A Very Compact Hardware Implementation of the KASUMI Block Cipher

Dai Yamamoto, Kouichi Itoh, and Jun Yajima

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

Abstract. For mobile devices, this paper proposes a compact hardware (H/W) implementation for the KASUMI block cipher, which is the 3GPP standard encryption algorithm. In [4], Yamamoto et al. proposed the method of reducing temporary registers for the MISTY1 FO function (YYI-08), and implemented a very compact MISTY1 H/W. This paper aims to design the smallest KASUMI H/W by the application of YYI-08 to KASUMI, which has a similarly structured MISTY1 FO function. We discussed the applicability and found the problems on register competition and logical equivalence in the simple application, so we propose the new YYI-08 improved for KASUMI and the compact H/W architecture. According to our logic synthesis on a 0.11-μm ASIC process, the gate size is 2.99 Kgates, which is the smallest as far as we know.

Key words: Block cipher, KASUMI, Hardware, ASIC, FPGA, Compact Implementation

1 Introduction

Recently, mobile devices have included not only the basic functions such as telephone calls and cryptographic function for preventing a wiretapping, but also the additional functions such as digital camera and digital television. Future mobile devices have more additional functions as smart devices, so the CPU load will be increasing. Also, there is required high-throughput for the cryptographic function because of future broadband network. Hence the cryptographic function is more suitable to be implemented in dedicated hardware (H/W) than in software on CPU. The mobile devices have only a limited H/W resource, so circuit size of the H/W must be as small and low-power as possible. This paper focuses on the KASUMI 64-bit block cipher [1]. It is estimated that 80% of the global mobile market is based on second generation mobile communications systems (GSM) [2], and more and more GSM mobile phones use the KASUMI (A5/3). Also, the KASUMI will be widely used in third generation mobile communications systems as the 3rd Generation Partnership Project (3GPP) standard encryption algorithm. It is well known that the KASUMI is suitable for compact H/W implementations. We assume an implementation of a very compact KASUMI circuit is suitable for the future mobile devices with many additional functions.

Hence we aim to implement KASUMI H/W with the throughput of over 100 Mbps, which is enough speed for mobile devices. Also, we aim to implement it with a few Kgates. The H/W is one of the smallest H/W implementation for block ciphers.

A number of KASUMI ASIC and FPGA implementations have been studied [8]-[16]. In [8], [10]-[16], the implemented H/W is based on two types of H/W architectures; the pipeline architecture and the loop architecture. These papers aim to improve the processing speed and the H/W efficiency rather than to reduce the gate count. In [9], the implemented H/W is based on the loop architecture for designing the compact circuit. According to the logic synthesis on a 0.13-μm ASIC process, the gate size is 3.4 Kgates, which is the smallest at present. However, the paper did not optimize the size of temporary registers for storing intermediate data in the processing of KASUMI. In general, the 1-bit register has larger gate count than other 1-bit logic gates, such as AND, OR, XOR, and NAND. So, it is very significant for the compact KASUMI H/W to maximally reduce the total bit length of the registers.

In this paper, we focus on four strategies for the compact design. First, we choose to implement the H/W by using the half of the FI function. Secondly, extended keys are generated on-the-fly by using shift registers. Thirdly, we use S-boxes implemented in the combinational logic. Fourthly, we optimize the total bit length of the registers, as the main topic of this paper.

In [4], Yamamoto et al. proposed the method of reducing temporary registers for the MISTY1 FO function from 32 bits to 16 bits (YYI-08), and implemented a very compact H/W of MISTY1 [5]. KASUMI has a similarly structured MISTY1 FO function. In this paper, we discuss applying YYI-08 to KASUMI in order to reduce the bit length of the registers, and aim to design the smallest KASUMI H/W. In this process, we found some problems on the application, because of the following two differences between KASUMI and MISTY1. First, MISTY1 has the FL function outside of the F-function in Feistel network, while KASUMI has the FL function inside of the F-function. This difference causes the problem of the logic equivalence. Second, it takes 1 cycle for the common compact MISTY1 H/W to execute the FI function, while it takes 2 cycles for the common compact KASUMI H/W. This causes the problem that the additional temporary register is required to execute the FI function. We propose the new YYI-08 improved for KASUMI to solve these two problems. Also, we propose the implemented algorithm and the compact H/W architecture based on the YYI-08 improved for KASUMI.

We synthesize our KASUMI H/W by a 0.11-μm CMOS standard cell library, then an extremely small size of 2.99 Kgates with 110.3 Mbps throughput is obtained. Also, we synthesize KASUMI H/W by using XCV300E-8BG432 FPGA device from Xilinx, then a very small size of 332 slices with 44.54 Mbps throughput is obtained. Through the synthesis on both ASIC and FPGA platforms, our KASUMI H/W is the smallest, as far as we know.

The rest of the paper is organized as follows. Section 2 explains the algorithm of KASUMI. Section 3 explains the outline of YYI-08 for MISTY1. Our strategy

for the smallest KASUMI H/W is discussed in Section 4. We analyze the problems with applying YYI-08 to KASUMI in Section 5. Section 6 proposes the new YYI-08 improved for KASUMI. Section 7 proposes the compact KASUMI H/W architecture base on our proposal. Section 8 evaluates the performance of our KASUMI H/W on both ASIC and FPGA platforms. Finally, we conclude with a summary and comment on future directions in Section 9.

2 KASUMI

Figure 1 shows the entire structure of KASUMI excluding the key scheduler [1]. KASUMI encrypts a 64-bit plaintext using a 128-bit secret key. KASUMI has the 8-round Feistel network with the F-function including FO functions and FL functions connected in series. The $FO_i(1 \leq i \leq 8)$ function uses two 48-bit extended key, KI_i and KO_i. The $FL_i(1 \leq i \leq 8)$ function is used in the encryption and 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, $KO_{ij}(1 \leq j \leq 3)$ and $KI_{ij}(1 \leq j \leq 3)$ are the j-th (from left) 16 bits 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. The extended keys KO_i, KI_i, and KL_i are easily generated by rotating the 128-bit secret key and XORing with constant values.

