IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 12, Issue 3, Ver. II (May - June 2017), PP 40-50
www.iosrjournals.org

Fpga Implementation of Image Encryption and Decryption Using Aes Algorithm Along With Key Encryption

Parul Rajoriya[1], Nilesh Mohota (Professor)[2]
[1, 2] *Department of E&TC J.D. College of Engineering and Management*

Abstract*: The importance of cryptography applied to security in electronic data transactions has acquired an essential relevance during the last few years. A proposed FPGA-based implementation of the Advanced Encryption Standard(AES) algorithm along with key encryption for images is presented in this paper. An efficient image encryption scheme based on FPGA using AES algorithm integrated with RC4 encryption standard is proposed. The use of RC4 algorithm imparts additional level of security to the encryption. The design converts the original image into its hex values using Matlab and then give it as input to the proposed AES. The key input given to AES is further encrypted using RC4 encryption algorithm. The encrypted image is decrypted using AES decryption algorithm. The encrypted key is decrypted using RC4 decryption algorithm so as to give it as input to the AES decryption algorithm. The encrypted and decrypted image can be analysed in Matlab. The design uses an iterative looping approach with block and key size of 128 bits, lookup table implementation of S-box. This gives low complexity architecture and easily achieves low latency as well as high throughput.*

Keywords: *AES algorithm, RC4 algorithm, image encryption, key encryption, image decryption, key decryption*

I. Introduction

Each day millions of users generate and interchange large volumes of information in various fields, such as financial and legal files, medical reports, and bank services via Internet. These and other examples of applications deserve a special treatment from the security point of view, not only in the transport of such information but also in its storage. In this sense cryptography techniques are especially applicable. For a long time, the Data Encryption Standard (DES) was considered as a standard for the symmetric key encryption. DES has a key length of 56 bits. However, this key length is currently considered small and can easily be broken. For this reason, the National Institute of Standards and Technology (NIST) opened a formal call for algorithms in September 1997. A group of fifteen AES candidate algorithms were announced in August 1998. Next, algorithms were subject to assessment process performed by various groups of cryptographic researchers all over the world. In August 2000, NIST selected five algorithms: Mars, RC6, Rijndael, Serpent and Twofish as the final competitors. These algorithms were subject to further analysis prior to the selection of the best algorithm for the AES. Finally, on October 2, 2000, NIST announced that the Rijndael algorithm was the winner. Field Programmable Gate Arrays (FPGAs) are hardware devices whose function is not fixed which can be programmed in system. The potential advantage of encryption algorithm implemented in FPGAs includes:
Algorithm agility- This term refers to the switching of cryptographic algorithm during operation.
Algorithm upload- It is perceivable that fielded devices upgraded with new encryption algorithm which did not exist at design time.
Algorithm modification- There are applications which require modification of a standardized algorithm.
Architecture efficiency- With FPGAs it is possible to design and optimize architecture for specific parameter set.
Throughput- Although typically slower than ASIC implementation, FPGA have potential of running substantially faster than software implementations.
Cost efficiency- Time and cost for developing an FPGA implementation of a given algorithm are much lower than for an ASIC implementation.

As shown in Fig. 1 the cryptographic primitives are classified into three main categories; not using key, symmetric key and asymmetric key [1]. Symmetric key ciphers are also known as secret key or single key ciphers. Secret key ciphers are further classified as block ciphers and stream ciphers. In block ciphers, a block of bits/bytes is processed at a time. DES, IDEA, RC5, AES, BLOWFISH, TWOFISH are the different available block ciphers. Whereas in stream ciphers one bit or a byte of data is processed at a time. Stream ciphers are further classified as synchronous and self-synchronous stream ciphers.Different synchronous stream ciphers available in the literature are RC4, E0 (a stream cipher used in Bluetooth), A5/1 and A5/2 (stream ciphers used in GSM), SNOW 3G, ZUC (4G stream ciphers), Rabbit, FISH, and HC-256 etc. [2-7].
A keystream is produced in stream ciphers which is a pseudorandom sequence of bits. A plaintext (a sequence of bits/bytes) is converted into ciphertext (again a sequence of bits/bytes of same length as that of plaintext) by

hiding the plaintext with a generated keystream, using a simple XOR operation. The strength of stream ciphers is a random keystream which ensures the computational security of the cipher.

In cryptography, the AES is also known as Rijndael. AES has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits. This paper deals with an FPGA implementation of an AES encryptor/decryptor using an iterative looping approach with block and key size of 128 bits along with key encryption and decryption using RC4 algorithm . This method imparts additional level of security.

