Investigation of Compact Hardware Implementation of the Advanced Encryption Standard

Namin Yu
Electrical and Computer Engineering
Memorial University of Newfoundland
Email: namin@engr.mun.ca

Howard M. Heys
Electrical and Computer Engineering
Memorial University of Newfoundland
Email: howard@engr.mun.ca

Abstract

A compact and efficient implementation of the Advanced Encryption Standard (AES) is the desirable encryption IP core for any practical low-end embedded application. In this paper, we investigate various architectures for compact AES implementations in 0.18-um CMOS technology. We first investigate a new compact digital hardware implementation of AES s-boxes applying the discovery of linear redundancy in AES s-boxes. Although the new circuit has a small size, the speed of this implementation is also reduced. Encryption architectures without key scheduling employing four s-boxes and only one s-box are implemented using our new AES s-boxes, as well as based on other compact s-box structures. The comparison of six implementations indicates that the implementation using four s-boxes based on arithmetic operations in GF(2^4) has the best trade-off of area and speed. Therefore, using this s-box implementation, a complete encryption-decryption architecture with key scheduling employing the four s-box structure is implemented. In order to be adaptive to various practical applications, we optimize the implementation with the four s-box structure to support five different operation modes.

Keywords: *Encryption, Cipher, Security, Digital Hardware*

1. Introduction

Since the National Institute of Standards and Technology (NIST) announced the selection of Rijndael as the Advanced Encryption Standard (AES) in November 2001 [1], AES has been accepted as the popular means to encrypt sensitive commercial and government data. Various hardware implementation architectures and optimizations have been suggested for different applications. Those to achieve high speed are usually very expensive in hardware complexity. The large area of such architectures may not be suitable for practical low-end embedded applications, such as smart cards, PDAs, cell phones, and other mobile devices. These small embedded applications do not require high speed or throughput, but are area critical. Therefore, reducing hardware resources to gain a compact and efficient implementation circuit is an increasing demand.

AES [1] is a symmetric-key block cipher, which supports different key lengths of 128, 192 or 256 bits. It is based on rounds of byte-oriented substitution and linear transforms with the fixed block length of 128 bits. There are five main functions for the encryption process, namely byte substitution, rotation, linear transformation, key addition, and key expansion. Unlike DES, the decryption process has a different structure from the encryption. However, with some change in key expansion, an equivalent decryption structure can be achieved using inverse functions for the byte substitution, rotation, and linear transformation.

2. New Implementation of AES S-box

For the byte substitution operation, the 128-bit input data is taken as 16 bytes and each byte is substituted by the corresponding element in the s-box. In terms of AES hardware implementation, the most costly components are s-boxes. Schemes employing look-up tables or direct implementation of 8-bit Boolean functions cost a lot of hardware resources.

In [2], J. Fuller and W. Millan reported the important discovery of linear redundancy in AES s-boxes. By investigating the local structure of the Hamming distance between Boolean functions, Fuller and Millan used a new method to determine the equivalence between the 8 Boolean functions of the AES s-box outputs. In generally, an n-input Boolean function $g(x)$, $x \in \{0,1\}^n$, can be represented by its equivalent Boolean function $f(x)$ using a binary matrix D, two binary vectors p and q, and a binary constant c. That is,

$$g(x) = f(Dx \oplus p) \oplus qx \oplus c$$

For the AES s-box, the relations are simpler. Only binary matrix D and binary constant c are needed. Therefore, the output Boolean function $b_j(x)$ can be easily represented by the form $b_j(x) = b_i(D_{ij}x) \oplus c_j$, based on the known b_i Boolean function.

As noted in [2], this property of s-boxes gives a hint for compact hardware implementation. We only need to implement one Boolean function for the s-box and then utilize the transformations between the output bits to get the 8-bit result of the whole s-box. The combinational logic implementation of a Boolean function and 7 mapping matrices should cost much less in hardware resources than a direct implementation of all eight 8-bit Boolean functions.

