The Optimized Design of Rijndael Algorithm Based on SOPC

Shunwen Xiao, Yajun Chen, Peng Luo
College of Physics and Electronic information
China West Normal University
Nanchong, China
rejoice222@126.com

***Abstract*—Based on the analysis of the round transformation and key expansion, the Advanced Encryption Standard (AES) algorithm is optimized through the Look-up Table. And then the optimized Rijndael algorithm based on SOPC (System on a programmable Chip) is designed and implemented through software and hardware. According to the software design flow chart of the optimized Rijndael algorithm, the program design of the corresponding B table and key generator are completed. The testing results are: the highest working frequency is 147 MHz; the biggest data flow is 2382 Mbps. The design of the Rijndael algorithm based on "NIOS Ⅱ + FPGA" can achieve a higher data processing speed while it occupies relatively low resources. The design has a big breakthrough compared to the traditional FPGA realization.**

Keywords-optimized design; Rijndael algorithm; SOPC

I. INTRODUCTION

As the new generation AES cipher algorithm, the Rijndael algorithm has the advantages of high efficiency in implementation, low demand for storage, and high speed of encryption and decryption [1]. The algorithm can be implemented through software as well as hardware. The software implementation is too slow although it is flexible. It is even more so with the block encryption algorithms getting more and more complex. Besides, the security system is difficult to avoid the appearance of the key plaintext in the computer during the implementation through software, which may result in the plaintext being stolen or modified. On the contrary, the hardware implementation has the advantages of high speed, high reliability and much higher security, so that an increasing number of applications require the implementation through hardware [2]-[4]. The key of hardware implementation operates in the interior of module and the algorithm solidifies in the hardware, which can guarantee that the key of plaintext doesn't flow outside and can achieve the true sense of confidentiality. The design based on "NIOS Ⅱ + FPGA" project, on the one hand, can configure CPU flexibly according to the application; on the other hand, it can realize the configuration of hardware logic directly in the FPGA interior. The software realization can be used for the peripheral device that has not so high demand for data transmission speed. While for the configuration that requires high speed, it can use the hardware interface to realize directly and then use the software to control it. In this paper, the Rijndael algorithm is optimized comprehensively using Look-up Table, and then the algorithm is designed systematically using "NIOS Ⅱ + FPGA" project.

II. THE DESIGN OF RIJNDAEL ALGORITHM BASED ON LOOK-UP TABLE

A. The design of round transformation based on Look-up Table

In SubBytes, the input high order 4 bits is used as the row value of the S-box, the low order 4 bits is used as the column value of the S-box, then the corresponding row and column element is taken out from the S-box as an output. The SubBytes in which each column is 32 bits can use matrix to express as Eq. (1) [2] - [9]:

$$\begin{bmatrix} b'_{0,i} \\ b'_{1,i} \\ b'_{2,i} \\ b'_{3,i} \end{bmatrix} = \begin{bmatrix} S[b_{0,i}] \\ S[b_{1,i}] \\ S[b_{2,i}] \\ S[b_{3,i}] \end{bmatrix} \tag{1}$$

Where $b_{0,i}$ is the 4-bytes element of the first row in the matrix, $S[b_{0,i}]$ is the S-box Look-up Table operation on the byte element of the matrix.

In ShiftRows, row 0 is not shifted; row 1 is shifted to the left by 1 byte cyclically; row 2 by 2 bytes and row 3 by 3 bytes. The ShiftRows transformation can use matrix to express as Eq. (2):

$$\begin{bmatrix} b''_{0,i} \\ b''_{1,i} \\ b''_{2,i} \\ b''_{3,i} \end{bmatrix} = \begin{bmatrix} S[b_{0,i(0)}] \\ S[b_{1,i(1)}] \\ S[b_{2,i(2)}] \\ S[b_{3,i(3)}] \end{bmatrix} \tag{2}$$

In MixColumns, $b'''(x) = c(x) \bullet b''(x) \bmod (x^4+1)$, in which, c(x) = {03} • x3 + {01} • x2 + {01} • x + {02}. The MixColumns can be written as a matrix multiplication (Eq.(3)), in which let the matrix through shift operation multiplywith the fixed matrix (by hexadecimal system expression).

