California State University, Northridge

Department of Electrical & Computer Engineering

ECE 526 Final Project Report

HDL and Cryptography

May 9, 2024

Professor: O. Haghighiara

Written By:

John Truong

INTRODUCTION:

In this experiment, I implemented an Advanced Encryption Standard (AES) algorithm, specifically the encryption process. The AES encryption process was modeled through a hierarchical process using instantiation. The experiment also tested the usage of indexed part selects for indexing vectors as a way to implement generate statements as referencing bit slices with a non-constant value in Verilog is not allowed. This experiment also utilized User Defined Function (UDF) for certain modules for arithmetical or logical functionality.

METHODOLOGY

The encryption process of the AES algorithm is shown in **Figure 1.0** and shows four different algorithms. This process begins by storing the input plaintext into a 4x4 state array which can be simplified in **Figure 1.1**. Afterward, the AES algorithm begins the key expansion or Rijndael's key schedule. This schedule utilizes four key operations, an 8-bit circular rotation on a 32-bit word, a round constant operation (rcon) that is introduced to destroy any symmetries that the other steps in the key expansion algorithm may have introduced, an s-box operation, and a key schedule which is too lengthy for this report but is described in Sam Tenholme's blog listed in my references. Because of its lengthy and outright complicated application, I utilized Trenholme's, Michael Ehab's Github, and the National Institute of Standards and Technology's publication as a reference for the implementation of the key expansion nevertheless, after the key expansion at the start of the encryption rounds which utilize four different algorithms.

