The 128-bit AES design by using FPGA

Hidayarni Hamzah1, Nabihah Ahmad2, Siti Hawa Ruslan3
Department of Computer Engineering, Faculty of Electrical and Electronic Engineering, Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, Johor, Malaysia.

1yanie6288@gmail.com, 2nabihah@uthm.edu.my, 3sitihawa@uthm.edu.my

Abstract Advanced Encryption Standard (AES) is a common symmetric encryption algorithm and widely implemented in Wireless Local Area Network (WLAN) and Bluetooth controller for security services in its application. This paper presents a 128-bit data path with 128-bit, 192-bit or 256-bit key size. The purpose of the design is high throughput and low area design of AES. The AES methodology is by using Field-Programmable Gate Array (FPGA) and Xilinx Virtex-7 XC7VX485T as a tool to obtain simulation results through Verilog Hardware Description Language (HDL). This design utilized a 2730 Slices with the throughput of 12.9 Gbps. The design is suitable for the portable device application which requires data security with a high throughput and speed.

1. Introduction
Advanced Encryption Standard (AES) is needed for security services in many applications such as Wireless Network, Radio Frequency Identification (RFID) tags and many more. It was selected as the United State (US) standard for encryption of unclassified information in 2001. As AES had announced, it replaced the Data Encryption Standard (DES) which had been the US standards for a number of years, since 1977. AES is the currently employed specification for encrypting electronic data from the United States, National Institute of Standards and Technology (NIST) [1].

2. AES Architecture
The process flow of AES-128 takes 10 rounds of operations per 128-bit data as shown in Figure 1. In each cipher round, there are four elementary operations, including SubBytes, ShiftRows, MixColumns and AddRoundKeys, which performed on a two-dimensional array of bytes called state matrix. SubBytes transform individual bytes of the state matrix into the values stored in a non-linear byte substitution table (Sbox). ShiftRows cyclically shifts the last three rows of the state matrix by different offsets, respectively. MixColumns mixes all the 4 bytes of a column of the state matrix to form a new column. AddRoundKeys was simply the exclusive-OR (XOR) between the state matrix and RoundKey with 128-bit data.

After 10 rounds of iterations of the four operations, a plaintext then converted to a ciphertext, which could be decrypted through an inverse flow of Figure 1. In Figure 1, there was also a module parallel to the main encrypting operations. This module, called Key Expansion is used to generate from the SeedKey a series of RoundKeys, which were then applied to the state matrix in the AddRoundKeys operation [2].
2.1. SubBytes and InvSubBytes
SubBytes and InvSubBytes operations substitute each byte of the data in state with 8-bit data from substitution table S-box and Inverse-S-box respectively. The S-box and Inverse-S-box value can be obtained by calculating the multiplicative inverse of Galois Field refer the intersection value as in Figure 2 should be used in substitution. SubBytes and InvSubBytes operation are to obscure the relationship between plain text and encrypted cipher text [3].

On the other hand, InvSubBytes operation, substitutes data “68” with “F7” by referring from Inverse-S-box as in Figure 3. The process is similar with SubBytes but referring on different table of Inverse-S-box instead of S-box.
2.2. ShiftRow and InvShiftRow
ShiftRows transformation consists of a cyclic shift of each row in a state. The underlying purpose is to spread out the original message. During the ShiftRows operation in encryption as shown in Figure 4, first row of the state is not shifted, while second row of the state is shifted cyclic left by one byte, third row shifted cyclic left by two bytes and last row by three bytes. On the other hand, InvShiftRows operation during decryption shifted the same way as ShiftRows but opposite in direction.

2.3. MixColumn and InvMixColumns
MixColumns and InvMixColumns operation multiply the state’s column with a matrix of constant, $C(x)$ as shown in Figure 5 during encryption and decryption respectively. The value of $B$ is the product of $A$ and constant $C(x). The only difference between MixColumns and InvMixColumns is the $C(x)$ value as shown in Equation (1) for encryption and Equation (2) for decryption respectively.
2.4. AddRoundKey

AddRoundKey operation is needed in both encryption and decryption process. AddRoundKey operation operates the same way for both encryption and decryption. During this operation, 128-bit of generated round key or key is added with the states as shown in Figure 6. It is one of the most straightforward processes among four operations in AES algorithm.

\[
C(x) = 
\begin{bmatrix}
02 & 03 & 01 & 01 \\
01 & 02 & 03 & 01 \\
01 & 01 & 02 & 03 \\
03 & 01 & 01 & 02 \\
\end{bmatrix}
\]

\[
C(x) = 
\begin{bmatrix}
0E & 0B & 0D & 09 \\
09 & 0E & 0B & 0D \\
0D & 09 & 0E & 0B \\
0B & 0D & 09 & 0E \\
\end{bmatrix}
\]

Figure 6 AddRoundKey operation adds 128-bit of round key generated with states

2.5. AddRoundKey

Key expansion generates round key in forward and backward direction. During encryption process, key expansion generates round key in forward direction. On the contrary, decryption process requires key expansion to generate round keys in backward direction. Although operation of key expansion during encryption and decryption are different, most of their sub-operations are similar. They differ in sequence of steps only where key expansion for decryption is the reverse steps in encryption.

