VERILOG IMPLEMENTATION OF AES

MEMBERS INVOLVED

PRABHAT REDDY (EE18B022)
SUDARSHAN H V (EE18B034)
UMAESHWER SHANKAR (EE18B036)
YASHAS S V (EE18B040)

PROJECT REPORT

ABSTRACT

Advanced Encryption Standard (AES) is a secure encryption used all around the world for different kinds of data transmission. AES-128 plays a major role in transmitting data over the internet, and has become extremely popular for regular use due to the balance between its security and performance.

Our project aims to implement 128-bit version of AES encryption algorithm using Verilog HDL. It contains synthesizable code that can be implemented on an FPGA board. Using pre-programmed hardware can improve security standards, as one cannot steal information via means of software.

INTRODUCTION

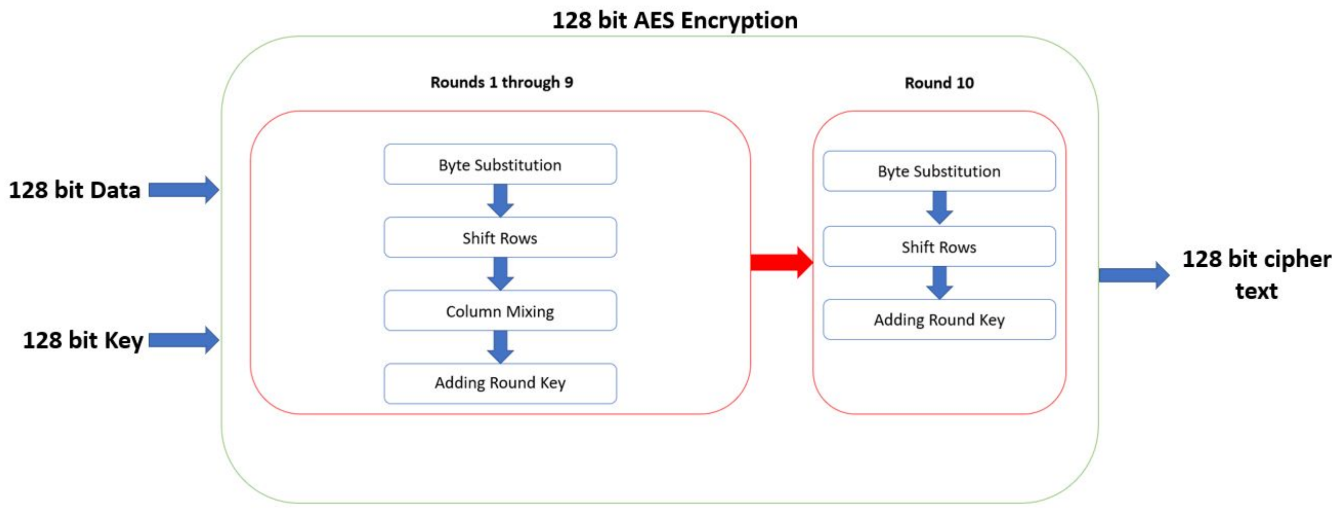
AES or Advanced Encryption Standard is a cipher, i.e., a method for encrypting and decrypting information. It supersedes the Data Encryption Standard(DES), and is better in terms of both security and performance. AES was established by the U.S. National Institute of Standards and Technology (NIST) in 2001.

AES is a part of the Rijndael block cipher, after its creators Vincent Rijmen and Joan Daemen. A block cipher takes a block of fixed length as input, and gives an encrypted block of same length as output. It also makes use of a key of some fixed length to encrypt the input block. The algorithm described by AES is a ‘symmetric key algorithm’, meaning the same key is used for both encrypting and decrypting the data.

AES contains 3 members of the Rijndael cipher: AES-128, AES-192, and AES-256. All of them have the same block size, but different key-lengths which are given by the numbers in their names.

In the United States, AES was announced by the NIST as U.S. FIPS PUB 197 (FIPS 197) on November 26, 2001. This announcement followed a five-year standardization process in which fifteen competing designs were presented and evaluated, before the Rijndael cipher was selected as the most suitable. AES became effective as a federal government standard on May 26, 2002.