3 YYI-08

In this paper, the FO function of MISTY1 is implemented based on the assumption that the FI function is executed in 1 cycle, and the FO function is executed in 3 cycles by repeatedly using one FI function module for the compact design. The FO function of MISTY1 differs from that of KASUMI in that the extended key KO_{i4} is XORed. This XOR operator is non-influential factor for the implemented algorithm of the FO function, so the following discussion excludes this XOR operator.

Figure 2 shows two methods of dividing an FO function into three cycles; the straightforward method and YYI-08. The data path in each cycle is illustrated by the thick line. In the straightforward method shown in Fig. 2(I), an FO function is separated horizontally for every cycle, so a 32-bit temporary register is required for left and right 16-bit data. REG_L and REG_R are two 32-bit data registers in which intermediate data is stored in the encryption process. The 32-bit output data from the FO function is XORed with data registers in Cycle3. In the YYI-08 shown in Fig. 2(II), an FO function is separated vertically for every cycle. YYI-08 differs from the straightforward method in that the output data from the FI function in Cycle2 is directly XORed with data registers, which

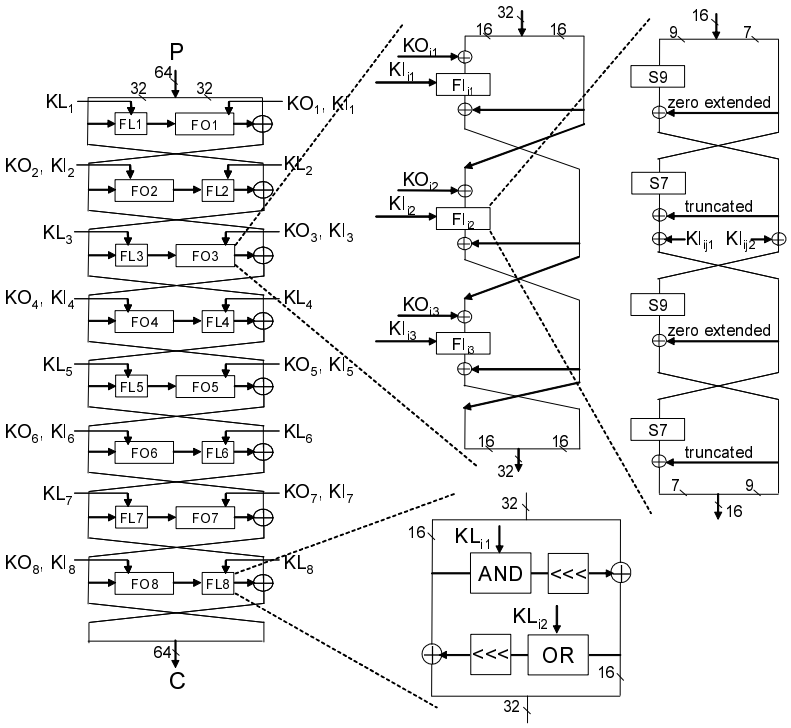


Fig. 1. KASUMI encryption algorithm

can reduce the 16-bit temporary register for the data, so only 16-bit temporary register is required.

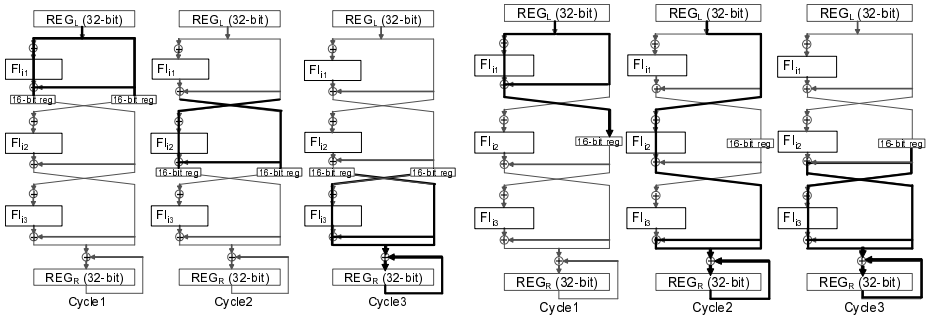
From the above analysis, we can see the following two conditions are necessary for cryptographic implementation algorithms in the application of YYI-08.

1. No logic gate exists between the FO function and data registers, and they are connected directly (To guarantee the logic equivalence when the output data from the FI function is partially XORed with data registers in Cycle2 and Cycle3 shown in Fig. 2(Ⅱ)).
2. The FI function is executed in 1 cycle.

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 KASUMI, so the FO/FI function is one of the most influential factors for gate counts of KASUMI H/W. Thus, it is important for the compact design to reduce the number of the implemented



(I) Straightforward method (II) YYI-08

Fig. 2. The implemented method for the FO function of MISTY1

FO/FI function module. The FI function of KASUMI can be divided into two "FI' function" shown in Fig. 3. So, we choose to implement only one FI' function module for the compact design. That is, the FO function is executed in six clock cycles by repeatedly using one FI' function module. This architecture is suitable for compact implementation.

4.2 Extended key generation method

We choose to generate the extended key by the on-the-fly method, in which a required extended key is generated in the encryption/decryption process sequentially. The on-the-fly method is suitable for the compact design because the register to store the extended key is not required. In general, KASUMI can easily generate the extended key by rotating the 128-bit secret key with shift registers. So, we choose to implement the key scheduler by using shift registers composed of eight 16-bit registers proposed in [9]. The key scheduler loads the 128-bit secret key into the shift registers from the 16-bit input port in 8 cycles, which can reduce selectors.

