A High-Throughput Low-Cost AES Cipher Chip

Tsung-Fu Lin, Chih-Pin Su, Chih-Tsun Huang, and Cheng-Wen Wu
Laboratory for Reliable Computing
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013
R.O.C.

Abstract

We propose an efficient hardware implementation of the AES (Advanced Encryption Standard) algorithm, with key expansion capability. Compared with the widely used table-lookup technique, the proposed basis transformation technique reduces the hardware overhead of the S-box by 64%. Our pipelined design has a very high throughput rate. Using a typical 0.35μm CMOS technology, a 200MHz clock is easily achieved, and the throughput rate is 2.381 Gbps for 128-bit keys, 2.008 Gbps for 192-bit keys, and 1.736 Gbps for 256-bit keys. Testability of the design also is considered. The hardware cost of the AES design is about 58.5K gates.

1. Introduction

The large and growing number of Internet and wireless communication users has led to an increasing demand of security measures and devices for protecting the user data transmitted over the open channels. Two kinds of cryptographic systems can be used for that purpose, i.e., the symmetric-key and asymmetric-key crypto-systems. The symmetric-key cryptography (such as DES [1], 3DES, and AES [2]) uses an identical key between the sender and receiver, both to encrypt the message text and decrypt the cipher text. The asymmetric-key cryptography (such as the RSA [3] and Elliptic Curve algorithms) uses different keys for encryption and decryption, eliminating the key transportation dilemma. Because of its high speed, the symmetric-key cryptography is more suitable for the encryption of a large amount of data. The Advanced Encryption Standard (AES) algorithm, which was announced by the National Institute of Standards and Technology (NIST) of the United States, has been widely accepted for replacing DES as the new symmetric encryption algorithm [2].

The AES encryption is considered to be efficient both for hardware and software implementations. Some works have been presented on hardware implementations of the AES algorithm using FPGA [4–6] and ASIC [7–10]. Most of them used the ROM/RAM-based look-up table (LUT) to implement the S-box operation in the AES algorithm. It is cost-effective for SRAM-based FPGAs, but may not be a good choice for ASIC implementation. In this paper, we present a more efficient hardware design for the AES algorithm. Instead of the LUT approach, we implement the inverse function of the S-box by using a basis transformation in the finite field. The proposed design is much more cost-effective than previous works. Furthermore, design-for-testability has been considered. With a 64% area reduction in S-box and 51% in total area reduction as compared with the best previous result, the speed also is enhanced. Using a typical 0.35μm CMOS technology, a 200MHz clock is easily achieved, and the throughput rate is 2.381 Gbps for 128-bit keys, 2.008 Gbps for 192-bit keys, and 1.736 Gbps for 256-bit keys. We used only about 58.5K gates for the design.

2. AES Algorithm

The AES algorithm is also known as the Rijndael algorithm [2], which is a symmetric block cipher that processes data blocks of 128 bits using the cipher key of length 128, 192, or 256 bits. Each data block consists of a 4×4 array of bytes called the *State*, on which the basic operations of the AES algorithm are performed. The top-level view of the AES encryption/decryption procedure is shown in Fig. 1. After an initial round key addition, a round function consisting of four different transformations—SubBytes(), ShiftRows(), MixColumns() and AddRoundKey()—is applied to the data block, i.e., the State array. The round function is performed iteratively for 10, 12, or 14 times, depending on the key length. Note that in the last round, MixColumns() does not applied. The four transformations are described briefly as follows [2]:

1. SubBytes(): a non-linear byte substitution that operates independently on each byte of the State using a substitution table (called the S-box).

0-7803-7363-4/02/$17.00 ©2002 IEEE.

2. ShiftRows(): a circular shifting operation on the rows of the State with different numbers of bytes (offsets).

3. MixColumns(): the operation that mixes the bytes in each column by the multiplication of the State with a fixed polynomial modulo $x^4 + 1$.

4. AddRoundKey(): an XOR operation that adds a round key to the State in each iteration, where the round keys are generated during the key expansion phase.

