Coding AES (Encryption and Decryption) using VHDL

Arjun Raj

December 2020

1 Introduction

It is a widely used symmetric cipher and finds its application in many commercial systems ,Internet security standard ,WiFi encryption etc. Its was initially used to encrypt classified documents from level secret to top secret. To date there has been only brute force attack on AES. Major inputs needed for encryption/decryption are plain text/cipher text and key. Here I made my key length fixed (128 bits) and used plain text of size 128 bits. Number of internal rounds(or iterations) depends on the size of the key. Here for a key length of 128 bits ,I need my algorithm to iterate 10 times. In each round the entire 128 bits are encrypted. There are 4 main layers in an encryption round, they are- Key addition , Byte substitution , Shift row layer , Mix column layer; Note the last round doesn't contain Mix column layer

2 Key addition

128 bit subkeys(in my case 11 of them) are derived from our unique main key and it is used to XOR with the state (ie, cipher text / plain text). The key addition and its inverse operation are essentially the same. The process of XORing each individual subkey with the state is called key whitening. The number of key whitening steps in our approach is 11(number of rounds+1)

2.1 Key schedule for AES

$k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15}$ be a 128 bits/16 bytes number, where k_i denotes $8bits/1byte$. Let W be array of length 44 and each element of the array is 32 bits .

$$W[0] = k_0k_1k_2k_3 \tag{1}$$

$$W[1] = k_4k_5k_6k_7 \tag{2}$$

$$W[2] = k_8 k_9 k_{10} k_{11} \tag{3}$$

$$W[3] = k_{12} k_{13} k_{14} k_{15} \tag{4}$$

We can find other W[i], $for\ i\ \epsilon [4, 43] by$

$$W[i] = W[i-1]\ xor\ W[i-4],\ for\ i\%4 \neq 0 \tag{5}$$

$$W[i] = g(W[i-1])\ xor\ W[i-4],\ for\ i\%4 = 0 \tag{6}$$

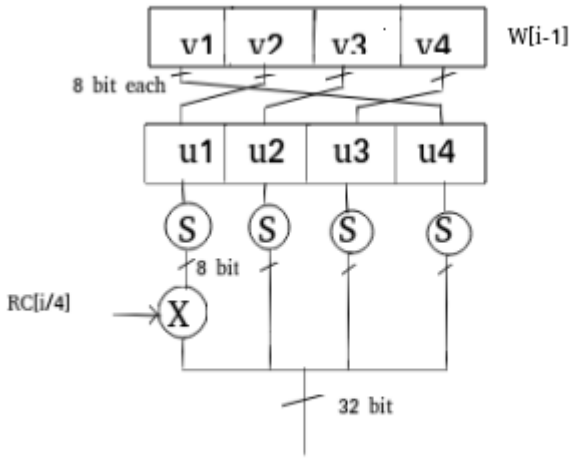

Figure 1 function g

Here S denotes subbytes operation (done on 8 bits) and X implies bitwise XOR. RC[i/4] is given in the table below.

i/4	RC val
1	00000001
2	00000010
3	00000100
4	00001000
5	00010000
6	00100000
7	01000000
8	10000000
9	00011011
10	00110110

Figure 2 RC table

subkey[j] is obtained as W[j*4]&W[j*4+1]&W[j*4+2]&W[j*4+3], $for j \epsilon [1, 10]$, where subkey is 128 bits and is used for j^{th} round

3 Encryption

128 bit plain text is encrypted by using a key X"11111111111111111111111111111111" .The key is 128 bits long , so we iterate 10 times through the entire encryption process. The following subsections we elaborate the main components in encryption module

3.1 Subbytes or S block

Here each 8 bits of 128 bits word is mapped into some other 8 bits. The underlying operation is inversion in GF(2^8), followed by an affine transformation. It results in one to one mapping. This is a non linear transformation block. We can write a function (combinational) that take in the 8 bits input , A_i and return the S(A_i), where S is transformation happening on A_i.This block provides high degree of non linearity , which in turn provide strong protection against some of the analytical attacks. Affine will destroy the structure imparted due to inversion in GF. We can use a look up table, for this operation.

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

Figure 3 S block table

3.2 Shift Row

It's a diffusion step and linear step. It spreads the influence of individual bytes over the entire state. It shifts second, third and fourth rows of the state matrix to left by 1, 2 and 3 units, this will result in new state matrix. This step will enhance the diffusion property of AES.

