Very Compact Hardware Implementations of the Blockcipher CLEFIA

Toru Akishita and Harunaga Hiwatari

Sony Corporation
{Toru.Akishita,Harunaga.Hiwatari}@jp.sony.com

Abstract. The 128-bit blockcipher CLEFIA is known to be highly efficient in hardware implementations. This paper proposes very compact hardware implementations of CLEFIA-128. Our implementations are based on novel serialized architectures in the data processing block. Three types of hardware architectures are implemented and synthesized using a 0.13 μm standard cell library. In the smallest implementation, the area requirements are only 2,488 GE, which are about half of the previous smallest implementation as far as we know. Furthermore, only additional 116 GE enable to support decryption.

Key words: blockcipher, CLEFIA, compact hardware implementation, ASIC

1 Introduction

CLEFIA [9, 11] is a 128-bit blockcipher supporting key lengths of 128, 192 and 256 bits, which is compatible with AES [2]. CLEFIA achieves enough immunity against known attacks and flexibility for efficient implementation in both hardware and software. It is reported that CLEFIA is highly efficient particularly in hardware implementations [12, 10, 13].

Compact hardware implementations are very significant for small embedded devices such as RFID tags and wireless sensor nodes because of their limited hardware resources. As for AES with 128-bit keys, low-area hardware implementations have been reported in [3] and [4]. The former uses a RAM based architecture supporting both encryption and decryption with the area requirements of 3,400 GE, while the latter uses a shift-register based architecture supporting encryption only with the area requirements of 3,100 GE. Both implementations use an 8-bit serialized data path and implement only a fraction of the *MixColumns* operation with additional three 8-bit registers, where it takes several clock cycles to calculate one column. Very recently, another low-area hardware implementation of AES was proposed in [5] requiring 2,400 GE for encryption only. Unlike the previous two implementations, it implements *MixColumns* not in a serialized way, where one column of *MixColumns* is processed in 1 clock cycle. Thus it requires 4 times more XOR gates for *MixColumns*, but requires no additional register and can reduce gate requirements for control logic.

In this paper, we present very compact hardware architectures of CLEFIA with 128-bit keys based on 8-bit shift registers. We show that the data processing part of CLEFIA-128 can be implemented in a serialized way without any additional registers. Three types of hardware architectures are proposed according to required cycles for one block process by adaptively applying clock gating technique. Those architectures are implemented and synthesized using a 0.13 μm standard cell library. In our smallest implementation, the area requirements are only 2,488 GE, which are to the best of our knowledge about half as small as the previous smallest implementation, 4,950 GE [10, 12], and competitive to the smallest AES implementation. Furthermore, only additional 116 GE are required to support decryption by switching the processing order of F-functions at even-numbered rounds.

The rest of the paper is organized as follows. Sect. 2 gives brief description of CLEFIA and its previously proposed hardware implementations. In Sect. 3, we propose three types of hardware architectures. Sect. 4 describes additional hardware resources to support decryption. Sect. 5 gives evaluation results for our implementations, compared with the previous results of CLEFIA and AES. Finally, we conclude in Sect. 6.

2 128-bit Blockcipher CLEFIA

2.1 Algorithm

CLEFIA [9, 11] is a 128-bit blockcipher with its key length being 128, 192, and 256 bits. For brevity, we consider 128-bit key CLEFIA, denoted as CLEFIA-128, though similar techniques are applicable to CLEFIA with 192-bit and 256-bit keys. CLEFIA-128 is divided into two parts: the data processing part and the key scheduling part.

The data processing part employs a 4-branch Type-2 generalized Feistel network [14] with two parallel F-functions F_0 and F_1 per round. The number of rounds r for CLEFIA-128 is 18. The encryption function ENC_r takes a 128-bit plaintext $P = P_0|P_1|P_2|P_3$, 32-bit whitening keys WK_i $(0 \leq i < 4)$, and 32-bit round keys RK_j $(0 \leq j < 2r)$ as inputs, and outputs a 128-bit ciphertext $C = C_0|C_1|C_2|C_3$ as shown in Fig. 1.

The two F-functions F_0 and F_1 consist of round key addition, 4 non-linear 8-bit S-boxes, and a diffusion matrix. The construction of F_0 and F_1 is shown in Fig. 2. Two kind of S-boxes S_0 and S_1 are employed, and the order of these S-boxes are different in F_0 and F_1. The diffusion matrices of F_0 and F_1 are also different; the matrices M_0 for F_0 and M_1 for F_1 are defined as

$$M_0 = \begin{pmatrix} \texttt{01} & \texttt{02} & \texttt{04} & \texttt{06} \\ \texttt{02} & \texttt{01} & \texttt{06} & \texttt{04} \\ \texttt{04} & \texttt{06} & \texttt{01} & \texttt{02} \\ \texttt{06} & \texttt{04} & \texttt{02} & \texttt{01} \end{pmatrix}, \quad M_1 = \begin{pmatrix} \texttt{01} & \texttt{08} & \texttt{02} & \texttt{0A} \\ \texttt{08} & \texttt{01} & \texttt{0A} & \texttt{02} \\ \texttt{02} & \texttt{0A} & \texttt{01} & \texttt{08} \\ \texttt{0A} & \texttt{02} & \texttt{08} & \texttt{01} \end{pmatrix}.$$

The multiplications between these matrices and vectors are performed in GF(2^8) defined by a primitive polynomial $z^8 + z^4 + z^3 + z^2 + 1$.