4.3 S-box Implementation method

The S-box performance of KASUMI, 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 KASUMI have been designed so that they can be easily implemented in combinational logic as well as by a lookup table [1]. On KASUMI H/W, 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. Therefore, we used S-boxes implemented in combinational logic based on the hand-optimized bit-level logic in [3].

4.4 Optimization of temporary registers

We aim to design the smallest KASUMI H/W, so maximally reduce the bit length of the temporary registers for the FO function because the register has larger gate count than other logic gates. The common compact H/W of KASUMI requires 32-bit temporary register for the FO function as well as that of MISTY1. KASUMI has a similarly structured MISTY1 FO function, so we consider it possible that YYI-08 is applied to KASUMI, and the temporary register is reduced from 32 bits to 16 bits. In Section 5, we discuss the application of YYI-08 to KASUMI.

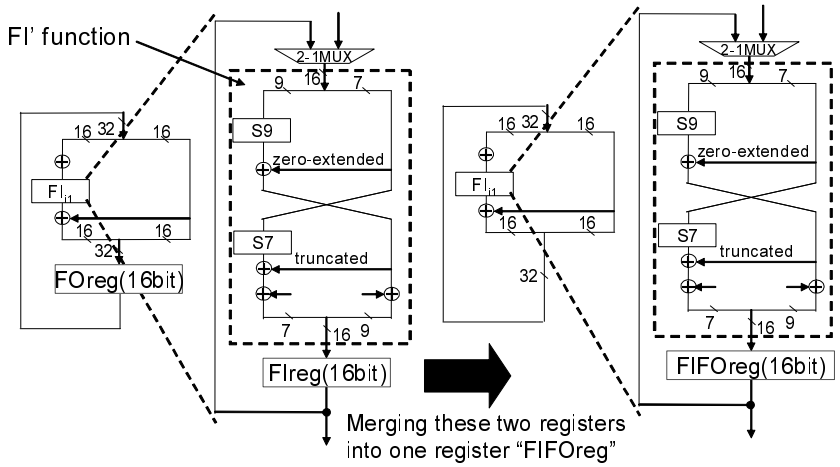


Fig. 3. Merging temporary registers

5 Problem in the application of YYI-08 to KASUMI

As described in Section 3, two conditions are necessary for cryptographic implementation algorithms in the application of YYI-08. However, the conditions are satisfied for the implementation algorithm of KASUMI discussed in Section 4. First, KASUMI has the FL function between the FO function and data registers, which causes a problem in the application. In order to apply YYI-08 to KASUMI and guarantee the logic equivalence, we must analyze the nature of FL function and improve the implementation algorithm of KASUMI. Second, it takes 2 cycles to execute the FI function by using FI' function for the compact KASUMI H/W, which causes the problem that the additional temporary register is required to execute the FI function. The details of these two problems are explained as follows.

5.1 Problem1: Property of the FL function

This Section explains the problem caused when KASUMI is implemented by using only 16-bit temporary register FIFOreg. We can see the cause of the problem by assuming following three implementation cases. Case(1) assumes to implement KASUMI with 32-bit temporary register straightforwardly, and Case(2) assumes to implement the modified version of KASUMI with 16-bit temporary register, and no problem occurs in these cases. But in Case(3), which assumes to implement the original version of KASUMI with 16-bit temporary register, the logical equivalence problem occurs. We can see the cause of the problem by remarking the gap between Case(2), which omits the FL function, and Case(3), which provides FL function inside of the F-function as original KASUMI.

Case(1) FIFOreg:32-bit, FL function is inside of the F-function.
Case(2) FIFOreg:16-bit, FL function is not inside of the F-function.
Case(3) FIFOreg:16-bit, FL function is inside of the F-function (Problem).

Figure 4 shows the logic in the even-numbered rounds where the FL function follows the FO function. The input data is loaded from 32-bit data register $REG_R = \{R_H||R_L\}$ (R_H, R_L: 16-bit). The output data obtained through the FO and FL functions are XORed with 32-bit data register $REG_L = \{L_H||L_L\}$ (L_H, L_L: 16-bit) as shown in the following equation.

$$(L_H||L_L) = (L_H||L_L) \oplus \mathrm{FL}(\underline{\mathrm{FI}(R_{H,2}, KO_{i,2}, KI_{i,2}) \oplus \mathrm{FI}(R_{H,1}, KO_{i,1}, KI_{i,1}) \oplus R_L||} \\ \underline{\mathrm{FI}(R_{H,3}, KO_{i,3}, KI_{i,3}) \oplus \mathrm{FI}(R_{H,2}, KO_{i,2}, KI_{i,2}) \oplus \mathrm{FI}(R_{H,1}, KO_{i,1}, KI_{i,1}) \oplus R_L)} \tag{1}$$

Here, let $\{R_{H,1}||R_{L,1}\}$, $\{R_{H,2}||R_{L,2}\}$, $\{R_{H,3}||R_{L,3}\}$ ($R_{H,1}$, $R_{L,1}$, $R_{H,2}$, $R_{L,2}$, $R_{H,3}$, $R_{L,3}$: 16-bit) be the data input into the first, second, and third steps of FO functions, respectively.

Case(1) In Case(1), the size of temporary registers is 32 bits, so the KASUMI H/W can be implemented straightforwardly without logic problems as follows. The output data from the FO function is stored into the 32-bit FIFOreg temporarily as shown in the following equation.

$$FIFOreg = \mathrm{FI}(R_{H,2}, KO_{i,2}, KI_{i,2}) \oplus \mathrm{FI}(R_{H,1}, KO_{i,1}, KI_{i,1}) \oplus R_L|| \\ \mathrm{FI}(R_{H,3}, KO_{i,3}, KI_{i,3}) \oplus \mathrm{FI}(R_{H,2}, KO_{i,2}, KI_{i,2}) \oplus \mathrm{FI}(R_{H,1}, KO_{i,1}, KI_{i,1}) \oplus R_L \tag{2}$$

Next, the 32-bit data stored into the FIFOreg are input into the FL function, and the output data from the FL function is XORed with data registers $\{L_H||L_L\}$ as shown in the following equation.