$$\begin{bmatrix} b_{0,i}'''' \\ b_{1,i}'''' \\ b_{2,i}'''' \\ b_{3,i}'''' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[b_{0,i(0)}] \\ S[b_{1,i(1)}] \\ S[b_{2,i(2)}] \\ S[b_{3,i(3)}] \end{bmatrix} \tag{3}$$

In AddRoundKey, XOR is performed on the result of MixColumns transformation and the key (Eq. (4)):

$$\begin{bmatrix} b_{0,i}'''' \\ b_{1,i}'''' \\ b_{2,i}'''' \\ b_{3,i}'''' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[b_{0,i(0)}] \\ S[b_{1,i(1)}] \\ S[b_{2,i(2)}] \\ S[b_{3,i(3)}] \end{bmatrix} xor \begin{bmatrix} w_{0,n} \\ w_{1,n} \\ w_{2,n} \\ w_{3,n} \end{bmatrix} \tag{4}$$

Where $W_{m,n}$ is the key matrix of the round "m". The four tables named B_0, B_1, B_2, and B_3 are defined respectively as follows:

$$B_0[x] = \begin{bmatrix} 02\bullet S[x] \\ S[x] \\ S[x] \\ 03\bullet S[x] \end{bmatrix}, \quad B_1[x] = \begin{bmatrix} 03\bullet S[x] \\ 02\bullet S[x] \\ S[x] \\ S[x] \end{bmatrix},$$

$$B_2[x] = \begin{bmatrix} S[x] \\ 03\bullet S[x] \\ 02\bullet S[x] \\ S[x] \end{bmatrix}, \quad B_3[x] = \begin{bmatrix} S[x] \\ S[x] \\ 03\bullet S[x] \\ 02\bullet S[x] \end{bmatrix}$$

Then Eq. (4) can be expressed as Eq. (5):

$$\begin{bmatrix} b_{0,i}'''' \\ b_{1,i}'''' \\ b_{2,i}'''' \\ b_{3,i}'''' \end{bmatrix} = B_0[b_{0,i(0)}] xor B_1[b_{1,i(1)}] xor B_2[b_{2,i(2)}] xor B_3[b_{3,i(3)}] xor W_{m,n} \tag{5}$$

As the column mixed transformation is not carried out in the final round, so it can also create table as the other rounds when it applies Look-up Table method in the final round. Tables B_0' to B_3' are defined as follows:

$$B_0'[x] = \begin{bmatrix} S[x] \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B_1'[x] = \begin{bmatrix} 0 \\ S[x] \\ 0 \\ 0 \end{bmatrix},$$

$$B_2'[x] = \begin{bmatrix} 0 \\ 0 \\ S[x] \\ 0 \end{bmatrix}, \quad B_3'[x] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ S[x] \end{bmatrix}$$

The final round transformation can be expressed as Eq. (6):

$$\begin{bmatrix} b_{0,i}''' \\ b_{1,i}''' \\ b_{2,i}''' \\ b_{3,i}''' \end{bmatrix} = B_0'[b_{0,i(0)}] xor B_1'[b_{1,i(1)}] xor B_2'[b_{2,i(2)}] xor B_3'[b_{3,i(3)}] xor W_{m,n} \tag{6}$$

It can be seen from Eqs. (5) and (6) that the calculation of each round only needs four Look-up Tables, four XORs and three shifts operation.

B. The design of key expansion based on Look-up Table

The key is divided into groups according to the column of matrix and 40 new columns are added to expand. If the former four columns (i.e. the initial key) is w (0), w (1), w (2) and w (3), then the new column will produce in a recursive way. The column "i" is determined by Eq. (7) [1]:

$$\begin{cases} w(i) = w(i-4) XOR w(i-1) & \text{i is the multiple of 4} \\ w(i) = w(i-4) XOR T[w(i-1)] & \text{i isn't the multiple of 4} \end{cases} \tag{7}$$