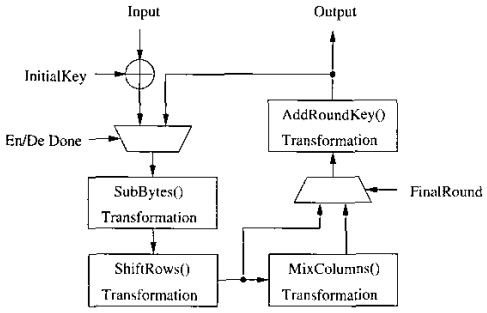

Figure 1. The top-level view of the AES algorithm.

The SubBytes() (S-box) transformation, which consists of a multiplicative inversion over $GF(2^8)$ and an affine transform, is the most critical part in the AES algorithm, so far as computational complexity is concerned. The S-box operation is required both for the encryption and key expansion. The S-box dominates the hardware complexity of the AES circuit. Conventionally, the coefficients of the S-box and inverse S-box are stored in LUTs [7], or a hard-wired multiplicative inverter over $GF(2^8)$ can be used, together with the affine transform function. The dedicated inverter, however, has a high area overhead. In the next section, we will present an efficient implementation by a transformation of the S-box over the finite field.

3. Efficient Implementation of the S-box

The S-box is an invertible function and is composed of two transformations: 1) take the multiplicative inverse in $GF(2^8)$ and the element 00 is mapped to itself; and 2) apply an affine transformation (over $GF(2)$): $b'_i = b_i \oplus b_{(i+4)8 \bmod} \oplus b_{(i+5)8 \bmod} \oplus b_{(i+6)8 \bmod} \oplus b_{(i+7)8 \bmod} \oplus c_i$ for $0 \le i < 8$, where b_i is the i^{th} bit of the byte, and c_i is the i^{th} bit of $c = 63$ (hexadecimal) or 01100011 (binary). We can represent elements in $GF(2^8)$ as polynomials of degree one (i.e., $px + q$, $p, q \in GF(2^4)$) to reduce the complexity [11]. The inverse operation in the S-box can then be computed by taking the inverse of the polynomial $px+q$, with its irreducible polynomial being x^2+Ax+B, $A, B \in GF(2^4)$. The multiplicative inverse for an arbitrary $px + q$ can be obtained from the following equation [11]:

$$\begin{aligned}(px+q)^{-1} &= p(p^2B + pqA + q^2)^{-1}x \\ &\quad +(q+pA)(p^2B + pqA + q^2)^{-1}\end{aligned}$$

Finding the inverse over $GF(2^8)$ is now reduced to finding the inverse over $GF(2^4)$, which can be done by a much smaller LUT. In order to minimize the hardware overhead in multiplication and squaring, A is assigned as the multiplicative identity in the normal basis, and B is chosen as an element with minimum Hamming weight, e.g., 0001.

To implement SubBytes() by basis transformation as suggested in [11], we search for an efficient S-box transformation by choosing a suitable basis between $GF(2^4)$ and $GF(2^8)$ such that it satisfies certain criteria: First of all, the irreducible polynomial defined in AES for constructing $GF(2^8)$ is $m(x) = x^8 + x^4 + x^3 + x + 1$, which is not a primitive polynomial. A proper primitive irreducible polynomial, $P(x)$, should be obtained before we can perform a basis transformation of the elements in $GF(2^8)$ to those over $GF(2^4)$. Let α be a root of $P(x)$ and β be a root of $m(x)$, then there exists a k such that $\beta = \alpha^k$ (see, e.g., Appendix C of [12]), i.e., for any set of standard basis, $\{\beta^7, \beta^6, \beta^5, \beta^4, \beta^3, \beta^2, \beta\ 1\} = \{\alpha^{7k}, \alpha^{6k}, \alpha^{5k}, \alpha^{4k}, \alpha^{3k}, \alpha^{2k}, \alpha^k, 1\}$. The corresponding basis transformation matrix T^α_β must satisfy $\mathfrak{a}^T = T^\alpha_\beta \cdot \mathfrak{b}^T$, where $\mathfrak{a} = \sum_{i=0}^{7} a_i\alpha^i$ and $\mathfrak{b} = \sum_{i=0}^{7} b_i\beta^i, a_i, b_i \in \{1, 0\}$.

