A Compact Rijndael Hardware Architecture with S-Box Optimization

Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh

IBM Research, Tokyo Research Laboratory, IBM Japan Ltd., 1623-14,
Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan
{akashi,e02716,chano,munetoh}@jp.ibm.com

Abstract. Compact and high-speed hardware architectures and logic optimization methods for the AES algorithm Rijndael are described. Encryption and decryption data paths are combined and all arithmetic components are reused. By introducing a new composite field, the S-Box structure is also optimized. An extremely small size of 5.4 Kgates is obtained for a 128-bit key Rijndael circuit using a 0.11-μm CMOS standard cell library. It requires only 0.052 mm^2 of area to support both encryption and decryption with 311 Mbps throughput. By making effective use of the SPN parallel feature, the throughput can be boosted up to 2.6 Gbps for a high-speed implementation whose size is 21.3 Kgates.

1 Introduction

DES (Data Encryption Standard) [14,1], which is a common-key block cipher for US federal information processing standards, has also been used as a de facto standard for more than 20 years. NIST (National Institute of Standard Technology) has selected Rijndael [2] as the new Advanced Encryption Standard (AES) [13]. Many hardware architectures for Rijndael were proposed and their performances were evaluated by using ASIC libraries [8,18,10,9] and FPGAs [3, 17,6,11,5]. However, they are simple implementations according to the Rijndael specification, and none are yet small enough for practical use. The AES has to be embeddable not only in high-end servers but also in low-end consumer products such as mobile terminals. Therefore, sharing and reusing hardware resources, and compressing the gate logic are indispensable to produce a small Rijndael circuit.

The SPN structure of Rijndael is suitable for highly parallel processing, but it usually requires more hardware resources compared with the Feistel structure used in many other ciphers developed after DES. This is because, all data is encoded in each round of Rijndael processing, while only half of data is processed at once in DES. In addition, Rijndael has two separate data paths for encryption and decryption.

In this paper, we describe a compact data path architecture for Rijndael, where the hardware resources are efficiently shared between encryption and decryption. The key arithmetic component S-Box has been implemented using

C. Boyd (Ed.): ASIACRYPT 2001, LNCS 2248, pp. 239–254, 2001.

look-up table logic or ROMs in the previous approaches, which requires a lot of hardware support. Reference [16] proposed the use of composite field arithmetic to reduce the computation cost of the S-Box, but no detailed hardware implementation was provided. Therefore, we propose a methodology to optimize the S-Box by introducing a new composite field, and show its advantages in comparison to the previous work.

2 Rijndael Algorithm

Fig. 1 shows a Rijndael encryption process for 128-bit plain text data string and a 128-bit secret key, with the number of rounds set to 10. These numbers are used throughout this paper, including for our hardware implementation. Each round and the initial stage requires a 128-bit round key, and thus 11 sets of round keys are generated from the secret key. The input data is arranged as a 4×4 matrix of bytes. The primitive functions SubBytes, ShiftRows and MixColumns are based on byte-oriented arithmetic, and AddRoundKey is a simple 128-bitwise XOR operation.

SubBytes is a nonlinear transformation that uses 16 byte substitution tables (S-Boxes). An S-Box is the multiplicative inverse of a Galois field $GF(2^8)$ followed by an affine transformation. In the decryption process, the affine transformation is executed prior to the inversion. The irreducible polynomial used by a Rijndael S-Box is

$$m(x) = x^8 + x^4 + x^3 + x + 1. \tag{1}$$

ShiftRows is a cyclic shift operation of the last three rows by different offsets. MixColumns treats the 4-byte data in each column as coefficients of a 4-term polynomial, and multiplies the data modulo $x^4 + 1$ with the fixed polynomial given by

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}. \tag{2}$$

In the decryption process, InvMixColumns multiplies each column with the polynomial

$$c^{-1}(x) = \{\texttt{0B}\}x^3 + \{\texttt{0D}\}x^2 + \{09\}x + \{\texttt{0E}\} \tag{3}$$

and InvShiftRows shifts the last three rows in the opposite direction from ShiftRows.

