A Very Compact Hardware Implementation of the MISTY1 Block Cipher

Dai Yamamoto, Jun Yajima, and Kouichi Itoh

FUJITSU LABORATORIES LTD.
4-1-1, Kamikodanaka, Nakahara-ku, Kawasaki, 211-8588, Japan
{ydai,jyajima,kito}@labs.fujitsu.com

Abstract. This paper proposes compact hardware (H/W) implementation for the MISTY1 block cipher, which is an ISO/IEC18033 standard encryption algorithm. In designing the compact H/W, we focused on optimizing the implementation of FO/FI functions, which are the main components of MISTY1. For this optimization, we propose two new methods; reducing temporary registers for the FO function, and shortening the critical path for the FI function. According to our logic synthesis on a 0.18-μm CMOS standard cell library based on our proposed method, the gate size is 3.95 Kgates, which is the smallest as far as we know.

Key words: Block cipher, MISTY1, Hardware, ASIC, Compact Implementation

1 Introduction

The MISTY1 64-bit block cipher [1] is an ISO/IEC18033 [2] standard encryption algorithm. MISTY1 can be implemented in various ways in order to meet different performance requirements, such as compact design or high-speed preference. So, MISTY1 is suitable for embedded systems, such as mobile phones.

A number of MISTY1 ASIC implementations have been studied [3] [4] [5]. In [3] [4], compact MISTY1 architectures were designed. To realize compact design, these architectures use the only one FI function module repeatedly, and use S-boxes that are implemented in combinational logic. However, these architectures do not use common methods for the compact design, in which extended keys are sequentially generated in the encryption/decryption process in order to limit the register size of extended keys to 16 bits. Furthermore, they do not optimize the implementation method of the FO/FI function in consideration of using one FI function module. This optimization is very significant for the compact MISTY1 H/W.

In this paper, we focus on four strategies for the compact design. First, we choose to implement the H/W by using one FI function. Secondly, we use S-boxes implemented in the combinational logic. Thirdly, extended keys are generated sequentially in our H/W. Fourthly, we optimize the implementation of the FO/FI function. To realize this optimization, we propose two new methods. One reduces the temporary register for the FO function by the optimization of an FO function structure. Another shortens the critical path around the FI function by reducing the number of XOR gates in the critical path.

With our strategies, we synthesize MISTY1 H/W by a 0.18-μm CMOS standard cell library (CS86 technology[18]), and the performance evaluations are shown. As a result, an extremely small size of 3.95 Kgates with 71.1 Mbps throughput is obtained for our MISTY1 H/W. This is the smallest MISTY1 H/W, as far as we know.

Our proposed methods can be applied not only to MISTY1 but also to MISTY2 [1] and KASUMI [6], which have a similarly structured MISTY1 FO function. In [7], the compact H/W of KASUMI is proposed. A further gate count reduction of the KASUMI H/W can be realized by using our proposal.

The rest of the paper is organized as follows. A survey of related work is found in Chapter 2. Chapter 3 explains the algorithm of MISTY1. Our strategy for the smallest H/W of MISTY1 is discussed in Chapter 4. Chapter 5 proposes two new methods for an effective H/W implementation of MISTY1. Chapter 6 presents evaluation results for gate counts and the performance of our H/W, compared with previous results. Finally, we conclude with a summary and comment on future directions in Chapter 7.

2 Previous work

A large number of MISTY1 H/W implementation evaluations on FPGA and ASIC have been studied.

The implementation on FPGA platform was reported in [8] [9] [10] [11] [12] [13] [14]. In [8] [9] [10], designers of MISTY1 have implemented MISTY1 H/W based on three types of H/W architectures; the fully loop unrolled architecture, the pipeline architecture, and the loop architecture. The two former architectures allow high processing speed, while the latter architecture allows a compact circuit. The implemented H/W based on the loop architecture uses a large 128-bit register for extended keys. In [11] [12], the implemented H/W was aimed not at compact design but high H/W efficiency, and it had an encryption function without a decryption function. In [13] [14], the implemented H/W had both the encryption and decryption function. Also, RAM blocks embedded in the considered FPGA devices were used for the implementation of S-boxes, so the implemented H/W realized higher H/W efficiency.

Implementation on the ASIC platform was reported in [3] [4] [5]. In [3] [4], developers of MISTY1 implemented and evaluated MISTY1 H/W. In particular, the research purpose in [4] is to reduce the gate count, and the implementation methods of FO/FI functions are well-studied. However, the gate size of their H/W is not small enough because one large 128-bit register is used for the extended key. The H/W performances of various block ciphers including MISTY1 are compared in [5]. In [5], the MISTY1 H/W is implemented straightforwardly based on the cipher specification. S-boxes are implemented by a lookup table in consideration of the fairness among ciphers, and a 128-bit register is used for extended keys, so the gate count of the implemented H/W is not small.