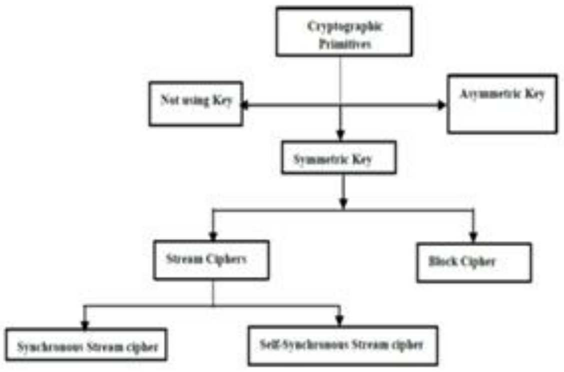


Fig. 1. Cryptographic primitives

II. Proposed Work

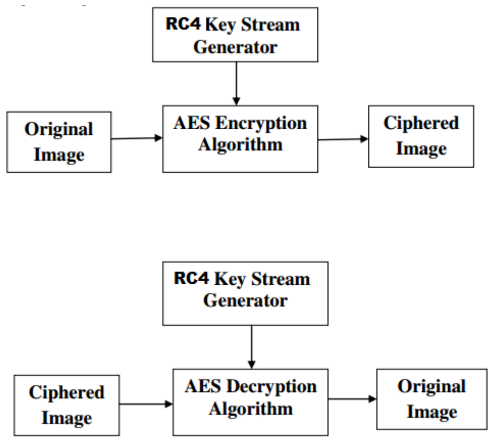


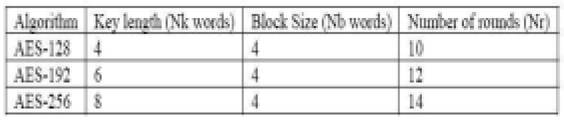
Fig 2. Proposed architecture

The figure 2 gives the proposed architecture in which image encryption and decryption is carried out using blend of AES algorithm and RC4 algorithm to provide additional level of security. The image encryption and decryption is carried out using AES algorithm and key encryption and decryption is carried out using RC4 algorithm.

A. Aes Encryption And Decryption

The AES Algorithm is a symmetric-key cipher, in which both the sender and the receiver use a single key for encryption and decryption. The data block length is fixed to be 128 bits, while the length can be 128,192,or 256 bits. In addition, the AES algorithm is an iterative algorithm. Each iteration can be called a round, and the total number of rounds is 10,12, or 14, when key length is 128,192, or 256, respectively. The 128 bit data block is divided into 16 bytes. These bytes are mapped to a 4x4 array called the State, and all the internal operations of the AES algorithm are performed on the State

Table 1: AES parameters



Algorithm	Key length (Nk words)	Block Size (Nb words)	Number of rounds (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

a. AES Encryption

An outline of AES encryption is given in Fig. 3a) In the encryption of the AES algorithm, each round except the final round consists of four transformations:

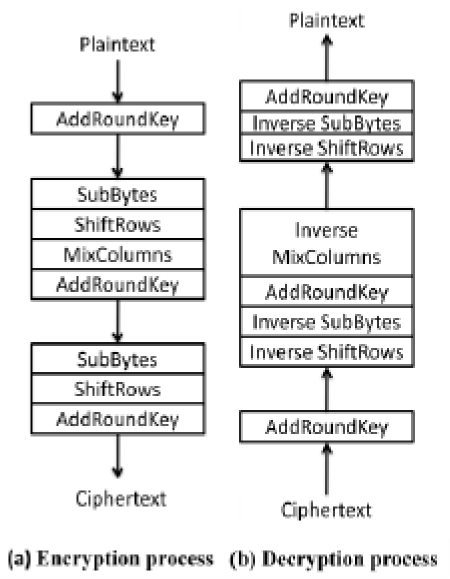


Fig 3. AES Encryption and Decryption process

SubBytes Transformation: Operates in each byte of the State independently. Each byte is substituted by corresponding byte in the S-box
ShiftRows Transformation: Cyclically shifts the rows of the State over different offsets.
MixColumns Transformation: In this operation the column of the State are considered as polynomials over GF(2^8) and are multiplied with a fixed polynomial. The MixColumn component doesnot operate in the last round of the algorithm
AddRoundKey Transformation: Involves bit-wise XOR operation.

b. AES Decryption

Decryption is a reverse of encryption which inverse round transformations to computes out the original plaintext of an encrypted cipher-text in reverse order shown in fig.3.b). The round transformation of decryption uses the functions AddRoundKey, InvMixColumns, InvShiftRows, and InvSubBytes successively.
AddRoundKey: AddRoundKey is its own inverse function because the XOR function is its own inverse. The round keys have to be selected in reverse order.
InvShiftRows Transformation: InvShiftRows exactly functions the same as ShiftRows, only in the opposite direction. The first row is not shifted, while the second, third and fourth rows are shifted right by one, two and three bytes respectively.
InvSubBytes transformation: The InvSubBytes transformation is done using a once precalculated substitution table called InvS-box. That InvS-box table contains 256 numbers (from 0 to 255) and their corresponding values.
*InvMixColumns Transformation:*The InvMixColumns transformation is done using polynomials of degree less than 4 over GF(28), which coefficients are the elements in the columns of the state, are multiplied modulo ($x4$ + 1) by a fixed polynomial $d(x) = \{0B\}x3 + \{0D\}x2 + \{09\}x + \{0E\}$, where {0*B*}, {0*D}*; {09}, {0*E*} denote hexadecimal values.

B. RC4 Encryption And Decryption

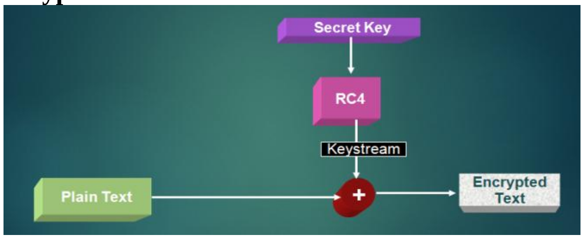


Fig 4. Block diagram of RC4 algorithm

RC4 follows the design strategy used in stream ciphers. To extract the pseudorandom data bytes from a pseudorandom permutation is the basic design principle of RC4 stream cipher. RC4 has two working modules: first there is a KSA with key K as input (with typical size of 40-256 bits), and second is PRGA which generates a pseudo-random output sequence. The pseudo code for RC4. Fig 4 presents the complete working of RC4 encryption algorithm. KSA generates the 256 byte initial state vector S, by scrambling input state vector with a random key *K.* The S contains a permutation of 8 bit words i.e. 256 bytes. The secret key *k* is generally of length between 8 to 2048 bits and the expanded key K (K of length N=256 bytes) is produced by performing simple repetitions. The expanded key is generated in the manner such that if secret key *k* is of length *l* bytes, the expanded key will be K[i] = *k* [*i* mod *l*] for $0 \leq i \leq N\text{-}1$. Further *S* pairs are swapped and an initial state SN-1 is achieved at the end which is the input to the second module PRGA. It generates the keystream of words and is further XORed with the plaintext to produce a ciphertext. Fig 5 depicts both the modules.