The key expander in Fig. 1 generates 11 sets of 128-bit round keys from one 128-bit secret key by using a 4-byte S-Box. These round keys can be prepared on the fly in parallel with the encryption process. In the decryption process, these sets of keys are used in reverse order. Therefore, all keys have to be generated and stored in registers in advance, or the final round key in the encryption process has to be pre-calculated for on-the-fly key scheduling. Because the first method requires the equivalent of a 1,408-bit register (128 bits $\times$ 11), and is not suitable

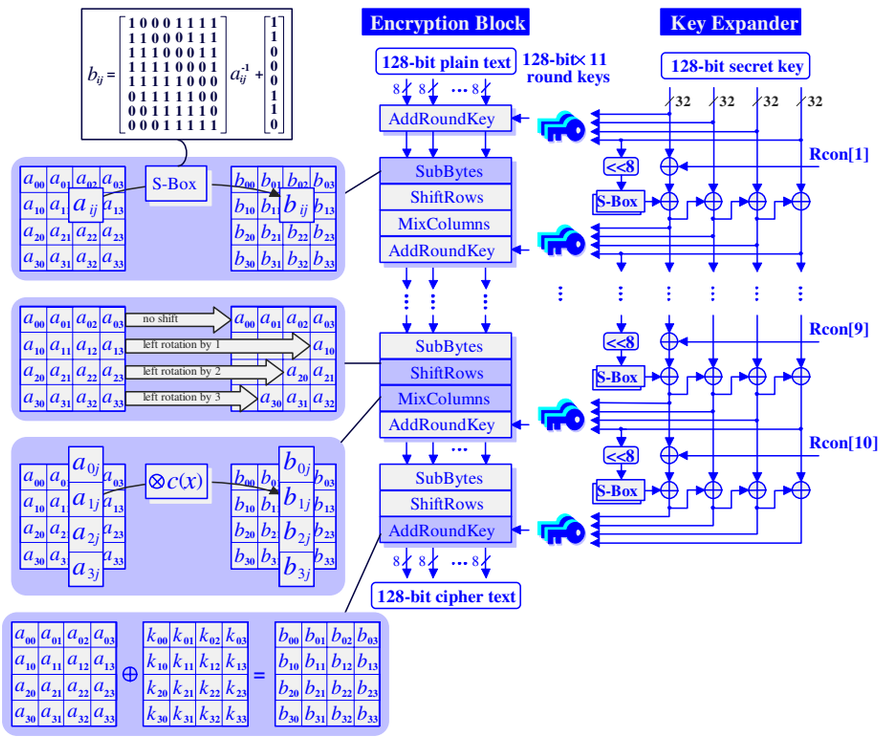

Fig. 1. Encryption process of Rijndael algorithm

for compact hardware, the second approach was chosen for the implementation described in the next section. Rcon[i] in Fig. 1 is a 4-byte value, and the lower 3 bytes are 0 for all i, and the highest byte is the bit representation of the polynomial x^i mod $m(x)$.

3 Data Path Architecture

3.1 Data Path Sharing between Encryption and Decryption

In order to minimize the size of our Rijndael hardware, resource sharing in the data path is fully employed as shown in Fig. 2. This circuit can execute both encryption and decryption. The 128-bit data (4×4 bytes) block is divided into four 32-bit columns, and is processed column by column through the 32-bit data bus. Therefore one round takes 4 clock cycles. It is not a good idea to make the bus width smaller than 32 bits, because the MixColumns operation needs 32-bits of data at one time. A smaller bus requires more registers and selectors, and resource sharing is hindered, resulting in an inefficient implementation.

The "Enc/Dec block" has 16-byte data registers, and they execute ShiftRows (or InvShiftRows) operations by themselves. Each 4-byte column is transformed by four parallel S-Boxes as SubBytes (or InvSubBytes). The order of ShiftRows and SubBytes is different from that in Fig. 1, though this does not affect the operations' results.

Selectors change the circuit state between encryption and decryption. The data path

$$\delta^{-1} \rightarrow x^{-1} \rightarrow \delta^{-1} \text{ and affine} \rightarrow \text{MixColumns}$$

is selected for encryption, and the path

$$\text{affine}^{-1} \text{ and } \delta^{-1} \rightarrow x^{-1} \rightarrow \delta^{-1} \rightarrow \text{InvMixColumns}$$

is used for decryption. δ^{-1} and δ are isomorphism functions for field conversions. Details are described in Section 4.

By moving InvMixColumns from the front of each S-Box to the back, MixColumns and InvMixColumns can be merged and some selectors are eliminated. As a result, the circuit size and the critical path length are reduced. An additional InvMixColumns is required in the key expander, but the area impact is minor.

3.2 S-Box Sharing with Key Expander

The key expander reuses the S-Boxes in the encryption/decryption block to generate a 128-bit key in each round. The S-Boxes are used once by the key expander, and four times by the encryption/decryption block, for a total of five times in every round. While the key expander uses the S-Boxes, the ShiftRows (or InvShiftRows) operation is executed simultaneously. As shown in Fig. 1, only the AddRoundKey operation is executed in the initial round, and the MixColumns (or InvMixColumns for decryption) is omitted in the final round. This operation switching is carried out by controlling the 4:1 selector at the bottom of Fig. 2. The first round key used in AddRoundKey is the initial key data stored in the key registers, and a transformation with the S-Boxes is not necessary. Therefore the first round takes four cycles, and the entire encryption process takes 54 ($= 4 + 5 \times 10$) cycles. The decryption process also takes 54 cycles. When a new secret key is provided, the key expander takes 10 cycles to generate the initial decryption key, which is the final round key in the encryption.