The round key "i" can be expressed as Eq. (8):

$$W_i = w_{4i+0} w_{4i+1} w_{4i+2} w_{4i+3} \tag{8}$$

With 4 × 4 matrix that is Eq. (9):

$$W_{m,n} = \begin{bmatrix} w_{0,4i+0} & w_{0,4i+1} & w_{0,4i+2} & w_{0,4i+3} \\ w_{1,4i+0} & w_{1,4i+1} & w_{1,4i+2} & w_{1,4i+3} \\ w_{2,4i+0} & w_{2,4i+1} & w_{2,4i+2} & w_{2,4i+3} \\ w_{3,4i+0} & w_{3,4i+1} & w_{3,4i+2} & w_{3,4i+3} \end{bmatrix} \tag{9}$$

If the initial key is w (0), w (1), w (2) and w (3), it can obtain w (4) ~ w (43) by Eq.(7). So the 10 round keys are composed of w (4) ~ w (43), as is shown in Table 1.

TABLE I. THE ROUND KEY

i	1	2	3	4	5
W(i)	$W_4W_5W_6W_7$	$W_8W_9W_{10}W_{11}$	$W_{12}W_{13}W_{14}W_{15}$	$W_{16}W_{17}W_{18}W_{19}$	$W_{20}W_{21}W_{22}W_{23}$
i	6	7	8	9	10
W(i)	$W_{24}W_{25}W_{26}W_{27}$	$W_{28}W_{29}W_{30}W_{31}$	$W_{32}W_{33}W_{34}W_{35}$	$W_{36}W_{37}W_{38}W_{39}$	$W_{40}W_{41}W_{42}W_{43}$

As can be seen from Table 1, if the initial key (w (0), w (1), w (2) and w (3)) is known, then the round key can be taken as a constant table and it can be implemented through Look-up Table circuit.

III. THE ALGORITHM DESIGN BASED ON SOPC SYSTEM

The Rijndael algorithm based on the SOPC system is shown in Fig.1. The standard Altera 32 bits NIOS Ⅱ embedded CPU provides guarantee for the large and systematic data processing. The system is composed of FPGA, memory and the external interface three parts. In the system, the peripheral circuit and NIOS Ⅱ CPU Soft-Core are integrated to realize the control functions. As the control core of the system, the NIOS Ⅱ CPU Soft-Core needs a balance between its resource occupation and function when it is created. The demand of system resources is greatly reduced by the SOPC Builder customization for NIOS Ⅱ CPU. The FPGA part is built in the FPGA chip, the core is NIOS Ⅱ processor CORE. A lot of data need to be processed in algorithm, so the algorithm round transformation is completed by using NIOS Ⅱ processor CORE and the key generation is conducted by the key generator in FPGA. The external interface of FPGA is a part including some interface devices and circuit modules, which is used for data input / output and display and so on.

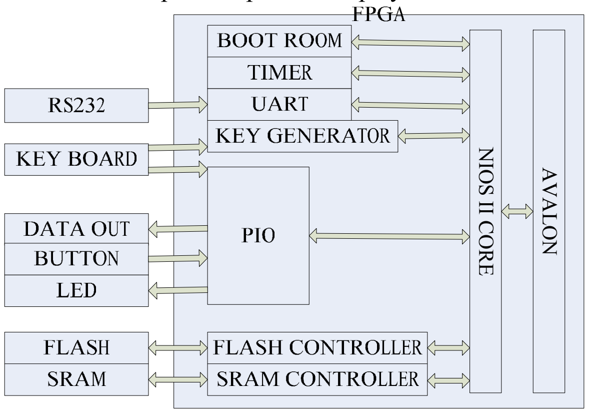

Figure 1. The scheme of SOPC system

IV. THE DESIGN OF OPTIMIZED ALGORITHM

The design of optimized algorithm is shown in Fig.2.