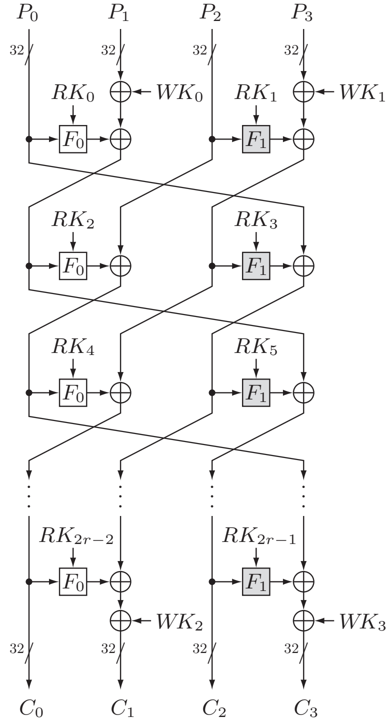


Fig. 1. Encryption function ENC_r

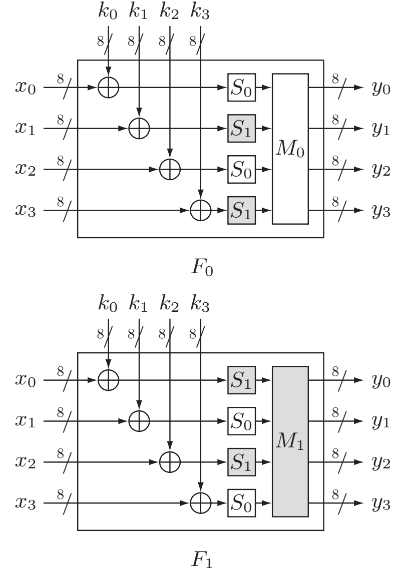


Fig. 2. F-functions F_0, F_1

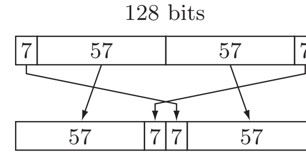


Fig. 3. *DoubleSwap* function Σ

The key scheduling part of CLEFIA-128 takes a secret key K as an input, and outputs 32-bit whitening keys WK_i $(0 \leq i < 4)$ and 32-bit round keys RK_j $(0 \leq j < 2r)$. It is divided into the following two steps: generating a 128-bit intermediate key L (step 1) and generating WK_i and RK_j from K and L (step 2). In step 1, the intermediate key L is generated by 12 rounds of encryption function which takes K as a plaintext and constant values CON_i $(0 \leq i < 24)$ as round keys. In step 2, the intermediate key L is updated by the $DoubleSwap$ function Σ, which is illustrated in Fig. 3. Round keys RK_j $(0 \leq j < 36)$ is generated by mixing K, L, and constant values CON_i $(24 \leq i < 60)$. Whitening keys WK_i is equivalent to 32-bit chunks K_i of K as $K = K_0|K_1|K_2|K_3$.

2.2 Previous Hardware Implementations

Hardware implementations of CLEFIA-128 have been studied in [12, 10, 13]. In [12], optimization techniques in data processing part including S-boxes and

diffusion matrices were proposed. The compact architecture, where F_0 is processed in one cycle and F_1 is processed in another cycle, was implemented, and its area requirements in area optimization are reported to be 4,950 GE.

In [10], two optimization techniques in key scheduling part were introduced. The first technique is related to implementation of the *DoubleSwap* function Σ. Σ is decomposed into the following *Swap* function Ω and *SubSwap* function Ψ as $\Sigma = \Psi \circ \Omega$.

$$\begin{aligned}
&\Omega : X \mapsto Y \\
&\quad Y = X[64\text{-}127] \mid X[0\text{-}63] \\
&\Psi : X \mapsto Y \\
&\quad Y = X[71\text{-}127] \mid X[57\text{-}70] \mid X[0\text{-}56]
\end{aligned}$$

$X[a\text{-}b]$ denotes a bit string cut from the a-th bit to the b-th bit of X. Please note that Ω and Ψ are both involutive. The 128-bit key register for the intermediate key L is updated by applying Ω and Ψ alternately. Round keys are always generated from the most significant 64-bit of the key register. After the final round of encryption, L is re-stored into the key register by applying the following *FinalSwap* function Φ.

$$\begin{aligned}
&\Phi : X \mapsto Y \\
&\quad Y = X[49\text{-}55] \mid X[42\text{-}48] \mid X[35\text{-}41] \mid X[28\text{-}34] \mid X[21\text{-}27] \mid X[14\text{-}20] \mid \\
&\qquad X[7\text{-}13] \mid X[0\text{-}6] \mid X[64\text{-}71] \mid X[56\text{-}63] \mid X[121\text{-}127] \mid X[114\text{-}120] \mid \\
&\qquad X[107\text{-}113] \mid X[100\text{-}106] \mid X[93\text{-}99] \mid X[86\text{-}92] \mid X[79\text{-}85] \mid X[72\text{-}78]
\end{aligned}$$

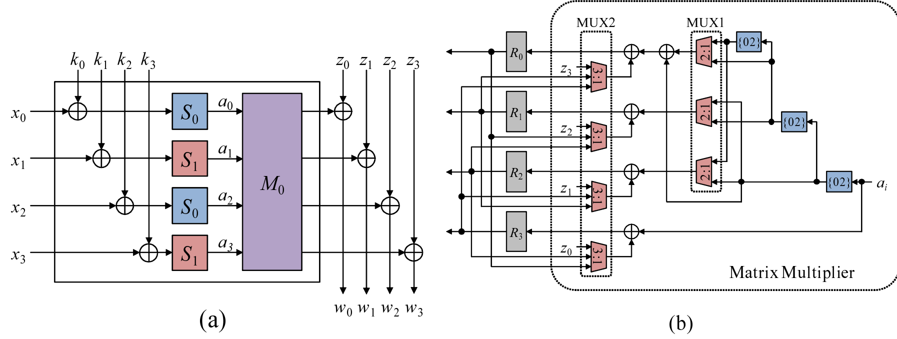
Please note that Φ is also involutive. In case of decryption, round keys are always generated from the most significant 64-bit of the key register by applying the inverse functions of Ω, Ψ and Φ in reverse order of encryption. Due to their involutive property, only three functions Ω, Ψ and Φ are required for encryption and decryption.

