encryption

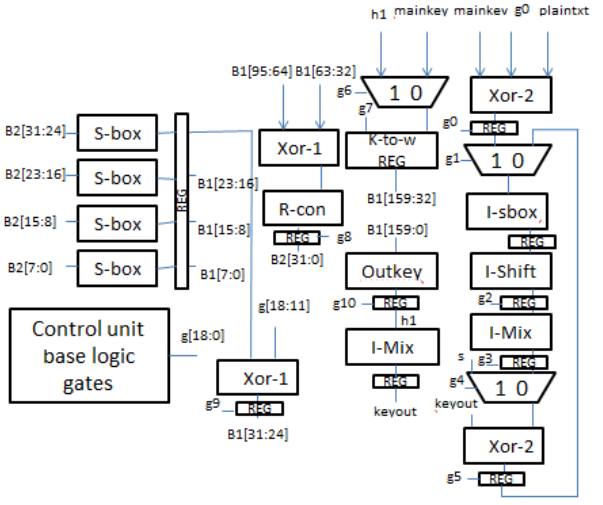

Fig 15. Our implementation of AES decryption

2.1. Power Optimization Techniques

Platform dependent power optimization techniques are implemented by using the opportunities, which are provided by the implementation platform. One of the power optimization techniques is sleep mode operation, which is called as power gating. The static power consumption of a CMOS circuit is caused by the leakage currents of transistors and P-N junctions [25]. Especially SRAM based FPGA platform causes the circuit consumes a huge amount of static power caused by the leakage currents when the circuit is off. Power gating method prevents the power consumption by using sleep mode for the states that the circuit is off [22]. There are memory blocks in the FPGA's, which cause the dynamic power consumption. If there are input ports to read or write on these memory blocks, these inputs are allowed to disable for the memory blocks, which are not used at that time. In this way, dynamic power consumption of unused memory blocks is prevented [22]. The clock signal in the FPGA has to reach for every single sequential block; so it has a long routing line. These long routing lines causes power dissipation by charging and discharging the nodes capacitance, which is referred to the capacitive power dissipation. It is also obvious that clock signal has a high frequency of logic level change; this is why its dynamic power consumption is high. So there is a way of power optimization by preventing of clock routing to the blocks which are unused. This feature is available on some of FPGA platforms [22].

2.2. Glitching

Glitches are unwanted transitions of a signal after an input change until the final output value is reached. This behavior is due to different arrival times of signals to a gate, called logic hazards. Figure 16 shows the circuit for the logic equation Q = AB + BC, which exhibits a static-1 hazard. When the inputs A and C are logic 1, any change on B will cause a transition on Q. There are two paths for B to the output Q where one path contains an inverter. This causes a slightly longer delay, resulting in a glitch in the output Q [25]. More complex circuits, e.g. ripple carry adders, amplify this problem. In typical combinational circuits glitching accounts for between 10% and 40% of the dynamic power consumption. Hazards and glitches can be avoided at the cost of more circuitry [26].

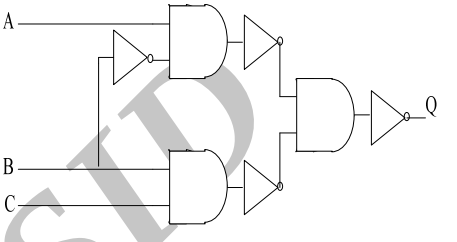

Fig.16. Glitch caused by hazard.

There are two types of ways in order to solve this glitching problem in the circuit: The first method is to place register blocks between large combinational circuits, which is widely used in our implementation. These register blocks not only decreases the logic deepness in the circuit, but also increases the clock frequency in the circuit. However, to place these register blocks increases the data processing time. This method is shown in Figure 17.

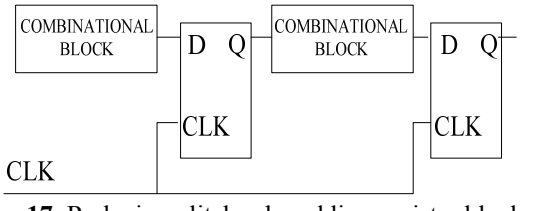

Fig. 17. Reducing glitches by adding register blocks.