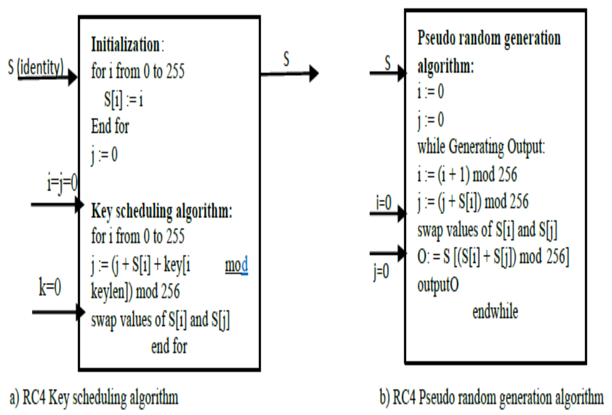

S (identity)
Initialization:
for i from 0 to 255
S[i] := i
End for
j := 0
i=j=0
Key scheduling algorithm:
for i from 0 to 255
j := (j + S[i] + key[i mod keylen]) mod 256
k=0
swap values of S[i] and S[j]
end for
S
S
Pseudo random generation algorithm:
i := 0
j := 0
while Generating Output:
i := (i + 1) mod 256
i=0
j := (j + S[i]) mod 256
swap values of S[i] and S[j]
O: = S [(S[i] + S[j]) mod 256]
j=0
outputO
endwhile
a) RC4 Key scheduling algorithm
b) RC4 Pseudo random generation algorithm

Fig 5.RC4 Stream Cipher

III. Design Steps

A. AES Encrytion steps

The encryption process is iterative in nature. Each iterations are known as rounds .For each round 128 bit input data and 128 bit key is required .That is,need 4 words of key in one round.So the input key must be expanded to the required number of words, which depends upon the number of rounds. The output of each round serves as input of next stage.In AES system,same secret key is used for both encryption and decryption.So it provides simplicity in design.

a. State array

The input to the encryption algorithm is a single 128-bit block .This block is copied into the State array,which is a square matrix of bytes. State array is modified at each stage of encryption.Similarly,the 128 bit key is depicted as a square matrix of bytes.The ordering of bytes within a matrix is by column.

b. Key Expansion

Key expansion is an important for both encryption and decryption.

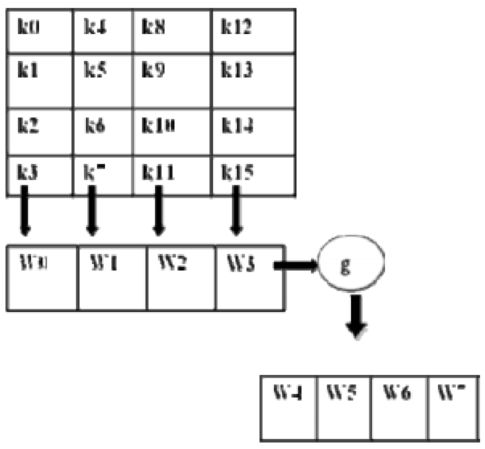


Fig 6. Key Expansion Algorithm

The AES key expansion algorithm takes as input a 4-word (16 bytes) key which is the output of the RC4 encryption module and produces a linear array of 44 words (176 bytes).This is sufficient to provide a 4-word round key for the initial Add Round Key stage and each of the 10 rounds of the cipher. The following pseudocode describes the expansion:

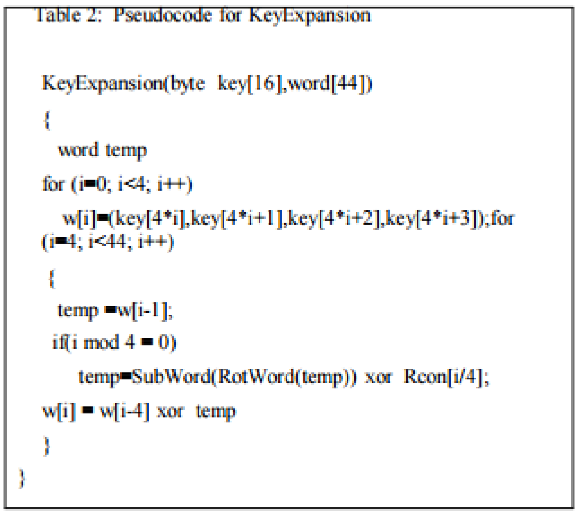


Table 2: Pseudocode for KeyExpansion

```
KeyExpansion(byte  key[16],word[44])
{
  word temp
for (i=0; i<4; i++)
   w[i]=(key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);for
(i=4; i<44; i++)
 {
  temp =w[i-1];
  if(i mod 4 = 0)
     temp=SubWord(RotWord(temp)) xor  Rcon[i/4];
w[i] = w[i-4] xor  temp
 }
}
```

The key which is the output of RC4 Encrytion is copied into the first 4 words of the expanded key.The reminder of the expanded key is filled in 4 words at a time.Each added word w[i] depends on the immediately preceding word, w[i-1] and the word four positions back, w[i-4]. In three out of four cases, a simple XOR is used. For a word:

1. RotWord performs a one-byte circular left shift on a word.This means that an input word [b0,b1,b2,b3] is transformed into [b1,b2,b3,b0]
2. SubWord performs a byte substitution on each byte of its input word,using the S-box.
3. The result of step 1 and step2 is XORed with a round constant Rcon[j].

