2020 3rd International Conference on Data Intelligence and Security (ICDIS) | 978-1-7281-9379-3/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDIS50059.2020.00009

A Fast Implementation of the Rijndael Substitution Box for Cryptographic AES

Aaron Barrera
Department of Electrical and Computer Engineering
The University of Texas Rio Grande Valley
Edinburg, TX, USA
ORCID ID: 0000-0001-8764-9272

Chu-Wen Cheng
Department of Electrical and Computer Engineering
The University of Texas Rio Grande Valley
Edinburg, TX, USA
ORCID ID: 0000-0003-3156-3255

Dr. Sanjeev Kumar
Department of Electrical and Computer Engineering
The University of Texas Rio Grande Valley
Edinburg, TX, USA
Email: SJ.KUMAR@UTRGV.EDU

Abstract—**Today's current standard in Cryptography is the symmetric Advanced Encryption Standard (AES) as selected by the US National Institute of Standards and Technology (NIST). It is also known as Rijndael Encryption Algorithm. With the emergence of high-performance cloud computing, it is important for the security encryption schemes to perform at higher speeds to provide fast, efficient, and secure data transmissions. The Rijndael S-box (substitution box) is the only non-linear component of the cryptosystem and significantly affects the overall performance of the AES encryption scheme. In this paper, we investigate various implementations for improving the hardware performance of the Rijndael S-box component of the AES algorithm in terms of delay and number of logic elements on the Altera Cyclone IV FPGA (Field programmable gate arrays) using the Intel Quartus II software and Verilog Hardware Description Language (Verilog HDL).**

Index Terms – Cyber Security, Cryptography, Encryption, Advanced Encryption Standard (AES), FPGA, Altera Cyclone IV, Verilog HDL, Intel Quartus II, Rijndael Affine Field

I. Introduction

The AES encryption algorithm accepts blocks of 128 or 192 or 256 bits and applies a series of substitutions and permutations [1-3]. A special substitution termed as "SubBytes Transformation" is also called Rijndael S-box, named after its designers. S-box is the main core structure of every block cipher system and controls the hardware complexity of Rijndael cipher elements due to its particular characteristics, a non-linear byte substitution and operating on each of the State bytes independently. The purpose of S-box is to produce confusion between the ciphertext and the secret key. There are 256 = 16 × 16 possible 8-bit numbers, and so the S-box can be represented as a 16 ×16 table mapping inputs to outputs.

S-box provides reversible conversion of plain text segments during the encryption process, while providing the opposite conversion during the decryption process. It is a single simple function that is applied to each byte over and over again during the encryption phase. Each of the 256 possible byte values is converted to another byte value by the transformation, which is a complete permutation as shown as Figure 1. As a result, no two different byte values are changed to the same byte values.

The AES S-box is shown as Figure 2[1-2]. To find the output from the S-box table, the byte input is split into two 4-bit halves. The first half provides the row number and the second half provides the column number for the Byte substitution. For example, an S-box transformation of 8'b11000011or 0xC3 can be found in a cell at the intersection of a row labeled C0 in hexadecimal and a column labeled 03 in hexadecimal. Therefore, 8'b11000011or 0xC3 becomes 8'b00101110 or 0x2E.

Several hardware implementation related work is available in literature [3]–[9]. Many literatures have proposed S-box hardware lookup table implementations [10-16]. To reduce LUT space requirements, the basic idea of Shannon's expansion theorem is applied. It helps achieve a logical design that has a greater number of levels with less implementation cost. This optimization technique reduces the complexity of the whole S-box which means it requires fewer arithmetic operations. As it simplifies table indexing, it simply consumes less power. Besides this optimization technique, including a smaller number of iterations will decrease the delay producing algebraic and matrix operations. In this paper, we simulate different design techniques for the AES non-linear byte substitution in the Quartus-II simulator for a Cyclone IV FPGA platform. We verified the output performance of various implementations of the S-box in terms of delay and size.