Second method is to solve the glitch problem by reducing the logic deepness of the circuit. This solution is applied to the circuit during the HDL code implementation by using some coding hints. For example, the circuit in Figure 5 can be converted into a circuit as Figure 18 by doing some changes in the HDL where if, elseif and else blocks are stated. In this way, both the logic deepness of the circuit and the amount of the glitches are reduced [27].

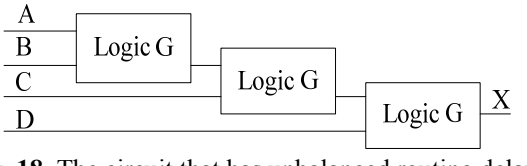

Fig. 18. The circuit that has unbalanced routing delays.

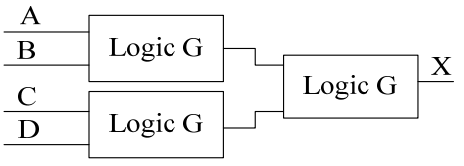

Fig. 19. The circuit that has balanced routing delays.

2.3. Clock gating

Figure 20 shows a typical implementation of a synchronous register with enable. We assume that a register is multiple bits wide and consists of one flip-flop per bit. The register is disabled when the enable signal is at logic 0. Its output is fed back to its input through the multiplexer. When the enable signal is at logic 1 the register can load new values from data in. In this design, each flip-flop of the register requires a multiplexer at its data input [25].

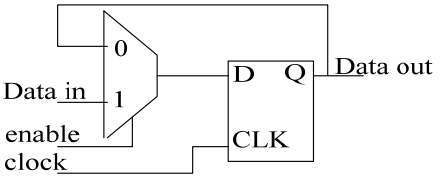

Fig. 20. Enable register with multiplexer.

Furthermore, the clock network has to drive each flip-flop. Clock gating provides a way to disable the clock signals for a register, and therefore, eliminating the need to separate multiplexers for each input bit. Figure 21 shows such a design. The enable signal is usually the output of some combinatorial logic and may contain glitches. The latch prevents glitches from the enable signal to propagate to the clock input of the register. The AND gate performs the actual gating. Clock gating replaces the multiplexers with a single clock gating cell and isolates the register clock from the global clock. The clock gating cell, containing a latch and an AND gate, consumes more power than a single bit multiplexer. However, when this technique is applied to the multiple bit registers, it can conserve both static and dynamic power. We observed savings even at registers that were only 8-bits wide [25].

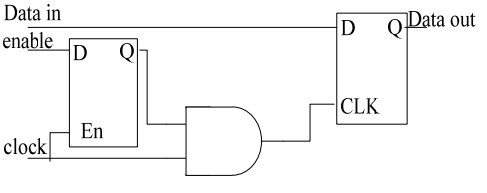

Fig. 21.Clock gated register.

2.4. Operand Isolation

Operand Isolation is a method to stop data selectively from entering a block of complex combinatorial logic, causing many transitions, and dynamic power consumption, when the output is discarded by either an unselected multiplexer or a currently disabled register. Figure 22 shows an example where changes to the input A consume power even when the output A' is not used.

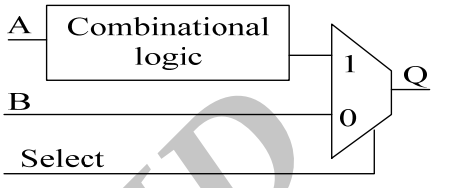

Fig. 22. Design without operand isolation.

To prevent this unnecessary power consumption isolation logic can be added at the input to the complex combinatorial logic. It prevents changes to input A from propagating through the combinatorial logic. The isolation logic usually consists of either AND or OR gates depending on the specific application. The example in Figure 22 uses an AND gate for operand isolation. The combinatorial logic only receives the input A when its output A' is selected by the multiplexer. Otherwise, its input is 0. In this way, it is prevented unnecessary power consumption when selected control signal is not logic 1, which means the output of combinatorial logic is not used [25].