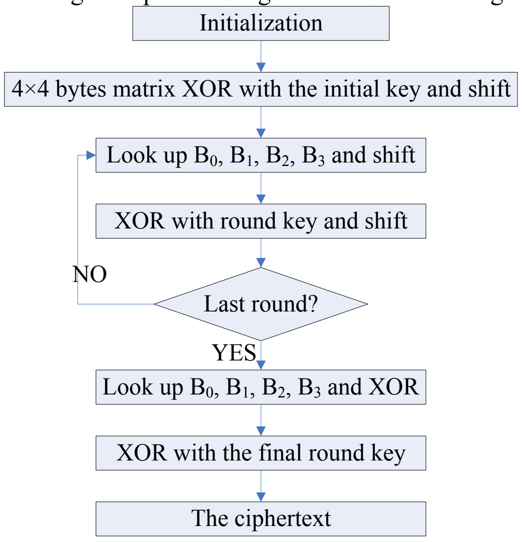
Initialization
4×4 bytes matrix XOR with the initial key and shift
Look up B0, B1, B2, B3 and shift
XOR with round key and shift
NO
Last round?
YES
Look up B0, B1, B2, B3 and XOR
XOR with the final round key
The ciphertext

Figure 2. The design of optimized algorithm

A. *Table B generation*

The rapid optimization of Rijndael algorithm is the Look-up Table operation. At first, the Look-up Table should be created. As has been discussed from Eqs. (5) and (6), the calculation of each round needs four Look-up Tables, four XORs and three shifts operation. As can be seen from Table B, only Table B_0 needs to be created, the other three Look-up Tables B_1, B_2 and B_3 can be obtained by cyclical shift of the bytes. While the round function Table B_0 can be realized using the S-box and byte multiplication. Table B_0 can be expressed as Eq. (10), in which each element is one byte.

$$B_0(x) = (S(x)\bullet\{03\} << 24) OR (S(x) << 16) OR (S(x) << 8) OR (S(x)\bullet\{02\}) \tag{10}$$

The algorithm of creation of the round function table B_O is as follows [10]:

```
Round (unsigned long int *b, unsigned long int *k, unsigned
                    long int *e)
        {
         for(int i=0; j<4; j++)
              {
                unsigned long int B0=0,B1=0,B2=0,B3=0;
                               B0=B[b[4*i]];
                   B1=(B[b[1+4*((j+l)%4)]]>>8)| (B [b
                      [1+4*((j+l)%4)]]<<24);
                  B2=(B[b[2+4*((j+2)%4)]]>>16) |(B [b
                     [2+4*((j+2)%4)]] <<16);
                      B3=(B[b[3+4*((j+3)%4)]]>>24)|
                   (B[b[3+4*((j+3)%4)]] <<8);
                           E[j]=BO^B1^B2^B3^k
              }
        }
```

Where *b is the head address of the State matrix, *k points to the head address of the round key of each round. *e returns to the head address of the State matrix after a round transformation, B[] is the created Table B_0.

As there is no mix-column for the final round in the round operation in the Rijndael algorithm, it simply needs to change Table B_0 to the S-box.

B. *Key generation*

According to the initial key (w (0), w (1), w (2) and w (3)), the key generator generates w (4) ~ w (43) automatically and stores them in the memory (complete the memory initialization). In the period of round transformation, the quadruple frequency of the round clock is conducted by the frequency multiplier and the counting value is taken as the Look-up Table circuit address. Four Look-up tables are implemented in a round clock period and $w_{4i+0} \sim w_{4i+3}$ are sent out. At the same time, the 128 bits round key is exported through the serial-in parallel-out shift register. The function description for the key generation circuit is conducted by VHDL and the generated module symbols are shown in Fig.3.

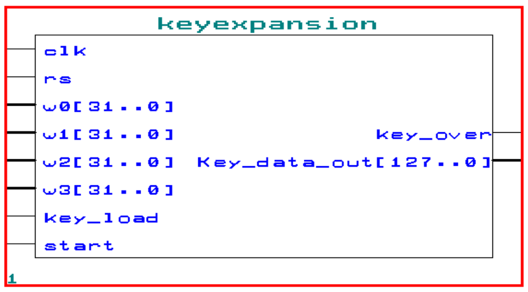

Figure 3. The module of key expansion circuit

In which, $W_0[31..0]$, $W_1[31..0]$, $W_2[31..0]$ and $W_3[31..0]$ is the initial key, "Key_load" is loading the initial key, "start" is the beginning of the round key generation, "key_data_out[127..0]" is the output of Sub-key, "key_over" is the completion of round key generation. When the initial key is W_0 = 173c4615, W_1 = 29bd08a1, W_2 = 123f4c55 and W_3 = 9167ce16, the simulation result is presented in Fig.4.