To construct an irreducible polynomial $(x^2 + Ax + B)$, let $\lambda = \alpha^i$ be a nonzero element in $GF(2^8)$. If λ and its conjugate λ^{16} are distinct, then $(x + \lambda)(x + \lambda^{16}) = x^2 + Ax + B$ is a degree-2 irreducible polynomial [12], where $A, B \in GF(2^4)$. Since A has to be 1, the objective is to find distinct conjugate pairs $\{\lambda, \lambda^{16}\}$ such that $\lambda \neq \lambda^{16}$ and $\lambda + \lambda^{16} = 1$. Also, to derive B with minimal hamming weight in the normal basis, the conjugate pairs $\{\lambda, \lambda^{16}\}$ must satisfy $\gamma = \lambda \cdot \lambda^{16} = \lambda^{17}$, $\gamma^2 = \lambda^{17}$, $\gamma^4 = \lambda^{17}$, or $\gamma^8 = \lambda^{17}$, so that $\{\gamma^8, \gamma^4, \gamma^2, \gamma\}$ form a basis.

Based on the previous discussion, we define the basis for computing the inverse as $\{\theta_7, \theta_6, \theta_5, \theta_4, \theta_3, \theta_2, \theta_1, \theta_0\}$, where $\theta_7 = \gamma^8\lambda$, $\theta_6 = \gamma^4\lambda$, $\theta_5 = \gamma^2\lambda$, $\theta_4 = \gamma\lambda$, $\theta_3 = \gamma^8$, $\theta_2 = \gamma^4$, $\theta_1 = \gamma^2$ and $\theta_0 = \gamma$. To compute the inverse of $\mathfrak{c}$, let $\mathfrak{c}$ be $\sum_{i=0}^{7} c_i\theta_i, c_i \in \{0, 1\}$, i.e., $\mathfrak{c} = (c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$. Then

$$\mathfrak{c}^T = T^\gamma_\alpha \cdot \mathfrak{a}^T = T^\gamma_\alpha T^\alpha_\beta \cdot \mathfrak{b}^T = T^\gamma_\beta \cdot \mathfrak{b}^T$$

There are many possibilities for choosing the primitive irreducible polynomial and the expression of B. In our implementation, a proper basis is selected to minimize the area and the delay of the AES chip. In addition, the affine transformation in the S-box is combined with the

basis transformation. The block diagram of the proposed S-box design is shown in Fig. 2. Four transformations (matrix operations) are designed and implemented such that SubBytes() and Inverse SubBytes() can use the same inverse module.

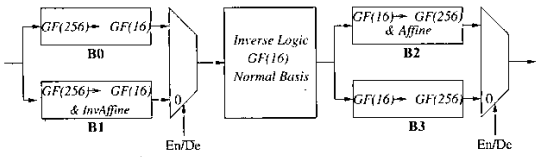

Figure 2. The proposed S-box design: the SubBytes and Inverse SubBytes transformations.

4. Architecture of the AES Chip

Figure 3 shows the architecture of our AES chip. The Main Controller generates control signals for data transportation, key expansion, encryption, and decryption. When an encryption/decryption process begins, the I/O Interface first gathers the slices of the initial key through a 32-bit data bus. The Key Controller then executes the key expansion routine. It controls the datapath to generate all the necessary round keys and stores them in four 512-bit SRAM modules. All the round keys will be retrieved during the encryption or decryption phase. Once the round keys are ready, the En/De Controller will take over the control and perform the encryption or decryption whenever a 128-bit data block is ready from the I/O Interface. The AES round function will then be applied for 10, 12, or 14 times depending on the key size. Finally, the processed data will be exported through the external data bus.