In the second technique, XOR operations with the parts of round keys related to a secret key K are moved by an equivalent transformation into the two data lines where key whitening operations are processed. Therefore, these XOR operations and key whitening operations can be shared.

In [13], five types of hardware architectures were designed and fairly compared to the ISO 18033-3 standard blockciphers under the same conditions. In their results, the highest efficiency of 400.96 Kbps/gates was achieved, which is at least 2.2 times higher than that of the ISO 18033-3 standard blockciphers.

3 Proposed Architectures

In this section we propose three types of hardware architectures. Firstly, we propose a compact matrix multiplier for CLEFIA-128. Next, in Type-I architecture, we propose a novel serialized architecture of the data processing block of CLEFIA-128. By adaptively applying clock gating logic to Type-I architecture,



l	1	2	3	4
R_0	$z_3 \oplus \{06\}a_0$	$z_2 \oplus \{04\}a_0 \oplus \{06\}a_1$	$z_1 \oplus \{02\}a_0 \oplus a_1 \oplus \{06\}a_2$	$z_0 \oplus a_0 \oplus \{02\}a_1 \oplus \{04\}a_2 \oplus \{06\}a_3$
R_1	$z_2 \oplus \{04\}a_0$	$z_3 \oplus \{06\}a_0 \oplus \{04\}a_1$	$z_0 \oplus a_0 \oplus \{02\}a_1 \oplus \{04\}a_2$	$z_1 \oplus \{02\}a_0 \oplus a_1 \oplus \{06\}a_2 \oplus \{04\}a_3$
R_2	$z_1 \oplus \{02\}a_0$	$z_0 \oplus a_0 \oplus \{02\}a_1$	$z_3 \oplus \{06\}a_0 \oplus \{04\}a_1 \oplus \{02\}a_2$	$z_2 \oplus \{04\}a_0 \oplus \{06\}a_1 \oplus a_2 \oplus \{02\}a_3$
R_3	$z_0 \oplus a_0$	$z_1 \oplus \{02\}a_0 \oplus a_1$	$z_2 \oplus \{04\}a_0 \oplus \{06\}a_1 \oplus a_2$	$z_3 \oplus \{06\}a_0 \oplus \{04\}a_1 \oplus \{02\}a_2 \oplus a_3$

(c)

Fig. 4. Matrix multiplier: (a) F-function F_0, (b) Data path, (c) Contents of registers R_j $(0 \le j < 4)$ at the l-th cycle

we can reduce the number of multiplexers (MUXes) in Type-II and Type-III architectures with increasing cycle counts.

Clock gating is a power-saving technique used in synchronous circuits. For hardware implementations of blockciphers, it was firstly introduced in [8] as a technique to reduce gate counts and power, and have been applied to KATAN family [1] and AES [5]. Clock gating works by taking the enable conditions attached to registers. It can remove feedback MUXes to hold their present state and replace them with clock gating logic. In case that several bits of registers take the same enable conditions, their gate counts will be saved by applying clock gating.

3.1 Matrix Multiplier

Among low-area AES implementations, *MixColumns* matrix operations are computed row by row in [3], while they are computed column by column in [4]. In our architecture, matrix operations are computed column by column in the following way.

The 4-byte output of M_0 operation is XORed with the next 4-byte data as shown in Fig. 4 (a). The matrix multiplier in Fig. 4 (b) performs the matrix multiplication together with the above XOR operation in 4 clock cycles. Fig. 4 (c) presents the contents of the registers R_i at the l-th cycle $(1 \le l \le 4)$. At the 1st cycle, the output a_0 of S_0 are fed to the multiplier and multiplied by {01}, {02}, {04}, and {06}. The products are XORed with the data z_i $(0 \le i < 4)$, and then the intermediate results are stored in the four registers R_j $(0 \le j < 4)$. As each