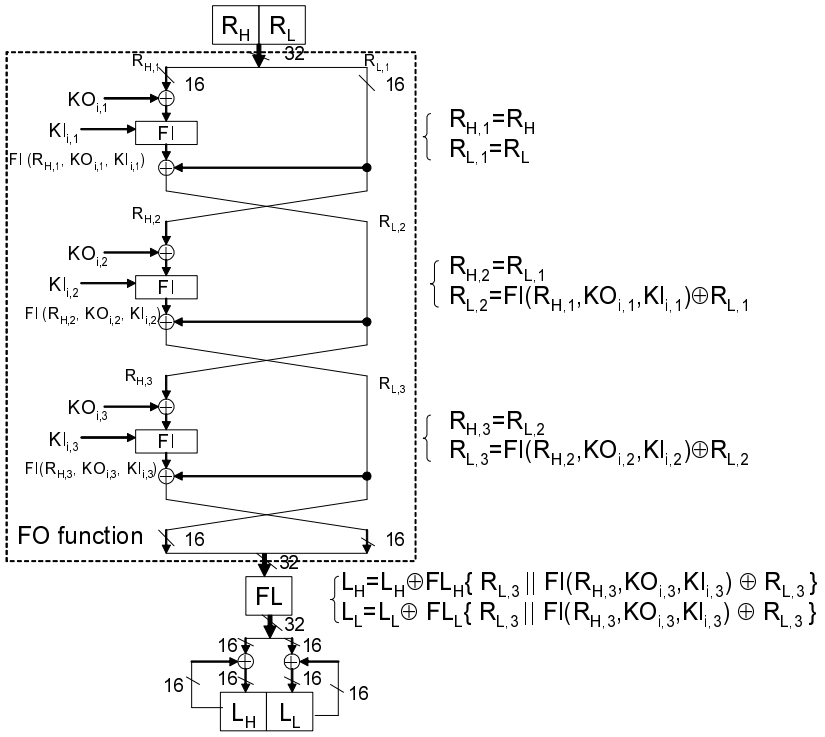


Fig. 4. The logic in the even-numbered rounds

$$(L_H||L_L) = (L_H||L_L) \oplus \mathrm{FL}(FIFOreg) \tag{3}$$

Case(2) The data shown in the underlined part of Eq. (1) can be divided into the data shown in the underlined part of the following equation.

$$\begin{aligned}
&(L_H||L_L) = (L_H||L_L) \oplus \underline{\mathrm{Sig1} \oplus \mathrm{Sig2} \oplus \mathrm{Sig3} \oplus \mathrm{Sig4}} \\
&\mathrm{Sig1} = (R_L||R_L) \\
&\mathrm{Sig2} = (\mathrm{FI}(R_L, KO_{i,2}, KI_{i,2})||\mathrm{FI}(R_L, KO_{i,2}, KI_{i,2})) \\
&\mathrm{Sig3} = (\mathrm{FI}(R_H, KO_{i,1}, KI_{i,1})||\mathrm{FI}(R_H, KO_{i,1}, KI_{i,1})) \\
&\mathrm{Sig4} = ((16'\mathrm{h0000\}})||\mathrm{FI}(R_{H,3}, KO_{i,3}, KI_{i,3}))
\end{aligned} \tag{4}$$

In Case(2), the size of temporary registers is 16 bits, so four 32-bit data from Sig1 to Sig4 cannot be stored as one 32-bit data, {Sig1 ⊕ Sig2 ⊕ Sig3 ⊕

Sig4}. Therefore, four data from Sig1 to Sig4 are partially XORed with data registers $\{L_H||L_L\}$. Each of the 32-bit data from Sig1 to Sig4 is generated by bit concatenating the 16-bit data stored in the 16-bit FIFOreg, so the KASUMI H/W can be implemented with the 16-bit FIFOreg.

Case(3) In Case(3) where the FL function is inside of the F-function and the bit length of temporary registers is 16 bits, the logic is shown as the following equation.

$$(L_H||L_L) = (L_H||L_L) \oplus \mathrm{FL}(\mathrm{Sig1} \oplus \mathrm{Sig2} \oplus \mathrm{Sig3} \oplus \mathrm{Sig4}) \tag{5}$$

We cannot calculate the 32-bit data FL(Sig1 $\oplus$ Sig2 $\oplus$ Sig3 $\oplus$ Sig4) by using the FL function after calculating {Sig1 $\oplus$ Sig2 $\oplus$ Sig3 $\oplus$ Sig4}. This is because the calculation of {Sig1 $\oplus$ Sig2 $\oplus$ Sig3 $\oplus$ Sig4} requires multiple cycles due to using one FI' function, so the 32-bit temporary register is necessary to store the 32-bit data. Therefore, we consider the following idea to use only 16-bit temporary register. In each cycle, four data from FL(Sig1) to FL(Sig4) are calculated by using only 16-bit temporary register, and the four data are partially XORed with data registers $\{L_H||L_L\}$ as shown in the following equation.

$$(L_H||L_L) = (L_H||L_L) \oplus \mathrm{FL}(\mathrm{Sig1}) \oplus \mathrm{FL}(\mathrm{Sig2}) \oplus \mathrm{FL}(\mathrm{Sig3}) \oplus \mathrm{FL}(\mathrm{Sig4}) \tag{6}$$

For the logic equivalence between Eq. (5) and Eq. (6), the following equation is required to be satisfied.

$$\mathrm{FL}(\mathrm{Sig1} \oplus \mathrm{Sig2} \oplus \mathrm{Sig3} \oplus \mathrm{Sig4}) = \mathrm{FL}(\mathrm{Sig1}) \oplus \mathrm{FL}(\mathrm{Sig2}) \oplus \mathrm{FL}(\mathrm{Sig3}) \oplus \mathrm{FL}(\mathrm{Sig4}) \tag{7}$$

In general, the equivalence in Eq. (7) is called linearity, which means that any two integer numbers, X and Y, satisfy $\mathrm{FL}(X \oplus Y) = \mathrm{FL}(X) \oplus \mathrm{FL}(Y)$. However, this equation and Eq. (7) does not hold because of the property of the FL function. Therefore, the logic equivalence cannot be guaranteed. Next Section proposes our methods to solve these problems.