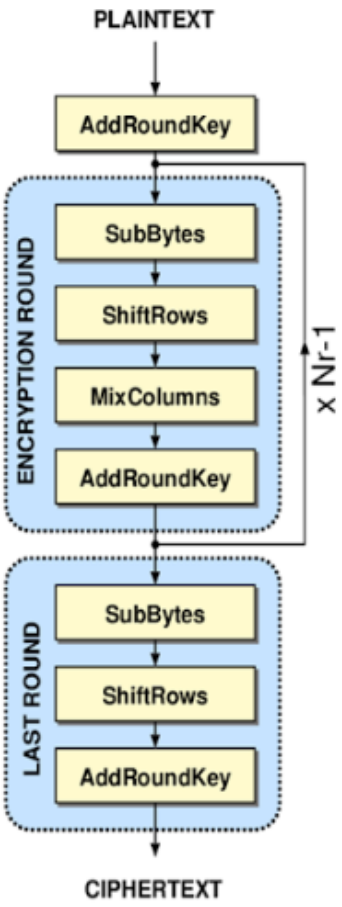


Figure 1.0 Encryption Process

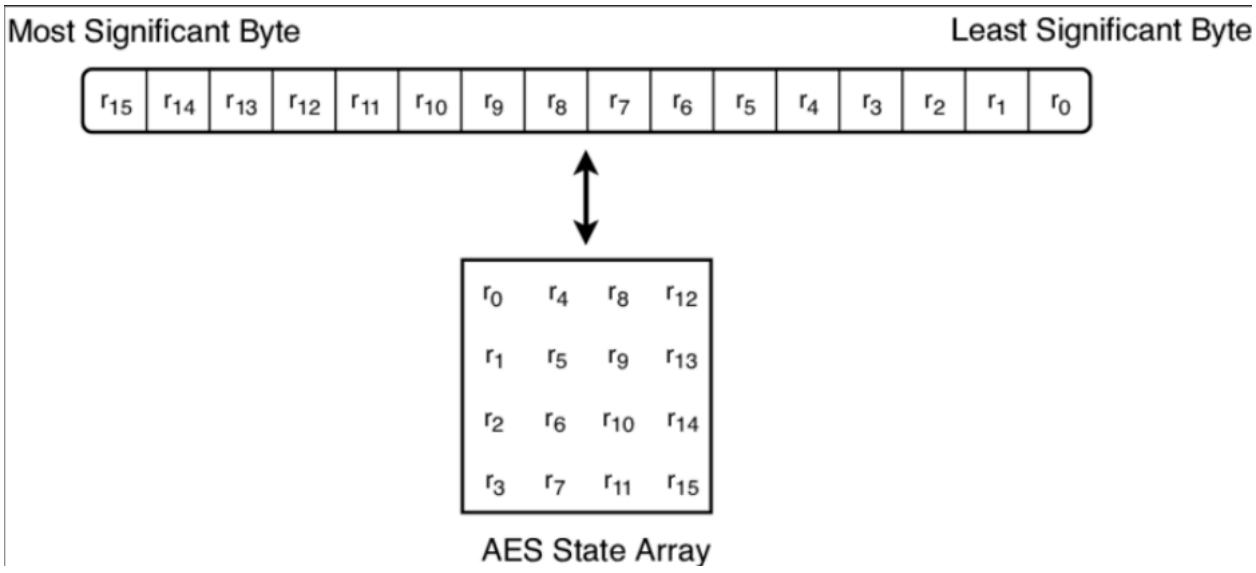


Figure 1.1 State Array Simplification

AddRoundKey is an algorithm that simply XORs the input state array with the input key. This operation in essence adds the key to the input state array due to the XOR being crucial for binary addition. Thus the implementation can be done with an XOR operation with the input and key.

Next, the SubBytes algorithm substitutes each byte of the state array with the corresponding s-box value equivalent which is shown in **Figure 1.2**. For the implementation, it can be done by instantiating the s-box 16 times to substitute all 128 bits. To shorten the amount of code I utilized Verilog's generate block to for-loop repeatedly 8 bits (1 byte) to the instantiated s-box module which substitutes a given value.

AES S-Box

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

Figure 7. S-box: substitution values for the byte xy (in hexadecimal format).

Figure 1.2 AES S-box

Following that is the ShiftRows algorithm which given a 4x4 state array will shift each row by r = n - 1, where n = row # (counting from 1~). This implementation can be done by choosing the necessary 8 bits for each row and shifting them depending on r.

After ShiftRows is the MixColumns algorithm which multiplies the input state array with a constant matrix shown in **Figure 1.3**. The matrix multiplication is done by multiplying the state array's column with the constant matrix's row. Doing this would produce a new state array's column which when repeated four times would produce a new state array. As the predefined constant matrix contains constants 1, 2, and 3 I would create UDF for them (except for constant 1) so that it can be reused for each state array column. After that, I would use the generate block to instantiate through each column where i represents the column number (from 0 onward). I would then multiply the constant matrix row with the state array's column and add them together through an XOR operation which can visualized by the equation in **Figure 1.4**. Due to the splicing of the bits now involving a non-constant variable in the form of i, I would now use indexed part select to select the range of bits that contains each byte of the state array.

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Figure 1.3 Constant Matrix

$$d_0 = 2 \bullet b_0 \oplus 3 \bullet b_1 \oplus 1 \bullet b_2 \oplus 1 \bullet b_3$$
$$d_1 = 1 \bullet b_0 \oplus 2 \bullet b_1 \oplus 3 \bullet b_2 \oplus 1 \bullet b_3$$
$$d_2 = 1 \bullet b_0 \oplus 1 \bullet b_1 \oplus 2 \bullet b_2 \oplus 3 \bullet b_3$$
$$d_3 = 3 \bullet b_0 \oplus 1 \bullet b_1 \oplus 1 \bullet b_2 \oplus 2 \bullet b_3$$

Figure 1.4 Constant Matrix Representation

All these four algorithms are instantiated in a top module which generate for-loops by the number of respective rounds for each variable key size. With the final round instantiating everything except the mixColumns algorithm.

My test plan involved utilizing input values from outside resources either through GitHub or other online resources. This was done to ensure the proper functioning of my design. There were two inputs shown below, which were used to test the functionality of my algorithm with key sizes 128, 192, and 256 bits which are also shown below. I confirmed correctness by comparing the output with a value labeled expected###.

in = 128'h_3243f6a8885a308d313198a2e0370734;

in2 = 128'h_00112233445566778899aabbccddeeff;

key128 = 128'h_2b7e151628aed2a6abf7158809cf4f3c;

key192 = 192'h_000102030405060708090a0b0c0d0e0f1011121314151617;

key256 = v256'h_000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f;

