Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter

Chih-Chung Lu and Shau-Yin Tseng
Internet Platform Application Department,
Computer & Communications Research Laboratories,
Industrial Technology Research Institute, Hsinchu, Taiwan, ROC

Abstract

This paper proposed a method to integrate the AES encrypter and the AES decrypter into a full functional AES crypto-engine. This method can make it a very low-complexity architecture, especially in saving the hardware resource in implementing the AES (Inv)SubBytes module and (Inv)Mixcolumns module, etc. Most designed modules can be used for both AES encryption and decryption. Besides, the architecture can still deliver a high data rate in both en/decryption operations. The proposed architecture is suited for hardware-critical applications, such as smart card, PDA, and mobile phone, etc.

1. Introduction

In January 1997, the National Institute of Standards and Technology (NIST) announced the initiation of an effort to develop the AES and made a formal call for algorithms on September 12, 1997. After reviewed the results of this preliminary research, the algorithms MARS, RC6TM, Rijndael [1], Serpent and Twofish were selected as finalist [2]. And further reviewed public analysis of the finalist, NIST has decided to propose Rijndael as the Advanced Encryption Standard (AES). It is expected to replace the DES and Triple DES so as to fulfil the stricter data security requirement because of its enhanced security levels [3].

Besides, an ASIC solution of AES is also required, because it can be more secure and consumes less power than that implemented by software. In this paper, a 21 cycle AES-128 assembled en/decrypter is presented. We also present some techniques to modify its hardware architecture and reduce its hardware complexity. By comparing it with other designs, we can find that it yields a very low-complexity hardware that can be used for both AES encryption and AES decryption. In next section, we will briefly introduce the AES system. Subsequently, in section 3, the design optimizations of AES operations will be proposed. The designed AES overall architecture was proposed in section 4. Finally, the simulation results and performance report will be discussed in section 5 and conclude it in section 6.

2. AES system

An AES system [4] is a symmetric-key system in which the sender and receiver of a message share a single, common key, which is used to encrypt and decrypt the message. The data length of

Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'02)
1063-6862/02 $17.00 © 2002 **IEEE**

a key or message may be chosen to be any of 128, 192, or 256 bits. The AES encryption/decryption algorithms are shown in Table 1. The AES encryption algorithm has five main operations, namely, AddRoundKey, SubBytes, ShiftRows, MixColumns, and KeyExpansion. The operations in decryption are basically the inverse of the operations in encryption. Besides, the number of rounds of the looping is set to Nr-1 in which Nr is specified according to the AES specification.

Table 1. AES en/decryption algorithm.

AES Encryption	AES Decryption
AddRoundKey	InvAddRoundKey
for round=1 to Nr-1	for round=1 to Nr-1
SubBytes	InvShiftRows
ShiftRows	InvSubBytes
MixColumes	InvAddRoundKey
AddRoundKey	InvMixColumes
end for	end for
SubBytes	InvShiftRows
ShiftRows	InvSubBytes
AddRoundKey	InvAddRoundKey

3. Design Optimization of AES main operations

In some applications, i.e. smart cards, cellular phones, and PDA's, etc. hardware complexity is a very important issue that directly affects to its cost and power consumption. Thus, it is necessary to optimize these AES main operations both in encryption and decryption.

3.1 Combination of SubBytes and InvSubBytes

We know that there are two large tables defined in AES standard [4]. One is for S-box operation, and another is for inverse S-box operation. The first table, S-box table, was used in two functions, i.e. SubBytes and KeyExpansion. And the inverse S-box table was used in function InvSubBytes. We know that these two tables are not the same, thus we must build two different ROMs(256*8bit) to store the table as illustrated in Figure 1 to realize the S-box and inverse S-box operations. Moreover, for a parallel architecture of AES design, it usually needs several tables. For example, in a high-speed design of 128-bit AES, we usually need total 20 S-box modules and 16 inverse S-box modules. In which, there are 16 S-box modules used in implementation of function SubBytes. Four S-box modules are used in implementation of function KeyExpansion. And 16 inverse S-box modules are used in implementation of function InvSubBytes. In this case, a substantial amount of hardware resource will be occupied if SubBytes and InvSubBytes use respective tables in encryption and decryption. It is desirable to obtain a simplified way so as to reduce the hardware complexity.