5.2 Problem2: Register competition

The FI function is executed in 2 cycles because only one FI' function module is implemented. Therefore, the 16-bit temporary register for FI function (i.e., is "FIreg") is necessary to store the intermediate data in Cycle1. Also, 16-bit temporary register for FO function (i.e., is "FOreg") is required due to the application of YYI-08 to KASUMI. If these two registers are implemented independently, then the total size of temporary registers is 32 bits as shown in the left side of Fig. 3. That is, the advantage of YYI-08, the temporary register is

reduced from 32 bits to 16 bits, is lost. So, we aim to integrate 16-bit FIreg and 16-bit FOreg into one 16-bit register (i.e., is "FIFOreg") as shown in the right side of Fig. 3.

However, only 16-bit temporary register causes the following problem. Figure 5 shows the data path in Cycle1, Cycle2, and Cycle3. By the process of the Cycle1 and Cycle2, the output data of the FI_{i1} is stored into the 16-bit register. But in Cycle3, intermediate data of the FI_{i2} must be stored into the 16-bit register with holding the previous 16-bit data of the FI_{i1}. Therefore, two 16-bit temporary registers are required, KASUMI cannot be implemented by using only 16-bit temporary register FIFOreg when YYI-08 is applied to KASUMI straightforwardly.

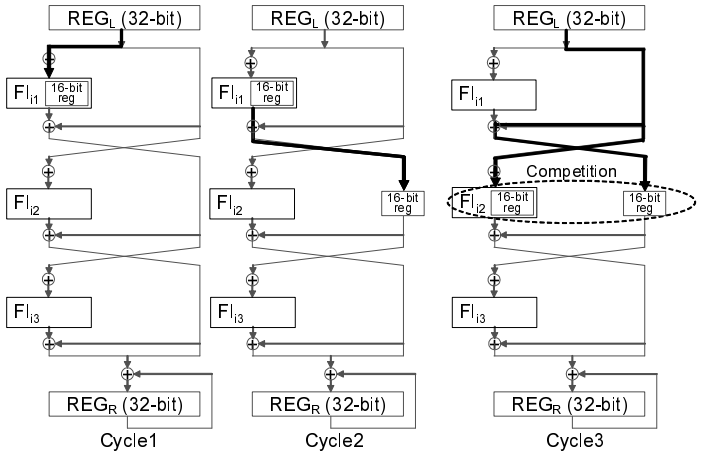


Fig. 5. YYI-08 straightforwardly applied to KASUMI

6 Proposed methods

6.1 Solution1:Correction for FL function

We solve the problem of the FL function described in Section 5.1. The FL function does not satisfy $\mathrm{FL}(X \oplus Y) = \mathrm{FL}(X) \oplus \mathrm{FL}(Y)$. Consequently, in our own analysis, we show that the FL function satisfies:

$$\mathrm{FL}(X \oplus Y) = \mathrm{FL}(X) \oplus \mathrm{FL}(Y) \oplus \{(KL_{i,2} \lll 1)||16'\mathrm{h0000}\}. \tag{8}$$

The following equation is also hold.

$$\mathrm{FL}(X_1 \oplus \ldots \oplus X_n) = \begin{cases} (n \text{ is an odd number.}) \\ \mathrm{FL}(X_1) \oplus \ldots \oplus \mathrm{FL}(X_n) \\ (n \text{ is an even number.}) \\ \mathrm{FL}(X_1) \oplus \ldots \oplus \mathrm{FL}(X_n) \oplus \{(KL_{i,2} \lll 1)||(16'\mathrm{h0000})\} \end{cases} \tag{9}$$

Here, let $X_k (k = 1, 2, \ldots, n)$ and $\mathrm{FL}(X_k)$ be the input data into FL function and the output data from FL function. In Eq. (9), when the number of the data input into FL function n is even, the correction data, $\{(KL_{i,2} \lll 1)||16'\mathrm{h0000}\}$, are XORed with data registers. This correction based on our own analysis can guarantee the logic equivalence. Therefore, the output data from FO function can be input into FL function in multiple cycles, so the KASUMI H/W can be implemented with only 16-bit temporary register FIFOreg.

6.2 Solution2:YYI-08 improved for KASUMI

We solve the problem described in Section 5.2 by reference to Fig. 6. We proposed new YYI-08 improved for KASUMI, in which the second FI function FI_{i2} is executed primarily, and then the first FI function FI_{i1} is executed. Our new YYI-08 can avoid the register competition because it is unnecessary to hold the 16-bit output data from the FI_{i1} during processing the FI_{i2} if the second FI function FI_{i2} is executed primarily. To avoid the register competition by our new YYI-08 can lead to merging FIreg and FOreg into FIFOreg.