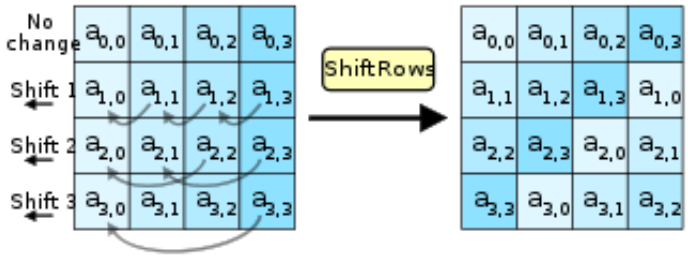

Figure 4 Shift row operation

3.3 Mix column

It is a transformation by which every input byte influences 4 bytes at the output. This is the major diffusion operation in AES and after 3 rounds ,it is made sure that every byte of state matrix depends on all 16 plain text bytes. Multiplication of polynomial is done in GF(2^8) with modulo irreducible polynomial, P(x)= $x^8+x^4+x^3+x+1$, addition of polynomial is bitwise XOR. This will make sure that every byte from state will return a byte after the operation.

$$\begin{bmatrix} a1' \\ a2' \\ a3' \\ a4' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a1 \\ a2 \\ a3 \\ a4 \end{bmatrix}$$

Figure 5. Mix column operation

4 Encryption Flowchart

The above said sub modules can be stacked in the given fashion to create an encryption block. This is for naive implementation, when we don't have any restrictions on area. There are 10 rounds when key length is 128 bits. The last round doesn't have mix column operation.

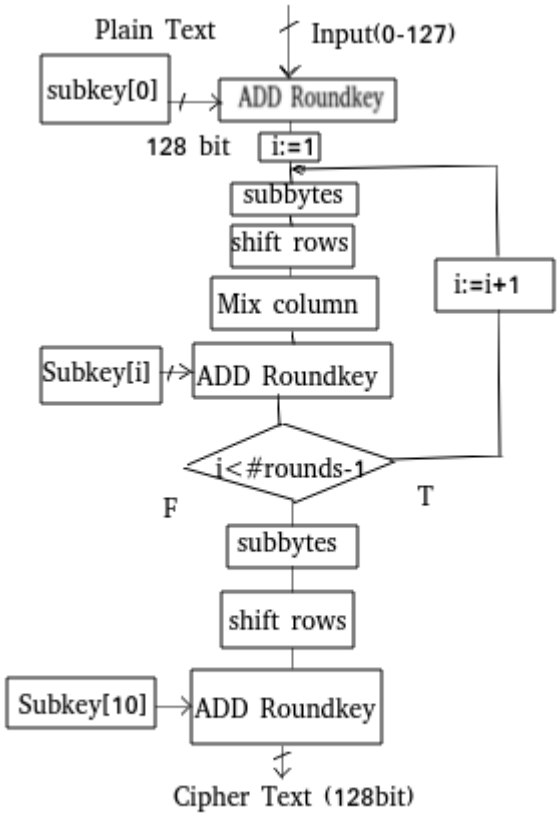

Figure 6. Target Image

5 Decryption

Decryption of the cipher text can be done by replacing the operations in the encryption flowchart by their inverse operation and reversing the operation order. Every operations done in encryption has got their inverse operation.