As described in Section 2, Rcon[i] is a 4-byte constant value, and the highest order byte is generated by modular multiplication on $GF(2^8)$. The circuit RC in Fig. 3 generates the constant values sequentially during the encryption process, starting from {01}, and RC^{-1} calculates the same values in reverse order from {36}. These circuits are also merged as shown in this figure.

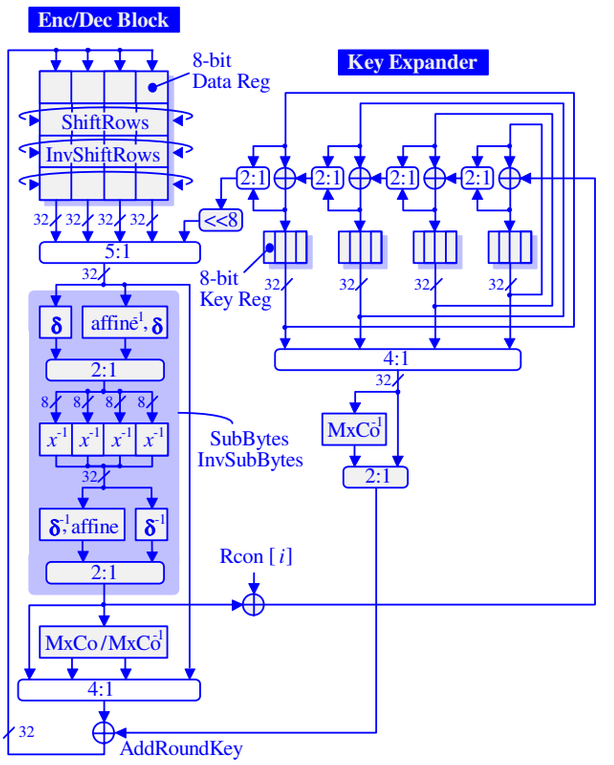

Fig. 2. Data path architecture

3.3 Factoring in MixColumns and InvMixColumns

MixColumns and InvMixColumns are modular multiplications with constant polynomials (2) and (3) that can be written as the constant matrix multiplications shown in Equations (4) and (5) respectively.

$$\begin{pmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

$$= \begin{pmatrix} 02\ 02\ 00\ 00 \\ 00\ 02\ 02\ 00 \\ 00\ 00\ 02\ 02 \\ 02\ 00\ 00\ 02 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 00\ 01\ 01\ 01 \\ 01\ 00\ 01\ 01 \\ 01\ 01\ 00\ 01 \\ 01\ 01\ 01\ 00 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \tag{4}$$

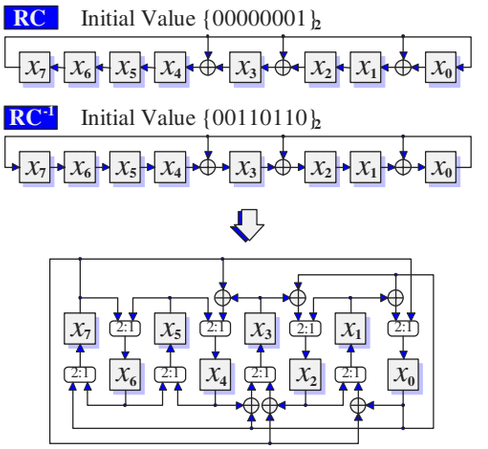

Fig. 3. Rcon[i] generator

$$\begin{aligned}\begin{pmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} &= \begin{pmatrix} \mathtt{0E\ 0B\ 0D\ 09} \\ \mathtt{09\ 0E\ 0B\ 0D} \\ \mathtt{0D\ 09\ 0E\ 0B} \\ \mathtt{0B\ 0D\ 09\ 0E} \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \\ &= \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \\ &+ \begin{pmatrix} 08\ 08\ 08\ 08 \\ 08\ 08\ 08\ 08 \\ 08\ 08\ 08\ 08 \\ 08\ 08\ 08\ 08 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 04\ 00\ 04\ 00 \\ 00\ 04\ 00\ 04 \\ 04\ 00\ 04\ 00 \\ 00\ 04\ 00\ 04 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \end{aligned} \tag{5}$$