ALGORITHM



There are four key steps involved in the Rijndael cipher:

1. *Byte Substitution*
2. *Shift Rows*
3. *Column Mixing*
4. *Adding Round Key.*

For the first nine rounds, the steps 1 to 4 are implemented iteratively in each round. The last round does not have the step 3.

SOME TERMS BEFORE WE BEGIN:

➔ *State Array*: It is the array which is generated by converting 128 bit data into 16 bytes and then arranging it in a 4 x 4 matrix.

THE ACTUAL ENCRYPTION PROCESS:

The four steps listed below will be run in the first nine rounds. In the tenth round, the Column Mixing step is omitted.

STEP 1: BYTE SUBSTITUTION

Byte Substitution is the first major step in AES, and has a huge role in scrambling the data bytes to a large extent. There are two ways of implementing Byte Substitution. The first way is to construct a precomputed 16 x 16 look up table called the SBox, which directly substitutes the input byte with another random unique byte value. The second method involves a more traditional approach of constructing the table each and every time by using the algorithm to find the corresponding substitute byte.

We have decided to go with the first method as we feel that the main focus should be on the implementation of AES in Verilog, and we have to speed up by using the predetermined SBox. Our code takes a byte as an input, looks it up in the table, and outputs the corresponding byte in the SBox as output.