3 MISTY1

Figure 1 shows the nested structure of MISTY1 excluding the key scheduler [1]. MISTY1 encrypts a 64-bit plaintext using a 128-bit secret key. MISTY1 has the Feistel network with a variable number of rounds n including FO functions and FL/FL^{-1} functions. Since $n = 8$ is recommended in [1], we set $n = 8$ in the rest of this paper. The $FO_i(1 \leq i \leq 8)$ function uses a 48-bit extended key KI_i and a 64-bit extended key KO_i. The $FL_i(1 \leq i \leq 10)$ function is used in the encryption, meanwhile the FL_i^{-1} function is used in the decryption with a 32-bit extended key KL_i. In Fig. 1, 16-bit KL_{i1} and KL_{i2} are the left and right data of 32-bit KL_i, respectively. The FO_i function has three FI functions $FI_{ij}(1 \leq j \leq 3)$. Here, $KO_{ij}(1 \leq j \leq 4)$ and $KI_{ij}(1 \leq j \leq 3)$ are left j-th 16-bit data of KO_i and KI_i, respectively. The FI function uses the 7-bit S-box S_7 and the 9-bit S-box S_9. Here, the zero-extended operation is performed to 7-bit blocks by adding two '0's. The truncate operation truncates the two most significant bits of a 9-bit string. KI_{ij1} and KI_{ij2} are the left 7 bits and the right 9 bits of KI_{ij}, respectively. Here, the key scheduler of MISTY1 is explained. $K_i(1 \leq i \leq 8)$ is the left i-th 16 bits of a 128-bit secret key. $K'_i(1 \leq i \leq 8)$ corresponds to the output of FI_{ij} where the input of FI_{ij} is assigned to K_i and the key KI_{ij} is set to $K_{(i mod 8)+1}$. The assignment between the 16-bit secret/extended keys K_i, K'_i and the 16-bit round key $KO_{ij},\ KL_{ij},\ KI_{ij}$ is defined in Table 1, where i equals $(i-8)$ when $(i > 8)$.

Table 1. The assignment between K_i, K'_i and $KO_{ij},\ KI_{ij}$

Round	KO_{i1}	KO_{i2}	KO_{i3}	KO_{i4}	KI_{i1}	KI_{i2}	KI_{i3}	KL_{i1}	KL_{i2}
Secret/ Extended	K_i	K_{i+2}	K_{i+7}	K_{i+4}	K'_{i+5}	K'_{i+1}	K'_{i+3}	$K_{\frac{i+1}{2}}$ (*odd i*) $K'_{\frac{i}{2}+2}$ (*even i*)	$K'_{\frac{i+1}{2}+6}$ (*odd i*) $K'_{\frac{i}{2}+4}$ (*even i*)

4 Four strategies for the compact design

4.1 The number of the FO/FI function module

The FO/FI function is the main component of MISTY1, so the FO/FI function is one of the most influential factors for gate counts of MISTY1 H/W. Thus, it is important to decide the number of the FO/FI function module. MISTY1 has a nested structure including FO functions and FL/FL^{-1} functions, so the number of the FO/FI function module can be variously selected. When MISTY1

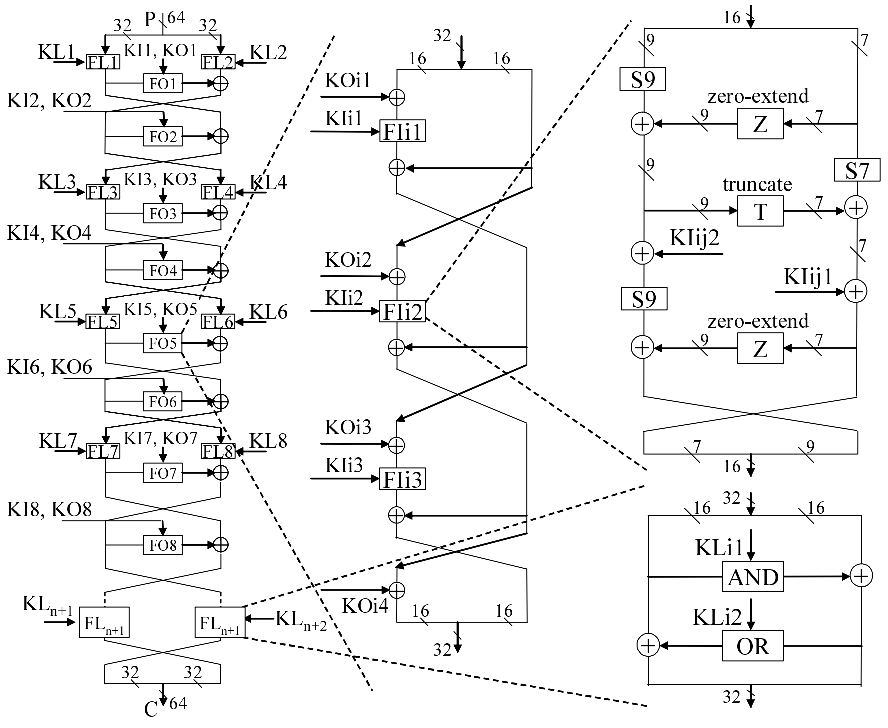

Fig. 1. MISTY1 encryption algorithm

is implemented with a pipelined H/W architecture, eight FO function modules are performed in the same clock cycle. This is suitable for high-speed implementation, but leads to a large circuit size. Therefore, we choose to implement only one FI function module for the compact design. That is, the FO function is executed in three clock cycles by repeatedly using one FI function module. This architecture leads to low speed processing, but is suitable for compact implementation.

4.2 Extended key generation method

The generation method of extended keys in MISTY1 is classified into two methods; called the "register method" and the "on-the-fly method". It is important for the compact design to choose between two methods. In the register method, a 128-bit extended key is generated and stored into a 128-bit register in advance of the encryption/decryption process, and the required extended key is read directly from the register. In the on-the-fly method, a 16-bit required extended key is generated in the encryption/decryption process sequentially. The on-the-fly method is more suitable for the compact design than the register method because of the 128-bit register. Therefore, we chose the on-the-fly method, which has not been reported in the existing H/W implementation of MISTY1, but has been employed in other algorithm implementations, such as AES. That is, all of the previous architectures of MISTY1 are based on not the on-the-fly method but the register method. Also, because of the MISTY1 algorithm, the FI function is used not only in the encryption/decryption process but also in the extended key generation process. Therefore, we chose the implemented method, in which one FI function module is shared with these two processes. Thus, both the encryption/decryption process and the extended key generation process cannot be performed in the same cycle. So, a 16-bit register is required to retain a 16-bit extended key generated sequentially. Here, our on-the-fly method requiring a 16-bit register is called the "sequential method".