The round constant is a word in which the three rightmost bytes are always 0.Thus the effect of XOR of a word with Rcon is to only perform an XOR on the left byte of the word.The round constant is for each round and is defined as Rcon[j] = (RC[j],0,0,0), with RC[1]=1; RC[j]=2*RC[j-1] and with multiplication over the field GF(2^8). The values of RC[j] in hexadecimal are :

c. Add Round Key

The 128 bits of State array are bitwise XORed with the 128 bits of the round key(4 words of the expanded key).The operation is viewed as a columnwise operation between the 4 bytes of the State array column and one word of the round key (Figure).

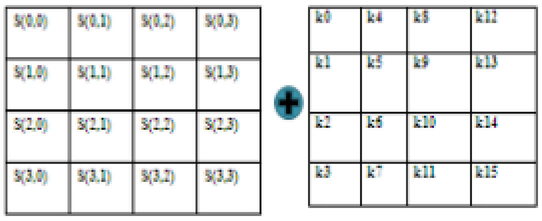


Fig 7. XOR Operation between State and Key word

d. Substitute Bytes

AES defines a 16 x 16 matrix of byte values, called an S-box, that contains a permutation of all possible 256 8-bit values. Each byte of State array is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the leftmost 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. (Figure 8).

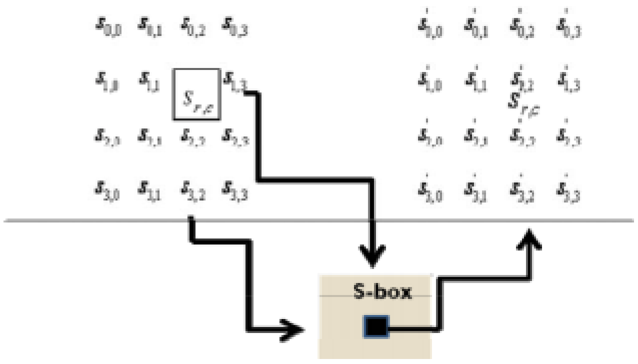


Fig 8. Sub-Byte Operation

e. Shift Rows

The first row of State array is not altered. For the second row , a 1-byte circular left shift is performed. For the third row,a 2- byte circular left shift is performed. For the third row, a 3- byte circular left shift is performed.

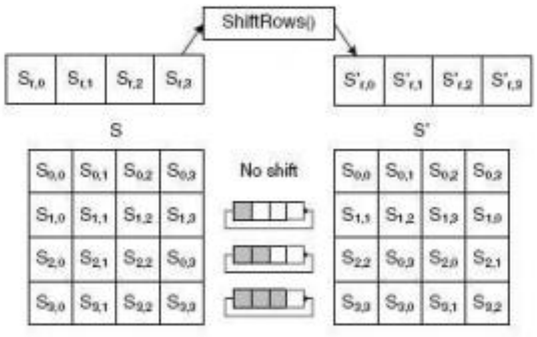


Fig 9. Shifting Operation

f. Mix Columns

It 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 the column.The transformation can be defined by following matrix multiplication on State array

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

$$\Downarrow$$

$$\begin{bmatrix} S'(0,0) & S'(0,1) & S'(0,2) & S'(0,3) \\ S'(1,0) & S'(1,1) & S'(1,2) & S'(1,3) \\ S'(2,0) & S'(2,1) & S'(2,2) & S'(2,3) \\ S'(3,0) & S'(3,1) & S'(3,2) & S'(3,3) \end{bmatrix} \quad - \quad (1)$$

Each element in the product matrix is the sum of products of elements of one row and one column.In this case,individual additions and multiplications are performed in GF(2^8).The Mix Column transformation on a single column j(0 <= j <= 3) of State array can be expressed as

$$S'(0,j) = (2.s(0,j)) \text{ xor } (3.s(1,j)) \text{ xor } (s(2,j)) \text{ xor } (s(3,j))$$
$$S'(1,j) = (s(0,j)) \text{ xor } (2.s(1,j)) \text{ xor } (3.s(2,j)) \text{ xor } (s(3,j))$$
$$S'(2,j) = (s(0,j)) \text{ xor } (s(1,j)) \text{ xor } (2.s(2,j)) \text{ xor } (3.s(3,j))$$
$$S'(3,j) = (3.s(0,j)) \text{ xor } (s(1,j)) \text{ xor } (s(2,j)) \text{ xor } (2.s(3,j)) \quad - \quad (2)$$

B. AES Decryption Steps

The Cipher transformations can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher - InvShiftRows(), InvSubBytes(),InvMixColumns(), and AddRoundKey() – process the State and are described in the following subsections. The Inverse Cipher is described in the pseudo code in Fig. 10 In Fig. 10, the array w[] contains the key schedule, described earlier.