Let us label the s-box output byte as $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$. The implementation of all Boolean functions is derived from the execution of least significant bit b_0. In our scheme, the s-box consists of three main parts, namely the D matrix block, MUX, and b_0_logic. Fig.1 shows the structure used to produce each output bit of the s-box. We refer to the new implementation structure as the LR s-box implementation since it is based on the concept of linear redundancy.

In the new s-box implementation, we apply factoring to minimize and reuse hardware resources for each block. After integrating all parts together, we use a 0.18-um CMOS standard cell library for the synthesis. The synthesis reports indicate that the circuit only needs the equivalent of 296 2-input NAND gates totally. The D matrix block occupies 40.9% of it. The b_0_logic block takes 31.4% and the multiplexer takes 28.7%.

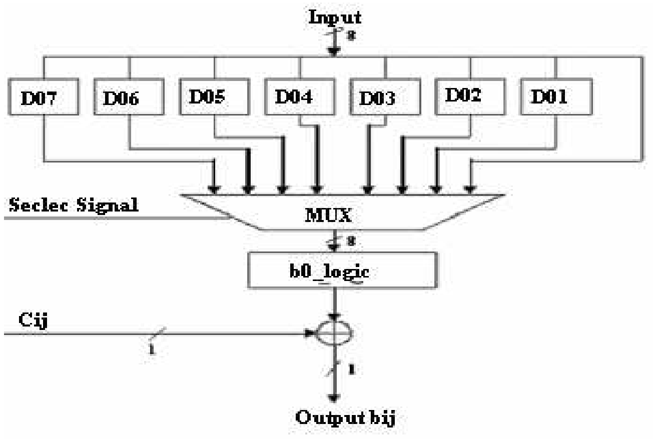

Fig.1. S-box Implementation Structure Based on Linear Redundancy

Since it is very difficult to compare the performance of implementations using different technology libraries and synthesis tools, we applied the same technology and tools to other compact s-boxes implementations using composite fields based on GF (2^4) [3] and GF (2^2) [4]. The synthesis results presented in Table 1 show that the new implementation requires 11% fewer gates than the other two methods.

Table 1. Area Complexity of S-box Implementations (1 gate = 2-input NAND)

Implementation	GF (2^2) (gates)	GF (2^4) (gates)	LR (gates)
Inverter	232	241	——
Isomorphism	27	23	——
Inv_isomorphism	31	30	——
Affine Transformation	37	37	——
b_0_logic	——	——	93
D matrix block	——	——	121
MUX	——	——	80
S-box (totally)	327	331	296

Since our s-box is processing the data bit by bit, not byte by byte as in the other two methods, our implementation is about 8 times slower than other implementations. Moreover, the area calculation does not include the additional 8-bit shift registers required for storing the outputs of the s-boxes.

3. Performance of Encryption Architectures

Based on the above studies on three compact implementations of the AES s-box, we implement the encryption architectures without key scheduling using an iterative loop structure. The encryption datapath is shown in Fig. 2, where the unlabelled boxes in the diagram represent registers. Since the architecture implements 4 s-boxes per iteration, a full round requires 4 iterations.

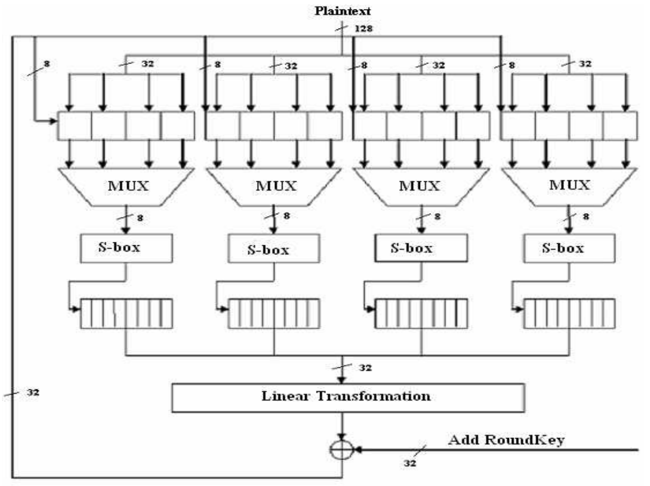

Fig.2. Encryption Datapath for Four S-boxes

The 32-bit shift registers not only work as data registers but also implement the rotation function. By using different s-box implementations, the architecture is changed a little. Because the LR s-boxes take 8 clock cycles to produce the 4 bytes, while the linear transformation needs the 32-bit data at one time, we have to insert four 8-bit shift registers to store the output of s-boxes to prepare the input for the linear transformation. This increases the count of gates in the circuit.