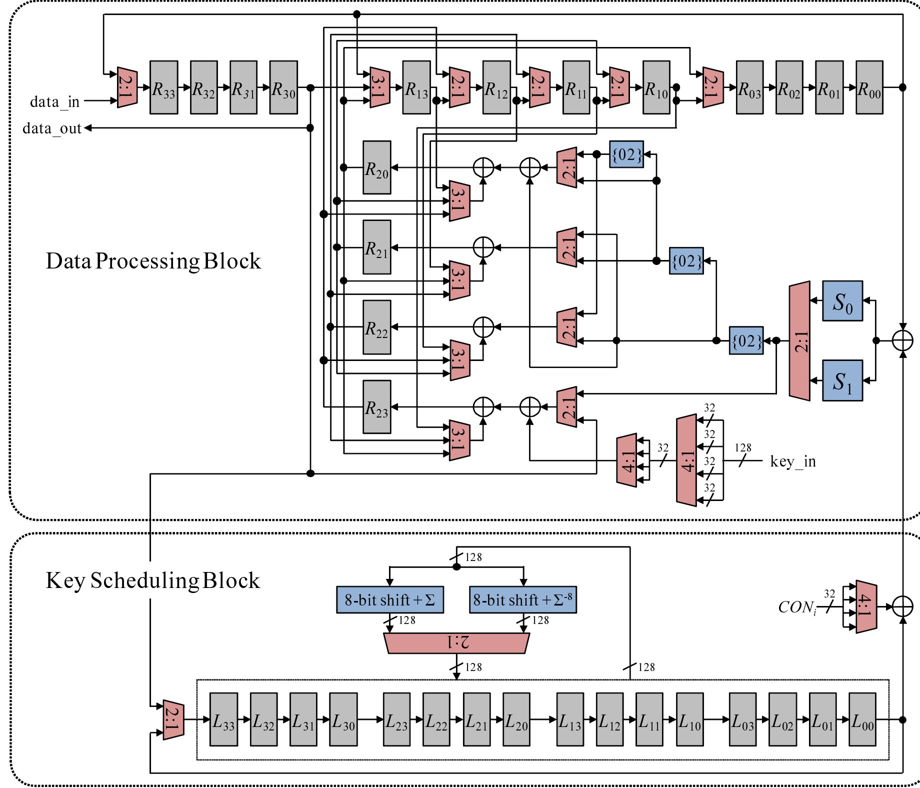


Fig. 5. Data path of Type-I architecture

column in M_0 consists of the same coefficients, the matrix multiplication can be performed by selecting the intermediate results through MUX2 and XORing the products of a_i $(i = 1, 2, 3)$ with them at the $(i+1)$-th cycle. After 4 clock cycles, w_i $(0 \leq i < 4)$ are stored in R_i. The multiplication by M_1 can be performed by switching MUX1.

In [4], three 8-bit registers are required for the construction of a parallel-to-serial converter due to avoiding register competition with the next calculation of a matrix. On the other hand, no competition occurs in our architecture because z_i is input at the 1st cycle of a matrix multiplication. w_i can be moved into the register where z_i for the newly processing F-function is stored.

3.2 Type-I Architecture

Fig. 5 shows the data path of Type-I architecture, where the width of data path is 8 bit except those written in the figure. It is divided into the following two blocks: the data processing block and the key scheduling block. Type-I architecture

processes a round of the encryption function in 8 clock cycles. We show, in appendix, the detailed data flow of the data registers R_{ij} $(0 \leq i, j < 4)$ in Fig. 5 for a round of the encryption processing. As described in Sect. 3.1, at the 1st and the 5th cycle in the 8 cycles, the data stored in R_{20}–R_{23} are moved into R_{03}–R_{12}, and simultaneously the data stored in R_{10}–R_{13} are input to the matrix multiplier. Therefore, no additional register but the 128-bit data register exists in the data processing block. Please note that R_{30}–R_{33} hold the current state at the 5–8th cycle by clock gating.

In the start of encryption, a 128-bit plaintext is located to R_{ij} in 16 clock cycles by inputting it byte by byte from data_in. After 18 rounds of the encryption function which require 144 cycles, a 128-bit ciphertext is output byte by byte from data_out in 16 clock cycles. Therefore, it takes 176 cycles for encryption. The reason why data_out is connected to R_{30} is that no word rotation is necessary at the final round of encryption. In the start of key setup, a 128-bit secret key K input from key_in is located to R_{ij} in 16 clock. After 12 rounds of the encryption function which require 96 cycles, a 128-bit intermediate key L is stored into the key registers L_{ij} $(0 \leq i, j < 4)$ by shifting R_{ij} and L_{ij} in 16 clock cycles. Therefore, it takes 128 cycles for key setup.

The two S-box circuits S_0 and S_1 are located in the data processing block, and one of those outputs is selected by a 2-to-1 MUX (8-bit width) and input to the matrix multiplier. The encryption processing of CLEFIA-128 is modified by a equivalent transformation as shown in Fig. 7 (a). The 32-bit XOR operation with 32-bit chunks K_i is reduced to the 8-bit XOR operation by locating it in the matrix multiplier. A 32-bit chunk K_i selected by a 32-bit 4-to-1 MUX is divided into four 8-bit data, and then one of the data is selected by a 8-bit 4-to-1 MUX and fed into the matrix multiplier one by one in 4 clock cycles.

In the key scheduling block, the intermediate key L stored in L_{ij} is cyclically shifted by one byte, and the 8-bit chunk in L_{00} is fed into the data processing after being XORed with the 8-bit chunk of CON_i. At the end of even-numbered rounds, L_{ij} is updated by (8-bit shift+Σ) operation; at the end of encryption, L_{ij} is updated by (8-bit shift + Σ^{-8}) operation in order to recover the intermediate key L. After re-storing the intermediate key L, L_{ij} hold it by clock gating until next start of encryption.

3.3 Type-II Architecture

In Type-II architecture, we aim the area optimization of the key scheduling block. Since *DoubleSwap* function Σ is decomposed as $\Sigma = \Psi \circ \Omega$, where Ψ and Ω are both involutive, as described in Sect. 2.2, Σ^{-8} satisfies the following

equations.

$$\begin{aligned}
\Sigma^{-8} &= (\Psi \circ \Omega)^{-8} \\
&= (\Omega \circ \Psi)^{8} \\
&= (\Omega \circ \Psi)^{8} \circ (\Omega \circ \Omega) \\
&= (\Omega \circ \Psi) \circ \cdots \circ (\Omega \circ \Psi) \circ (\Omega \circ \Omega) \\
&= \Omega \circ (\Psi \circ \Omega)^{8} \circ \Omega \\
&= \Omega \circ \Sigma^{8} \circ \Omega
\end{aligned}$$

Swap function Ω is realized by 8 iterations of cyclic shifting. Thus Σ^{-8} operation can be achieved by 8 iterations of cyclic shifting, 8 iterations of Σ operation, and 8 iterations of cyclic shifting again, which require 24 cycle counts.

During the encryption processing the intermediate key L is updated by Σ operation at the 17th cycle after 16 iterations of cyclic shifting every two rounds. At the 17th cycle, the data registers must hold the current data by clock gating. Accordingly, both 8 additional cycles for the encryption processing and 8 additional cycles to recover the intermediate key L after outputting a ciphertext are required, which results in 192 cycles for encryption. In compensation for the increase of 16 cycle counts, a 128-bit input of MUX in the key scheduling block can be removed.