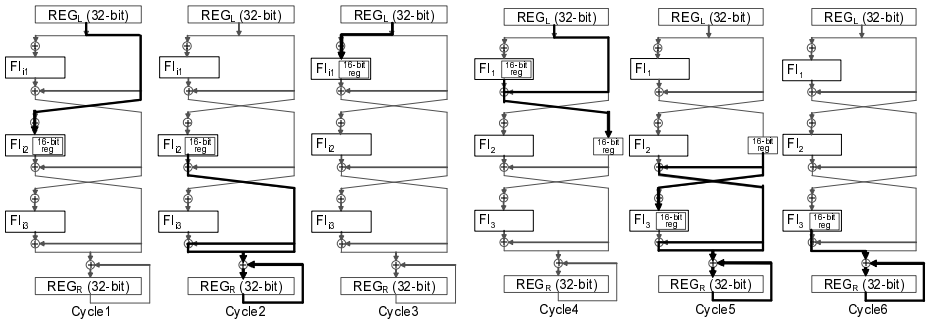


Fig. 6. YYI-08 improved for KASUMI

7 Proposed architecture

This Section proposes new implementation algorithm of KASUMI H/W based on our proposed method described in Section 6. First, Alg.1 shows the implemented algorithm in odd-numbered rounds where the FO function follows the FL function. We assume that the FI function is executed in 2 cycles by using

only one FI' function module, so let FI′() and FI() be the intermediate data in Cycle1 and the output data in Cycle2, respectively. Also, "<=" and "=" mean non-blocking and blocking assignments, respectively. Let $\mathrm{FL_H}()$ and $\mathrm{FL_L}()$ be the upper and the lower 16-bit data of FL(), respectively. In Cycle1, 3, and 5, the output data from the FL function module is input into FI' function module at the same cycle. This means that the FL function module is directly connected to the FI' function module. In Cycle2, 4, and 6, the output data from FI' function module is XORed with data register $\{R_H||R_L\}$ at the same cycle.

Next, Alg.2 shows the implemented algorithm in even-numbered rounds. Here, the output data are XORed with data register in Cycle3, 5, and 7, it takes one more cycle in comparison to odd-numbered rounds. This is because the temporary register FIFOreg exists on the data path from FI' to FL. This can avoid a feedback loop structure, and an increase in area size due to two implemented FL function module. In Alg.2, the output data are XORed with data register even number of times (Cycle3, 5, 6, and 7), so the correction data, $\{(KL_{i,2} \lll 1)||16'\mathrm{h0000}\}$, are XORed with data registers in Cycle4. Note that the logic equivalence can be guaranteed by the corrective operation in any cycle including Cycle4.

Alg. 1. Odd-numbered rounds

Cycle1 : FIFOreg <= $\mathrm{FI'_{i,2}}(\mathrm{FL_L}(\{L_H||L_L\}), KO_{i,2}, KI_{i,2})$
Cycle2 : FIFOsig = $\mathrm{FI_{i,2}}(\mathrm{FL_L}(\{L_H||L_L\}), KO_{i,2}, KI_{i,2})$
Cycle2 : $\{R_H||R_L\}$ <= $\{R_H||R_L\} \oplus \{\mathrm{FIFOsig}||\mathrm{FIFOsig}\}$
Cycle3 : FIFOreg <= $\mathrm{FI'_{i,1}}(\mathrm{FL_H}(\{L_H||L_L\}), KO_{i,1}, KI_{i,1})$
Cycle4 : FIFOsig = $\mathrm{FI_{i,1}}(\mathrm{FL_H}(\{L_H||L_L\}), KO_{i,1}, KI_{i,1}) \oplus \mathrm{FL_L}(\{L_H||L_L\})$
Cycle4 : FIFOreg <= FIFOsig
Cycle5 : $\{R_H||R_L\}$ <= $\{R_H||R_L\} \oplus \{\mathrm{FIFOreg}||\mathrm{FIFOreg}\}$
Cycle5 : FIFOreg <= $\mathrm{FI'_{i,3}}(\mathrm{FIFOreg}, KO_{i,3}, KI_{i,3})$
Cycle6 : FIFOsig = $\mathrm{FI_{i,3}}(\mathrm{FIFOreg}, KO_{i,3}, KI_{i,3})$
Cycle6 : $\{R_H||R_L\}$ <= $\{R_H||R_L\} \oplus \{16'\mathrm{h0000}||\mathrm{FIFOsig}\}$

Alg. 2. Even-numbered rounds

Cycle1 : FIFOreg <= $\mathrm{FI'_{i,2}}(R_L, KO_{i,2}, KI_{i,2})$
Cycle2 : FIFOreg <= $\mathrm{FI_{i,2}}(R_L, KO_{i,2}, KI_{i,2})$
Cycle3 : $\{L_H||L_L\}$ <= $\{L_H||L_L\} \oplus \mathrm{FL}(\{\mathrm{FIFOreg}||\mathrm{FIFOreg}\})$
Cycle3 : FIFOreg <= $\mathrm{FI'_{i,1}}(R_H, KO_{i,1}, KI_{i,1})$
Cycle4 : $\{L_H||L_L\}$ <= $\{L_H||L_L\} \oplus \{(KL_{i,2} \lll 1)||16'\mathrm{h0000}\}$
Cycle4 : FIFOreg <= $\mathrm{FI_{i,1}}(R_H, KO_{i,1}, KI_{i,1})$
Cycle5 : $\{L_H||L_L\}$ <= $\{L_H||L_L\} \oplus \mathrm{FL}(\{\mathrm{FIFOreg}||\mathrm{FIFOreg}\})$
Cycle5 : FIFOreg <= $\mathrm{FI'_{i,3}}(\mathrm{FIFOreg} \oplus R_L, KO_{i,3}, KI_{i,3})$
Cycle6 : $\{L_H||L_L\}$ <= $\{L_H||L_L\} \oplus \mathrm{FL}(\{R_L||R_L\})$
Cycle6 : FIFOreg <= $\mathrm{FI_{i,3}}(\mathrm{FIFOreg} \oplus R_L, KO_{i,3}, KI_{i,3})$
Cycle7 : $\{L_H||L_L\}$ <= $\{L_H||L_L\} \oplus (\{16'\mathrm{h0000}||\mathrm{FIFOreg}\})$