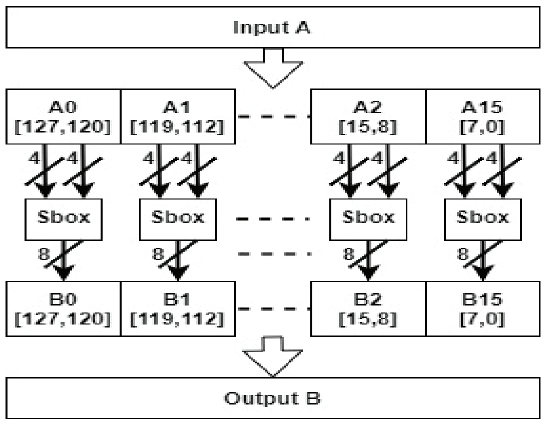
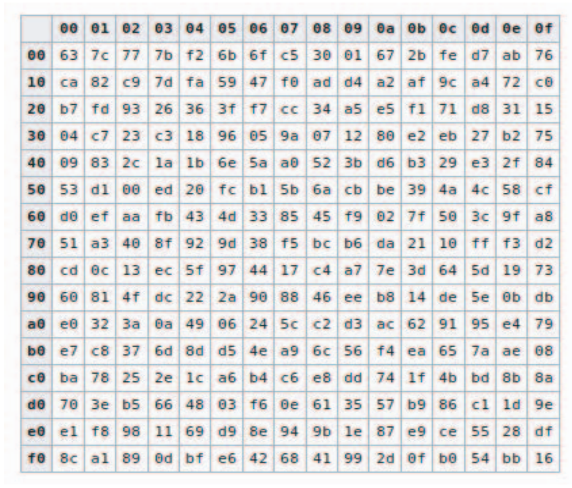


Figure 1: Substitute Bytes



	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 2: S-box table [1-2]

$$\begin{bmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \end{bmatrix} \times \left[\begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \right]$$

Figure 3: S-box Matrix Computation in GF (2^8)

II. RIJNDAEL S-BOX COMPUTATION

S-box treats the values as a polynomial in Galois Field in factor 2^8(GF (2^8)) [1-3], irreducible polynomial uses $x^8 + x^4 + x^3 + x^1 + 1$. It's calculation basically involves two steps:

1. The inverse multiplication: derived from multiplicative inverse over GF (2^8). "00" is mapped to itself.
2. The affine transformation: applying the affine (on GF (2^8)) transformation.

Using Rijndael's finite field for affine transformations the following expresses it as an equation for an input vector signal "X".

$$GF(2^8) = \frac{GF(2)[X]}{(x^8+x^4+x^2+x+1)} \qquad \text{-(Equation1.1)}$$

Equation 1.1 shows the affine transform in Galois Field (2^8) as a function, with the characteristic irreducible polynomial as the denominator. This polynomial is represented in hardware as the binary string "100011011" for Boolean addition operations in the algorithm. The matrix representation of this function for S-box is shown in Figure 3. The input signal "b[7:0]" represents the 8-bit multiplicative inverse vector which undergoes Boolean XOR and addition operations to obtain the S-box output vector "a[7:0]." Figure 3 has various equations displayed in Boolean logic for quick implementation that represent the vector computation that occurs within the Rijndael multiplication and are shown in sequence as equations below.

$$s = b \oplus (b \ll 1) \oplus (b \ll 2) \oplus (b \ll 3) \oplus (b \ll 4) \ \oplus 63_{16}$$

-(Equation 1.2)

$$s_i = \ b_i \ \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \ \oplus \ c_i \qquad \text{-(Equation 1.3)}$$

$$s = (b \ x \ 31_d) \ mod \ 257_d \oplus 99_d \qquad \text{-(Equation 1.4)}$$

However, the Rijndael S-box is not economical because it is based on the LUTs and uses more resources during implementation. To achieve high throughput and low power consumption, many literatures have proposed S-box hardware lookup table implementations [8-16]. In this paper, we focused on how to efficiently implement pipeline technology utilizing FPGA platform to make S-box fast and verify the performance of various implementations of S-box in terms of latency and size.

III. EXPERIMENTAL SETUP

We use the EDA tools available on the Altera website to evaluate our designs [4-7]. The Intel Quartus II Web Software and the University Program Simulator, allow code to be built, compiled, synthesized, simulated, and programmed into the designated FPGA hardware with optional input elements. In this work, we use Altera's Cyclone IV FPGA on the DE2-115 board with the module hierarchy implemented on the EP4CE115F29 model shown as Figure 4. This model for Cyclone IV has a density of 114,480 LE's and it contains an internal 50 MHz clock which is used during simulations.