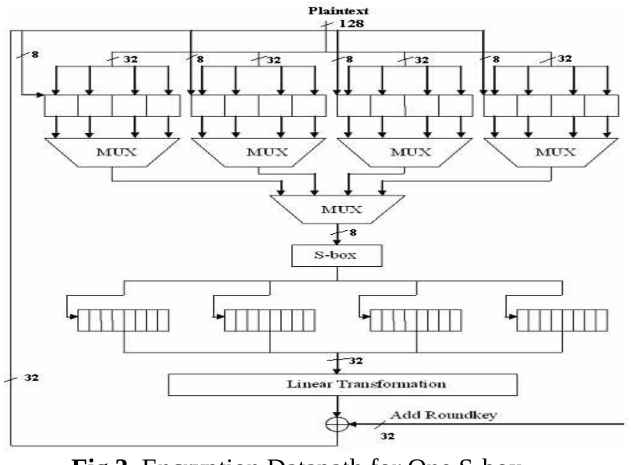

Fig.3. Encryption Datapath for One S-box

In order to gain a more compact circuit, we have also explored the method of using only one s-box instead of four in the whole encryption architecture. The encryption datapath is shown in Fig. 3. Obviously, the new encryption architecture is really minimized a lot since the s-boxes are the most costly components in the circuit. However, the reduction of area is at the cost of speed. The encryption architectures of one s-box is 4 times slower than those of four s-boxes.

We have used 0.18 um CMOS library in our implementations and applied Synopsis as our synthesis tool. After simulation and synthesis, we find that for the four s-box architecture using LR s-boxes, about 3.6k gates are required to implement the round operation with a maximum clock frequency of 131 MHz. The implementation with only one s-box requires about 2.5k gates with a maximum clock frequency of 123 MHz. The results are shown in Table 2. We also carry out other compact encryption implementations, which are based on the s-boxes implemented in composite GF (2^4) and GF (2^2). The results are also shown in Table 2 for comparison.

Table 2. Implementations Performance Comparison

Encryption Datapath	Area (gates)	Throughput (Mbps)	Throughput/Area (kbps/gates)
Based on 4 S-boxes in GF (2^4)	3569	179.78	50.37
Based on 4 S-boxes in GF (2^2)	3540	152.29	43.02
Based on 4 LR S-boxes	3581	50.84	14.20
Based on 1 S-box in GF (2^4)	2612	62.33	23.86
Based on 1 S-box in GF (2^2)	2624	50.53	19.26
Based on 1 LR S-box	2545	12.22	4.80

4. Encryption-Decryption Architecture with Key Scheduling

Based on the study of the AES s-boxes and encryption datapath, the comparison of six implementations in Table 2 indicates that the implementation using four s-boxes based on arithmetic operations in GF(2^4) has the best trade-off of area and speed. Therefore we have implemented an encryption-decryption architecture with key scheduling using four s-boxes in GF(2^4). In doing so, we merged the encryption and decryption functionality and generated circuitry to provide the on-the-fly key scheduling for encryption and decryption. In this implementation, we try to reuse and share the hardware components as much as possible to reduce the circuit size. The encryption-decryption architecture and key expander are shown in Fig. 4 and Fig. 5, respectively.

4.1 Sharing Between Encryption and Decryption Processes

As was mentioned, the decryption process of AES has a different structure from the encryption. So in order to share the hardware resources between the encryption and decryption processes, we have modified the order of operations for the structure. Firstly, we exchange the order of byte substitution and rotation for the encryption process. Since both of the operations are byte-oriented, this does not alter the result of the round. The second change of structure is exchanging the order of liner transformation and adding the round key for the decryption process. This change causes a corresponding change in the key expander to add the inverse linear transformation at the end of key scheduling [2]. Also we can share the multiplicative inverse in GF (2^8) for the s-box and inverse s-box, as well as share hardware between the linear transformation and its inverse operation.