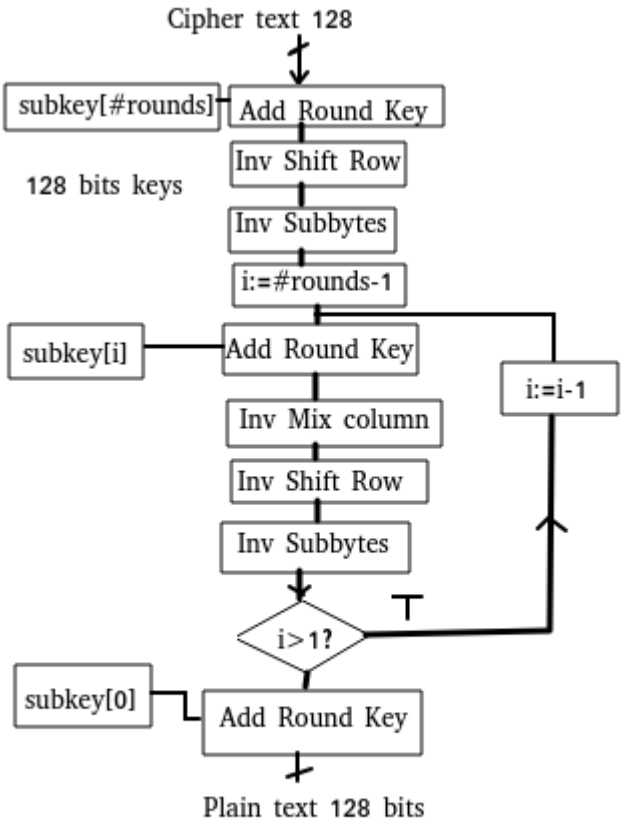

Figure 7. Decryption flowchart

5.1 Inverse subbytes

Inverse S block table can be used.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
dx	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 8. Inverse S block

5.2 Inverse shift row

It shifts second, third and fourth rows of the state matrix to right by 1, 2 and 3 units, this will result in new state matrix. This is exactly the opposite of the operation done during shift row.

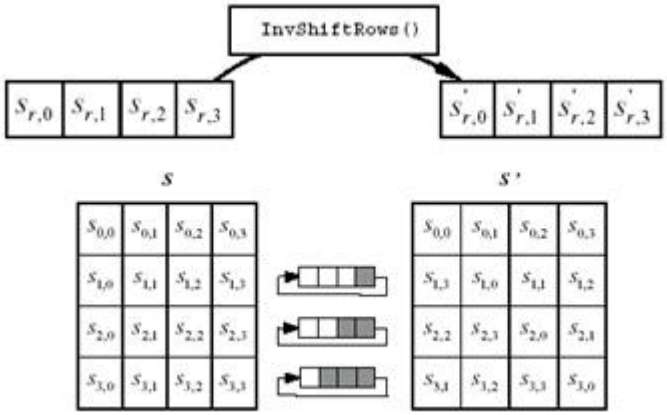

Figure 9. Inverse shift row

5.3 Inverse Mix column

Inverse mix column multiplies each column of the state by matrix of polynomial in GF(2^8). Multiplication is modulo P(x) multiplication and addition is bitwise XOR, so we end up getting 8 bytes for each 8 byte input.P(x)=$x^8+x^4+x^3+x+1$.

$$\begin{bmatrix} a1' \\ a2' \\ a3' \\ a4' \end{bmatrix} = \begin{bmatrix} 0E\ 0B\ 0D\ 09 \\ 09\ 0E\ 0B\ 0D \\ 0D\ 09\ 0E\ 0B \\ 0B\ 0D\ 09\ 0E \end{bmatrix} \begin{bmatrix} a1 \\ a2 \\ a3 \\ a4 \end{bmatrix}$$

Figure 10. Inverse mix column

Here a_1 ,a_2 ,a_3 and a_4 are the current bytes of a column in state matrix and a'_1 ,a'_2 ,a'_3 and a'_4 are their transformed version

6 Implementation detail

Here pipelined version of AES unit is coded and simulated. Registers are placed after every rounds, so there is a latency 10 clock periods and throughput is 1 output (128 bits) per every clock period. Inputs to the entity are endcbar, clk, input$(0-127)$ and output$(0-127)$. If endcbar='1', then input is considered as plain text and output obtained after latency will be cipher text. And if endcbar='0' then input is cipher text and output obtained will be plain text.