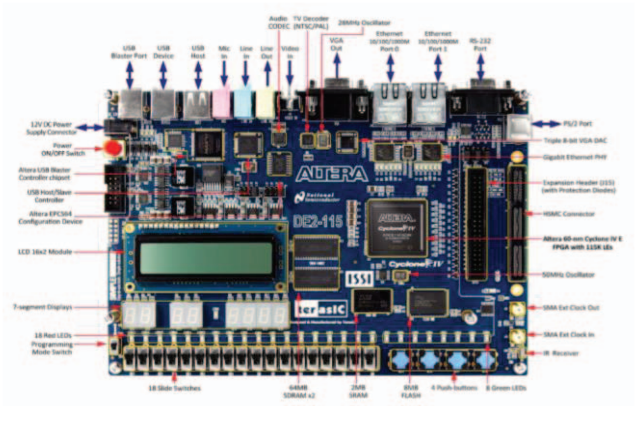


Figure. 4: Altera Cyclone IV 4CE115 FPGA Device [6]

IV. DESIGN METHODS AND DISCUSSION

This research proposes three unique new designs based on restricting the way the S-box is implemented [1-2]. The baseline implementation is a 256-line Look-Up-Table (LUT) that is conducted using sequential logic. Using the LUT in the design logic can significantly impact the amount of logic elements (LE's) that get used up by the FPGA. This method of adopting the basic principles of Shannon's expansion theorem achieves a logical design, with a greater number of levels, reduces the space requirement of the LUT, and lowers the implementation cost. This optimization technique reduces the complexity of the S-box module which results in fewer arithmetic operations. As it simplifies table indexing, it additionally consumes less power. This design, along with the reduced number of iterations, significantly lowers the delay for the algebraic and matrix operations. The designs are shown in the figures below and depict how the S-box is segmented to create smaller LUT's combined with multiplexer (MUX) logic for the correct output selection.

Baseline: The baseline Verilog HDL module implements the Substitution Box as a single LUT. The Register Translation Language (RTL) shows the baseline LUT module in Figure 5. This module is a direct implementation of the standard Rijndael Substitution Box shown in Figure 2, where an 8-bit input vector corresponds to a specific 8-bit output vector. This module relies on sequential logic, which consumes both a significant amount of hardware Logic Elements (LE's) and processing time from input to output.

Approach 1 (Design 1 and 2): Parallelization of S-Box

In this approach, we aim to parallelize the Substitution box by determining each corresponding 4-bit output in the same amount of clock cycles. There are two variations. Design 1- this approach for the Substitution Box focused on separating the rows and columns of the 256-byte LUT (Figures 6-8). Each value in the row and column of the Substitution Box uses a nibble of the input 8-bit vector to determine the corresponding substitute value. This design utilizes 2 LUT's for each nibble (HI and LO) of the input 8-bit vector and generates 16 possible output values that feed into a 16-to-1 Multiplexer (MUX) which uses the opposite nibble to select the correct 4-bit output. The module then concatenates the output of each MUX to create the corresponding output 8-bit vector. Design 2- In this variation, shown in Figures 9-11, extends this concept and creates 4 LUT's instead of 2 (as in Design1). This design uses smaller LUT's but introduces MUX logic which affects the trade-off between cost and delay reduction.

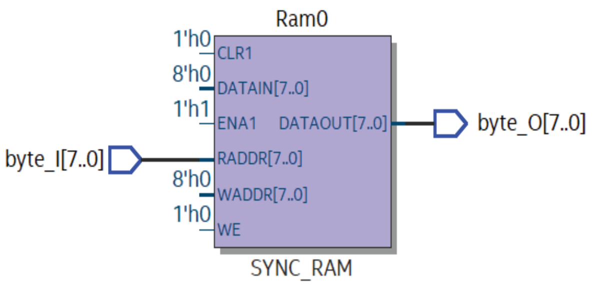
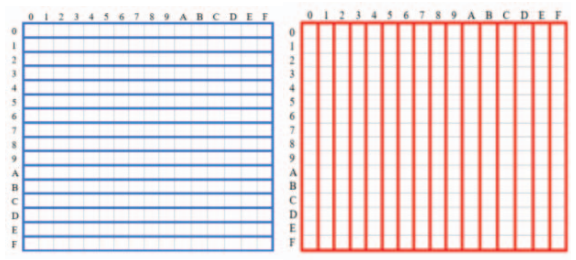