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                    // See Sec. 5.3.1
      InvSubBytes(state)                     // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                   // See Sec. 5.3.3
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

Fig 10. Pseudo code for Inverse Cipher

a. Inverse Shift Rows() Transformation

Inv Shift Rows () is the inverse of the Shift Rows () transformation. The bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, r = 0, is not shifted. The bottom three rows are cyclically shifted by Nb - shift(r, Nb) bytes, where the shift value shift(r, Nb) depends on the row number. Specifically, the InvShift Rows () transformation proceeds as follows:

$$s'_{r,(c+shift(r,Nb)) \bmod Nb} = s_{r,c} \quad \text{for } 0 < r < 4 \quad \text{and} \quad 0 \le c < Nb$$

Figure 11 illustrates the Inv Shift Rows () transformation.

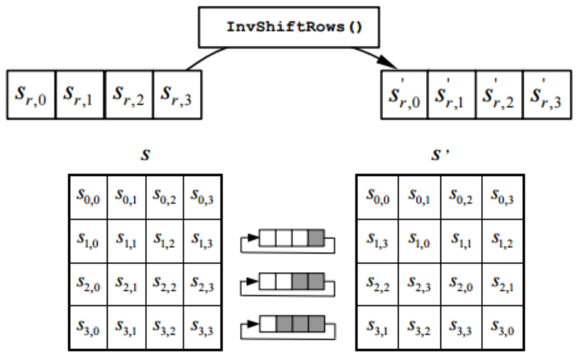

InvShiftRows()
$s_{r,0}$ $s_{r,1}$ $s_{r,2}$ $s_{r,3}$
$s'_{r,0}$ $s'_{r,1}$ $s'_{r,2}$ $s'_{r,3}$
s
s'
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,3}$ $s_{1,0}$ $s_{1,1}$ $s_{1,2}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,1}$ $s_{3,2}$ $s_{3,3}$ $s_{3,0}$

Fig 11. Inv Shift Rows ()

b. InvSubBytes Transformations

InvSubBytes() is the inverse of the byte substitution transformation, in which the inverse Sbox is applied to each byte of the State. This is obtained by applying the inverse of the affine transformation followed by taking the multiplicative inverse in GF(28). The inverse S-box used in the InvSubBytes() transformation is presented in Fig. 12

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
x	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Fig 12. Inverse S-box substitution values for the byte xy(in hexadecimal format)

c. InvMixColumns() Transformation

InvMixColumns() is the inverse of the MixColumns() transformation. InvMixColumns() operates on the State column-by-column, treating each column as a fourterm polynomial. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a^{-1}(x)$, given by

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

Let

$$s'(x) = a^{-1}(x) \otimes s(x):$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}.$$

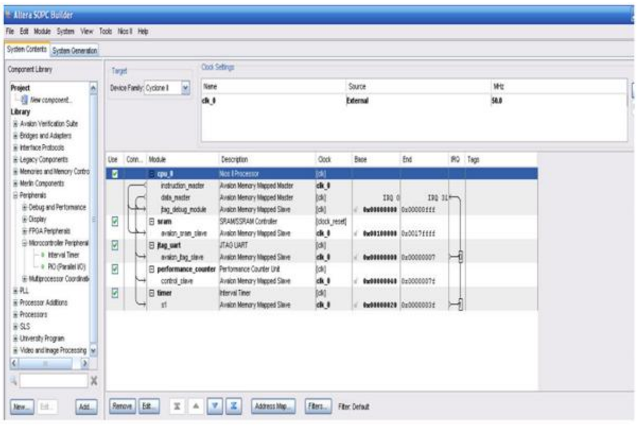
d. Inverse of the AddRoundKey() Transformation

AddRoundKey(),is its own inverse, since it only involves an application of the XOR operation.

IV. Development Steps

A. Development of Platform Using NIOS II Processor

In QUARTUS II (version 8.1*)* environment the development flow begins with selection of Cyclone II (EP2C35F672*)* family. Then according to system hardware requirements, we use the SOPC (System on programmable chip) builder tool which is available in Quartus II environment to design the system. As shown in Fig.13 using SOPC Builder a single core system is designed which is having a NIOS II processor core, SRAM memory for storing the data, for the calculation of clock and timing analysis performance counter is added and JTAG UART is connected to processor for interface. SOPC Builder generates the interconnect logic to integrate the components in the hardware system. When the system is generated successfully as shown in Fig.14.sopc file is created. Once single core NIOS II system is generated, we integrate it into the overall Quartus II project. After this the system is complied and compilation report is generated as shown in Fig.15. We can also calculate the total thermal power dissipated by the single core system which is found to be 116.66 mW using the PowerPlay power analyzer tool which is available in QUARTUS II environment as shown in Fig.16 Then the system designed using NIOS II processor is downloaded on Altera DE2 FPGA board as shown in Fig.17.