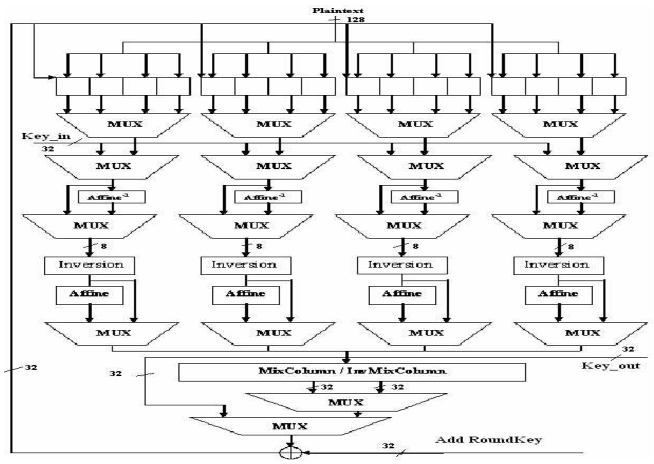

Fig. 4. Encryption-Decryption Datapath

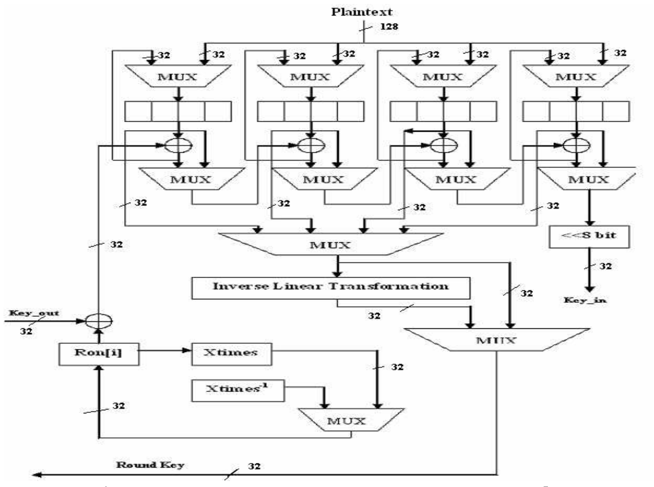

Fig. 5. Encryption-Decryption Key Expander

4.2 Sharing Between Datapath and Key Expander

The key expander needs byte substitution operations to generate the key for encryption and decryption. Since the s-boxes are one of the most costly components in the whole circuit, sharing between the datapath and the key expander is a good method to reduce the circuit size. We use multiplexers after the rotation operation to select to process cipher data or the round key. The key is taken after byte substitution back to the key expander to continue the key process. This sharing of s-boxes increases one clock cycle in datapath for each round in the encryption and decryption.

After simulation and synthesis, we find that the complete four s-box structure for encryption-decryption incorporating the key scheduling architecture requires about 6.7k gates with a maximum clock frequency of 52.7 MHz. The throughput of the circuit is 112.40 Mbps.

5. The Five-Mode System

In order to be used for different applications, we have optimized the implementation to support five different operation modes: Electronic Codebook mode (ECB), Cipher Block Chaining mode (CBC), Cipher Feedback mode (CFB), Output Feedback mode (OFB), and counter mode (CTR) [5]. The five-mode architecture is shown in Fig.6.