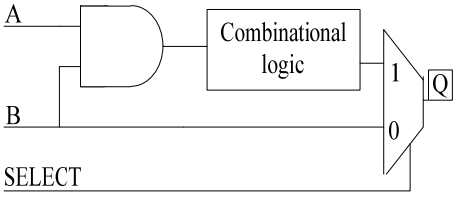

Fig. 22. Design with operand isolation.

2.5. Re-timing

Retiming for low-power is the process of positioning new or moving existing flip-flops so they separate parts of the circuit that cause glitching from parts which have high input capacitance. As glitches do not get propagated through flip-flops this technique significantly reduces the switching activity of the high input capacitance part of the circuit and reduces the dynamic power consumption [25]. The critical path in Figure 23 is decreased by changing the places of registers. The circuit in Figure 11 is redrawn in Figure 24 after being applied this retiming method [27].

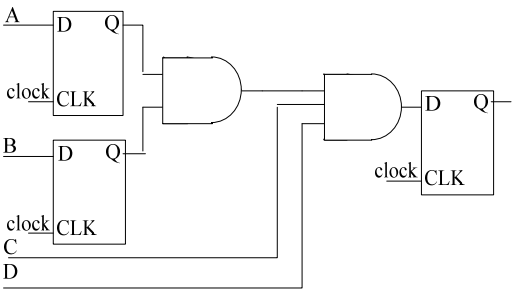

Fig. 23.Design without re-timing.

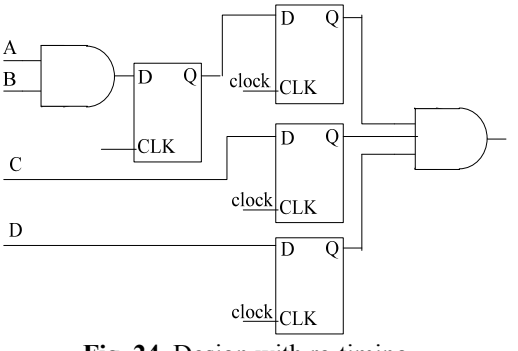

Fig. 24. Design with re-timing.

2.6. Implementation of AES Algorithm

Block of k-to-w is register that gets 128bit input, and output includes w_1, w2, w3, w4 that each is 32 bits. The original image can be regenerated using the encrypted image, and the final key produced at the last stage of encryption by the image decryption circuit, which is also implemented. In this implementation, the image used is of 32∗32 sizes and the Hex code of the image is given to the designed AES encrypting. Encrypted data of the original image, which is the encrypted image, are obtained. The time needed to generate the encrypted image is 1.25 *ms* which is very shorter of [20]. Figure 25 shows the original image and the encrypted image obtained by this implementation. The histograms of the original and encrypted images are shown in Figure 26. We can see that the histogram of the ciphered image is fairly uniform and is significantly different from the original image. Therefore, it does not provide any indication to employ any statistical attack on the image under consideration.

Fig. 25.Original image and the encrypted image.

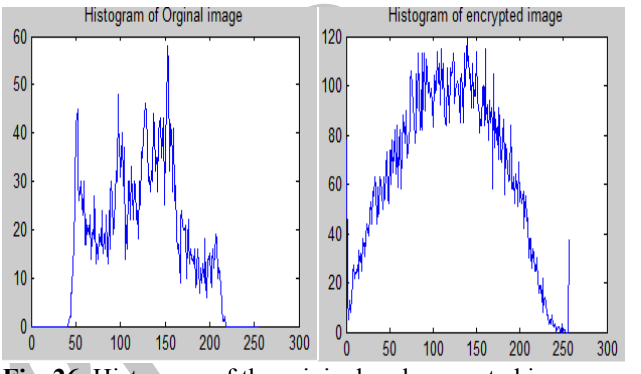

Fig. 26. Histogram of the original and encrypted image.