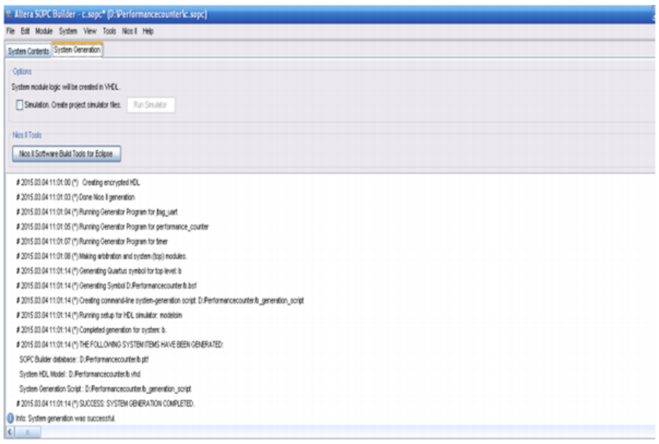


Device Family: Cyclone II

Name	Source	MHz
clk_0	External	50.0

Use	Conn...	Module	Description	Clock	Base	End	IRQ	Tags
✓		cpu_0	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	clk_0				
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00000000	0x00000fff		
✓		sram	SRAM/SSRAM Controller	[clock_reset]				
		avalon_sram_slave	Avalon Memory Mapped Slave	clk_0	0x00100000	0x0017ffff		
✓		jtag_uart	JTAG UART	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00000000	0x00000007		
✓		performance_counter	Performance Counter Unit	[clk]				
		control_slave	Avalon Memory Mapped Slave	clk_0	0x00000040	0x0000007f		
✓		timer	Interval Timer	[clk]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00000020	0x0000003f		

Fig 13. Development of platform using single NIOS II processor

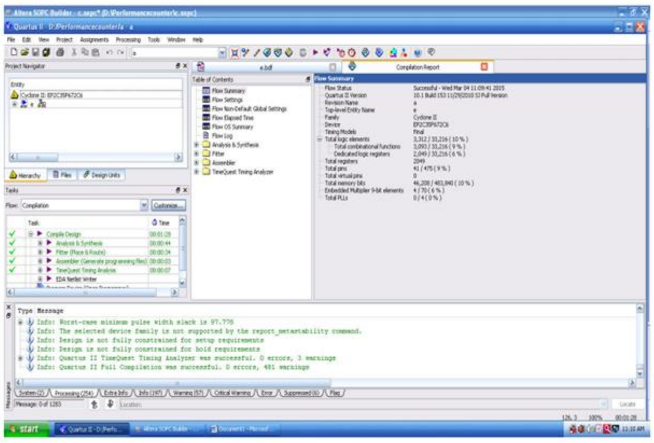

Altera SOPC Builder - c.sopc* (D:\Performancecounter\c.sopc)
File Edit Module System View Tools Nios II Help
System Contents
System Generation
Options
Run Simulator
Nios II Tools
Nios II Software Build Tools for Eclipse

System module logic will be created in VHDL.

☐ Simulation. Create project simulator files.

```
# 2015.03.04 11:01:00 (*)   Creating encrypted HDL
# 2015.03.04 11:01:03 (*) Done Nios II generation
# 2015.03.04 11:01:04 (*) Running Generator Program for jtag_uart
# 2015.03.04 11:01:05 (*) Running Generator Program for performance_counter
# 2015.03.04 11:01:07 (*) Running Generator Program for timer
# 2015.03.04 11:01:08 (*) Making arbitration and system (top) modules.
# 2015.03.04 11:01:14 (*) Generating Quartus symbol for top level: b
# 2015.03.04 11:01:14 (*) Generating Symbol D:/Performancecounter/b.bsf
# 2015.03.04 11:01:14 (*) Creating command-line system-generation script: D:/Performancecounter/b_generation_script
# 2015.03.04 11:01:14 (*) Running setup for HDL simulator: modelsim
# 2015.03.04 11:01:14 (*) Completed generation for system: b.
# 2015.03.04 11:01:14 (*) THE FOLLOWING SYSTEM ITEMS HAVE BEEN GENERATED:
  SOPC Builder database : D:/Performancecounter/b.ptf
  System HDL Model : D:/Performancecounter/b.vhd
  System Generation Script : D:/Performancecounter/b_generation_script
# 2015.03.04 11:01:14 (*) SUCCESS: SYSTEM GENERATION COMPLETED.
Info: System generation was successful.
```

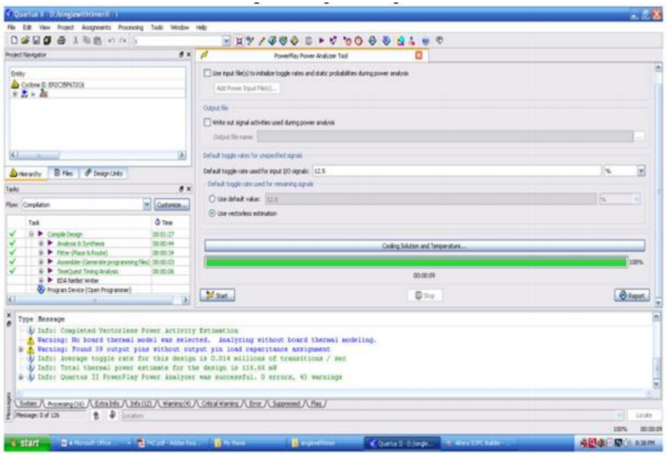
Fig 14. Single core system generated using NIOS II processor



Flow Status	Successful - Wed Mar 04 11:09:41 2015
Quartus II Version	10.1 Build 153 11/29/2010 SJ Full Version
Revision Name	a
Top-level Entity Name	a
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	3,312 / 33,216 (10 %)
Total combinational functions	3,093 / 33,216 (9 %)
Dedicated logic registers	2,049 / 33,216 (6 %)
Total registers	2049
Total pins	41 / 475 (9 %)
Total virtual pins	0
Total memory bits	46,208 / 483,840 (10 %)
Embedded Multiplier 9-bit elements	4 / 70 (6 %)
Total PLLs	0 / 4 (0 %)

```
Type Message
Info: Worst-case minimum pulse width slack is 97.778
Info: The selected device family is not supported by the report_metastability command.
Info: Design is not fully constrained for setup requirements
Info: Design is not fully constrained for hold requirements
Info: Quartus II TimeQuest Timing Analyzer was successful. 0 errors, 3 warnings
Info: Quartus II Full Compilation was successful. 0 errors, 481 warnings
```