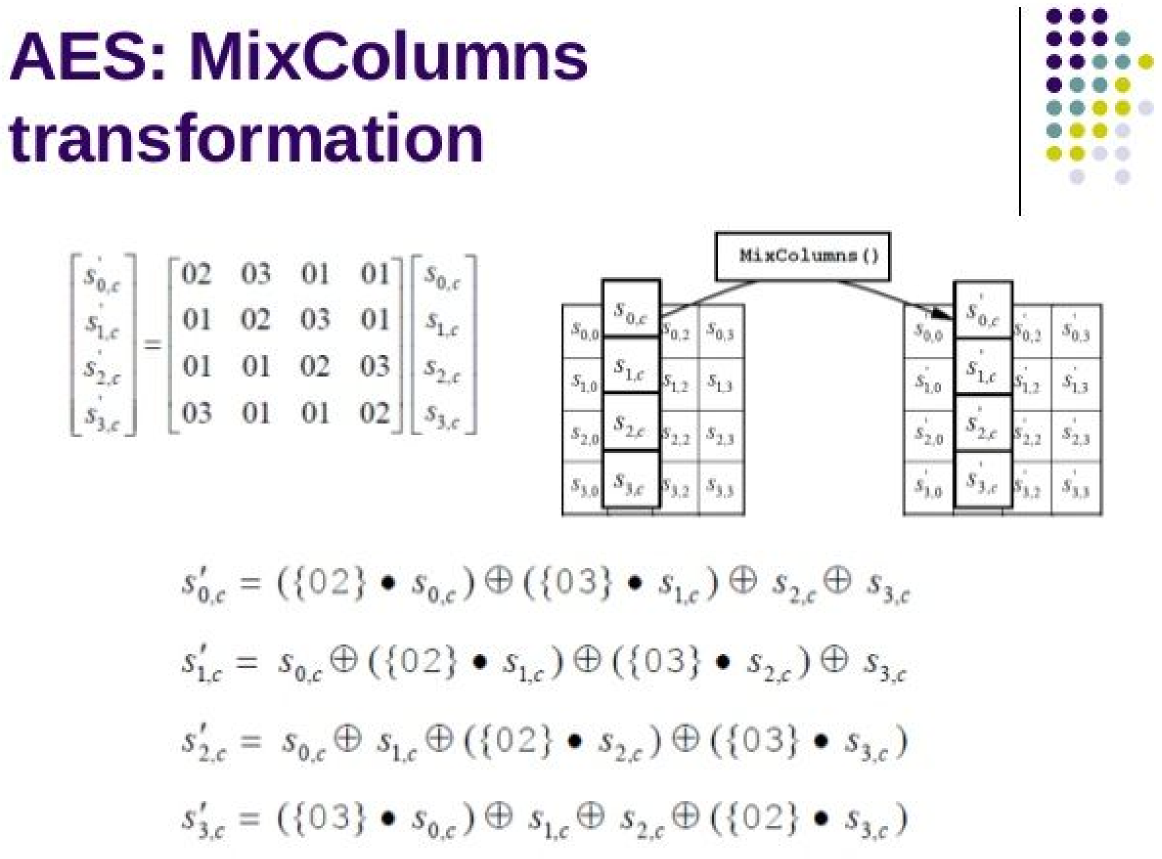
STEP 2: SHIFT ROWS

This step is a fairly simple step compared to the other steps. We take the 4 x 4 state array obtained after byte substitution. We leave the first row without modifying it. We perform a circular left shift on the second row by one byte, a circular left shift on the third row by two bytes and a circular left shift on the fourth row by three bytes. Though, we will not be implementing decryption, it is worthwhile to note that the reverse of this procedure is used during decryption

STEP 3: COLUMN MIXING

In this step, we basically rewrite every byte in the 4 x 4 state array after passing it through a function which expresses each byte in terms of all the bytes in that column.

For example, if we take the bytes of a specific row, we get the following relation.



AES: MixColumns transformation

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

MixColumns()

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$

Basically, we multiply the concerned byte by 3, add it with the next byte (in circular fashion) multiplied by 2, and add the remaining two bytes to get the result. Here, by addition, we use XOR operation in the Galois field GF(2^8), and multiplication refers to hexadecimal multiplication in GF (2^8) field.

The whole operation can also be expressed as a transformation matrix as shown below.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0.0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1.0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2.0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3.0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

STEP 4: **ADDING ROUND KEY**

In the Round key generation, we have two initial pieces of data. A 4 x 4 array of bytes

$$\begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} w_0 & w_1 & w_2 & w_3 \end{bmatrix}$$

$$\begin{aligned} w_{i+5} &= w_{i+4} \otimes w_{i+1} \\ w_{i+6} &= w_{i+5} \otimes w_{i+2} \\ w_{i+7} &= w_{i+6} \otimes w_{i+3} \end{aligned}$$

$$w_{i+4} = w_i \otimes g(w_{i+3})$$

(message to be encrypted) and a 4 x 4 array (128 bit) key. Each column of the key matrix will be grouped as a four byte word.

The algorithm then expands these four words into a 44 word key schedule using the following steps:

1. We determine the words w(i+5), w(i+6),w(i+7) by using the XOR operations mentioned in the figure above.
2. For w(i+4) we use a special function g(w).
3. The function g(w) does three things:
    a. Perform a left circular byte rotation on w.
    b. Get the byte substitution form of w by using the S Box.
    c. Then XOR the previous result with a Round Constant of the form 0xAB_00_00_00, where AB varies according the round number, and is predetermined.
4. Hence all the keys for each round are generated.

<u>VERILOG IMPLEMENTATION</u>

The entire encryption was divided into functions which were then called in the parent module. This was done to reduce the circuit area and increase the efficiency of the program. The following functions were defined:

- S-box
- ByteSubstitution
- ShiftRows

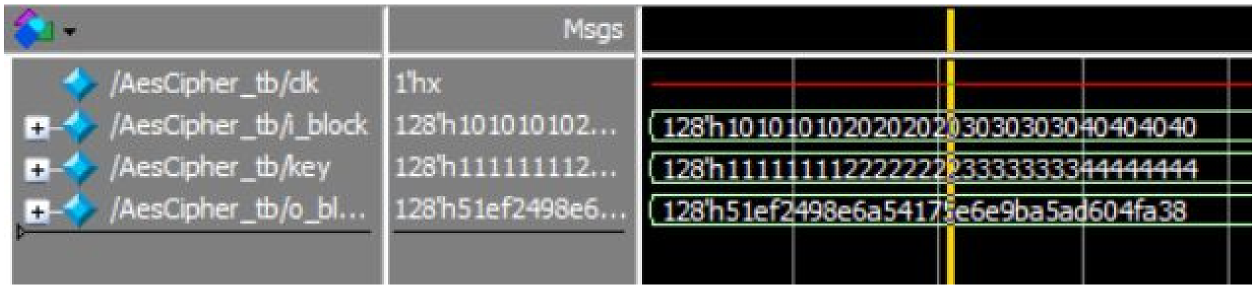
- MixColumns
- GenerateRoundKey
- AddRound Keys