7 Simulation result

7.1 Encryption

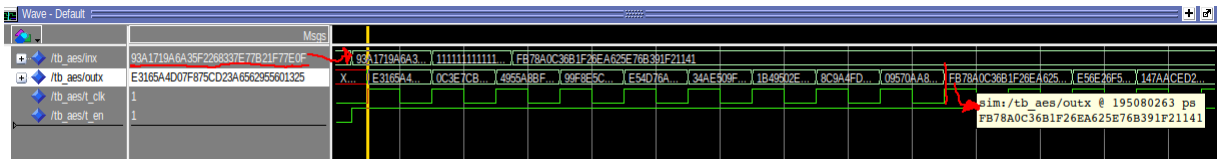

Figure 11.a. RTL simulation of encryption mode

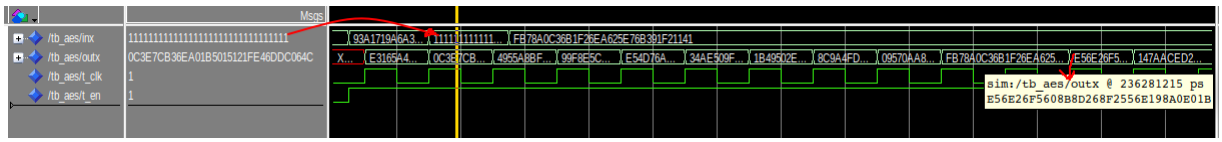

Figure 11.b. RTL simulation of encryption mode

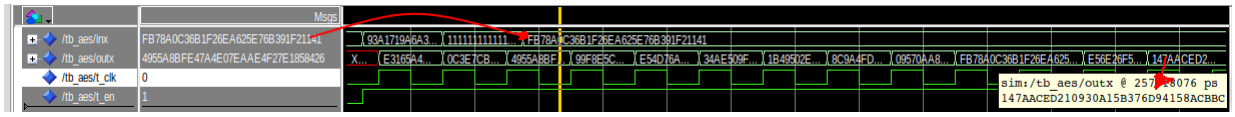

Figure 11.c. RTL simulation of encryption mode

7.2 Decryption

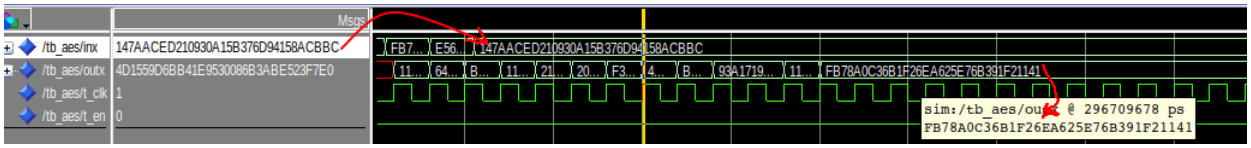

Figure 12.a. RTL simulation of decryption mode

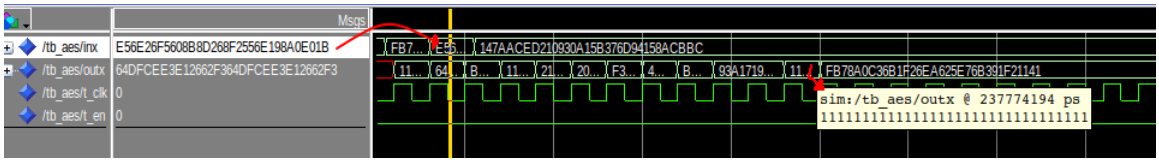

Figure 12.b. RTL simulation of decryption mode

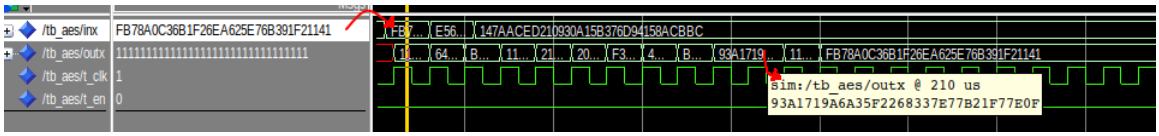

Figure 12.c. RTL simulation of decryption mode

8 Tabulation

Plain Text	Cipher Text
93A1719A6A35F2268337E77B21F77E0F	FB78A0C36B1F26EA625E76B391F21141
11111111111111111111111111111111	E56E26F5608B8D268F2556E198A0E01B
FB78A0C36B1F26EA625E76B391F21141	147AACED210930A15B376D94158ACBBC

Figure 13. Cipher text obtained after using a key =X"11111111111111111111111111111111"

Cipher Text	Plain Text
FB78A0C36B1F26EA625E76B391F21141	93A1719A6A35F2268337E77B21F77E0F
E56E26F5608B8D268F2556E198A0E01B	11111111111111111111111111111111
147AACED210930A15B376D94158ACBBC	FB78A0C36B1F26EA625E76B391F21141

Figure 14. Plain text retrieved after using a key =X"11111111111111111111111111111111"

9 Reference

. Federal Information Processing Standards Publication 197 ,AES
. http://www.facweb.iitkgp.ac.in/ sourav/AES.pdf