4.3 S-box Implementation method

The S-box performance of MISTY1, including gate counts, depends on the S-box implementation method, so it is important for the compact design to discuss them. The implementation method of two S-boxes (S_7 and S_9) is considered as follows. The two S-boxes of MISTY1 have been designed so that they can be easily implemented in combinational logic as well as by a lookup table [1]. On MISTY1, S-boxes in combinational logic show better performance both in terms of the area size and the delay time than that by a lookup table [3]. We confirmed that the same results are obtained when the implemented S-boxes are synthesized by using a 0.18-μm CMOS standard cell library. Therefore, we used S-boxes implemented in combinational logic.

4.4 Optimization of FO/FI function

The proposed H/W uses one FI function module repeatedly. Furthermore, it is very significant for the smallest MISTY1 H/W to discuss the following two methods; a concrete implementation method of FO function in three cycles by using one FI function module, and the method of reducing the gate count of the FI function itself. In Chapter 5, we propose these two new methods.

5 Proposed methods for the compact design

5.1 Reducing the temporary register for the FO function

When an FO function is executed in three cycles by repeatedly using one FI function module, an intermediate result in each cycle must be stored into a register. The FO function transforms 32-bit data, so a 32-bit temporary register for the intermediate result (i.e., is "temporary register") is required. We reduced the size of the temporary register to 16 bits.

The concept of the proposed method is explained by reference to Fig. 2. It shows the method of dividing an FO function into three cycles. The previous method is straightforward based on MISTY1 specification. An FO function is separated horizontally for every cycle in the previous method, so a 32-bit temporary register is required for left and right 16-bit data. Meanwhile, an FO function is separated vertically for every cycle in the proposed method. In fact, the output data from the FI function in Cycle2 is directly XORed with a data register, so the 16-bit temporary register for the data is reduced by the proposed separation.

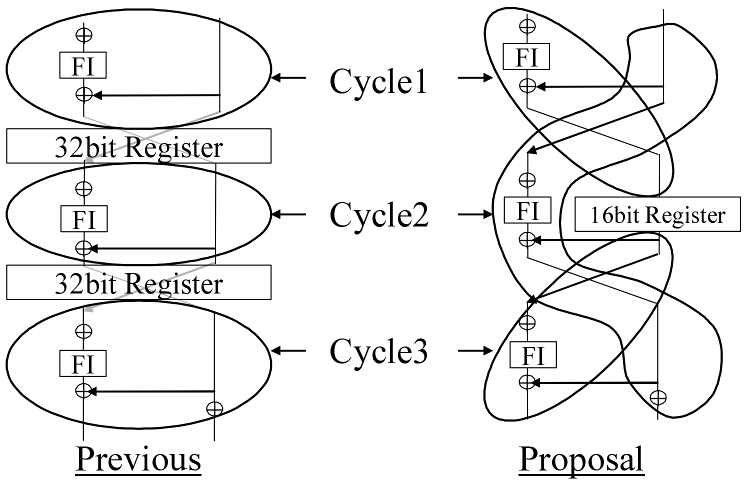

Fig. 2. Concept of temporary register reduction

The detail and effectiveness of the proposed method is explained in the following steps. First, the straightforward architecture based on MISTY1 specification is explained as "existing method". Next, the proposed architecture based on the proposed concept shown in Fig. 2 is explained as "proposed method (a)". Then, we propose the second proposed architecture, which is maximally optimized for compact design as "proposed method (b)". Finally, the gate counts of FI functions are estimated. By using proposed method (b), the size of the FO function is estimated to be reduced 17% of the existing method.

Existing method The common partition algorithm based on MISTY1 specification is shown in Fig. 3 (I) as the existing algorithm. Equation (1) in Appendix shows each process for three cycles in this algorithm. Let *Reg*-R_H, *Reg*-R_L, *Reg*-L_H, and *Reg*-L_L be 16-bit registers (called "Feistel data register"), respectively. The Feistel data register means the register for storing intermediate results for each round plaintext/ciphertext. Also, let *Reg*-*FOR* and *Reg*-*FOL* be the 16-bit temporary registers, so the total size of temporary registers is 32 bits in the existing algorithm. Next, the existing architecture based on the existing algorithm is shown in Fig. 3 (II). Note that the registers described in the following architecture figures include a 2-1MUX. This 2-1MUX can select the value stored in the register or the value of the external input. The value stored into the register can be updated to the selected value. Consequently, two 16-bit registers, $Reg\text{-}FOR$ and $Reg\text{-}FOL$, are required in the existing method.

Proposed method (a) We discuss the proposed algorithm (a) shown in Fig. 4 (I), which is designed based on the proposed concept shown in Fig. 2. Comparing Fig. 4 (I) with Fig. 3 (I), the output data from the FI function is directly XORed with $Reg\text{-}R_H$ and $Reg\text{-}R_L$ in Cycle2 in Fig. 4 (I). This makes it possible to remove a 16-bit temporary register because the output data from the FI function in Cycle2 does not need to be stored into the temporary register. The proposed architecture (a) based on the proposed algorithm (a) is shown in Fig. 4 (II). Equation (2) in Appendix shows each process for three cycles shown in Fig. 4 (I). Although the proposed architecture (a) shown in Fig. 4 (II) can remove the 16-bit temporary register, there is another issue. The issue is that the number of MUX operators is increased compared with the existing method, because the circuit structure differs in every three cycles, which comes from the vertical separation of the proposed method. Concretely, different values are input into input1 and input4 in the FI function in three cycles, so the existing architecture has two 16-bit 2-1MUX, meanwhile the proposed architecture (a) has two 16-bit 3-1MUX. Also, the proposed architecture (a) has a 16-bit 2-1MUX instead of a 16-bit XOR operator located above $Reg\text{-}R_H$. This 2-1MUX selects the output data from the FI function in Cycle2, and the extended key KO_{i4} in Cycle3. In other words, the proposed method (a) reduces the 16-bit temporary register, but increases the 48-bit 2-1MUX compared with the existing method.