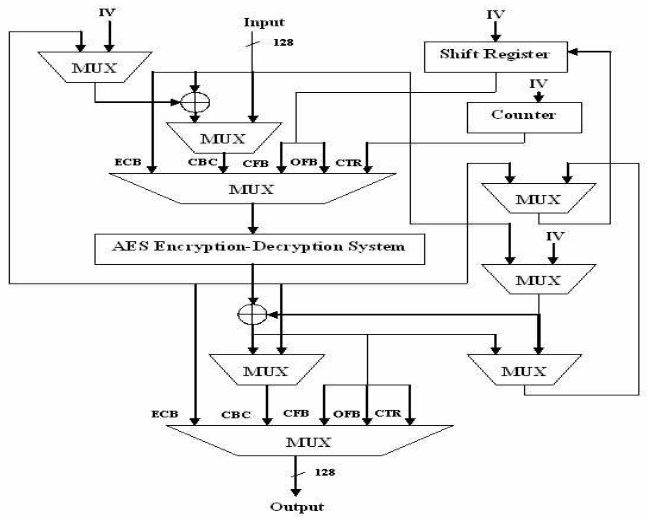

Fig.6. Five-mode System

ECB is the simplest operation mode since it uses the same key for each block of data and the input to the encryption-decryption system is the current plaintext.

In CBC mode, the input to the encryption-decryption system is the XOR of the current plaintext and preceding ciphertext. The first input is the XOR of the first block of plaintext and an initialization vector (IV). Thus the system can generate different ciphertext for the same plaintext. We add multiplexers to choose between the IV and preceding ciphertext and plaintext. Also we insert a multiplexer to choose between the encryption and decryption data.

CFB mode uses AES as a stream cipher. The input to the encryption-decryption system is the preceding ciphertext and the ciphertext is the XOR of the plaintext and the output of the encryption-decryption system. Rather than process the data block by block, CFB divides the plaintext into small segments. So we use a shift register to shift the data by segment and feed it into the encryption-decryption system.

OFB mode is similar to CFB. The only difference is that the input to encryption-decryption system is the preceding output of the encryption-decryption system. So we use a multiplexer to choose between the CFB and OFB to feed back into the shift register. The ciphertext is produced as the XOR of the current plaintext and the output of the encryption-decryption system.

In CTR mode, the input to the encryption-decryption system is a counter. The counter is initialized as IV and increases for each block. The ciphertext is the XOR of the plaintext and the output of the encryption-decryption system. There is no chaining in counter mode.

The five-mode system is implemented by using 0.18-um CMOS standard cell library technology. The resulting circuit has the size of 11.3k gates (based on a 64-bit counter) with a throughput of 100.67 Mbps.

6. Conclusion

By applying the discovery of linear redundancy in the implementation of the AES s-boxes, we have realized a new compact implementation of the s-box. Although the new resulting circuit of the s-box is about 11% less than the other compact s-box implementations, it is 8 times slower than the other compact implementations. For the complete encryption datapath, the implementation using four s-boxes based on arithmetic operations in GF(2^4) shows the best trade-off of area and speed. The one s-box structure uses 16% less hardware resources but only has 26% of the speed of the four s-box structure. Employing sharing between the encryption and decryption processes and reusing common components for different blocks, an compact encryption-decryption system using four s-boxes in GF(2^4) is obtained. A five-mode system is also presented that is able to support various low-end embedded applications. The final result shows that, with the architecture becoming complete, the difference between various compact implementations of AES s-box is not so important in the area consideration, and the four s-box structure is most suitable in terms of both size and speed.

References

[1] W. Stallings, "The Advanced Encryption Standard", *CRYPTOLOGIA*, Volume XXVI No. 3, July 2002, pp. 165-186.

[2] J. Fuller and W. Millan, "Linear Redundancy in S-Boxes", *Fast Software Encryption 2003*, Lecture Notes in Computer Science 2887, Springer, 2003, pp. 74-86.

[3] J. Wolkerstorfer, E. Oswald and M. Lamberger, "An ASIC implementation of the AES SBoxes", *The Cryptographer's Track at the RSA Conference 2002*, Lecture Notes in Computer Science 2271, Springer, 2002, pp. 67-78.

[4] A. Satoh, S. Morioka, K. Takano, S. Munetoh, "A Compact Rijndael Hardware Architecture with S-box Optimization", *ASIACRYPT 2001*, Lecture Notes in Computer Science 2248, Springer, 2001, pp. 239-254

[5] W. Stallings, *Cryptography and Network Security: Principles and Practice*, third edition, Prentice Hall 2003, pp. 90-99.