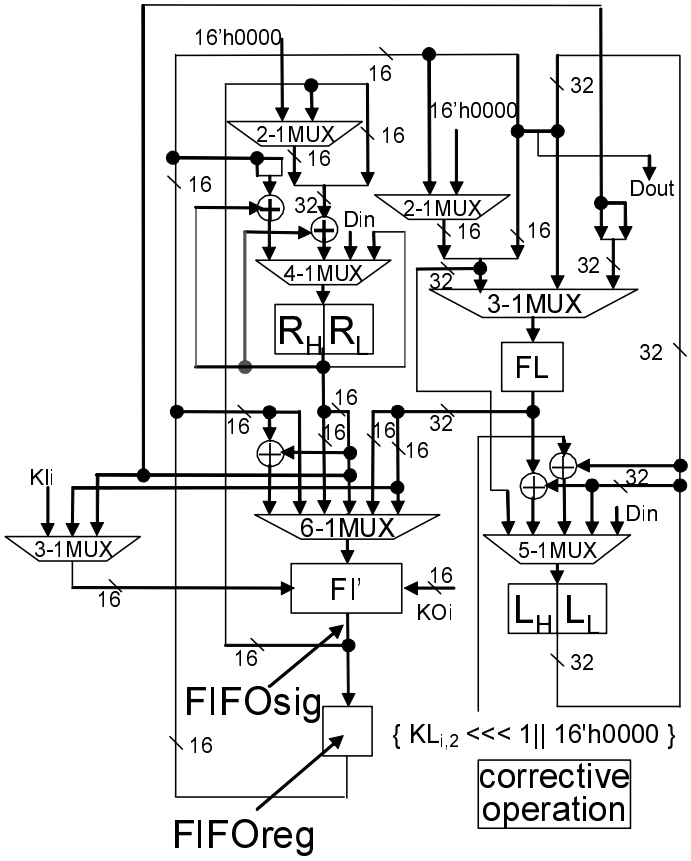


Fig. 7. Proposed KASUMI H/W architecture

Figure 7 shows the proposed architecture of the KASUMI H/W based on Alg. 1 and Alg. 2. Our proposed method can implement the KASUMI H/W with only 16-bit temporary register FIFOreg. The implemented H/W generates a 64-bit ciphertext (plaintext) from a 64-bit plaintext (ciphertext) in 54 clock cycles. The data input and output requires 2 cycles. The odd-numbered rounds requires 6 (cycles) $\times$ 4 (rounds) = 24 cycles, and the even-numbered rounds requires 7 (cycles) $\times$ 4 (rounds) = 28 cycles.

8 Performance evaluation

This Section evaluates the ASIC and FPGA performance of our KASUMI H/W. The KASUMI H/W is verified by using the test vectors provided by the NESSIE submission package [17]. The proposed H/W is not based on scan design, and is synthesized on a 0.11-μm CMOS standard cell library with the Design Compiler

2006.06-SP5-1 under the worst case conditions, and with size optimization and ungroup command. Also, one gate is equivalent to 2-1NAND gate.

Table 1 shows the ASIC performance of our proposed KASUMI H/W. The proposed H/W realized the smallest-area of less than 3K gates. We estimate the effect of our proposal on the gate counts. First, we focus on the registers. The straightforward H/W uses the 32-bit temporary register, while our KASUMI H/W used only 16-bit one due to our proposal. We suppose 1-bit register = 13.5 NAND gates, so our proposal can reduce about 216 NAND gates. Next, we focus on the multiplexers (MUXs). We show that our proposal do not increase MUXs in the following discussion. MUXs which have N-bit length and M input signals equal $(N/16)(M-1)$ 16-bit 2-1MUXs. So, we evaluate the gate count based on the numbers of 16-bit 2-1MUXs. We compare our KASUMI H/W with the previous smallest H/W in [9]. The H/W in [9] uses 26 16-bit 2-1MUXs. Note that we add six 16-bit 2-1MUXs to the original H/W in [9]. This is because the original H/W in [9] omits the MUXs into which the data output from registers are directly input. These MUXs are necessary to keep the value of registers. Meanwhile, our H/W in Fig. 7 uses less than 26 16-bit 2-1MUXs. Our H/W uses two 2-1MUXs where one-sided input data become 0, so they can be transformed into an AND gates smaller than MUX gates. Hence we estimate less than 26 16-bit 2-1MUXs, while 27 16-bit 2-1MUXs are used in our H/W shown in Fig. 7. Also, the FIFOreg in Fig. 7 is not connected to the MUX, because our proposal can always input the value of FIFOsig into the FIFOreg. As the result, our KASUMI H/W can reduce the gate counts of registers and MUXs compared with the previous smallest H/W in [9].

Next, "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. The throughput and efficiency of the proposed KASUMI H/W are comparatively lower because the maximum frequency of ours is smaller than that of the other reports. We discuss the reason by comparison with the KASUMI H/W in [9]. The critical path of the KASUMI H/W in [9] is shorter than that of ours. This is because the temporary register exists on the data path between FI' and FL in the KASUMI H/W in [9], while the FL function module is directly connected to the FI' function module in our KASUMI H/W. The difference is due to only 16-bit temporary register by the application of YYI-08 to KASUMI. Our proposed KASUMI H/W can achieve throughput of over 100 Mbps, so has enough performance for embedded systems, such as 3GPP mobile phones. This paper aims to implement the smallest H/W of KASUMI, so it is significant to maximally reduce the gate count even though the throughput and the H/W efficiency are not the highest.