Proposed method (b) The algorithm (b) shown in Fig. 5 (I) aims to reduce the 2-1MUX increased in the proposed method (a). One of the redundant MUX operators is a 2-1MUX located above $Reg\text{-}R_H$ in Fig. 4 (II). If KO_{i4} is XORed with $Reg\text{-}R_H$ without this 2-1MUX, then the 2-1MUX can be removed. To remove the 2-1MUX, we focused on the input4 in the FI function. KO_{i4} is input into the input4 in both Cycle2 and Cycle3 as shown in Fig. 5 (I). That is, KO_{i4} is XORed with both $Reg\text{-}R_H$ and $Reg\text{-}R_L$ in Cycle2, and KO_{i4} is XORed with only $Reg\text{-}R_L$ in Cycle3, so KO_{i4} is cancelled on $Reg\text{-}R_L$, and XORed with only $Reg\text{-}R_H$ finally. This algorithm can remove the 2-1MUX located above $Reg\text{-}R_H$. Moreover, the input4 in both Cycle2 and Cycle3 in this algorithm is the same value KO_{i4}. Therefore, not 3-1MUX but 2-1MUX is assigned above the input4 in the proposed method (b). In other words, the proposed method (b) reduces the 32-bit 2-1MUX compared with the proposed method (a). The proposed architecture (b) based on the proposed algorithm (b) is shown in Fig. 5 (II). Equation (3) in Appendix shows each process for the three cycles shown in Fig. 5 (I).

The gate counts of FI functions based on the above three architectures are estimated as shown Table 2. Let MUX, REG, and XOR be the 2-1multiplexer, the register, and the exclusive-OR, respectively. We supposed 1-bit 2-1MUX = 3.5 NAND gates, 1-bit REG = 13.5 NAND gates, 1-bit XOR = 2.5 NAND gates. In the row of MUX in Table 2, the value is based on a 1-bit 2-1MUX. For example, 16-bit 3-1MUX is regarded as two 16-bit 2-1MUX (= one 32-bit 2-1MUX). From Table 2, the gate count of the FI function based on the proposed architecture (b) is 17% smaller than the existing architecture.

Table 2. Comparison of gate counts of FI function

	Existing	Proposed (a)	Proposed (b)
# 1-bit MUX	32	80	48
# 1-bit REG	64	48	48
# 1-bit XOR	80	64	64
Total [gate] (†)	1176	1088	976

(†) MUX = 3.5gate/bit (in 2-1MUX) REG = 13.5gate/bit XOR = 2.5gate/bit

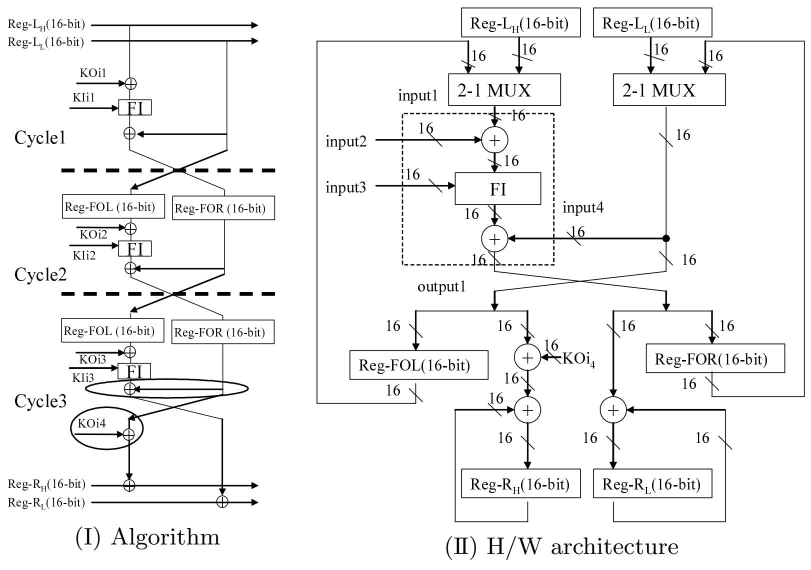

Fig. 3. Existing method

5.2 Shortening the critical path around an FI function

MISTY1 has a Feistel network with eight FO functions, and an FO function comprises three FI functions. Also, MISTY1 extended key is obtained by using the

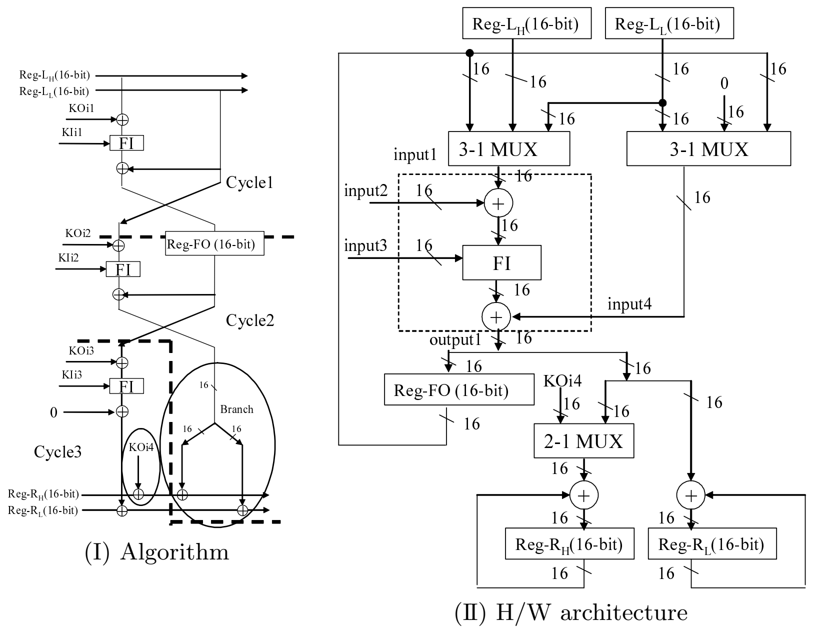

Fig. 4. Proposed method (a)