$$\begin{cases} b_3 &= 02X_3 + X_1 + a_2 \\ b_2 &= 02X_2 + X_1 + a_3 \\ b_1 &= 02X_1 + X_3 + a_0 \\ b_0 &= 02X_0 + X_3 + a_1 \end{cases} \quad \begin{cases} X_3 &= a_3 + a_2 \\ X_2 &= a_2 + a_1 \\ X_1 &= a_1 + a_0 \\ X_0 &= a_0 + a_3 \end{cases} \tag{6}$$

$$\begin{cases} c_3 &= b_3 + Z_1 \\ c_2 &= b_2 + Z_0 \\ c_1 &= b_1 + Z_1 \\ c_0 &= b_0 + Z_0 \end{cases} \quad \begin{cases} Z_1 &= Y_2 + Y_1 \\ Z_0 &= Y_2 + Y_0 \end{cases} \quad \begin{cases} Y_2 &= 02(Y_1 + Y_0) \\ Y_1 &= 04(a_3 + a_1) \\ Y_0 &= 04(a_2 + a_0) \end{cases} \tag{7}$$

As seen in Equation (5), InvMixColumns contains a complete MixColumns matrix. Therefore we merged these two functions into one circuit as shown in Fig. 4. In addition, both functions can be broken into regular matrices whose non-zero elements are only one of the values $\{08, 04, 02, 01\}$. Therefore the number of common terms can be greatly reduced by factoring, finally resulting in Equations (6) and (7). The result, shown in Table 1, is that the XOR logic gates are decreased by 2/3 (from 592 XORs to 195 XORs) with only 2 XOR gates of additional delay.

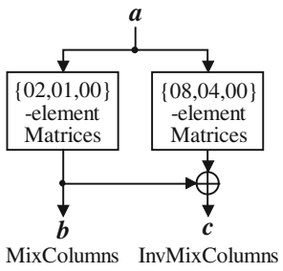

Fig. 4. MixColumns/InvMixColumns circuit

Table 1. Factoring effects of MixColumns and InvMixColumns

	Original Matrices			Our Implementation
	MixColumns	InvMixColumns	Total	
Number of XOR	152	440	592	195
Delay (gates)	3	5	5	7

4 S-Box Optimization

4.1 Structure of New S-Box

Designing a compact S-Box is one of the most critical problems for reducing the total circuit size of the Rijndael hardware. It is possible to implement the S-Box as a practical circuit based on its functional specification by using automatic logic synthesis tools, because the size of the S-Box function table is small; 256 entries $\times$ 1 byte. However, a significant reduction in the size of the S-Box was achieved in [16], by using composite field arithmetic [7]. In the following, we propose further optimization of S-Box by introducing a new composite field.

Fig. 5 shows the outline of our S-Box implementation. The most costly operation in the S-Box is the multiplicative inversion over a field A, where A is an extension field over $GF(2)$ with the irreducible polynomial $m(x)$. To reduce the cost of this operation, we adopted the following 3-stage method.

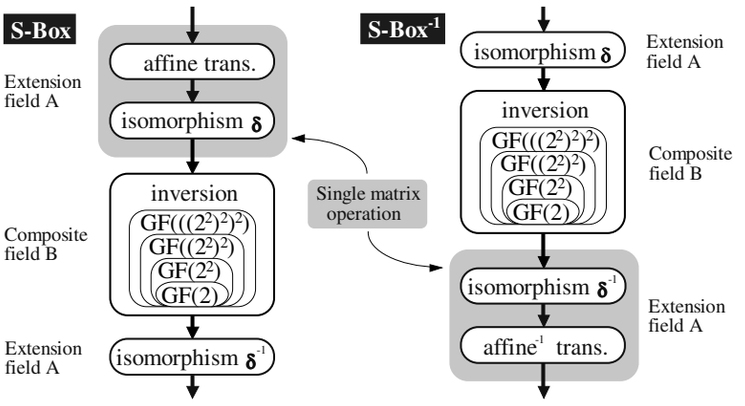

Fig. 5. The computation sequence of our S-Box implementation

(Stage 1) Map all elements of the field A to a composite field B, using an isomorphism function δ.
(Stage 2) Compute the multiplicative inverses over the field B.
(Stage 3) Re-map the computation results to A, using the function δ^{-1}.

Even though isomorphism functions are required in this method, the cost of those functions can mostly be hidden by merging them with the affine transformations.