For further comparison with eliminating the CMOS technology difference, we evaluate the FPGA performance of our KASUMI H/W as shown in Tab.2. A lot of previous works choose the same FPGA chip (Xilinx vertex-E series, XCV300E-8BG432) [18] for fair evaluation. We synthesize the proposed H/W by using XCV300E-8BG432 FPGA device with Xilinx ISE 10.1 with size optimization,

Table 1. H/W performance comparison in ASICs

Source	Process [μm]	Functions	Cycle	MaxFreq. [MHz]	Thr'put [Mbps]	Area [Kgates]	Efficiency [Kbps/gates]
This work	0.11	Enc.+Dec.	54	97.6	110.3	2.99	36.9
[6]	0.18	N/A	N/A	N/A	130	6.5	20
[7]	0.09	Enc.+Dec.	N/A	226	850	9	94.4
[8]	0.25	Enc.	8	90.42	723.37	15.697	46.08
[8]	0.25	Enc.	1	94.05	5786.94	47.660	126.29
[9]	0.13	Enc.+Dec.	56	251.89	287.87	3.437	83.76

Table 2. H/W performance comparison in FPGAs

Source	Functions	Cycle	MaxFreq. [MHz]	Thr'put [Mbps]	Area [Slice]	Efficiency [Kbps/Slice]	Device
This work	Enc.+Dec.	54	31.928	36.09	332	108.70	300E-6BG432
This work	Enc.+Dec.	54	39.403	44.54	332	134.15	300E-8BG432
[8]	Enc.	8	20.88	167.04	1287	129.79	300E-6BG432
[9]	Enc.+Dec.	56	58.14	66.45	435	152.75	300E-6BG432
[9]	Enc.+Dec.	56	68.13	77.86	435	179.00	300E-8BG432
[10]	N/A	8	20.00	110.00	650	169.23	Virtex-E
[11]	Enc.	16	41.14	165	488	338.11	300E-8BG432
[12]	Enc.	12	41.625	222	566	392.22	300E-8BG432
[13]	Enc.+Dec.	8	71	568	1174	483.81	N/A
[14]	Enc.+Dec.	8	54.00	432	3452	125.14	300E-8BG432
[15]	Enc.	16	79.453	318	625	508.80	300E-8BG432
[16]	Enc.	16	96.33	385.32	448	860.08	Virtex-II

then a very small size of 332 slices with 44.54 Mbps throughput is obtained. This is the smallest KASUMI H/W, as far as we know.

9 Conclusion

We presented the very small KASUMI H/W, which used only 16-bit register due to the application of YYI-08 to KASUMI. We found that the application required the solution for the following two problems, and proposed our methods of solving them. First, the FL function did not unconditionally satisfy the linearity, which is essential for the application of YYI-08. We showed that the FL function could satisfy the linearity with correction. Second, the KASUMI H/W takes 2 cycles to execute the FI function, which requires the additional register, so we proposed new YYI-08 improved for KASUMI, in which the order of execution is changed. The implemented KASUMI H/W based on our proposal was synthesized by a 0.11-μm CMOS standard cell library, then an extremely small size of 2.99 Kgates with 110.3 Mbps throughput was obtained. This is the smallest

KASUMI H/W, as far as we know. Future work will include discussion on the fastest KASUMI H/W implementation.

References

1. Third Generation Partnership Project: 3GPP TS 35.202 v7.0.0 Document 2: KASUMI Specification. Jun. 2007.
2. GSM Association, Market Data Summary (Q2 2009), http://www.gsmworld.com/newsroom/market-data/market_data_summary.htm
3. Matsui, M., Nakajima, J.: On the Power of Bitslice Implementation on Intel Core2 Processor. In: Proc. CHES 2007, LNCS 4727, pp.121–134, 2007.
4. Yamamoto, D., Yajima, J., Itoh, K.: A Very Compact Hardware Implementation of the MISTY1 Block Cipher. In: Proc. CHES 2008, LNCS 5154, pp.315–330, 2008.
5. Matsui, M.: New Block Encryption Algorithm MISTY. In: Proc. FSE 1997, LNCS 1267, pp.54–68, 1997.
6. Mitsubishi Electric Web Site, http://global.mitsubishielectric.com/bu/security/rd/rd05/kasumi_b.html
7. Elliptic Semiconductor Web Site, http://www.ellipticsemi.com/pdf/CLP-38_80102.pdf
8. Marinis, K., Moshopoulos, NK., Karoubalis, F., Pekmestzi, KZ.: On the Hardware Implementation of the 3GPP Confidentiality and Integrity Algorithms. In: Proc. ISC 2001, LNCS 2200, pp.248–265, 2001.
9. Satoh, A., Morioka, S.: Small and High-Speed Hardware Architectures for the 3GPP Standard Cipher KASUMI. In: Proc. ISC 2002, LNCS 2433, pp.48–62, 2002.
10. Kim, H., Choi, Y., Kim, M., Ryu, H.: Hardware implementation of the 3GPP KASUMI crypto algorithm. In: Proc. ITC-CSCC-2002, pp.317–320, 2002.
11. Balderas, T., Cumplido, R.: An Efficient Hardware Implementation of the KASUMI Block Cipher for Third Generation Cellular Networks. In: Proc. GSPx2004, 2004.
12. Balderas, T., Cumplido, R.: An Efficient Reuse-Based Approach to Implement the 3GPP KASUMI Block Cipher. In: Proc. ICEEE 2004, pp.113–118, 2004.
13. Kim, H., Lee, S.: Design and Implementation of a Private and Public Key Crypto Processor and Its Application to a Security System. IEEE Transactions on Consumer Electronics, Vol.50, No.1, pp.214–224, 2004.
14. Kitsos, P., Galanis, MD., Koufopavlou, O.: High-speed hardware implementations of the KASUMI block cipher. In: Proc. ISCAS 2004, Vol.2, pp.549–552, 2004.
15. Balderas, T., Cumplido, R.: "High Performance Encryption Cores for 3G Networks," In: Proc. DAC2005, pp.240–243, 2005.
16. Balderas, T., Cumplido, R., Feregrino-Uribe, C.: On the design and implementation of a RISC processor extension for the KASUMI encryption algorithm. Computers and Electrical Engineering, Vol.34, Issue.6, pp.531–546, 2008.
17. NESSIE: New European Schemes for Signatures, Integrity, and Encryption, https://www.cosic.esat.kuleuven.be/nessie/
18. Virtex-E 1.8V FPGA Complete Data Sheet, http://japan.xilinx.com/support/documentation/data_sheets/ds022.pdf