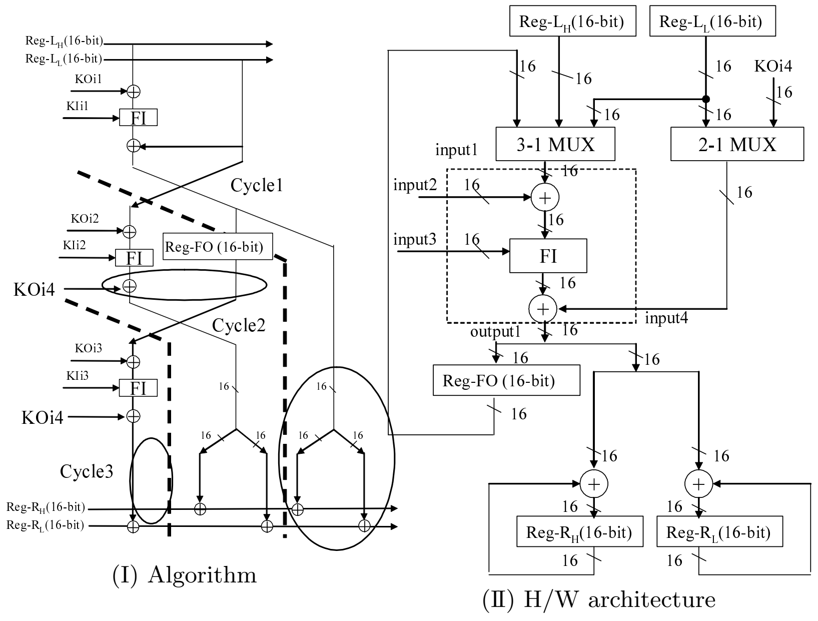

Fig. 5. Proposed method (b)

FI function. Thus, the performance of any MISTY1 H/W depends on the processing speed of the FI function. The proposed method improves the processing speed, which is important as well as the area size for the compact H/W.

Figure 6 shows the straightforward and the proposed algorithm of the FI function. XOR gates under a FI function in a FO function are described in Fig. 6. In Fig. 6, the critical path with two S-boxes S_9 in the FI function including these XOR gates is illustrated by the thick line. The XOR gate into which KI_{ij2} is input in the straightforward algorithm is transferred just below the first zero-extend operation in the proposed algorithm (Move1). This movement reduces one XOR gate on the critical path. To guarantee the logic equivalence in both algorithms of the FI function, the KI_{ij1} input in the straightforward algorithm is modified to the (KI_{ij1} XOR KI_{ij2}) input (Move2). Here, the two most significant bits of KI_{ij2} are truncated, and XORed with KI_{ij1}. Next, the 9-bit XOR gate under the FI function is transferred just below the second zero-extend operation (Move3). In other words, the proposed algorithm reduces two XOR gates on the critical path in the FI function, and realizes higher processing speed of the FI function almost without the gate counts increase.

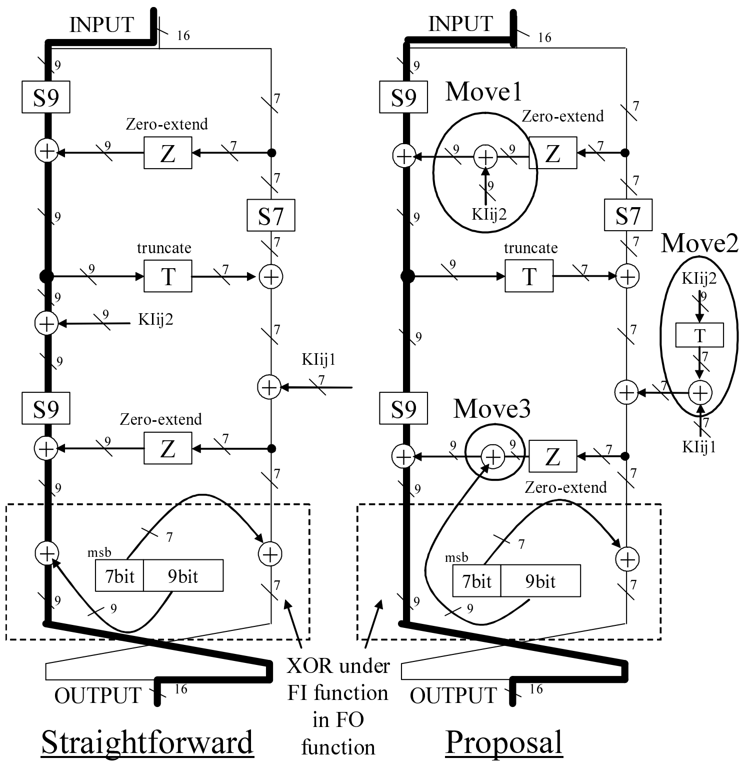

Fig. 6. Method for shortening the critical path around FI function

6 ASIC performance evaluation

6.1 Structure of implemented H/W

The implemented H/W comprises two circuits; the interface circuit(called the "I/F circuit") and the core circuit. The I/F circuit comprises a plaintext/ciphertext register and a secret key register. The core circuit comprises FI/FL/FL^{-1} function, selectors, counter circuit, and various registers (four 16-bit Feistel data registers, 16-bit temporary register, and 16-bit extended key register). The core circuit has only one FI function module, and the module is shared with the extended key generation process and the encryption/decryption process. Also, the core circuit has a 16-bit temporary register because of the proposed method, and has a 16-bit small extended key register due to the sequential method. By connecting the core circuit to the I/F circuit, MISTY1 H/W can be implemented as a VLSI chip. The block structure of the I/F circuit and the core circuit is shown in Fig. 7.