4.2 Multiplicative Inversion over A New Composite Field

The composite field B in Stage 2 is constructed not by applying a single degree-8 extension to $GF(2)$, but by applying multiple extensions of smaller degrees. To reduce the cost of Stage 2 as much as possible, we built the composite field B by repeating degree-2 extensions under a polynomial basis using these irreducible polynomials:

$$\begin{cases} GF(2^2) & : x^2 + x + 1 \\ GF((2^2)^2) & : x^2 + x + \phi \\ GF(((2^2)^2)^2) & : x^2 + x + \lambda \end{cases} \tag{8}$$

where $\phi = \{10\}_2$, $\lambda = \{1100\}_2$. The inverter over the field above has fewer $GF(2)$ operators compared with the composite field used in [16]

$$\begin{cases} GF(2^4) & : x^2 + x + 1 \\ GF((2^4)^2) & : x^2 + x + \omega_{14} \end{cases} \tag{9}$$

where $\omega_{14} = \{1001\}_2$.

Our hardware implementation of Stage 2 is shown in Fig. 6. For any composite fields $GF((2^m)^n)$ which are constructed using a degree-n extension after a degree-m extension, computing the multiplicative inverses can be done as a combination of operations over the subfields $GF(2^n)$, using the equation described in [7,4]

$$P^{-1} = (P^r)^{-1} \cdot P^{r-1}, \text{ where } r = (2^{nm} - 1)/(2^m - 1). \tag{10}$$

In our case ($n = 2$, $m = 4$), so Equation (10) becomes

$$P^{-1} = (P^{17})^{-1} \cdot P^{16}. \tag{11}$$

The circuit in Fig. 6 is an implementation of Equation (11), with additional optimizations. In the circuit, P^{16} is computed first (note that the hardware costs for computing 2-powers over Galois fields are very small) and then P^{17} is obtained by multiplying P by P^{16} over $GF(((2^2)^2)^2)$. This operation requires only two multiplications, one addition and one constant multiplication over $GF((2^2)^2)$. Because P^{17} is always an element of $GF((2^2)^2)$ according to Fermat's Little Theorem (i.e., the upper 4 bits of P^{17} are always 0), computing the upper 4 bits of P^{17} is unnecessary [7]. $(P^{17})^{-1}$ is computed recursively over $GF((2^2)^2)$, then multiplied by P^{16} over $GF(((2^2)^2)^2)$, and finally P^{-1} is obtained. This multiplication requires fewer circuit resources than usual, because P^{17} is an element of $GF((2^2)^2)$. Note that our multipliers and inverter over subfield $GF((2^2)^2)$ are also small [15]. Further gate reduction is possible by sharing parts of the three $GF((2^2)^2)$ multipliers in Fig. 6, where common inputs are used.

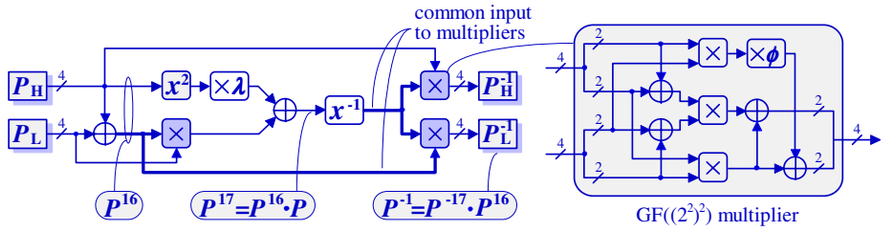

Fig. 6. Our implementation of an inverter over a composite field $GF(((2^2)^2)^2)$.

4.3 Generating Isomorphism Functions

The isomorphism functions δ and δ^{-1} are located at the both ends of the S-Boxes, and one of them is merged with an affine transformation. On the other hand, Reference [16] proposes locating these isomorphism functions at the circuit's primary input and output, and thus it cannot be merged with an affine transformation. The function is also required between the AddRoundKey and the key expander in Fig. 1. Therefore our approach is much more suitable for a reduced hardware implementation. The isomorphism functions δ and δ^{-1} in Stages 1 and 3 were constructed as follows. First, search for a generator element α in A and a generator β in B, where both α and β are roots of the same primitive irreducible polynomial. Any primitive polynomial can be applied, and here we use

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1. \tag{12}$$

Once such elements are found, the definition table of the isomorphism function δ (or δ^{-1}) is immediately determined, where α^k is mapped to β^k (or β^k to α^k) for any $1 \leq k \leq 254$. The hardware implementation of these functions can be obtained by mapping only the basis elements of A (or B) into B (or A), and these mappings are described as multiplications of constant matrixes over $GF(2)$. The functions δ and δ^{-1} are as follows:

$$\delta = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}_2 \quad \delta^{-1} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}_2 \tag{13}$$

where the least significant bits are in the upper left corners.

All of these isomorphism functions and the constant multipliers in the S-Boxes are implemented as XOR arrays, and their Boolean logic is compressed by applying a factoring technique based on a greedy algorithm [12].