Figure 5: Baseline RTL



0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3 4 5 6 7 8 9 A B C D E F

Figure 6: Design 1 S-box Segmentation

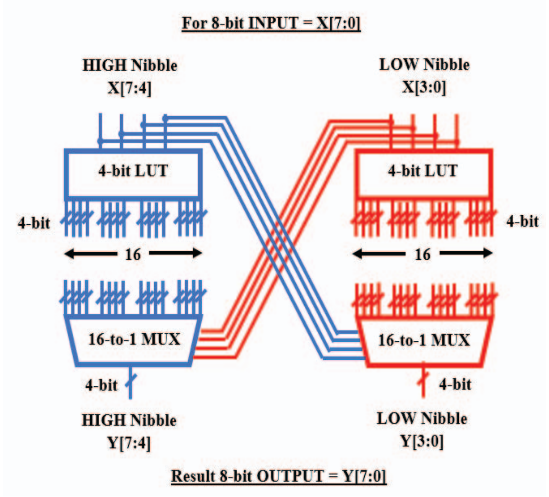


Figure 7: Design 1 Module Map

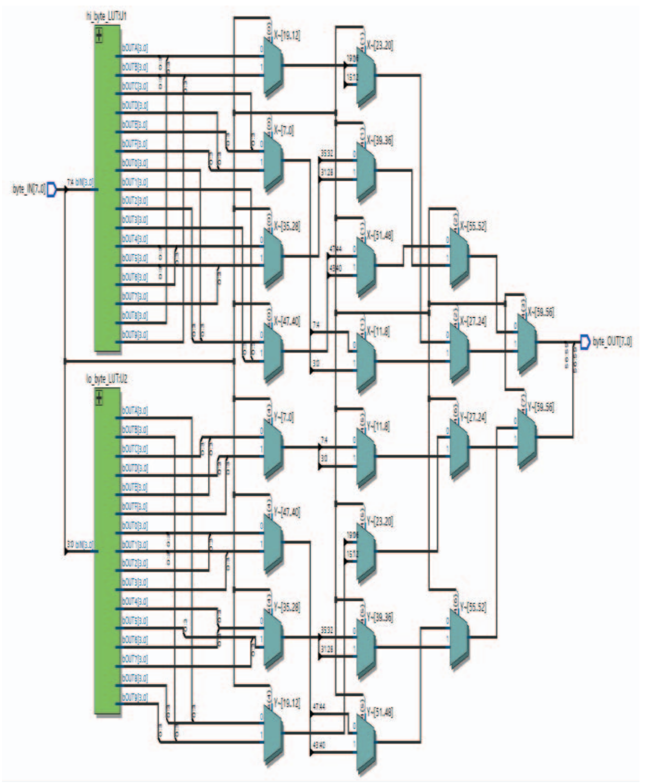


Figure 8: Design 1 RTL

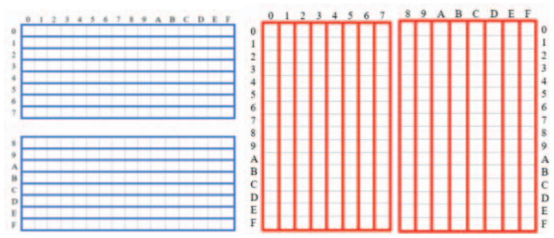


Figure 9: Design 2 S-box Segmentation

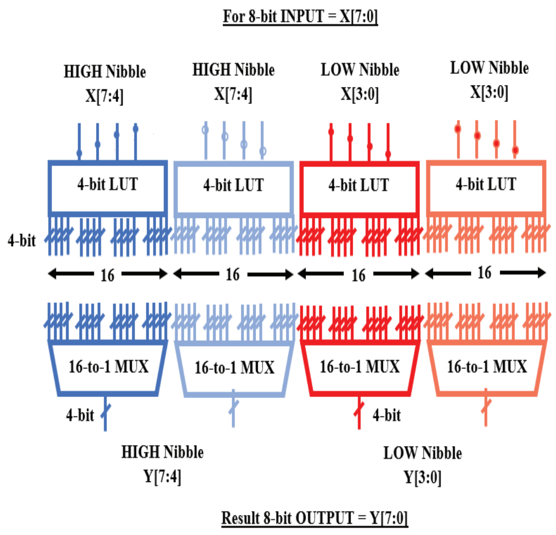


Figure 10: Design 2 Module Map

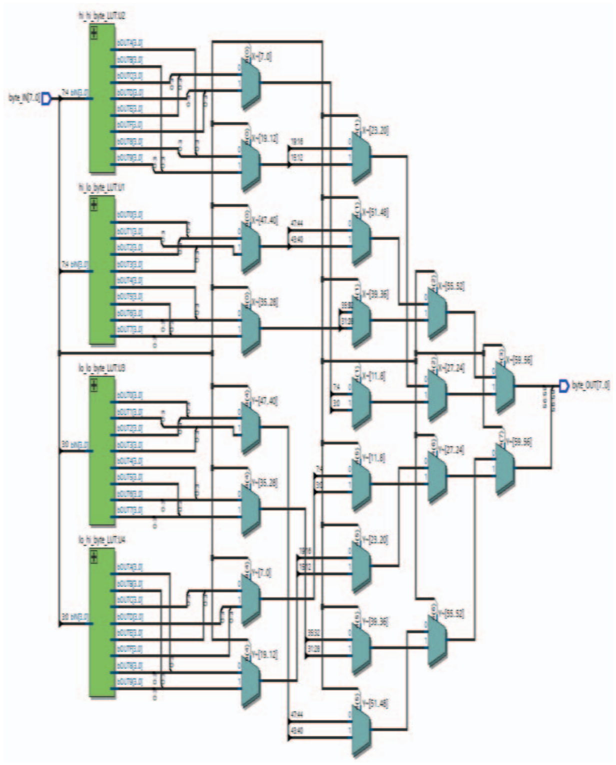


Figure 11: Design 2 RTL

Approach 2 (Design 3 and 4): The second approach, shown in Figure 12-13, focuses on creating smaller LUT's from the standard Substitution Box shown in Figure 2. The standard Substitution Box is separated into 4 LUT's one for each 8-byte by 8-byte quadrant. The 8-bit input vector goes through each of the 4 LUT's with only a single LUT generating the correct output vector. The output of all LUT's are then fed into a 4-to-1 byte-wise adder that combines all values to determine the output 8-bit substitution value. The aim of this approach is like approach 1's intent, which is to parallelize the process of determining the correct value by reducing the sequential logic case statement's memory consumption. The use of an adder to determine the output increases the LE cost of the module but significantly reduces the processing delay.

Approach 3 (Design 5): This design, shown in Figure 14, is a Verilog HDL implementation in Quartus based off the research presented in [4] and evaluated on the Cyclone IV FPGA versus the Virtex-4 and Virtex-5 FPGA's. This design uses Shannon's expansion to reduce the processing time and cost consumption of the Substitution Box. The initial baseline of this design uses the 8th and 7th bit of the input vector to determine the correct output substituted vector through MUX logic and utilizes 4 possible 6-byte x 6-byte LUT's. This design is then extended by using the 8th, 7th, and 6th bits of the input vector through MUX logic similar to the baseline.