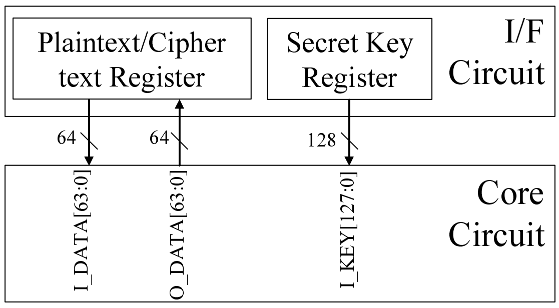

Fig. 7. The block structure of the I/F circuit and the core circuit

The implemented H/W generates a 64-bit ciphertext (plaintext) from a 64-bit plaintext (ciphertext) in 60 clock cycles. The details are as follows. The data input and output requires 2 cycles. The encryption/decryption processes in FO and FL/FL^{-1} function require 24 cycles and 10 cycles, respectively. The extended key generation process in the FI function requires 3 (cycles) $\times$ 8 (rounds) = 24 cycles, because an extended key generated by using the FI function is input into FL/FL^{-1} function in the same cycle.

6.2 Comparison of our MISTY1 H/W results

This section evaluates the ASIC performance of the proposed H/W based on the structure shown in Section 6.1. The evaluation environment is as follows.

H/W description language Verilog-HDL

Design library Fujitsu 0.18-μm CMOS standard cell library (CS86 technology [18])
Logic synthesizer Design Compiler 2006.06-SP5-1
Synthesis condition Worst case condition (Supply voltage: 1.65V, Junction temperature: 125˚C)

In this evaluation, the proposed H/W is not based on scan design, and is synthesized with the Design Compiler with size optimization and ungroup command. Also, one gate is equivalent to 2-1NAND gate.

Table 3 shows the ASIC performance of three types of the proposed H/W; the core circuit (Proposed 1), the core circuit with the secret key register (Proposed 2), and the core and I/F circuit (Proposed 3). In Table 3, "Block Structure" in the last column means the above-mentioned three types of the proposed H/W. Also, "H/W efficiency" means the throughput per gate, so the implementation with higher throughput and smaller gate counts show higher values. In this paper, the H/W efficiency is defined as the throughput divided by area size by reference to [5]. From Table 3, it is confirmed that a size of 3.95 Kgates with 71.1 Mbps throughput is obtained for our MISTY1 core (Proposed 1).

Table 3. H/W performance comparison in ASICs

Source	Process [μm]	Cycle	S-box	Freq. [Mhz]	Thr'put [Mbps]	Area [Kgates]	Efficiency [Kbps/gates]	Block Structure
Proposed 1	0.18	60	Logic	66.7	71.1	3.95	18.0	Core
Proposed 2	0.18	60	Logic	66.7	71.1	4.79	14.9	(‡)
Proposed 3	0.18	60	Logic	66.7	71.1	5.29	13.4	Core + I/F
[4]	0.60	35	(†)	29.9	66.3	8.099	8.19	Core
[5]	0.18	30	Table	92.6	197.5	9.3	21.3	(‡)
[15]	0.18	(†)	(†)	(†)	70.2	5.39	13.0	(†)
[16]	0.18	35	(†)	(†)	78.4	6.10	12.85	(†)

(†) Unknown, (‡) Core + Secret Key Register

6.3 Further comparison

This section compares the performance of existing and proposed architectures. The lower rows in Table 3 show the performance of existing architectures reported in [4] [5] [15] [16]. The implemented architectures shown in [4] [5] are based on the register method and have the only one FI function module. The core circuits in [4] [5] include a 128-bit extended key register. Meanwhile, the information of implementation methods is not clear in [15] [16]. From Table 3, our H/W is implemented with the smallest size and good efficiency.

Because the synthesis condition, such as design library, S-box implementation method, and Block Structure, are different from one another, it might be difficult to fairly compare the performance of each implementation shown in Table 3. In the following evaluation, both MISTY1 H/W based on the available RTL code [17] in [5] and the proposed H/W are synthesized under the same synthesis condition in order to compare performance fairly. These two architectures are based on the same implemented method except for two differences, one is to apply the proposed methods or not, the other is the extended key generation method. The following evaluation compares the performance of three implemented architectures. First, implementation (a) is the MISTY1 H/W based on the RTL code [17], which is implemented straightforwardly based on MISTY1 specification. Second, implementation (b) is obtained from the RTL code [17], where the S-box code was changed from the lookup table to combinational logic. Finally, implementation (c) is our MISTY1 H/W (Proposed 2 in Table 3), which is the core circuit with a secret key register, because implementation (a) and (b) have the only secret key register. Here, implementation (b) and (c) are based on the same implementation methods of S-boxes and Block Structure, so the performance of both implementations can be compared fairly.

Figure 8 shows a comparison of the gate counts of the above three implementations under various delay requirements. From Fig. 8, the gate count of implementation (c) is about 2K gates smaller than that of implementation (b). The reasons are as follows. First, implementation (b) has the 128-bit extended key register due to the register method, while implementation (c) has the 16-bit small one due to the sequential method. Second, implementation (c) has reduced the temporary register due to our proposal described in Section 5.1.

Next, Fig. 9 shows a comparison of the H/W efficiency of the above three implementations under various delay requirements. From Fig. 9, the H/W efficiency of implementation (c) is lower than that of implementation (b). This is mainly because implementation (b) is based on the register method, while implementation (c) is based on the sequential method.