2.7. COMPARISONS

This design is accomplished via Verilog HDL hardware description language by QuartusII9.0 software simulated with MATLAB, and finally implemented on FPGA in Stratix II family. This design has a high speed, high throughput and low power consumption. It is really suitable for highly secured image encryption; and also the time of its converting is low. Table 3 shows the comparison of the frequency throughputs, Numbers of registers and devices and the type of devices that have been used in different articles. Table 4 presents the characteristic of our encryption image.

Table 3. Compare implementation of different AES algorithm

Implementation	Device	Frequency (mhz)	Throughput (mbps)	Nbr of register
M.zeghid[4]	-------	129	1651	-------
l.thulasimani [19]	Xc2v600 bf957-6	-------	666.7	2943
Parhi[20]	xc2vp30	150.5	221.4	536
Wang[17]	-------	125.38	1604	395
f.burns[18]	-------	132	156	4800
Parikh[13]	Xcv1000 e-8	168.4	30556	11022
Chang[9]	Spartan-3xc3s200	287	647	148
Cheng[15]	Vertex-2pxc2vp2273	273	749	104
Elkeelanv[16]	Single core	-------	12.6	1475

Our measurement result of implementation image encryption by AES is shown in table 4.

Table 4.Our result of impelementation of image encryption by AES

Implementation	Nbr of register	Frequency (mhz)	Throughput (mbps)	Encryption time(ms)
Proposed method	--	441.5	6500	1.25 (32*32)
Kuo-huang [20]	104	273	749	8243 (120*160)

2.8. CONCLUSION

In this article hardware implementation of AES algorithm is used to encrypt the image. For decrease power applying 4 retiming,glitch and operand isolation stages, designing the control unit based on logical gates, Implementation of mixcolumn and invmixcolumn by mult 2 and mult 3 units and synchronizing the key production phase with each round phase. This algorithm has been improved in terms of hardware and is appropriate for encrypting an image in a short time with low power that is 130mw.

3. ACKNOWLEDGMENT

Special thanks to my advisors Dr. Ghader Karimian and Dr. Javad Fronchi, professors in the Department of Computer Science Engineering, university of tabriz for allowing me to choose such an interesting area of security in VLSI Design. I am also very thankful to Professor kozeh kanani, the Dean of department electrical and computer for advise me in this section.

REFERENCES

[1] N. Sloss, D. Symes, and C. Wright, **"ARM System Developer's Guide, Designing and Optimizing System Software"**, *Morgan Kaufmann*, 2004.

[2] B. Gladman, **A specification for Rijndael, the AES Algorithm**. Available at http://fp.gladman.plus.com, May 2002.

[3] XYSSL Crypto Library, GNU Lesser General Public License, 2003.

[4] M. Zeghid, M. Machhout, L. Khriji, A. Baganne, and R. Tourki, **"A Modified AES Based Algorithm for Image Encryption"**, *International journal of computer science engineering IEEE* ,2007.

[5] Kuo-huang chang,yi-cheng,chung-cheng, **"Embedded a Low Area 32-bit AES for Image Encryption/Decryption Application"** *IEEE* 2009.

[6] Shuuen-shyang wang and wan-sheng ni, **"Anefficient FPGA implementation of advanced encryption standard algorithm"**, *IEEE* 2004.

[7] Alireza hodjat,david d.hwang,bocheng lai,Ingrid verbauwhede, **"a 3.84 gbits/s AES crypto corprocessor with modes of operation in a0.18um cmos technology"**, *IEEE* 2005.

[8] Xinmiao Zhang and Keshab K. Parhi, **"High-Speed VLSI Architectures for the AES Algorithm"** published in IEEE Transactions OnVery Large Scale Integration (VLSI) Systems,VOL. 12, NO. 9, September 2004.

[9] Chi-jeng Chang, Chi-Wu Husang,Hung-Yun Tai,Mao-Yuan Lin and Teng-KueiHu, **"8-bit AES FPGA Implementation ussing Block RAM"**, *The 33 Annual Conference of the IEEE Industrial Electronics Society(IECON)*,Nov.5-8,2007 , Taipei , Taiwan.

[10] carl dreyer, **"A pipelined Implementation of AES for Altera FPGA platforms"** 2004.