Fig 15. Compilation report



```
Type Message
Info: Completed Vectorless Power Activity Estimation
Warning: No board thermal model was selected.  Analyzing without board thermal modeling.
Warning: Found 39 output pins without output pin load capacitance assignment
Info: Average toggle rate for this design is 0.014 millions of transitions / sec
Info: Total thermal power estimate for the design is 116.66 mW
Info: Quartus II PowerPlay Power Analyzer was successful. 0 errors, 43 warnings
```

Fig 16. Powerplay power analyzer tool result

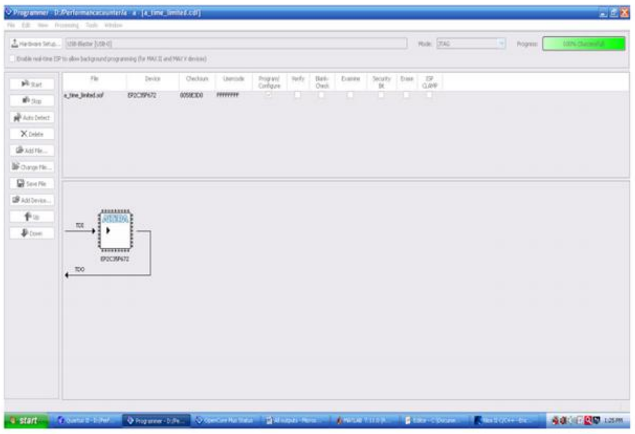


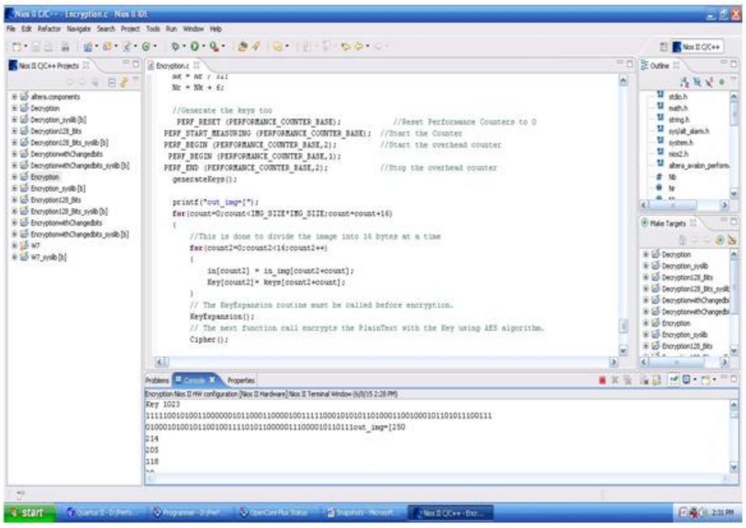
Fig 17. Single core NIOS II system downloaded into Altera DE2 board

B. Implementation on the developed platform

Using NIOS II IDE , we specify path of the developed system. After this .ptf file is selected which is created when the system hardware is downloaded into Altera DE2 board. Then we can begin designing our C/C++ application code immediately with NIOS II IDE. Altera provides component drivers and a hardware abstraction layer (HAL) which allows us to write NIOS II programs quickly and independently of the low-level hardware details. NIOS II IDE provides complete facilities for downloading software to a target board, and running or debugging the program on hardware. The IDE debugger allows us to start and stop the processor.

C. Implementation of Proposed AES blended with RC4 Encryption Algorithm

AES algorithm is designed for image encryption using C language in NIOS II IDE and verified. Then proposed AES algorithm is formed by including the RC4 cipher in the encryption code of AES to further enhance the encryption performance. Then a grey scale image is read using MATLAB and given as input to encryption algorithm. Implementation of this is done on the Altera DE2 FPGA (Cyclone II EP2C35F672) board using single core NIOS II system and the output obtained in NIOS II IDE is shown in Fig 18.



```
    Nr = Nk + 6;

    //Generate the keys too
     PERF_RESET (PERFORMANCE_COUNTER_BASE);             //Reset Performance Counters to 0
  PERF_START_MEASURING (PERFORMANCE_COUNTER_BASE);  //Start the Counter
  PERF_BEGIN (PERFORMANCE_COUNTER_BASE,2);          //Start the overhead counter
   PERF_BEGIN (PERFORMANCE_COUNTER_BASE,1);
  PERF_END (PERFORMANCE_COUNTER_BASE,2);            //Stop the overhead counter
    generateKeys();

    printf("out_img=[");
    for(count=0;count<IMG_SIZE*IMG_SIZE;count=count+16)
    {
        //This is done to divide the image into 16 bytes at a time
        for(count2=0;count2<16;count2++)
        {
            in[count2] = in_img[count2+count];
            Key[count2]= keys[count2+count];
        }
        // The KeyExpansion routine must be called before encryption.
        KeyExpansion();
        // The next function call encrypts the PlainText with the Key using AES algorithm.
        Cipher();
```