Through the above evaluation, it is confirmed that the H/W efficiency of our MISTY1 H/W is lower than implementation (b), but is better than that of the other reports. The proposed H/W realized the smallest-area of less than 4K gates, which is about 2K gates smaller than the area of straightforward implementation. This is because our MISTY1 H/W is based on the sequential method and our proposed methods described in Section 5.1, 5.2. This paper aims to implement the smallest H/W of MISTY1, so it is significant to maximally reduce the gate count even though the H/W efficiency is not the highest.

7 Conclusion

In this paper, we presented the smallest H/W of the MISTY1 64-bit block cipher, and proposed two new methods. The first method reduced the temporary register for the FO function from 32 bits to 16 bits. The second method shortened the critical path around the FI function by the reduction of the number of XOR gates on the critical path. The implemented MISTY1 H/W was synthesized by a 0.18-μm CMOS standard cell library, then an extremely small size of 3.95

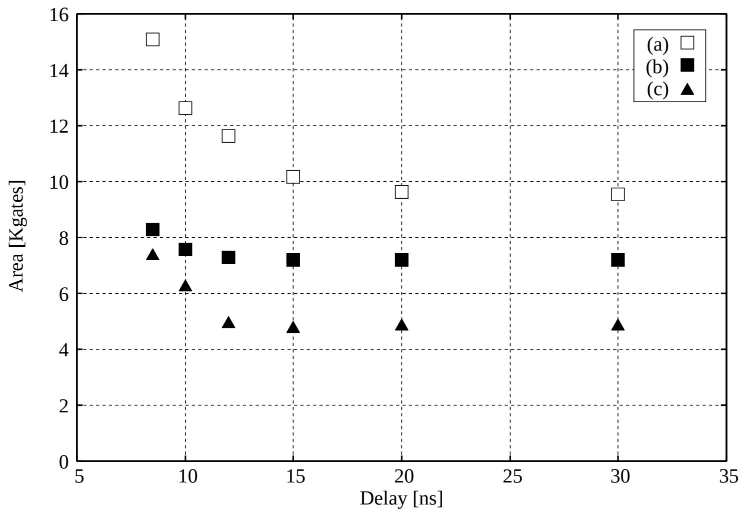

Fig. 8. Comparison of the gate counts under various delay requirements

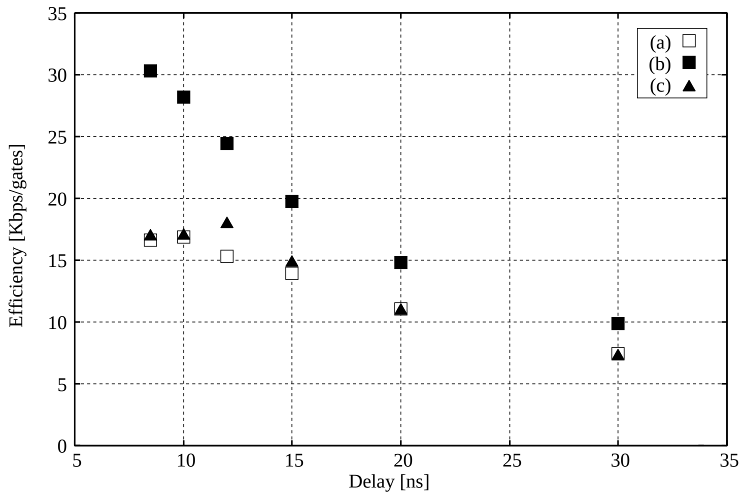

Fig. 9. Comparison of the H/W efficiency under various delay requirements

Kgates with 71.1 Mbps throughput was obtained for our MISTY1 core circuit. In this paper, it was first shown that MISTY1 H/W is implemented with a size of less than 4K gates. Our two proposed methods described in Section 5.1, 5.2 can be applied to MISTY2 [1] and KASUMI [6]. Future work will include discussion on the smallest H/W implementation of MISTY2 and KASUMI by using the proposed methods.

References

1. M. Matsui: "New Block Encryption Algorithm MISTY," Fast Software Encryption, FSE'97, LNCS 1267, pp.54–68, Springer–Verlag, 1997.
2. ISO/IEC18033-3, `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37972`
3. T. Ichikawa, T. Sorimachi and M. Matsui: "A Study on Hardware Design for Block Encryption Algorithms," in *Proc. 1997 Symposium on Cryptography and Information Security*, SCIS1997, 9.D, Jan. 1997 (written in Japanese).
4. T. Ichikawa, J. Katoh and M. Matsui: "An Implementation of MISTY1 for H/W Design," in *Proc. 1998 Symposium on Cryptography and Information Security*, SCIS1998, 9.1.A, Jan. 1998 (written in Japanese).
5. T. Sugawara, N. Homma, T. Aoki and A. Satoh: "ASIC Performance Comparison for the ISO Standard Block Ciphers," in *Proc. 2007 Joint Workshop on Information Security*, JWIS2007, pp.485–498, Aug. 2007.
6. Third Generation Partnership Project: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification (Release 7)," 3GPP TS 35.202 v7.0.0, Jun. 2007.
7. A. Satoh and S. Morioka: "Small and High-Speed Hardware Architectures for the 3GPP Standard Cipher KASUMI," in *Proc. 5th International Conference on Information Security*, ISC2002, pp.48–62, Sep. 2002.
8. A. Sorimachi, T. Ichikawa and T. Kasuya: "On Hardware Implementation of Block Ciphers using FPGA," in *Proc. 2003 Symposium on Cryptography and Information Security*, SCIS2003, 12D-3, Jan. 2003 (written in Japanese).
9. T. Ichikawa, T. Sorimachi and T. Kasuya On Hardware Implementation of Block Ciphers Selected at the NESSIE Project Phase I (1) in *Proc. 2002 Symposium on Cryptography and Information Security*, SCIS2002, 12C-3, Jan. 2002 (written in Japanese).
10. T. Ichikawa, T. Sorimachi, T. Kasuya and M. Matsui: "On the criteria of hardware evaluation of block ciphers," in *Technical Report of IEICE*, ISEC2001-54, Sep. 2001 (written in Japanese).
11. F.-X. Standaert, G. Rouvroy, J.-J. Quisquater and J.-D. Legat: "Efficient FPGA Implementations of Block Ciphers KHAZAD and MISTY1," The Third NESSIE Workshop, Nov. 2002.
12. G. Rouvroy, F.-X. Standaert, J.-J. Quisquater and J.-D. Legat: "Efficient FPGA Implementation of Block Cipher MISTY1," in *Proc. Parallel and Distributed Processing Symposium 2003*, IPDPS2003, Apr. 2003.
13. P. Kitsos and O. Koufopavlou: "A TIME AND AREA EFFICIENT HARDWARE IMPLEMENTATION OF THE MISTY1 BLOCK CIPHER," in *Proc. 46th IEEE Midwest Symposium on Circuits and Systems 2003*, ISCAS2003, Dec. 2003.