[11] Chih-Pin Su, Tsung-Fu Lin, Chih-Tsun Huang, and Cheng-Wen Wu, **"A High-Throughput Low-Cost AES Processor"** *IEEE Communications Magazine National Tsing Hua University*, December 2003.

[12] Namin Yu, Howard M. Heys , **"Investigation of Compact Hardware Implementation of the Advanced Encryption Standard"**, *IEEE CCECE/CCGEI, Saskatoon*, May 2005.

[13] Yi-Cheng Chen,Chung-Cheng Hsieh , Chi-WuHuang and Chi-Jeng Chang Kuo-Huang Chang, **"Embedded a Low Area 32-bit AES for Image Encryption/Decryption Application"** ,*IEEE* 2009.

[14] Alireza Hodjat, Student Member, IEEE and Ingrid Verbauwhede, Senior Member, IEEE, **"Area-Throughput Trade-Off for Fully Pipelined 30to70Gbits/s AES Processors"**, *IEEE TRANS ACTIONSON COMPUTERS* , VOL.55,NO.4,APRIL2006.

[15] Chi-Jeng Chang, Chi-Wu Huang, Kuo-Huang Chang, Yi-Cheng Chen and Chung-Cheng Hsieh, **"High throughput 32-bit AES implementation in FPGA, IEEE Asia pacific conference on circuits and systems"**, *December 2008 , MACAO* , pp.1806-1809.(EI).

[16] Jyothi Yenuguvanilanka , Omar Elkeelany, **"Performance Evaluation of Hardware Models of Advanced Encryption Standard (AES) Algorithm"**, *978-1-4244-1884-8/08, IEEE* 2008.

[17] Dazhong Wang , Xiaoni Li , **"Improved Method to Increase AES system Speed"**, *The Ninth International Conference on Electronic Measurement & Instruments* ,ICEMI'2009.

[18] F.BurnsJ.MurphyA.KoelmansA.Yakovlev, **"Efficient advanced encryption standard Implementation using lookup and normal basis"**, *Published in IET Computers & Digital Techniques*, *IET Comput .Digit.Tech.*, 2009, Vol. 3, Iss. 3, pp. 270–280.

[19] L.Thulasimani, M.Madheswaran, **"A single chip design and implementation of AES 128/192/256 encryption algorithms"** *International Journal of Engineering Science and Technology* Vol. 2(5), 2010, 1052-1059.

[20] XinmiaoZhang, Student Member, IEEE , and Keshab K.Parhi , Fellow, IEEE **"High-Speed VLSI Architectures for the AES Algorithm"** *IEEE transactions on very large scale integration (VLSI) systems* , VOL.12,NO.9 , September 2004.

[21] Dessalegn Atnafu, **"Optimizing AES Implementation for high-speed embedded Application"**, Feb. 2008, addisa baba.

[22] A.P. Chandrakasan, S.S. and Brodersen, R., 1992. **"Low-power CMOS Digital design"**, IEEE *Journal of Solid-State Circuits*, Apr., 27(4), pp. 473-484.

[23] McIvor,C., McLoone, M., and McCanny, J.V., **"Modified Montgomery modular multiplication and RSA exponentiation techniques"**, *Proceedings of Computers and Digital Techniques*, 151, pp. 402-408 2004.

[24] Walter, C.D., 1999. Montgomery Exponentiation Needs No Final Subtraction, Electronic Letters, 35, pp. 1831-1832.

[25] Kaps, J.P., 2006. **Cryptography for Ultra-Low Power Devices**, Ph.D. Thesis, Worcester Polytechnic Institue.

[26] Ghosh, A., Devadas, S., Keutzer, K. and White, J., 1992. **"Estimation of average switching activity in combinational and sequential circuits"**, *DAC '92: Proceedings of the 29th ACM/IEEE conference on Design automation*, pp. 253-259.

[27] Doğan, A.Y., **AES Algoritmasının FPGA Üzerinde Düük Güçlü Tasarımı**, M.Sc. Thesis, Istanbul Technical University, Istanbul 2008.