3.4 Type-III Architecture

In Type-III architecture, we achieve the area optimization of the data processing block by applying clock gating effectively. Fig. 6 shows the data path of Type-III architecture. Instead of using MUXes, the data stored in R_{10}–R_{13} and those stored in R_{20}–R_{23} are swapped by cyclically shifting these registers in 4 clock cycles, while the other data register and the key registers hold the current state by clock gating. Simultaneously, the XOR operation with a 32-bit chunk K_i is done by XOR gates in the matrix multiplier, which leads the savings of 8 XOR gates. These data swaps are required twice for a round of the encryption processing. Therefore, it takes 16 cycles for a round of the encryption processing; in total 328 and 224 clock cycles are required for encryption and key setup, respectively. In compensation for the increase of many cycle counts, several 8-bit inputs of MUXes together with 8 XOR gates for secret key chunks can be removed.

4 Supporting Decryption

Any encryption-only implementation can support decryption by using the CTR mode. Nevertheless, if an implementation itself supports decryption, it can be used for more applications, for example, an application requiring the CBC mode. Accordingly, we consider the three types of hardware architectures supporting decryption.

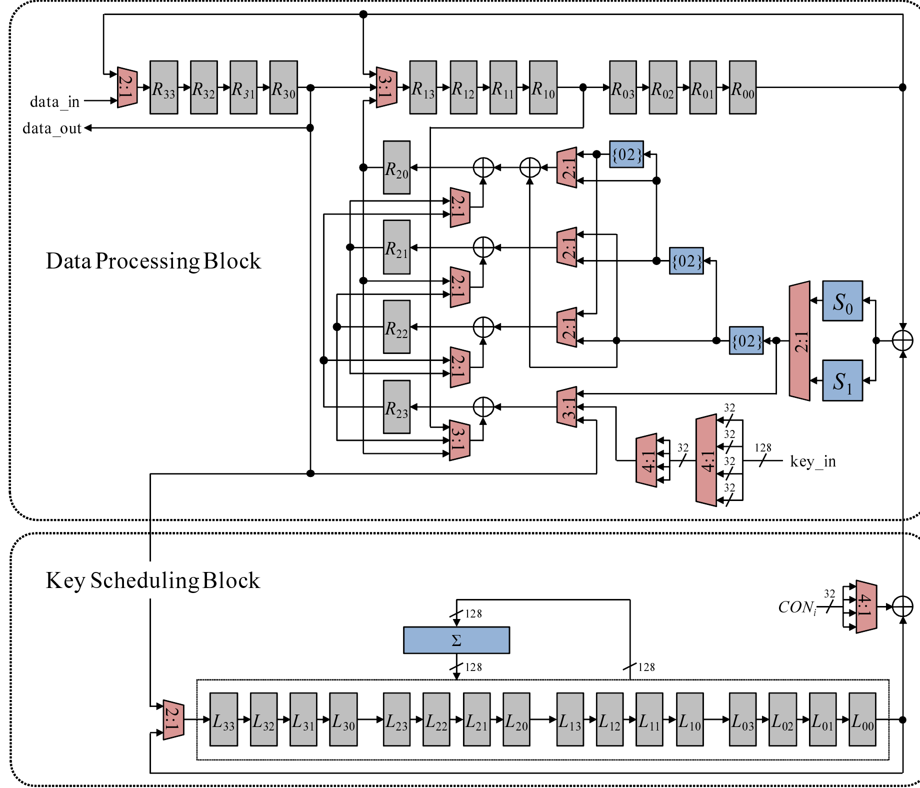


Fig. 6. Data path of Type-III architecture

Since the data processing part of CLEFIA employs a 4-branch Type-2 generalized Feistel network [14], the directions of word rotation are different between the encryption function and the decryption function. The encryption and decryption processing of CLEFIA-128 is shown in Fig. 7 (a) and (b), respectively. When the hardware architectures described in Sect. 3 support the decryption processing straightforwardly, many additional multiplexers are considered to be required due to these different directions of word rotation. For avoiding this, we switch the positions of F_0 and those of F_1 at even-numbered rounds as shown in Fig. 7 (c), and then the direction of word rotation becomes the same as the encryption processing shown in Fig. 7 (a). Thus we do not have to largely modify the data path of the above three architectures by processing F_1 ahead of F_0 at even-numbered rounds. However, as the order of round keys fed into the data processing block has been changed, the 8-bit round keys are fed from L_{10} when F_1 is processed at even-numbered rounds and from L_{30} when F_0 is processed at even-numbered rounds. Accordingly, a 8-bit 3-to-1 MUX is required for selecting the source registers of appropriate round keys including L_{00}. Since the leading

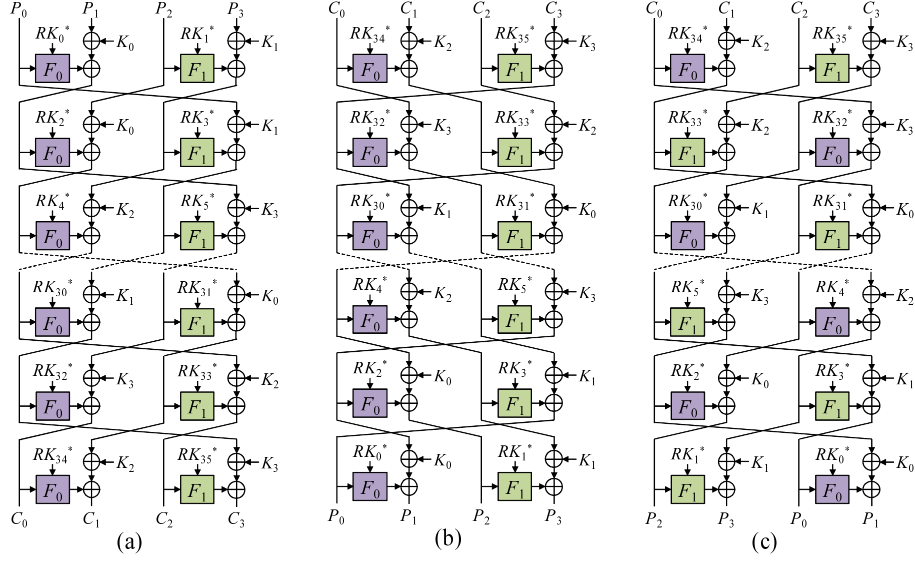


Fig. 7. (a) Encryption processing, (b) Decryption processing, (c) Modified decryption processing. XOR operations with the part of round keys related to secret key K are moved by an equivalent transformation, and thus RK_j^* $(0 \leq j < 36)$ denote the remaining part of round keys.