3. Result and discussion

Sets of 128-bit test vector inserted sequentially at input “new_data” and given an instruction to start the encryption at input “control” as shown in Figure 7. While the second set of test is loading into the design, first set of test vector has started the encryption immediately. Output “data_ready” is set to “1” once the encrypted data is ready at the output “data_out_reg” as shown in Figure 8. The encrypted data is compared with expected cipher. Observing the “counter” value in Figure 8, 4 clock cycles are needed to complete 4 sets of 128-bit data encryption beginning from “counter = 45” to “counter = 49” when state is not “IDLE”.
Table 1. Encryption key, test vector and its encrypted cipher text are used in simulation to test its functionality.

| Encryption key: | Cipher Text                        |
|----------------|-----------------------------------|
| 2b7e151628aed2a6abf7158809cf4f3c |                                    |
| 6bc1bee22e409f96e93d7e117393172a  | 3ad77bb40d7a3660a89ecaf32466ef97  |
| ae2d8a571e03ac9c9eb76fac45af8e51   | f5d3d58503b9699de785895a96fdbaaf |
| 30c81c46a35ce411e5fbc1191a0a52ef   | 43b1cd7f598ece23881b00e3ed030688 |
| f69f2445df4f9b17ad2b417be66c3710   | 7b0c785e27e8ad3f8223207104725dd4 |

Figure 7 Encryption process begins by asserting "01" to control input."new_data" input accepts 4 sets of test vectors sequentially.

Figure 8 "data_ready" is set to "1" when encrypted data is available at "data_out" output. "data_out" output 4 sets of encrypted data sequentially.

Throughput = 128 bits/ (Cycles per Encrypted Block * Time period) (3)
The cycle is the clock cycle per byte to encrypt a block data while time period is the reciprocal of the frequency clock.

Figure 9 is the RTL viewer for overall design with 128-bit input data and key from 128-bit to 256-bit key length.

![Fig. 9 RTL viewer](image)

### 4. Performance Comparison

**Table 2.** Comparison result AES 128-bit from year 2014 until 2018

| Author     | Data Path   | Key size                  | FPGA Device       | No. of Slices | Throughput (Gbps) | Percentage of throughput over slices (%) |
|------------|-------------|---------------------------|-------------------|---------------|-------------------|----------------------------------------|
| Our design | 128-bit, 128-bit, 192-bit, 256-bit | Virtex-7 XC7VX485T | 2730              | 12.8          | 0.46              |
| [5]        | 128-bit     | 128-bit, 128-bit, 192-bit, 256-bit | Kintex-7 XC7K325T | 5110          | 1.67              | 0.033                                  |
| [4]        | 128-bit     | 128-bit, 128-bit, 192-bit, 256-bit | Virtex-5 XC5VFX130 | 919           | 1.3               | 0.14                                   |
| [9]        | 128-bit     | 128-bit, 128-bit, 192-bit, 256-bit | Virtex-7 XC7VX690T | 10773         | 1.28              | 0.012                                  |
| [10]       | 128-bit     | 128-bit, 128-bit, 192-bit, 256-bit | Spartan-3 XC3S400-FG456 | 1403      | 2.059             | 0.15                                   |

Through the table of comparison author [5] has 1.67 Gbps throughput with 5110 number of slices, author [4] has 1.3 Gbps throughput with 919 number of slices, author [9] has 1.28 Gbps with 10773 number of slices and author [10] has 2.059 Gbps with 1403 number of slices. Overall, the highest throughput is our design with 12.8 Gbps and the percentage of throughput over slices is 0.46 percent. As a number of slices are small it reflected the
utilization of the design. The device will require less area and complexity compares to the design with larger number of slices.

5. Conclusion
The result for this AES design is 12.9 Gbps throughput with 2730 Slices. The design implemented by using FPGA Xilinx Vivado Virtex-7. FPGAs typically provide higher efficiency compared to software. On the other hand they offer more flexibility and much lower design and debug costs compared to specifically-built hardware.

6. Acknowledgement
The authors would like to express gratitude for financial support and contributions from Ministry of Education (MOE) Malaysia under Fundamental Research Grant Scheme (FRGS) vot number 1630.