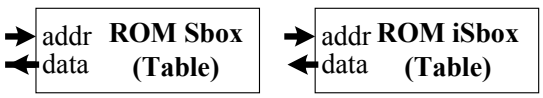

Fig. 1. The S-box ROM and inverse S-box ROM.

As described in standard, the operation of the S-box can be expressed as

$$y = M * \text{multiplicative\_inverse}(x) + c, \quad (1)$$

where

$$M = \begin{pmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{pmatrix} \text{ and } c=[0\ 1\ 1\ 0\ 0\ 0\ 1\ 1] \quad (2)$$

Since the multiplicative inverse (multiplicative_inverse) is a complicated function, the mostly used approach to S-box is to use look-up table to obtain y from x. The inverse equation of Eq.1 can be obtained as

$$x = \text{multiplicative\_inverse}^{-1}(M^{-1}*(y+c)). \quad (3)$$

Since multiplicative_inverse^{-1} () is equivalent to multiplicative_inverse(), the Eq.3 can be expressed as

$$x = \text{multiplicative\_inverse}(M^{-1}* (y+c)). \quad (4)$$

Finally, we must find M^{-1}, the finite field inverse matrix of the matrix M. By the finite field inverse matrix operation, we calculated the inverse of matrix M, named M', and determined it as

$$M' = M^{-1} = \begin{pmatrix} 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \end{pmatrix}. \quad (5)$$

Thus, Eq.3 can be expressed as

$$x = \text{multiplicative\_inverse}(M'*(y+c)). \quad (6)$$

By examine Eq. 1 and Eq. 6, a common look-up table, i.e., multiplicative_inverse(), is employed so the S-box and inverse S-box can be integrated to reduce the hardware requirement for SubBytes and InvSubBytes. Figure 2 shows an integrated SubBytes/InvSubBytes module (inverse-optional S-box module), capable of use in encryption and decryption of AES.

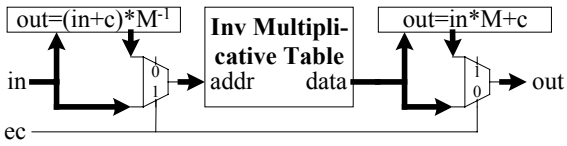

Fig. 2. Final architecture of the inverse-optional S-box module.

The inverse-optional S-box module performs the functions of both S-box and inverse S-box with only one look-up table so that the amount of hardware for implementation of SubBytes and InvSubBytes has a significant decrease of 57%, as compared with the original hardware requirement without the functional integration.

Table 2. Comparisons of S-box's under 0.25μ VLSI technology.

Module	Gate count
inverse S-box	694
S-box	690
inverse-optional S-box	***789***

3.2 MixColumns and InvMixColumns

In the operation of MixColumns and InvMixColumns, two main operations are defined by the following two equations

$$outx = [2\ 3\ 1\ 1]*[a\ b\ c\ d]^T \tag{9}$$

and

$$outy = [E\ B\ D\ 9]*[a\ b\ c\ d]^T \tag{10}$$

In common realization of these two equations, it takes 2 Xtime (AES byte double [x]) and 4 additions in calculating Eq. 11; 12 Xtime and 9 additions in calculating Eq. 12, respectively. In our design, we ungroup these two equations and express it as

$$outx = 2(a + b) + b + (c + d) \tag{11}$$

and

$$outy = 4(2(a + b) + 2(c + d) + (a + c))+ 2(a + b) + b + (c + d). \tag{12}$$

The operations for obtaining the results of Eq.8 and Eq.9 are listed in Table 3. Execution of the first five steps listed results in outx, and then executing the five steps after obtaining outx results in outy. Accordingly, in implementation, the hardware for the first five steps can be used for obtaining both results of the equations above. As shown in Figure 3, the new architecture (Byte_MixColumn module) needs only 8 additions and 4 Xtime operations with comparing to original operation. It reduces the hardware complexity and saves operation resource.

