Low-area AES encryption design and implementation

Cho Ho Ching, Feiyu Di, Hao He, Wangyang Ji, Jiayu Shi

Abstract

In many applications, the Advanced Encryption Standard (AES) algorithm is now the standard option for security services. We describe an AES encryption hardware core in our study that is suitable for low-cost, low-power devices. The transformations are to be monitored by a control device. After implementing the suggested design on the Xilinx Spartan FPGA, it was determined that the suggested approach had less area coverage and latency.

Keywords: Advanced Encryption Standard (AES), Field Programmable Gate Array (FPGA), Encryption, Decryption, Cryptography, RTL, low area

1. Introduction

As the number of crimes rises, networks and storage devices must be guaranteed to be secure. Data must be effectively protected from potential threats by taking preventative measures. One security mechanism to protect data from unwanted access is cryptography. The area of the algorithm must be modified to fit the device for the hardware to implement it. less area also offers less expense and electricity usage. Whenever feasible, low-order datapaths and an iterative structure are used to optimize the AES architecture's area with minimal performance loss. Using the Xilinx ISE tool, the design is built in Verilog and synthesized for the Xilinx Spartan device (encryption).
Our iterative AES core can encrypt using 128-bit keys and has an 8-bit (or byte) data route. An outline of AES is provided in Section 2.

2. Outline of AES

As seen in Fig.1, each loop has four transformations: AddRoundKey, MixColumns, SubBytes, and ShiftRows. AddRoundKey is used to initialize, and the MixColumns transformation is not performed in the last round.

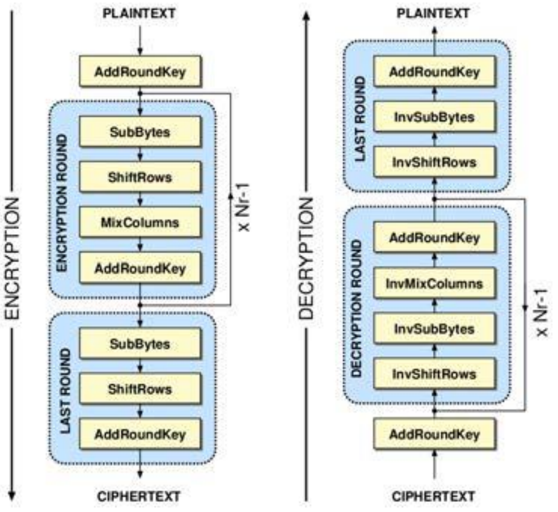


Fig. 1 Outline of AES

The SubBytes transformation uses a substitution table (S-box) to perform independent byte substitution on each byte of the State non-linear. The multiplicative inverse in the finite field GF(2^8) and the affine transformation are the two transformations that are combined to create this S-box. MixColumns treat the 4-byte data blocks in each column as coefficients of a 4-term polynomial, and a fixed polynomial is used to multiply the data modulo x^4 + 1. A straightforward bit-wise XOR operation on the data and the 128-bit round keys is AddRoundKey. The decryption structure can be obtained by reversing the aforementioned encryption algorithm.

2.1 ShiftRows operation

The ShiftRows operation is implemented as described by EFFICIENT BYTE PERMUTATION REALIZATIONS FOR COMPACT AES IMPLEMENTATIONS[1]. Row shift is a simple left loop shift operation, The principle is easy to understand, line 0 of

the status matrix is moved 0 bytes left, line 1 is moved 1 byte left, line 2 is moved 2 bytes left, and line 3 is moved 3 bytes left. The shift operation can increase a certain degree of confusion but also increase a certain degree of security so that relatively small changes can also have a large impact, which will improve the security of the AES algorithm. Still, also because of such a relatively simple step, the algorithm can be executed in hardware and software more efficiently. We use hardware description language to frame this part.
In the byte permutation module, there are three 2-to-1 multiplexors, twelve 8-bit shift registers, and a 4-to-1 multiplexor. The whole module is controlled by the five-bit counter, and the counter will count the number of clock cycles and set the values of ‘select’ for the multiplexors in order to control the whole dataflow process. The values of ‘select’ for the multiplexors at different time slots are provided below.

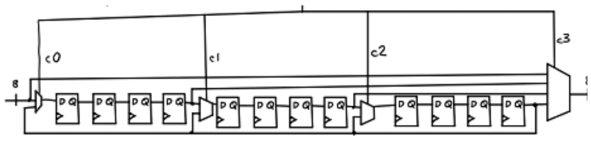
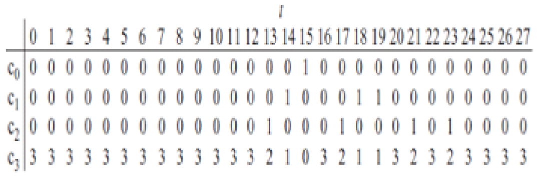


Fig. 2 Module structure



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
c_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
c_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0
c_2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0
c_3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	1	0	3	2	1	1	3	2	3	2	3	3	3	3

Fig. 3 Select value in the left shift

The dataflow process is illustrated below.