4.4 Implementation Results of the S-Box

Table 2 shows the performance of our multiplicative inverter and S-Box described above in comparison with that using Equation (9). The S-Box implementations are also compared with the one automatically generated by a synthesis tool from a look-up table. A 0.11-μm CMOS standard cell library (one gate is equivalent to a 2-way NAND) is used here, and the delay time is evaluated under the worst-case conditions. The hardware size of our S-Box using the field $GF(((2^2)^2)^2)$ is 294 gates, which is about 20% smaller and slightly faster than the one using the field $GF((2^4)^2)$.

Table 2. S-Box features of proposed method. (gate = 2-way NAND)

Method	Inverter		S-Box		S-Box^{-1}	
	Area (gates)	Delay (ns)	Area (gates)	Delay (ns)	Area (gates)	Delay (ns)
Ours, Equation (8)	173	2.55	294	3.69	← merged	
Equation (9)	241	2.50	362	3.75	← merged	
Look-up Table	-	-	696	2.71	700	2.29

Our S-Box consists of affine transformations, isomorphism functions, inverters and selectors, and can be applied to both encryption and decryption. On the other hand, the look-up table method requires two different circuits, an S-Box for encryption and an S-Box^{-1} for decryption. The S-Box tables appear as random numbers to CAD tools, and therefore logic compression is very hard. As a result, a large amount of hardware, 1,396 (= 696 + 700) gates, is required for each one-byte S-Box based on the look-up table method, while our method is less than 1/4 of that size.

By applying our new composite field, merging the isomorphism functions with affine transformations, using a factoring technique, and combining the encryption and decryption paths, a very small S-Box was produced.

5 Performance Comparison in ASICs

The architecture described in Section 3 has been implemented by using 0.11-μm CMOS technology, and the extremely small size of 5.4 Kgates was obtained with a 7.62-ns cycle time (131.24 MHz) under the worst-case conditions. The gate size of each component and the critical path delay are detailed in Table 3 and Fig. 7, respectively. The function SubBytes (S-Box) occupies about 22% of the circuit area, and accounts for almost half of the delay time. The second major component is neither MixColumns nor AddRoundKey, but the selectors. The requirement to use selectors is not obvious from the Rijndael algorithm specification, where they appear as conditional branches and data selections. However, they require 1,099 (= 699 + 400) gates (20.36% of the circuit), because of the wide data width. In order to drive those selectors, drivers with high fan out are also required. Therefore, we carefully analyzed the critical data path and optimized the order of data selection, and adjusted the driver size. As a result, the delay time of the selector and driver section was reduced from 3.46 ns down to 1.95 ns, without changing the total gate count.

Using our proposed architecture, we designed and synthesized five implementations as shown in Fig. 8. Higher throughputs with higher parallelism were achieved by increasing the number of S-Boxes and the bus width. Four S-Boxes are shared between the data encryption block and the key expander in the 5- and

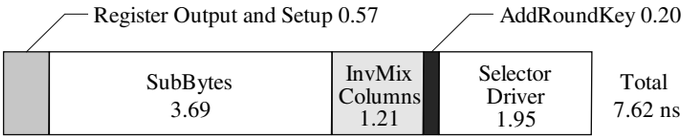

Fig. 7. Critical path delay

Table 3. Factoring effects of MixColumns and InvMixColumns

Components	Gates	%
Encryption/Decryption Block	(3,305)	(61.23)
Data Register	864	16.01
ShiftRows	160	2.96
SubBytes	1,176	21.79
MixColumns/InvMixColumns	350	6.48
AddRoundKey	56	1.04
Selector	699	12.95
Key Expander	(1,896)	(35.12)
Key Register	864	16.01
InvMixColumns	294	5.45
RC/RC^{-1}	100	1.85
XOR	238	4.41
Selector	400	7.41
Controller, Selector, Driver	197	3.65
Total	5,398	100.00

3-cycle/round versions. In the other three implementations, the key expanders have their own S-Boxes. Two circuits were synthesized from each implementation (a total of ten implementations), one optimized for size and the other for speed. The sizes and speeds are also shown in Table 4, in comparison with other ASIC implementations [8,18,10,9] under the worst-case conditions. Data and the key sizes are both 128 bits in all implementations, except that of [10], where 128-bit data and a 256-bit key (14 rounds) are used. A gate wireability of 80% is assumed to calculate the silicon area of our implementations. Reference [16] shows a throughput of 7.5 Gbps with 32 parallel cores, with a circuit size for encryption of 256 Kgates. However, this number was not evaluated by any synthesis tool, so we did not include it in the table.