These were then called wherever necessary in the module AES_main.v

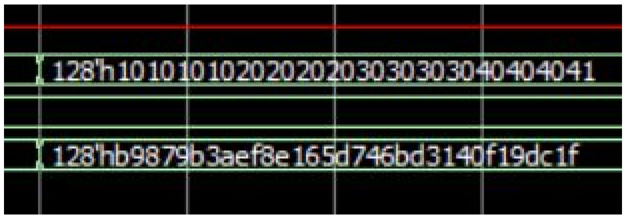
To keep track of the round of encryption an up counter was defined in the parent module. This is because the familiar for loop construct is not synthesizable in FPGAs.

EXPERIMENTAL RESULTS:

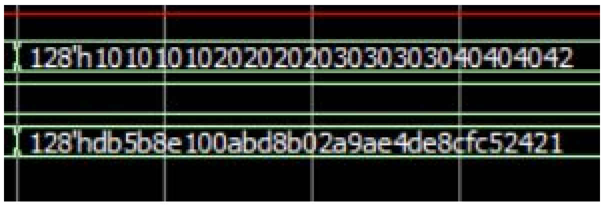
The images below have demonstrated that our code has given the correct output after cross checking with online server. We have shown three examples below as samples, though the user may test upon more examples. The online server to check the codes is given in the appendix section below, along with instructions to give input on the website.



```
                          Msgs
/AesCipher_tb/clk         1'hx
/AesCipher_tb/i_block     128'h101010102...   128'h10101010202020203030303040404040
/AesCipher_tb/key         128'h111111112...   128'h11111111222222223333333344444444
/AesCipher_tb/o_bl...     128'h51ef2498e6...  128'h51ef2498e6a54175e6e9ba5ad604fa38
```



```
128'h10101010202020203030303040404041
128'hb9879b3aef8e165d746bd3140f19dc1f
```



```
128'h10101010202020203030303040404042
128'hdb5b8e100abd8b02a9ae4de8cfc52421
```

Key: 11111111222222223333333344444444
Input 1: 10101010202020203030303040404040
Server Output 1:`51ef2498e6a54175e6e9ba5ad604fa38`
Input 2: 10101010202020203030303040404041
Server Output 2:`b9879b3aef8e165d746bd3140f19dc1f`
Input 3: 10101010202020203030303040404042
Server Output 3:`db5b8e100abd8b02a9ae4de8cfc52421`

All the outputs matched with the server generated outputs

CONCLUSION:

We managed to implement a Verilog code for the 128 bit AES Encryption. We have implemented both a sequential and combinational version of the code, and the combinational circuit implementation is submitted. Both the codes have been uploaded in the Github Link that we have attached below in the Appendix Section.
Another key takeaway from this project was the usage of Galois field multiplication to get the Column Mixing Results.

CONTRIBUTIONS:

All the team members put in an equal amount of effort in general. Equal number if modules were written by each of the team members, and this project would not have been possible without the contributions of everyone.

REFERENCES AND BIBLIOGRAPHY

K Avinash(2019, January). Lecture Notes on Computer and Network Security. *Purdue University*. Retrieved from: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf

P Christof(2014, January). Lecture 8: Advanced Encryption Standard by Christof Paar. *University of Massachusetts Amherst*. Retrieved from: https://www.youtube.com/watch?v=NHuibtoL_qk

J Moser(2009, September). A Stick Figure Guide to the Advanced Encryption Standard (AES) . *Moserware, Medium.org*. Retrieved from: http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

APPENDIX

Online Server used to check output:
https://www.cryptool.org/en/cto-highlights/aes
(NOTE: Choose 10 rounds for AES 128 bit encryption, and also choose the None option in Chaining. Then enter the key and input and find the result in the Round 10 output or under Encoded Heading.)

GITHUB LINK FOR PROJECT: https://github.com/SudarshanHV/AESVerilog
The final project is in the Combinational folder, with the report.