Table.1 dataflow process

t/data													input
12	12	11	10	9	8	7	6	5	4	3	2	1	0
13	13	12	11	10	9	8	7	6	5	1	4	3	5
14	14	13	12	11	2	9	8	7	6	1	4	3	10
15	3	14	13	12	11	2	9	8	7	6	1	4	15
16		3	14	13	12	11	2	9	8	7	6	1	4
17			3	14	13	12	11	2	1	8	7	6	9
18				3	6	13	12	11	2	1	8	7	14
19					7	6	13	12	11	2	1	8	3
20						7	6	13	12	11	2	1	8
21							7	6	1	12	11	2	13
22								7	6	1	12	11	2
23									11	6	1	12	7
24										11	6	1	12
25											11	6	1
26												11	6
27													11

2.2 SubBytes transformation

The S-box maps an 8-bit input, c, to an 8-bit output, s = S(c). Both the input and output are interpreted as polynomials over GF(2).

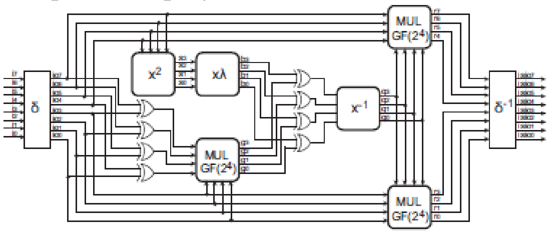


Fig. 4 Multiplicative inverse in Verilog

First, the input is mapped to its multiplicative inverse in GF(2^8) = GF(2) [x]/(x^8 + x^4 + x^3 + x + 1), Rijndael's finite field. Zero, as the identity, is mapped to itself. This transformation is known as the Nyberg S-box after its inventor Kaisa Nyberg. Rather than applying an 8-degree extension field explicitly, the multiplicative inversion module is executed using the method of numerous 2-degree extensions under bias, as proposed by Satoh et al. [2]. An element may be expressed as n1x + n0 in this GF(2^8) field, where n1 is the most significant nibble and n0 is the least significant. The following formula can be used to calculate the multiplicative inverse:

$$(n_1x+n_0)^{-1} = n_1\left(n_1^2B+n_1n_0A+n_0^2\right)^{-1}x + (n_0+n_1A)\left(n_1^2B+n_1n_0A+n_0^2\right)^{-1}$$

The irreducible polynomial of x2 + Ax + B allows every polynomial to be expressed as n1x+n0, so this equation may be simplified. The irreducible polynomial becomes x^2 +x+λ when A = 1 and B = λ are chosen, which enables the formula to be simplified to

$$(n_1x+n_0)^{-1} = n_1\left(n_1^2\lambda+n_0(n_1+n_0)\right)^{-1}x + (n_0+n_1)\left(n_1^2\lambda+n_0(n_1+n_0)\right)^{-1}$$

The multiplicative inverse is then transformed using the following affine transformation:

$$\begin{bmatrix} s_0\\ s_1\\ s_2\\ s_3\\ s_4\\ s_5\\ s_6\\ s_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0\\ b_1\\ b_2\\ b_3\\ b_4\\ b_5\\ b_6\\ b_7 \end{bmatrix} + \begin{bmatrix} 1\\ 1\\ 0\\ 0\\ 0\\ 1\\ 1\\ 0 \end{bmatrix}$$

2.3 MixColumn operation

In the Mixcolumns step, a reversible linear transform is employed to combine the 4 bytes in every column of a state. The MixColumns function receives 4 bytes of data and 4 bytes of data, with every entry having an effect on the 4 output bytes. Along with Shiftrows, MixColumns provides diffusion in the encryption.

In this action, you convert every column with a fixed matrix

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \qquad 0 \le j \le 3$$

Matrix multiplication consists of adding and multiplying. Entries are bytes treated as coefficients of polynomials of order x^7.Adding is just an XOR. Multiplication is modulo irreducible polynomial x^8+x^4+x^3+x+1.. If it is handled bitwise, then if the offset is greater than FF_{16}, then a conditional XOR with $1B_{16}$ will be executed.

2.4 Key Expansion

Explanation of Add Round Key

The Add Round Key Transformation consists of two parts: Key Expansion and bitwise XOR.
In the 128-bit length AES algorithm, 10 rounds of 'Round Keys' are needed. In general, The input of Key Expansion is a 16-byte Cypher Key and the output is 10 Round Keys. To get the Round Key, three operations are required: RotWord, SubBytes, and Rcon XOR.

- RotWord:

RotWord is the operation of shifting the data in the 4th column (note that the '4th' here starts at 1) in the initial key, that is, the transformation process from [a0, a1, a2, a3] to [a1, a2, a3, a0].

- SubBytes:

SubBytes is the operation of replacing the data in the 4th column after RotWord with S-box.

- Rcon XOR:

After SubBytes, The data should be done the XOR operation with Rcon and the 1st column. The Rcon is the Round Constant, contains the values given by [xi-1,{00},{00},{00}], with xi-1 being powers of x(x is denoted as {02}) in the field GF(28).[3] The detailed values of Rcon are as shown in Table 1.