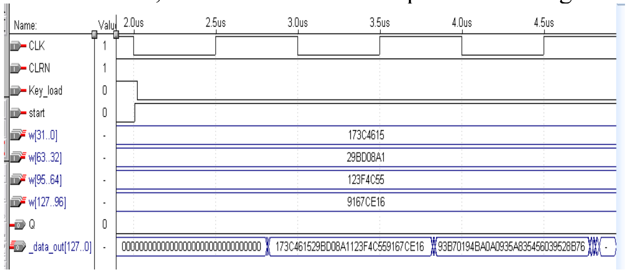

Figure 4. The local enlargement of process simulation

V. RESULTS

The design based on Altera EP1C12 chips is synthesized under the integrated environment of QUARTUSII6.0. The testing results are shown in Table 2. As can be seen, the highest working frequency is 147 MHz; the biggest data flow is 2382 Mbps. Compared with the results in literature (see Table 2), the design has a clear superiority in speed and performance under the relatively low resource request.

TABLE II. THE RESULTS OF EXPERIMENT

References	Keylength (bit)	Devices	LE	Frequency (MHz)	Data flow (Mbps)
[2]	128	XC2V1000	554(slice)	133	1504
[3]	128	XC2S400E	2822(slice)	39.7	254
[4]	128	XCV300E	881(slice)	67	779
[5]	128	EP1S10F484C5	3235	29.4	376.5
The paper	128	EP1C12	4813	147	2382

VI. CONCLUSIONS

The Rijndael algorithm is optimized in Look-up Table implementation on the basis of not lowering security of algorithm. The hardware design of the Rijndael algorithm is achieved mainly from the hardware and software implementation. The overall design of the Rijndael algorithm based on SOPC is performed and then the creation of NIOSⅡ CPU is completed to form an integrated SOPC system. The testing results show that the design of the Rijndael algorithm based on "NIOS Ⅱ + FPGA" can achieve a higher data processing speed while it occupies relatively low resources. The design has a big breakthrough compared to the traditional realization based on FPGA [2]-[5].

REFERENCES

[1] Richard Spillman (writer).YE Ranjian, CAO Ying, ZHANG Changfu (translator). Classical and Contemporary Cryptology [M]. Beijing: Tsinghua University. Press, 2005.

[2] LV Xiaobin, YANG Feng, ZHAO Zhixin, Design and Implementation of AES Crypto Coprocessor Based on FPGA [J]. Microelectronics & Computer, 2005, 22(5): 121-123, 127.

[3] ZHANG De-xue, GUO Li, FU Zhong-qian. Design and implementation of AES algorithm based on FPGA [J]. Journal of University of Science and Technology of China，2007，37(12) : 1461-1465.

[4] SHEN Qi-feng, HUANG Shi-tan, YANG Liang. FPGA Optimization Implementation for AES Algorithm[J]. Journal of Xi'an University of Technology, 2006, 22(2): 203-206.

[5] WANG Jian-yu, ZHANG Lu-guo. Reconfigurable Design for Encryption/Decryption of AES Based on FPGA [J]. Computer Engineering, 2008, 34(7): 163-164, 167.

[6] ZHANG Ju-ying, WANG He-ming. A Very Small FPGA Implementation of Application-Specific Instruction Processor for AES [J]. Microelectronics & Computer, 2008, 25(4): 165-168.

[7] Máire McLoone, John V McCanny. Rijndael FPGA Implementations Utilising Look-Up Tables [J]. The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, 2003, 34(3): 261-275.

[8] F. Rodriguez-Henriquez, N. A. Saqib, A. Diaz-Perez. 4.2Gbit/s single-chip FPGA implementation of AES algorithm [J]. Electronics Letters, 2003, 39(15): 1115-1116.

[9] S. Mangard, M. Aigner, S. Dominikus. A Highly Regular and Scalable AES Hardware Architecture [J]. Computers, IEEE Transactions on, 2003, 52(4): 483-491.

[10] Cao ZhiGang,The Research and FPGA Implemention of Rijndael Algorithm [D]. Harbin, Harbin Engineering University, 2006, 37(12): 1461-1465.