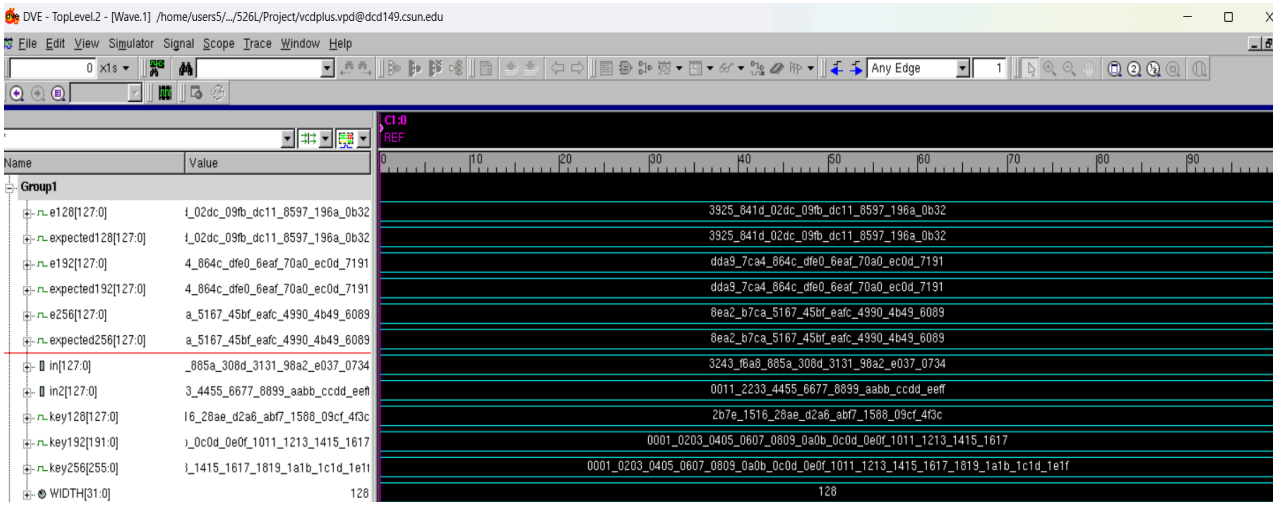
expected128 = 128'h_3925841d02dc09fbdc118597196a0b32;

expected192 = 128'h_dda97ca4864cdfe06eaf70a0ec0d7191;

expected256 = 128'h_8ea2b7ca516745bfeafc49904b496089;

ANALYSIS:

The results of my tests are shown in the waveform in **Figure 1.5** which outputs both the expected and resultant output of my design. My tests involved all three variable key sizes and the results and expected results are explicitly labeled by their key sizes.



Something to note is that the result for each key size is still 128 bits meaning the input and output are a fixed width of 128 bits. Based on the results it would appear that my design is working as intended as the expected and resultant outputs are the same for each key size.

CONCLUSION:

In conclusion, this experiment has allowed me to implement a standardized encryption process using Verilog. It has also allowed me to utilize generate blocks, indexed part select, and UDFs for the design. Overall, I have gained a practical amount of time to utilize what I have learned to research and implement the AES algorithm.

I hereby attest that this lab report is entirely my own work. I have not copied either code or text from anyone, nor have I allowed or will I allow anyone to copy my work.

Name (printed) John

Name (signed) John Date 05/11/24

References:

"An Improvement of Both Security and Reliability for AES Implementations." *Journal of King Saud University - Computer and Information Sciences*, Elsevier, 13 Jan. 2022, www.sciencedirect.com/science/article/pii/S1319157821003578.

Jena, Baivab Kumar. "What Is AES Encryption and How Does It Work? - Simplilearn." *Simplilearn.Com*, Simplilearn, 9 Feb. 2023, www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption.

Michaelehab. "Michaelehab/AES-Verilog: Advanced Encryption Standard (AES128, AES192, AES256) Encryption and Decryption Implementation in Verilog HDL." *GitHub*, github.com/michaelehab/AES-Verilog. Accessed 11 May 2024.

Moser, Jeff. *A Stick Figure Guide to the Advanced Encryption Standard (AES)*, www.moserware.com/2009/09/stick-figure-guide-to-advanced.html. Accessed 11 May 2024.

National Institute of Standards and Technology. "Advanced Encryption Standard (AES)." *CSRC*, 9 May 2023, csrc.nist.gov/pubs/fips/197/final.

Pnvamshi. "Pnvamshi/Hardware-Implementation-of-AES-Verilog: Hardware Implementation of Advanced Encryption Standard Algorithm in Verilog." *GitHub*, github.com/pnvamshi/Hardware-Implementation-of-AES-Verilog. Accessed 11 May 2024.

Rimkiene, Ruta. *What Is AES Encryption and How Does It Work? | Cybernews*, cybernews.com/resources/what-is-aes-encryption/. Accessed 11 May 2024.

Singh, Kirat Pal, and Shiwani Dod. *An Efficient Hardware Design and Implementation Of ...*, eprint.iacr.org/2016/789.pdf. Accessed 11 May 2024.

Trenholme, Sam. "Rijndael's Key Schedule." *Sam Trenholme RSS*, www.samiam.org/key-schedule.html. Accessed 11 May 2024.