Table 3: Ungroup operation of Eq. 11 and Eq. 12.

Step	Operation
1	w1 = a + b
2	w2 = a + c
3	w3 = c + d
4	w4 = 2 * w1
5	w5 = 2 * w3
6	w6 = w2 + w4 + w5
7	w7 = 2 * w6
8	w8 = 2 * w7
9	outx = b + w3 + w4
10	outy = w8 + outx

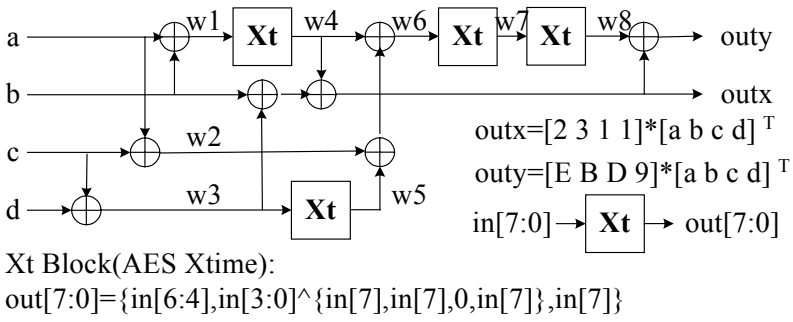

Fig. 3. Block diagram of Byte_MixColumn

Moreover, to realize the overall inverse-optional MixColumns module, we first must integrate 4 Byte_MixColumn module to a new one named Word_MixColumn as illustrate in Figure 4. We then integrate 4 Word_MixColumn modules to create the 128 bits MixColumns module. That is the parallel inverse-optional (block) MixColumns module as illustrated in Figure 5. The inverse-optional MixColumns module performs the functions of both MixColumns and inverse MixColumns with partial shared hardware and saving hardware resource.

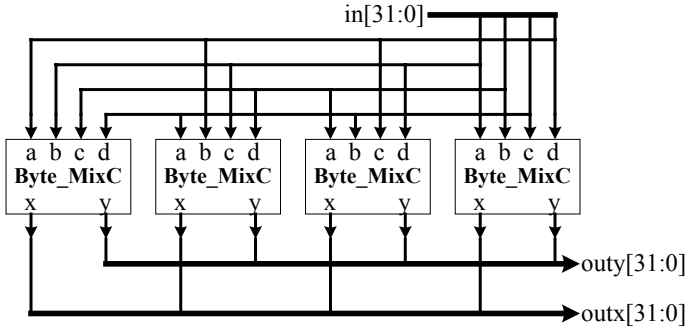

Fig. 4. Block diagram of Word_MixColumn

Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP'02)
1063-6862/02 $17.00 © 2002 **IEEE**

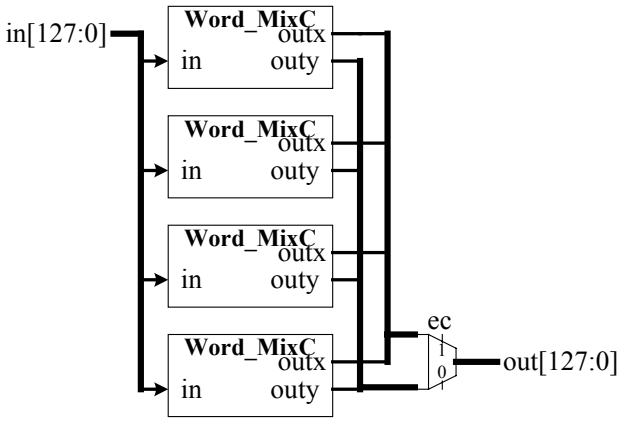

Fig. 5. Parallel Inverse-optional (Block) MixColumns

3.3 KeyExpansion, InvKeyExpansion

As described in AES standard, the function KeyExpansion performs many XOR operations. We assemble the architecture of KeyExpansion and InvKeyExpansion, and the result was shown in Figure 6. A port ec was used as inverse switch. The operations of this module were shown in Figure 7. The input sub key is denoted by sub_key(i). While we want to get the next sub key, sub_key(i+1), we set ec port high and the result can be obtained from the output port. Otherwise, while we want to get the previous sub key, sub_key(i-1), from the output port, we can set the ec port low.