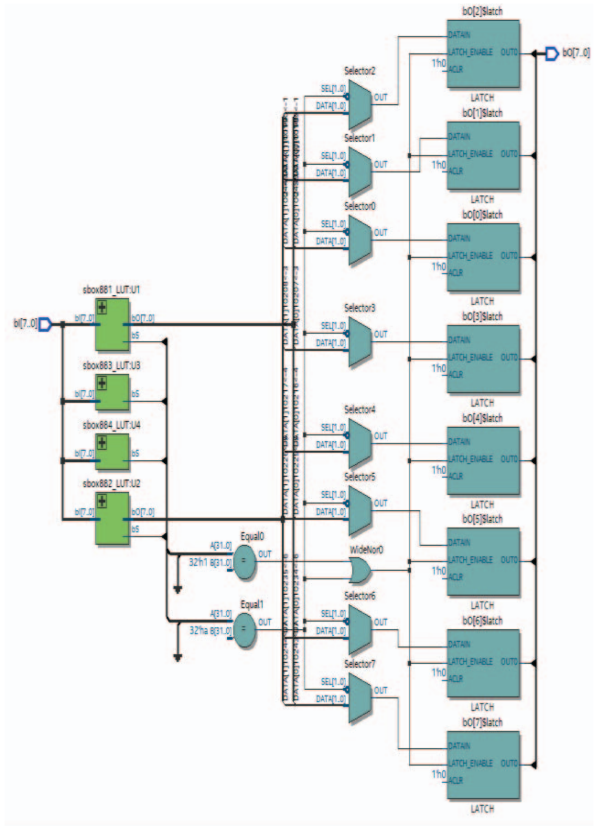


Figure 12: Design 3 RTL

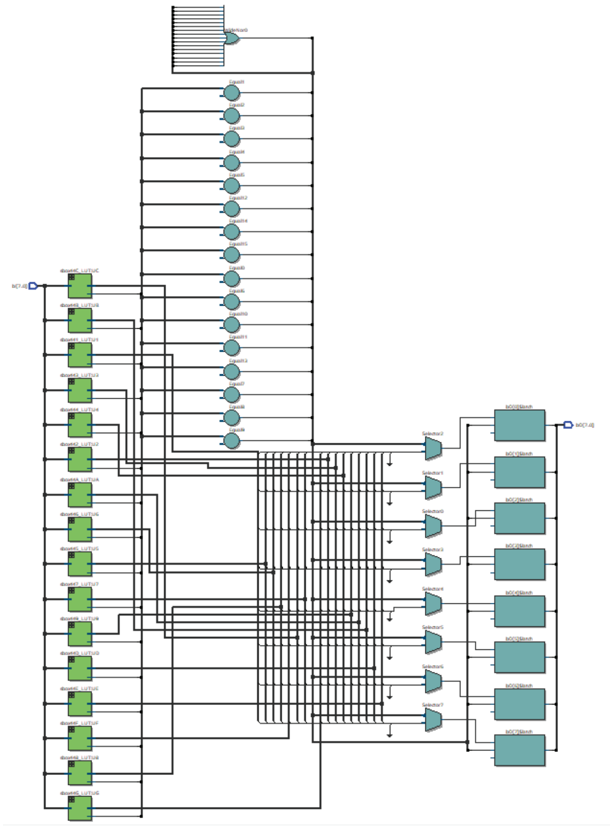


Figure 13: Design 4 RTL

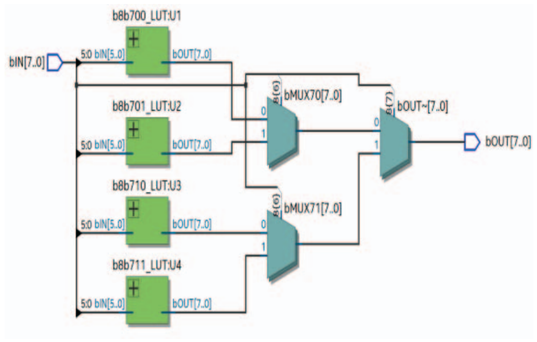


Figure 14: Design 5 RTL

V. RESULT AND ANALYSIS

The implementation of all Substitution Box designs was simulated on the Cyclone IV FPGA using NIST test vectors; the results are shown in Table 1. The baseline design is the initial direct implementation of the Rijndael Substitution Box from [1-2]. The baseline generated an average of 11.41 nanoseconds of delay while consuming 208 LE's. The results from the first implementation, displayed in Table 1, showed an increase in both average delay by 0.11 ns for design 1 and 0.52 ns for design 2. Design 3 showed the most efficient output compared to the baseline LUT; design 3 was able to generate the correct output 1.08 ns faster and with a 31.3% decrease in LE hardware consumption.