byte of a ciphertext is stored in R_{10}, not R_{30} for encryption, at the end of decryption because of the modified decryption processing, a 8-bit 2-to-1 MUX is required for selecting data_out.

5 Implementation Results

We designed and evaluated the three types of hardware architectures presented in Sect. 3 together with their versions supporting both encryption and decryption. The environment of our hardware design and evaluation is as follows:

Language	Verilog-HDL
Design library	0.13 μm CMOS ASIC library
Simulator	VCS version 2006.06
Logic synthesis	Design Compiler version 2007.03-SP3

One Gate Equivalent (GE) is equivalent to the area of a 2-way NAND with the lowest drive strength. For synthesis, we use a clock frequency of 100 KHz, which is widely used operating frequency for RFID applications.

Recently, scan flip-flops have been used in the low-area implementations of blockciphers instead of combinations of D flip-flops and 2-to-1 MUXes $[8, 1, 5]$ to reduce area requirements. In our evaluation, a D flip-flop and a 2-to-1 MUX cost

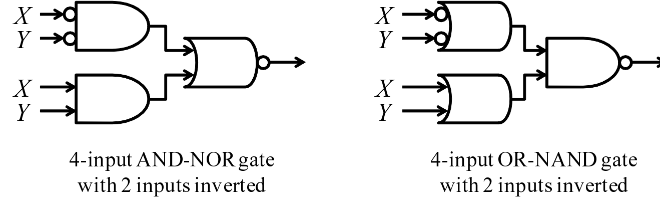


Fig. 8. 4-input AND-NOR and 4-input OR-NAND gate with 2 inputs inverted, which correspond to XOR and XNOR gate

4.5 and 2.0 GE, respectively, while a scan flip-flop costs 6.25 GE. Thus, we can save 0.25 GE per bit of storage. Moreover, the library we used has the 4-input AND-NOR and 4-input OR-NAND gates with 2 inputs inverted described in Fig. 8. The outputs of these cells are corresponding to those of XOR or XNOR gates when the inputs X, Y are set as shown in Fig. 8. Since these cells cost 2 GE instead of 2.25 GE required for XOR or XNOR cell, we can save 0.25 GE per XOR or XNOR gate. Clock gating logics are inserted into the design manually by instantiating Integrated Clock Gating (ICG) cells to gate the clocks of specific registers.

Table 1 shows the detailed implementations figures of the three types of hardware architectures presented in Sect. 3. CON generator and selector, ICG cells, and buffers are included in controller.

The area savings for the key scheduling block of Type-II/III implementation over Type-I implementation are 128 GE. In the library we used, a register with a 3-to-1 MUX costs 7.25 GE per bit; a register with a 4-to-1 MUX costs 8.25 GE per bit. The key register of Type-I implementation consists of 120 registers with a 3-to-1 MUX (870 GE) and 8 registers with a 4-to-1 MUX (66 GE), while the key register of Type-II/III implementation consists of 120 scan flip-flops (750 GE) and 8 registers with a 3-to-1 MUX (58 GE). Thus, the area savings of 128 GE are achieved.

The area savings for the data processing block of Type-III implementation over Type-I/II implementation are 78 GE. As for the data register of Type-III implementation 32 scan flips-flops (200 GE) is replaced with 32 D flip-flops (144 GE), which leads savings of 56 GE. 24 3-to-1 MUXes with output inverted (54 GE) can be replaced with 24 2-to-1 MUXes with output inverted (42 GE) in the matrix multiplier, leading to savings of 12 GE. In addition, 8 XOR gates (16 GE) for secret key XOR is merged to XOR gates in the matrix multiplier. Therefore, the area savings of 78 GE are achieved despite the additional 6 GE for the other MUX.

Table 2 shows the implementation results of the proposed architectures together with their versions supporting both encryption and decryption. We also show, for comparison, the best known result of CLEFIA and low-area implementation results of AES. Our implementations supporting encryption only achieve

Table 1. Detailed implementation figures

Components [GE]	Type-I	Type-II	Type-III
Data Processing Block	1392.5	1392.5	1314.5
Data Register (including MUX)	668	668	612
S-box (including MUX)	332.5	332.5	332.5
S0	117.25	117.25	117.25
S1	201.25	201.25	201.25
Matrix Multiplier	212	212	200
Secret Key MUX	136	136	136
Secret Key XOR	16	16	0^*
Round Key XOR	16	16	16
Other MUX	12	12	18
Key Scheduling Block	952	824	824
Key Register (including MUX)	936	808	808
CON XOR	16	16	16
Controller	333	377.25	349.25
Total [GE]	2677.5	2593.75	2487.75
Encryption [cycles]	176	192	328
Key Setup [cycles]	128	128	224
Throughput @100KHz [Kbps]	73	67	39

$*$: Secret key XOR is merged to XOR gates in matrix multiplier

46–50% reduction of the area requirements compared to the smallest implementation [10, 12] of CLEFIA. As for implementations supporting both encryption and decryption, our implementations are 44–47% smaller. Type-III implementation is 4% larger than the smallest encryption-only implementation [5] of AES, but its encryption/decryption version achieves 23% reduction of the area requirements compared to the smallest encryption/decryption implementation [3] of AES.