It is obvious that in our implementation that more hardware resources yield higher throughput. For instance, the number of operation clock cycles can be reduced by increasing the size of the S-Box, which allows more parallel computation. Increases in fan out can also be used to increase the speed. In order to clarify the total efficiency of each implementation, we show the throughput per gate on the right side of Table 4. In general, it is not easy to compare the

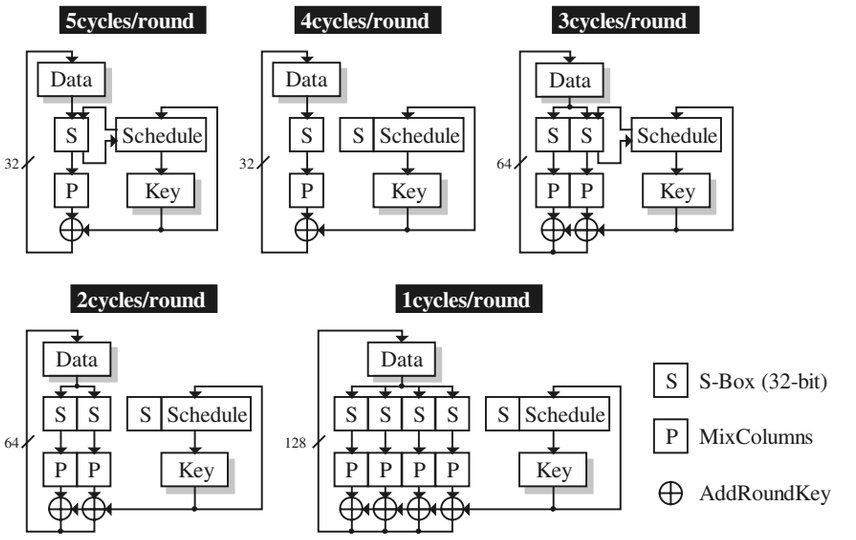

Fig. 8. Data Path Architectures of each implementation

implementations using different CAD tools and different technology libraries. However, even considering this difficulty of precise comparison, our hardware architecture is by far the best. Our smallest implementation is less than 1/6 of the 33.8 Kgates in the best previous approach [9]. Our 1-cycle/round version sets a new record for throughput at 2.6Gbps, not only in ECB mode but also in CBC cipher feedback mode. Reference [8] shows the second best throughput as 1.95 Gbps, but it uses 113.5 times as many gates, because all 11 rounds are unrolled. Throughout these comparisons of ASIC implementations, our hardware architecture has advantages in both size and speed.

6 Conclusion

In this paper, a compact yet high-speed architecture for Rijndael was proposed and evaluated through ASIC implementations. In order to minimize the hardware size, the order of the arithmetic functions was changed, and encryption and decryption data paths were efficiently combined. Logic optimization techniques such as factoring were applied to the arithmetic components, and gate counts were greatly reduced.

Our architecture provides high flexibility from a compact 32-bit bus implementation to a high-speed implementation using a 128-bit bus. The S-Box has been implemented as look-up table logic in the previous work, and has required extensive hardware resources. We introduced a new composite field $GF(((2^2)^2)^2)$ and proposed an optimization method for the S-Box. Our S-Box requires less

Table 4. Performance comparison in ASIC implementations. (worst case)

Cycles /Round	S-Box (bytes)	Area (gates)	Area (mm^2)	Max. Freq. (MHz)	Through -put (Mbps)	Throughput /Area (Kbps/gate)	Notes
Ours 0.11μm							
5	4	5,398	0.052	131.24	311.09	57.63	Total 54 cycles
		10,338	0.099	222.22	526.74	50.95	
4	8	6,292	0.060	137.55	400.15	63.60	Total 44 cycles
		10,990	0.106	219.30	637.96	58.05	
3	8	7,998	0.077	137.17	548.68	68.60	Total 32 cycles
		14,777	0.142	218.82	875.28	59.23	
2	12	8,836	0.085	137.17	798.08	90.32	Total 22 cycles
		17,016	0.163	217.86	1,267.55	74.49	
1	20	12,454	0.130	145.35	1,691.35	135.81	Total 11 cycles
		21,337	0.205	224.22	2,609.11	122.28	
[10] 0.18μm							
1	48	184,000	4.23	48	435 (256-bit key)	2.51	256-bit data and key supported Decryption not supported
[9] 0.35 μm							
1	40	33,850	-	-	509.70	15.06	
[8] 0.35 μm							
1/11	400	612,834	-	15.23	1,950.03	3.18	11 round unrolled
[18] 0.5 μm							
1	40	68,872	20.74	21.18	271.13	3.94	4 transistors/gate is assumed
1	40	160,421	33.85	47.36	605.77	3.78	

than 1/4 the size of one using a look-up table, and also showed 20% better performance in comparison with the one using a $GF((2^4)^2)$ field.