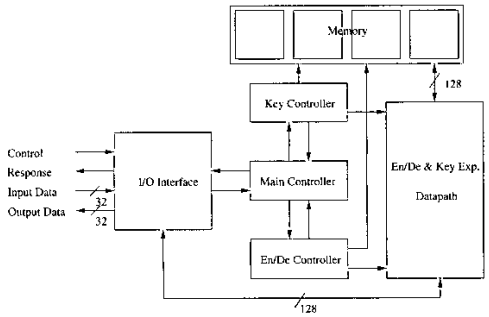

Figure 3. The architecture of our AES chip.

A 4-stage pipeline is used to enhance the performance of the datapath, as shown in Fig. 4. The first three stages implement SubBytes(), while the last one implements ShiftRows(), MixColumns() and AddRoundKey(). Note that in SubBytes(), three $GF(2^4)$ multipliers and two $GF(2^4)$ squaring circuits are used to compute $(px+q)^{-1}$. By choosing an optimized normal basis over $GF(2^4)$, the area of the multipliers is greatly reduced, and the squaring circuits simply perform the circular shifting. The multiplicative inverse of $GF(2^4)$ is implemented by random logic, since it is very small.

In addition to high performance and low hardware cost, increasing the testability is also one of our major goals. Scan design is used on the controllers. Four scan chains are implemented. With only 89 test patterns, the fault coverage is 97.1%. The memory blocks are directly multiplexed out in the test mode, since they are quite small. Of course the chip integrator can use a peripheral test method or implement a memory BIST circuit (can be shared with other memory blocks at the system level) [13]. As to our datapath design, it consists of 16 identical byte-oriented circuitries for SubBytes(), so pseudo-exhaustive patterns are applied. The fault coverage is about 99.8% reported by a commercial fault simulator) with 512 test patterns (the 8-bit exhaustive patterns are applied twice to cover the faults in the multiplexers).

5. Experimental Results

In Table 1 we compare the hardware overhead among different S-box implementations using a typical 0.35μm CMOS technology. In our design, the S-box (see Fig. 2) was synthesized using a standard-cell library. It was done for all possible bases which satisfy the the criteria discussed in Sec. 3. Based on the results of the experiments, we selected the basis with $\beta = \alpha^{25}$ and $\lambda = \alpha^{123}$, since it results in the shortest critical path, i.e., a delay of only 5ns using the 0.35μm cell library. As a result, our S-box design achieves a 64% area reduction as compared with the SRAM-based design, and a 59% reduction as compared with the ROM-based one.

Table 1. Comparison of 8-bit S-boxes.

	SRAM 256x8	ROM 2x256x8	Our design
Gate Count	2138.00	1866.00	762.67

Table 2 shows a comparison of area and performance among different AES implementations. In [9], all the AES finalists are evaluated, where the area is not the primary concern. Unlike the approach in [7], the throughput of our design varies for different key length. The key length of 128 bits achieves the highest throughput because the fewest rounds are required. The total gate count of the proposed AES design is 58.5K. With a critical path delay of 5ns, the throughput is about 2.381 Gbps for 128-bit keys, 2.008 Gbps for 192-bit keys, and 1.736 Gbps for 256-bit keys. Considering the ratio of throughput over gate count, our design is more than 3 times better than previous ones. Note that our chip contains the

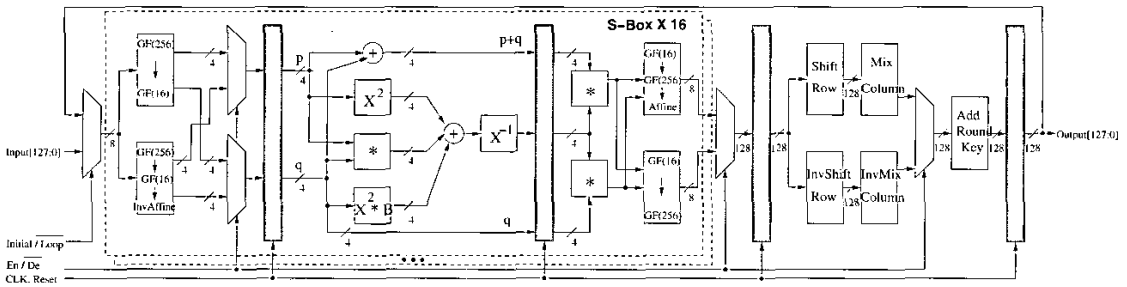

Figure 4. The pipelined datapath of our AES chip.