6 Conclusion

In this paper, we have proposed very compact hardware architectures of CLEFIA with 128-bit keys based on 8-bit shift registers. We showed that the data processing part of CLEFIA-128 can be implemented in a serialized way without any additional registers. Three types of hardware architectures were proposed according to required cycles for one block process by adaptively applying clock gating technique. Those architectures were implemented and synthesized using a 0.13 μm standard cell library. In our smallest implementation, the area requirements are only 2,488 GE, which is 50% smaller than the smallest implementation of CLEFIA-128, and competitive to the smallest AES-128 implementation. Moreover, the area requirements for its version supporting both encryption and decryption are only 2,604 GE, which achieve 23% reduction of area requirement compared to the smallest encryption/decryption implementation of AES-128.

Table 2. Implementation results and comparison

Algorithm	Source	Mode	Enc/Dec [cycles]	Key Setup [cycles]	Area [GE]	Throughput @100KHz [Kbps]	Tech. [μm]
CLEFIA	Type-I	Enc	176	128	2,678	73	0.13
		Enc/Dec	176	128	2,781	73	
	Type-II	Enc	192	128	2,594	67	
		Enc/Dec	192/184	128	2,678	67/70	
	Type-III	Enc	328	224	2,488	39	
		Enc/Dec	328/320	224	2,604	39/40	
	[10, 12]	Enc/Dec	36	24	4,950	356	0.09
AES	[3]	Enc/Dec	1,032/1,165	-	3,400	12/11	0.35
	[4]	Enc	177	-	3,100	72	0.13
	[5]	Enc	226	-	2,400	57	0.18

Future work will include the application of side-channel countermeasures such as threshold implementations [6, 7] to the proposed architectures.

References

1. C. D. Cannière, O. Dunkelman, and M. Knezevic, "KATAN and KTANTAN – a Family of Small and Efficient Hardware-Oriented Block Ciphers", *CHES 2009*, LNCS 5747, pp. 272–278, Springer-Verlag, 2009.
2. J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard (Information Security and Cryptography)*, Springer, 2002.
3. M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES Implementation on a Grain of Sand", *IEE Proceedings Information Security*, vol. 152, pp. 13–20, 2005.
4. P. Hämäläinen, T. Alho, M. Hännikäinen, and T. Hämäläinen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core", *DSD 2006*, pp. 577–583, IEEE Computer Society, 2006.
5. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the Limits: A Very Compact and a Threshold Implementation of AES", *EUROCRYPT 2011*, LNCS 6632, pp. 69–88, Springer-Verlag, 2011.
6. S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations against Side-channel Attacks and Glitches", *ICICS 2006*, LNCS 4307, pp. 529–545, Springer-Verlag, 2006.
7. S. Nikova, V. Rijmen, and M. Schläffer, "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches", *ICICS 2008*, LNCS 5461, pp. 218–234, Springer-Verlag, 2008.
8. C. Rolfes, A. Poschmann, G. Lender, and C. Paar, "Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents", *CARDIS 2008*, LNCS 5189, pp. 89–103, Springer-Verlag, 2008.
9. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-bit Blockcipher CLEFIA (Extended Abstract)", *FSE 2007*, LNCS 4593, pp. 181–195, Springer-Verlag, 2007.
10. T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "Hardware Implementations of the 128-bit Blockcipher CLEFIA", *Technical Report of IEICE*, vol. 107, no. 141, ISEC2007–49, pp. 29–36, 2007 (in Japanese).

11. "The 128-bit Blockcipher CLEFIA: Algorithm Specification", Revision 1.0, 2007, Sony Corporation.
`http://www.sony.net/Products/cryptography/clefia/download/data/clefia-spec-1.0.pdf`
12. "The 128-bit Blockcipher CLEFIA: Security and Performance Evaluations", Revision 1.0, 2007, Sony Corporation.
`http://www.sony.net/Products/cryptography/clefia/download/data/clefia-eval-1.0.pdf`
13. T. Sugawara, N. Homma, T. Aoki, and A. Satoh, "High-Performance ASIC Implementations of the 128-bit Block Cipher CLEFIA", *ISCAS 2008*, pp. 2925–2928, 2008
14. Y. Zheng, T. Matsumoto, and H. Imai, "On the Construction of Block Ciphers Provably Secure and not Relying on Any Unproved Hypotheses", *Crypto'89*, LNCS 435, pp. 461–480, Springer-Verlag, 1989.

Appendix

In this appendix, we show the detailed data flow of the registers R_{ij} in Fig. 5 during a round of the encryption processing for Type-I architecture. Fig. 9 defines the data structure of a round of the encryption processing. The contents of the registers R_{ij} $(0 \leq i < 4)$ are clarified in Table 3.