The Substitute Bytes module is repeated a total of 200 times, shown in Table 2, in the final implementation of AES-128 [3]. Compared to the other submodules that make up the AES-128 algorithm, only the byte-wide 2-input XOR (XOR2) module was instanced more than Substitute Bytes by a total of 120 times (Table 2). To improve the AES-128 algorithm in efficiency and throughput, it is crucial to target the most repetitive submodule since improvements in its implementation would be magnified. However, the XOR2 submodule is simply an expanded XOR gate that requires significantly less LE consumption than the Substitute Bytes module which is a 256-byte size LUT. The I/O delay and LE consumption for design 3 would be multiplied by the module count and would reduce the total delay of the AES encryption algorithm by approximately 200 ns compared to the baseline. The concept for design 3 and the other implementations, show the effectiveness of pipelining in processor operations. By segmenting the LUT into separate modules, each can be executed on the same clock cycle and reduce the overall delay of the process. This was expected to bring a trade-off between the hardware consumption to reduce the delay however design 3 was able to improve on both areas.

Many literatures have proposed S-box hardware lookup table implementations [10-16]. The logic design using the basic principles of Shannon's expansion theorem is achieved. By comparing our proposed designs to other reported techniques in literature [10-16], our analysis of the results indicates that our proposed design 3 is capable of significantly outperforming the other four designs in terms of delay and area as measured by the simulation. From the simulations of the proposed designs, we observed that the throughput can be increased by reducing the delay of the critical path taken by the input byte-wise vector specifically in the LUT segment. By targeting one of the most utilized modules in the AES computation scheme, the efficiency of the overall implemented algorithm can be improved on. The achieved lower LE cost and faster throughput will multiply based on the number of instances of the submodule included in the final design of AES-128 and can lead to improvements in the extended iterations of AES such as AES-192, AES-256, and AES-512.

Table 1: Design Parameter Comparison

Design	Average Delay (ns)	Logic Elements	Virtual Pins
Baseline	11.41	208 (<1%)	16 (3%)
Design 1	11.52	208 (<1%)	16 (3%)
Design 2	12.33	294 (<1%)	16 (3%)
Design 3	10.73	65 (<1%)	16 (3%)
Design 4	18.41	352 (<1%)	16 (3%)
Design 5	11.36	208 (<1%)	16 (3%)

Table 2: Total Module Count for AES-128

AES-128 Top Level Module				
Sub module	**Instances**	**Peak Virtual Memory (MB)**	**Pins/529**	**Logic Elements /114,480**
Substitute Bytes	**200**	**4788**	**16 (3%)**	**280 (<1%)**
Mix Column	**144**	**4770**	**16 (3%)**	**3 (<1%)**
XOR2	**320**	**4782**	**24 (5%)**	**8 (<1%)**
XOR4	**144**	**4770**	**40 (8%)**	**8 (<1%)**
Increment Bytes	**10**	**4789**	**70 (13%)**	**857 (<1%)**
Subkey Round	**10**	**4769**	**288 (54%)**	**128 (<1%)**