14. P. Kitsos, M.D. Galanis and O. Koufopavlou: "A RAM-Based FPGA Implementation of the 64-bit MISTY1 Block Cipher," in *Proc. 46th IEEE Midwest Symposium on Circuits and Systems 2005*, ISCAS2005, May. 2005.
15. Information-technology Promotion Agency, and Telecommunications Advancement Organization of Japan: "CRYPTREC Report 2002," Mar. 2003.
16. Mitsubishi Electric Web Site, `http://global.mitsubishielectric.com/bu/security/rd/rd01_01d.html`
17. HDL codes used for the performance evaluation in [5], `http://www.aoki.ecei.tohoku.ac.jp/crypto/items/JWIS2007.zip`
18. CS86 technology, http://edevice.fujitsu.com/fj/DATASHEET/e-ds/e620209.pdf

Appendix

Existing method

$$\begin{aligned}
\text{Cycle1}: \;& \textit{Reg-FOR} = \mathrm{FI}(\textit{Reg-}L_H \oplus KO_{i1}) \oplus \textit{Reg-}L_L \\
& \textit{Reg-FOL} = \textit{Reg-}L_L \\
\text{Cycle2}: \;& \textit{Reg-FOR} = \mathrm{FI}(\textit{Reg-FOL} \oplus KO_{i2}) \oplus \textit{Reg-FOR} \\
& \textit{Reg-FOL} = \textit{Reg-FOR} \\
\text{Cycle3}: \;& \textit{Reg-}R_H = \textit{Reg-}R_H \oplus (\textit{Reg-FOR} \oplus KO_{i4}) \\
& \textit{Reg-}R_L = \textit{Reg-}R_L \oplus \mathrm{FI}(\textit{Reg-FOL} \oplus KO_{i3}) \oplus \textit{Reg-FOR} \qquad (1)
\end{aligned}$$

Proposed method (a)

$$\begin{aligned}
\text{Cycle1}: \;& \textit{Reg-FO} = \mathrm{FI}(\textit{Reg-}L_H \oplus KO_{i1}) \oplus \textit{Reg-}L_L \\
\text{Cycle2}: \;& \textit{Reg-}R_H = \textit{Reg-}R_H \oplus \mathrm{FI}(\textit{Reg-}L_L \oplus KO_{i2}) \oplus \textit{Reg-FO} \\
& \textit{Reg-}R_L = \textit{Reg-}R_L \oplus \mathrm{FI}(\textit{Reg-}L_L \oplus KO_{i2}) \oplus \textit{Reg-FO} \\
\text{Cycle3}: \;& \textit{Reg-}R_H = \textit{Reg-}R_H \oplus (KO_{i4} \oplus 0) \\
& \textit{Reg-}R_L = \textit{Reg-}R_L \oplus \mathrm{FI}(\textit{Reg-FO} \oplus KO_{i3}) \oplus 0 \qquad (2)
\end{aligned}$$

Proposed method (b)

$$\begin{aligned}
\text{Cycle1}: \;& \textit{Reg-FO} = \big\{\mathrm{FI}(\textit{Reg-}L_H \oplus KO_{i1}) \oplus \textit{Reg-}L_L\big\} \\
& \textit{Reg-}R_H = \textit{Reg-}R_H \oplus \big\{\mathrm{FI}(\textit{Reg-}L_H \oplus KO_{i1}) \oplus \textit{Reg-}L_L\big\} \\
& \textit{Reg-}R_L = \textit{Reg-}R_L \oplus \big\{\mathrm{FI}(\textit{Reg-}L_H \oplus KO_{i1}) \oplus \textit{Reg-}L_L\big\} \\
\text{Cycle2}: \;& \textit{Reg-}R_H = \textit{Reg-}R_H \oplus \big\{\mathrm{FI}(\textit{Reg-}L_L \oplus KO_{i2}) \oplus KO_{i4}\big\} \\
& \textit{Reg-}R_L = \textit{Reg-}R_L \oplus \big\{\mathrm{FI}(\textit{Reg-}L_L \oplus KO_{i2}) \oplus KO_{i4}\big\} \\
\text{Cycle3}: \;& \textit{Reg-}R_H = \textit{Reg-}R_H \\
& \textit{Reg-}R_L = \textit{Reg-}R_L \oplus \big\{\mathrm{FI}(\textit{Reg-FO} \oplus KO_{i3}) \oplus KO_{i4}\big\} \qquad (3)
\end{aligned}$$