Our smallest implementation using 0.11-μm CMOS technology is 5.4 Kgates, which is less than 1/6 the size of the best hardware of previous work. Making the best use of the parallel processing allowed by Rijndael, a high-speed version obtained the best performance of 2.6 Gbps with 21.3 Kgates. Thus, our Rijndael hardware can be applied to various targets from mobile equipment to high-end security servers.

Our continuing research is to develop and evaluate even faster hardware for 10 Gbps class high-speed communication links and beyond.

Acknowledgements. We are grateful to Mr. G. Zang for supporting us generously with his CAD tool expertise, and also would like to thank Mr. A. Rudra and Dr. N. Ohba for their helpful comments on this work.

References

[1] ANSI (American National Standards Institute). *Triple Data Encryption Algorithm Modes of Operation*, 1998.

[2] J. Daemen and V. Rijmen. AES Proposal: Rijndael. NIST AES Proposal, June 1998. Available at http://csrc.nist.gov/encryption/aes /rijndael/Rijndael.pdf.

[3] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists. In *The Third Advanced Encryption Standard Candidate Conference*, pages 13–27. NIST, April 2000. Available at http://csrc.nist.gov/encryption/aes/round2/conf3/papers/08-aelbirt.pdf.

[4] J.L. Fan and C. Paar. On Efficient Inversion in Tower Fields of Characteristic Two. In *International Symposium on Information Theory*, page 20. IEEE, June 1997.

[5] V. Fischer and M. Drutarovsky. Two Methods of Rijndael Implementation in Reconfigurable Hardware. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES2001)*, pages 81–96, May 2001.

[6] K. Gaj and P. Chodowiec. Comparison of the Hardware Prformance of the AES Candidates using Reconfigurable Hardware. In *The Third Advanced Encryption Standard Candidate Conference*, pages 40–56. NIST, April 2000. Available at http://csrc.nist.gov/encryption/aes/round2/conf3/papers/22-kgaj.pdf.

[7] J. Guajardo and C. Paar. Efficient Algorithms for Elliptic Curve Cryptosystems. In Jr. Burton S. Kaliski, editor, *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 342–356. Springer-Verlag, August 1997.

[8] T. Ichikawa, T. Kasuya, and M. Matsui. Hardware Evaluation of the AES Finalists. In *The Third Advanced Encryption Standard Candidate Conference*, pages 279–285. NIST, April 2000. Available at http://csrc.nist.gov/encryption/aes/round2/conf3/papers/15-tichikawa.pdf.

[9] T. Ichikawa, T. Tokita, and M. Matsui. On Hardware Implementation of 128-bit Block Ciphers (III). In *2001 Symposium on Cryptography and Information Security (SCIS 2001)*, pages 669–674, January 2001. (Japanese).

[10] H. Kuo and I. Verbauwhede. Architectural Optimization for a 1.82 Gbits/sec VLSI Implementation of the AES Rijndael Algorithm. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES2001)*, pages 53–67, May 2001.

[11] M. McLoone and J.V. McCanny. High performance Single-chip FPGA Rijndael Algorithm Implementations. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES2001)*, pages 68–80, May 2001.

[12] S. Morioka and Y. Katayama. Design Methodology for a One-Shot Reed-Solomon Encoder and Decoder. In *International Conference on Computer Design (ICCD '99)*, pages 60–67. IEEE, October 1999.

[13] National Institute of Standards and Technology (U.S.). Advanced Encryption Standard (AES). Available at http://csrc.nist.gov/publications/drafts/dfips-AES.pdf.

[14] National Institute of Standards and Technology (U.S.). Data Encryption Standard (DES). FIPS Publication 46-3, NIST, 1999. Available at http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[15] C. Paar. A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields. *IEEE Transactions on Computers*, 45(7):856–861, July 1996.

[16] A. Rudra, P.K. Dubey, C.S. Jutla, V. Kumar, J.R. Rao, and P. Rohatgi. Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES2001)*, pages 175–188, May 2001.

[17] N. Weaver and J. Wawrzynek. A Comparison of the AES Candidates Amenability to FPGA Implementation. In *The Third Advanced Encryption Standard Candidate Conference*, pages 28–39. NIST, April 2000. Available at http://csrc.nist.gov/encryption/aes/round2/conf3/papers/13-nweaver.pdf.

[18] B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke. Hardware Performance Simulation of Round 2 Advanced Encryption Standard Algorithm. Available at http://csrc.nist.gov /encryption/aes/round2/NSA-AESfinalreport.pdf.