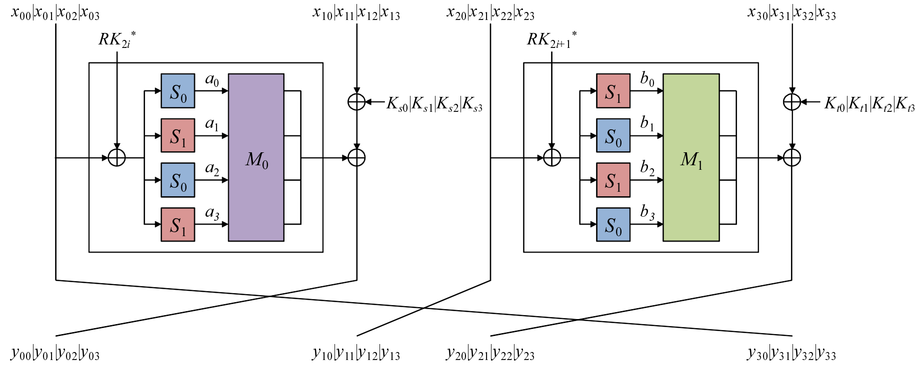


Fig. 9. A round of encryption processing

Table 3. Contents of registers R_{ij} $(0 \leq i, j < 4)$ at the l-th cycle

l	0	1	2	3	4
R_{00}	x_{00}	x_{01}	x_{02}	x_{03}	x_{20}
R_{01}	x_{01}	x_{02}	x_{03}	x_{20}	x_{21}
R_{02}	x_{02}	x_{03}	x_{20}	x_{21}	x_{22}
R_{03}	x_{03}	x_{20}	x_{21}	x_{22}	x_{23}
R_{10}	x_{10}	x_{21}	x_{22}	x_{23}	x_{30}
R_{11}	x_{11}	x_{22}	x_{23}	x_{30}	x_{31}
R_{12}	x_{12}	x_{23}	x_{30}	x_{31}	x_{32}
R_{13}	x_{13}	x_{30}	x_{31}	x_{32}	x_{33}
R_{20}	x_{20}	$x_{13} \oplus \{\texttt{06}\}a_0$	$x_{12} \oplus \{\texttt{04}\}a_0 \oplus \{\texttt{06}\}a_1$	$x_{11} \oplus \{\texttt{02}\}a_0 \oplus a_1 \oplus \{\texttt{06}\}a_2 \oplus K_{s1}$	y_{00}
R_{21}	x_{21}	$x_{12} \oplus \{\texttt{04}\}a_0$	$x_{13} \oplus \{\texttt{06}\}a_0 \oplus \{\texttt{04}\}a_1$	$x_{10} \oplus a_0 \oplus \{\texttt{02}\}a_1 \oplus \{\texttt{04}\}a_2 \oplus K_{s0}$	y_{01}
R_{22}	x_{22}	$x_{11} \oplus \{\texttt{02}\}a_0$	$x_{10} \oplus a_0 \oplus \{\texttt{02}\}a_1 \oplus K_{s0}$	$x_{13} \oplus \{\texttt{06}\}a_0 \oplus \{\texttt{04}\}a_1 \oplus \{\texttt{02}\}a_2$	y_{02}
R_{23}	x_{23}	$x_{10} \oplus a_0 \oplus K_{s0}$	$x_{11} \oplus \{\texttt{02}\}a_0 \oplus a_1 \oplus K_{s1}$	$x_{12} \oplus \{\texttt{04}\}a_0 \oplus \{\texttt{06}\}a_1 \oplus a_2 \oplus K_{s2}$	y_{03}
R_{30}	x_{30}	x_{31}	x_{32}	x_{33}	x_{00} $(= y_{30})$
R_{31}	x_{31}	x_{32}	x_{33}	x_{00}	x_{01} $(= y_{31})$
R_{32}	x_{32}	x_{33}	x_{00}	x_{01}	x_{02} $(= y_{32})$
R_{33}	x_{33}	x_{00}	x_{01}	x_{02}	x_{03} $(= y_{33})$

l	4	5	6	7	8
R_{00}	x_{20}	x_{21}	x_{22}	x_{23}	y_{00}
R_{01}	x_{21}	x_{22}	x_{23}	y_{00}	y_{01}
R_{02}	x_{22}	x_{23}	y_{00}	y_{01}	y_{02}
R_{03}	x_{23}	y_{00}	y_{01}	y_{02}	y_{03}
R_{10}	x_{30}	y_{01}	y_{02}	y_{03}	x_{20} $(= y_{10})$
R_{11}	x_{31}	y_{02}	y_{03}	x_{20}	x_{21} $(= y_{11})$
R_{12}	x_{32}	y_{03}	x_{20}	x_{21}	x_{22} $(= y_{12})$
R_{13}	x_{33}	x_{20}	x_{21}	x_{22}	x_{23} $(= y_{13})$
R_{20}	y_{00}	$x_{33} \oplus \{\texttt{0A}\}b_0$	$x_{32} \oplus \{\texttt{02}\}b_0 \oplus \{\texttt{0A}\}b_1$	$x_{31} \oplus \{\texttt{08}\}b_0 \oplus b_1 \oplus \{\texttt{0A}\}b_2 \oplus K_{t1}$	y_{20}
R_{21}	y_{01}	$x_{32} \oplus \{\texttt{02}\}b_0$	$x_{33} \oplus \{\texttt{0A}\}b_0 \oplus \{\texttt{02}\}b_1$	$x_{30} \oplus b_0 \oplus \{\texttt{08}\}b_1 \oplus \{\texttt{02}\}b_2 \oplus K_{t0}$	y_{21}
R_{22}	y_{02}	$x_{31} \oplus \{\texttt{08}\}b_0$	$x_{30} \oplus b_0 \oplus \{\texttt{08}\}b_1 \oplus K_{t0}$	$x_{33} \oplus \{\texttt{0A}\}b_0 \oplus \{\texttt{02}\}b_1 \oplus \{\texttt{08}\}b_2$	y_{22}
R_{23}	y_{03}	$x_{30} \oplus b_0 \oplus K_{t0}$	$x_{31} \oplus \{\texttt{08}\}b_0 \oplus b_1 \oplus K_{t1}$	$x_{32} \oplus \{\texttt{02}\}b_0 \oplus \{\texttt{0A}\}b_1 \oplus b_2 \oplus K_{t2}$	y_{23}
R_{30}	y_{30}	y_{30}	y_{30}	y_{30}	y_{30}
R_{31}	y_{31}	y_{31}	y_{31}	y_{31}	y_{31}
R_{32}	y_{32}	y_{32}	y_{32}	y_{32}	y_{32}
R_{33}	y_{33}	y_{33}	y_{33}	y_{33}	y_{33}