Key Expansion routine, i.e., Key Schedule, but Li's design does not.

Table 2. Comparison of AES designs.

	Ichikawa [9]	Kuo [10]	Li [7]	Ours
Technology	$0.35\mu m$	$0.18\mu m$	$0.35\mu m$	$0.35\mu m$
Clock Rate	N/A	N/A	166MHz	200MHz
Throughput (Gbps)	1.95	1.82	1.328	2.381 2.008 1.736
Gate Count	612K	173K	120K	58.430K
Throughput/ Gate Count	3.18	10.52	11.07	41.49 34.98 30.24

6. Conclusions

We have presented a high throughput, area efficient implementation of the AES algorithm. The complexity of the S-box is greatly reduced by a basis transformation from $GF(2^8)$ to $GF(2^4)$. There is a 64% hardware reduction in S-box and 51% in total area reduction compared with the previous LUT approaches. The pipelined AES chip provides a very high throughput while keeping the area small. In addition, our design also can perform key expansion, so the AES encryption can be done, too. Finally, testability has been stressed in the proposed design.

Acknowledgment

The authors would like to thank Prof. Chi-Chao Chao of NTHU for his valuable comments on the basis transformation.

References

[1] National Institute of Standards and Technology (NIST), *Data Encryption Standard (DES)*, National Technical Information Service, Springfield, VA 22161, Oct. 1999.

[2] National Institute of Standards and Technology (NIST), *Advanced Encryption Standard (AES)*, National Technical Information Service, Springfield, VA 22161, Nov. 2001.

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[4] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-Based performance evaluation of the AES block cipher candidate algorithm finalists", *IEEE Trans. VLSI Systems*, vol. 9, no. 4, pp. 545–557, Aug. 2001.

[5] N. Weaver and J. Wawrzynek, "A comparison of the AES candidates amenability to FPGA implementation", in *Proc. 3rd AES Candidate Conference*, 2000.

[6] K. Gaj and P. Chodowiec, "Comparison of the hardware performance of the AES candidates using reconfigurable hardware", in *Proc. 3rd AES Candidate Conference*, 2000.

[7] M.-H. Li, "A Gbps AES cipher", Master Thesis, Dept. Computer Science, National Tsing Hua University, Hsinchu, Taiwan, June 2001.

[8] B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke, "Hardware performance simulations of round 2 Advanced Encryption Standard algorithm", in *Proc. 3rd AES Candidate Conference*, 2000.

[9] T. Ichikawa, T. Kasuya, and M. Matsui, "Hardware evaluation of the AES finalists", in *Proc. 3rd AES Candidate Conference*, 2000.

[10] H. Kuo and I. Verbauwhede, "Architectural optimization for a 1.82 Gbits/sec VLSI implementation of the AES Rijndael algorithm", in *Cryptographic Hardware and Embedded Systems (CHES) 2001*, Ç. K. Koç, D. Naccache, and C. Paar, Eds. 2001, number 2162 in LNCS, Springer-Verlag.

[11] V. Rijmen, "Efficient implementation of the Rijndael S-box", http://www.esat.kuleuven.ac.be/rijmen/rijndael/sbox.pdf.

[12] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, MIT Press, Cambridge, MA, 2 edition, 1972.

[13] K.-L. Cheng, C.-M. Hsueh, J.-R. Huang, J.-C. Yeh, C.-T. Huang, and C.-W. Wu, "Automatic generation of memory built-in self-test cores for system-on-chip", in *Proc. Tenth IEEE Asian Test Symp. (ATS)*, Kyoto, Nov. 2001, pp. 91–96.