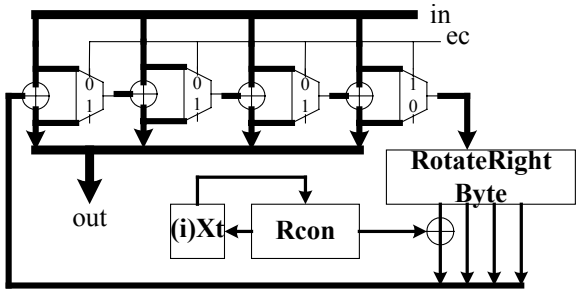

Fig. 6. Architecture of inverse-optional KeyExpansion.

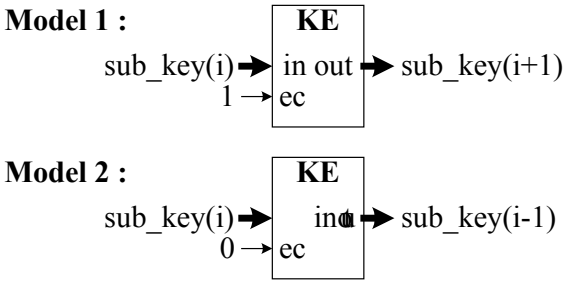

Fig. 7. Operation models of inverse-optional KeyExpansion module

4. Architectures of AES round module

We combine the AES encryption algorithm and decryption algorithm listed in Table 1, and the result was shown in Table 4. The corresponding hardware architecture was shown in Figure 8. There is a control signal "ec" used as a switch for encryption/decryption selection. As illustrated in Table 4, while in decryption operation, the ec signal will be set low. We first must calculate the last sub_key for reverse operation. And then, we start the AES round operation to get encryption result. While in encryption process, the "ec" signal will be set high. We skip the key reverse process and directly start the AES round operation to get encryption result. The hardware implementation of this function was shown in Figure 8.

Table 4. Combination of AES encryption and decryption algorithm

Control: ec=1 for encryption, ec=0 for decryption

if (ec==0) //inverse key
for (i=0;i<10;i++)
Inv_Opt_keyexpansion(key,1);
for (i=0;i<=round;i++)
{ addroundkey;
if (i==10) break;
Inv_Opt_keyexpansion(key,ec);
if (ec==1)
{ Inv_Opt_subbytes(ec);
Inv_Opt_shiftrows(ec);
if (i<9) Inv_Opt_mixcolumns(ec);
} else
{ if (i>0) Inv_Opt_mixcolumns(ec);
Inv_Opt_subbytes(ec);
Inv_Opt_shiftrows(ec);
} }

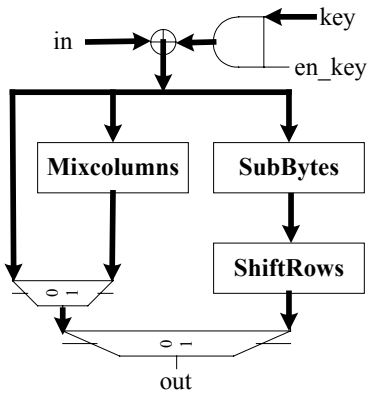

Fig. 8. Modified inverse-optional AES algorithm.

Figure 9 shows the overall architecture of our AES hardware design. In this design, there are total 3 128-bits registers used in our design, i.e. the data registers, key registers and the key backup registers. The data register is for data storage, and the other two registers are for key storage. We can easily start next en/decryption operation by select a right sub key from these two key registers and get a desired middle sub key by KeyExpansion module. In each round operation, the required sub key and data will be transfer into the Round module and the result of each round will write back to data register. Finally, the en/decryption result can be read from the data register.