02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00

Table 1 Values of Rcon

After generating the round keys, the Round Keys need to be done with the XOR operation with the values from the mix column.

Implementation in Verilog

- Architecture of Key Expansion

The architecture of Key Expansion is shown in Figure 1.

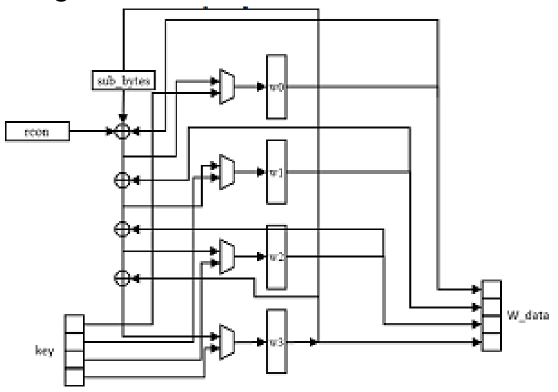


Fig. 5 Architecture of Key Expansion

The 'key' in the cipher key we input. The 'w_data' is the Round Key we output and it is assigned to w0, w1, w2, and w3. w0 equals the bit 96 to bit 127 data in Initial Key XOR with bit 0 to bit 31 data in Initial Key(after SubBytes) XOR with Rcon. w1 equals the bit 64 to bit 95 in Initial Key XOR with the new w0 we got from the previous step. w2 equals bit 32 to bit 63 in Initial Key XOR with the new w1 we got from the previous step. w3 equals the bit 0 to bit 31 in Initial Key XOR with the new w2 we got from the previous step. The value of S-box is generated from the sub_bytes module implemented before and Rcon is a look-up table. After generating the round key, XOR the round key with the data from MixColumns completes the AddRoundKey part.

3. Integration of AES

The top-level design of our AES encryption core is based on hämäläinen's design. The AES core's data flow is shown in Fig. 6. Through the input ports data_in and key_in, a plaintext block and the encryption key are concurrently loaded to the core, one byte at a time. During loading, using the XOR gate in the byte permutation unit's input, the first AES step—AddRoundKey between the block and the key—is carried out. The encrypted text block can be unloaded from the output port data_out, byte by byte, after 10 rounds of execution. The encryption key needs to be sent to the core again along with a fresh block of plaintext since round keys overwrite it.

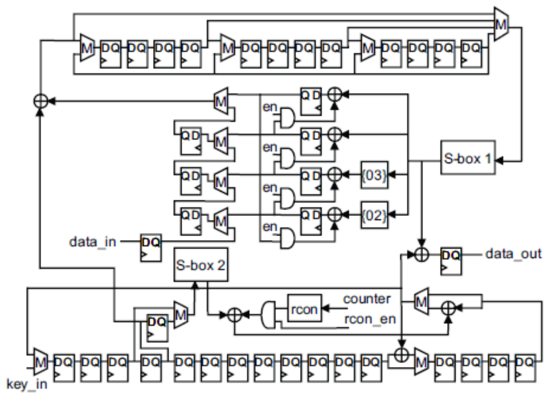
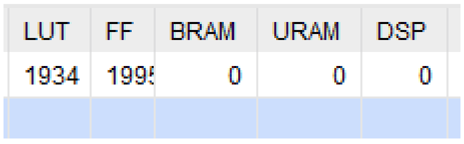


Figure 6. AES encryption core.

4. Simulation results

We synthesized the AES core at the gate level and defined it at the register transfer level in Verilog. Figure 7 displays the area represented as the number of LUTs and FFs, the use of BRAM, URAM, and DSP under standard working circumstances (1.2 V, 25 ◦C). We are only using 1934 LUTs and 1995 FFs for implementing both the encryption and decryption core of AES, which is less than many designs of AES. Since our design uses an 8-bit datapath, it consumes less power and resources than other AES modules.



LUT	FF	BRAM	URAM	DSP
1934	1995	0	0	0

Figure 7. Area of encryption core.

According to the synthesis report, the module block of key expansion
Our parallel operation results in a far reduced cycle count and greater throughput than the earlier 8-bit solutions. Our achieved cycle count of 160, which is one cycle per byte every round, can be considered the lowest for an iterated 8-bit AES implementation. From the simulation result, the cipher text is output at the 160-clock cycle.

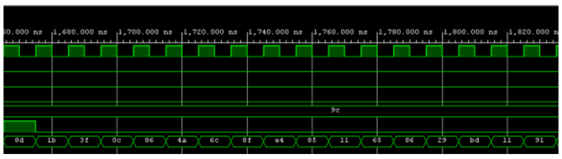


Fig. 8 Simulation result

Reference

[1] EFFICIENT BYTE PERMUTATION REALIZATIONS FOR COMPACT AES IMPLEMENTATIONS, Tuomas J¨arvinen, Perttu Salmela, Panu H¨am¨al¨ainen, and Jarmo Takala
[2] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact rijndael hardware architecture with s-box optimization," in International Con- ference on the Theory and Application of Cryptology and Information Security. Springer, 2001, pp. 239–254.
[3] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication 197, November 2001.