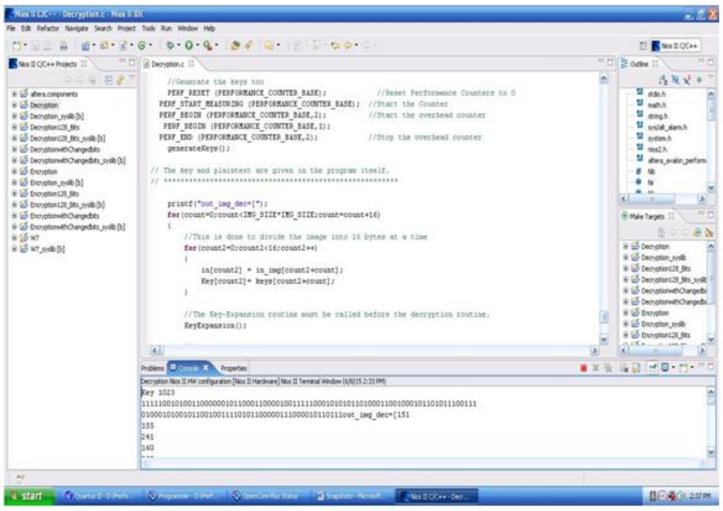
```
Encryption Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (6/8/15 2:28 PM)
Key 1023
11111001010011000000101100011000010011111000101010110100011001000101101011100111
0100010100101100100111101011000001110000101101111out_img=[250
214
205
118
```

Fig 18. Output of Proposed AES blended with RC4 Encryption Algorithm

As the size of image that is to be encypted is 128*128 so total 1024 keys are generated for its encryption i.e from key0 to key1023 as shown in Fig.7.8, this is because as AES encrypt 128 bits i.e 16 bytes at a time when the size of state matrix is 4*4. CPU clock cycles required for image encryption is calculated using performance counter.

D. Implementation of Proposed AES Decryption Algorithm

Proposed AES algorithm is designed for image decryption by including the RC4 cipher in the same manner as the design of encryption algorithm is done. CPU clock cycles required for image decryption is calculated using performance counter. The decrypted output obtained in NIOS II IDE by implementing the decryption algorithm on Altera DE2 FPGA (Cyclone II EP2C35F672) board using single core NIOS II system is shown in Fig. 19.



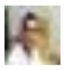
```
    //Generate the keys too
     PERF_RESET (PERFORMANCE_COUNTER_BASE);             //Reset Performance Counters to 0
  PERF_START_MEASURING (PERFORMANCE_COUNTER_BASE);  //Start the Counter
  PERF_BEGIN (PERFORMANCE_COUNTER_BASE,2);          //Start the overhead counter
   PERF_BEGIN (PERFORMANCE_COUNTER_BASE,1);
  PERF_END (PERFORMANCE_COUNTER_BASE,2);            //Stop the overhead counter
    generateKeys();

// The key and plaintext are given in the program itself.
// **********************************************************

    printf("out_img_dec=[");
    for(count=0;count<IMG_SIZE*IMG_SIZE;count=count+16)
    {
        //This is done to divide the image into 16 bytes at a time
        for(count2=0;count2<16;count2++)
        {
            in[count2] = in_img[count2+count];
            Key[count2]= keys[count2+count];
        }

        //The Key-Expansion routine must be called before the decryption routine.
        KeyExpansion();
```

```
Decryption Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (6/8/15 2:33 PM)
Key 1023
11111001010011000000101100011000010011111000101010110100011001000101101011100111
0100010100101100100111101011000001110000101101111out_img_dec=[151
155
241
160
```

Fig 19. Output of Proposed AES Decryption Algorithm

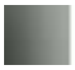
TEST IMAGE

IMG SIZE: 16X16



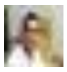
ENCRYPTED IMAGE

IMG SIZE :16X16



DECRYPTED IMAGE

IMG SIZE : 16X16



V. Conclusion

The encryption and decryption of image on the blended platform of AES and RC4 is successfully performed, i.e. FPGA implementation of image encryption and decryption using AES algorithm along with key encryption is executed successfully.

References

[1] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Hand-book of Applied Cryptography. CRC Press, August 2011 edition, 1996. Fifth Printing.

[2] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.

[3] Bluetooth T M. Bluetooth specification, v4.0, June 2010. E0 encryption algorithm described in volume 2, pages 1072–1081. Available online at http://www.bluetooth.org.

[4] Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms. Available online at http://www.scard.org/gsm/a51.html, 1998.

[5] 3rd Generation Partnership Project. Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2. ETSI/SAGE Specification Document 2: SNOW 3G Specification, v1.1, September 6, 2006.

[6] ECRYPT Stream Cipher Project eSTREAM. The current eSTREAM portfolio. Available online at http://www.ecrypt.eu.org/stream/ index.html.

[7] ECRYPT Stream Cipher Project eSTREAM. Software performance results from the eSTREAM project. Available online at http://www. ecrypt.eu.org/stream/perf/#results.

[8] El. Maraghy M, Hesham S and Abd El. Ghany M.A, "Real-time Efficient FPGA implementation of AES algorithm", IEEE International SOC Conference(SOCC), page 203-208, Sept 2013.

[9] M.Sambasiva Reddy and Mr.Y.Amar Babu, "Evaluation Of Microblaze and Implementation Of AES Algorithm using Spartan-3E", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 2, Issue 7, page 3341-3347, July 2013.

[10] Hoang Trang and Nguyen Van Loi, "An Efficient FPGA Implementation of The Advanced Encryption Standard algorithm", IEEE International Conference on Computing and Communication Technology, page 1-4, Ho Chi Minh city, 2012.

[11] Kamali S.H, Shakerian R, Hedayati M and Rahmani M, "A new modied version of Advanced Encryption Standard based algorithm for image encryption", (ICEIE) International Conference On Electronics and Information Engineering, volume 1, page 1250-1255, Aug 2010.