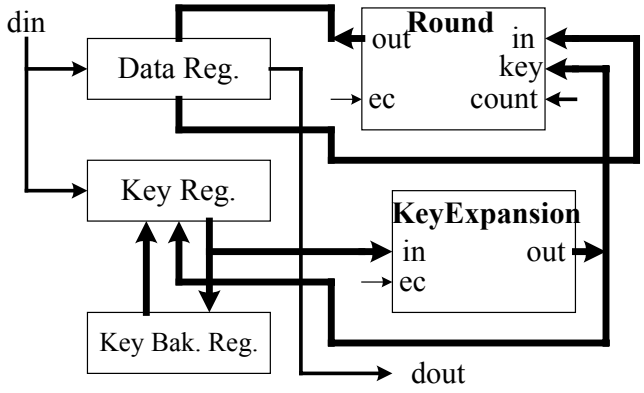

Fig. 9. Overall architecture of our AES hardware design.

5. Performance

In our design, the simulation results show that the critical path was less than 10ns using TSMC 0.25µ technology at SLOW case. The operation clock frequency can be 100Mhz. It takes 21 clock cycles to complete an AES-128 en/decryption. The throughput is about 609Mbit/sec. The integrated design of AES-128 encryption and decryption results in a simply hardware of 31957 gate counts. It shows that the proposed design can deliver very high data rate.

6. Conclusions

We have shown that the AES encrypter and decrypter are efficiently integrated. The overall architecture is low-complexity and is suited for smart card or other hardware critical applications. Besides, it can still deliver a high data rate in both en/decryption operations. The advantages of our design are listed in the following:

Combination of SubBytes and InvSubBytes module: We solve the dual lookup table problem using inverse finite field matrix and replace them with a new multiplicative inverse lookup table. Thus, the new inverse-optional SubBytes module utilizes less hardware resource with comparing to original dual lookup table architecture. Hardware complexity is reduced to 57% of original.

Optimization of MixColumns operation for both decryption and encryption: Ungrouping the equations so that there are five steps can be used for both MixColumn and InvMixColumn operation. Hardware complexity can be reduced by more than 50% of original.

High-utilized design: The encryption and decryption share most part of the hardware circuit. However, overall hardware complexity is reduced to 68% of other designs that perform encryption and decryption using different modules and ROMs [5][6][7][8].

References

[1] J. Daemen, V. RijnMen, "The Rijndael Block Cipher," AES proposal, First AES Candidate Conference (AES1), Aug. 20-22, 1998.

[2] T. Ichikawa, T. Kasuya, M. Matsui, "Hardware Evaluation of the AES finalists," The Third Advanced Encryption Standard (AES3) Candidata Conference, April, 13-14, 2000.

[3] E. Barker, L. Bassham, W. burr, M. Dworkin, J. Foti, J. Nechvatal, and E. Roback, "Report on the Development of the Advanced Encryption Standard (AES)." Available at http://home.ecn.ab.ca/~jsavard/crypto/co040801.htm.

[4] "Advanced Encryption Standard (AES)" Federal Information Processing Standards Publication 197, Nov. 26, 2001.

[5] K. Gai, P. Chodowiec, "Comparison of the hardware Performance of the AES Candidates using Reconfigurable Hardware." The Third Advanced Encryption Standard (AES3) Candidata Conference, 13-14 April 2000.

[6] A. Dandalis, V.L. Prasanna, J.D.P. Rolim, "Acomparative Study of performance of AES Candidates Using FPGAs." The Thied Advanced Encryption Standard of AES Candidate Conference, 13-14 April 2000.

[7] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists." The Thied Advanced Encryption Standard of AES Candidate Conference, 13-14 April 2000.

[8] M. Mcloone, J.V McCanny, "High Performance Single-Chip FPGA Rijndael Algorithm Implementation." Cryptographic Hardware and Embedded Systems, Workshop on Cryptographic Hardware and Embedded Systems, May 14-16, 2001.