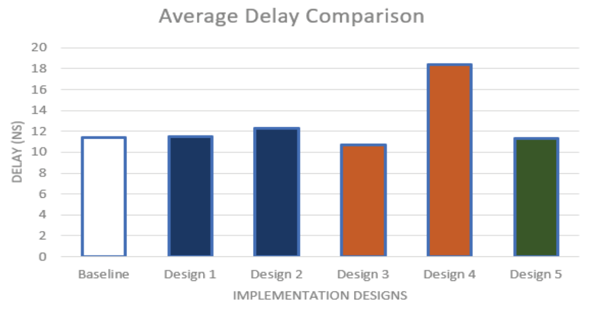


Figure 14: Average Delay (in nanoseconds) Comparison

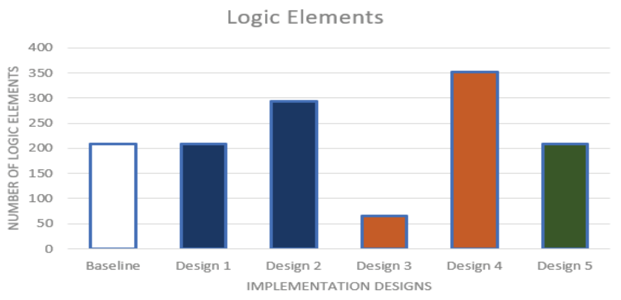


Figure 15: Comparison of Number of Logic Elements used

VI. CONCLUSION

This paper discusses the design and simulation of the AES non-linear byte substitution techniques. Even though, this topic has been studied a lot in the past. It is very difficult to compare different architectures objectively so comparing the number of LUTs, LEs, or delay on same device is the common way. From this perspective, by focusing on simulating the designs in the Quartus-II software on a Cyclone IV FPGA platform, we verified the output performance of various implementations of the S-box module in terms of delay and number of logic elements. From the simulations of the proposed designs, we observed that the throughput can be increased by reducing the delay of the critical path taken by the input byte-wise vector specifically in the LUT segment. We were able to generate the correct output 1.08 ns faster and with a 31.3% decrease in LE hardware consumption. The techniques proposed in our most optimized design 3 shows us how to achieve high throughput using the Shannon's expansion theorem on multiplexer synthesis in processor operations which can significantly impact the production market today for large data devices.

ACKNOWLEDGEMENTS

The support for this research is provided in part by the US National Science Foundation under Grant No. 041585 Houston Endowment Chair in Science, Math, and Technology Fellowship, and Lloyd Benston Jr. Endowment Fellowship.

REFERENCES

[1] "Advanced Encryption Standard (AES)." Federal Information Processing Standards, US National Institute of Standards and Technology, 26 November 2001, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[2] William Stallings. "Cryptography and Network Security 5th ed." ISBN-10: 0136097049.

[3] Aaron Barrera, Chu-Wen Cheng, Dr. Sanjeev Kumar. "Hardware Implementation of Improved Mix Column Computation of Cryptographic AES." Jun 2019, 2nd International Conference on Data Intelligence and Security.

[4] Altera (201) Quartus Prime Design Software, http://fpgasoftware.intel.com/17.0/?edition=lite

[5] John, L. (2016) "Altera Parts History. " University of California, Berkeley, http://www-inst.eecs.berkeley.edu/~cs294-59/fa10/resources/Altera-history/Alterahistory

[6] Altera (2017) Cyclone IV DE2-115 User Manual, Version 2.3, http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502&PartNo=4

[7] Altera DE2 Board, Terasic, N.P. (2016) , https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=502

[8] M.R.M. Rizk, M. Morsy. "Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA." 22 January 2008, https://ieeexplore.ieee.org/document/4437462/

[9] Del Rosal, Edni, and Sanjeev Kumar. "A Fast FPGA Implementation for Triple DES Encryption Scheme." Circuits and Systems 8.09 (2017): 237, https://m.scirp.org/papers/79306

[10] Sridevi Sathya Priya, S., Karthigaikumar, P. "FPGA implementation of High speed compact S-Box", https://acadpubl.eu/hub/2018-119-16/1/165.pdf

[11] Nalini C,Dr. Anandmohan P.V, Poomaiah D.V, and V.D.kulkami, "Compact Designs of SubBytes and MixColumn for AES", https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4809193

[12] Rijmen, Vincent. "Efficient Implementation of the Rijndael S-box." Katholieke Universiteit Leuven, Dept. ESAT. Belgium (2000). http://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/rijndael-sbox.pdf

[13] Satoh, Akashi, et al. "A compact Rijndael hardware architecture with S-box optimization." International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg, 2001. https://link.springer.com/content/pdf/10.1007%2F3-540-45682-1_15.pdf

[14] Mentens, Nele, et al. "A systematic evaluation of compact hardware implementations for the Rijndael S-box." Cryptographers' Track at the RSA Conference. Springer, Berlin, Heidelberg, 2005. https://www.esat.kuleuven.be/cosic/publications/article-560.pdf

[15] Canright, David. "A very compact S-box for AES." International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, 2005. https://www.iacr.org/archive/ches2005/032.pdf

[16] Leventis, Paul, et al. "Cyclone/spl trade: a low-cost, high-performance FPGA." Proceedings of the IEEE 2003 Custom Integrated Circuits Conference, 2003.. IEEE, 2003. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1249357