Implementation of Lightweight Encryption Algorithm LEA

Mi-Ji Sung[1], Gi-Chur Bae, Kyung-Wook Shin[a]
School of Electronic Engineering, Kumoh National Institute of Technology
E-mail : smj920307@kumoh.ac.kr[1]

***Abstract* – This paper describes a hardware implementation of the lightweight block cipher algorithm LEA. The LEA crypto-processor designed in this paper supports three key lengths of 128, 192, and 256 bits. To achieve area-efficient and low-power implementation, both round block and key scheduler are optimized to share hardware resources used for encryption and decryption as well as for key scheduling. In addition, a parallel register structure with a novel key scheduling scheme is devised to reduce clock cycles for key scheduling, which results in an increase of performance by 20~30%. The LEA crypto-processor implemented with 21,000 gates was fabricated with a 0.18-um CMOS process. Test results show that all of the 22 chips tested functions correctly, resulting in 100% yield.**

I. INTRODUCTION

The IoT technology enables various devices to be connected to internet, and to share information with each other without the intervention of human beings. It has been widely applied to various fields such as smart home, smart security, intelligent transportation information systems, and etc. Because the IoT system processes data in various types through complex and heterogeneous networks, it can be exposed to the risks of information security including information spill, fabrication, and falsification. The IoT has security-related issues similar to sensor network and Internet. The IoT security-related issues include privacy, access control, authentication, data storage, and management [1, 2].

A representative technology used for information security is cryptography. The cryptography is a word of Greek origin. It means not only secret writing but also technology that transforms information to be safe from various types of security attacks. The cryptography is used as a mean to protect information from security attacks, such as cases when someone tries to intercept information stored in computer or transferred via networks, to expose its contents, or to fabricate it intentionally. Security is considered as one of the core technologies for IoT system.

For the security of IoT system, both symmetric key cryptography and public key cryptography can be used. Since most of the IoT networks and devices including sensor network, RFID tag, smart card have limited hardware and software resources, the cryptographic algorithms that consume low-power and small hardware/software resource are required [3]. Recently, various lightweight block cipher algorithms for IoT security have been proposed, including HIGHT (HIGh security and lightweigHT) [4], LEA (Lightweight Encryption Algorithm) [5], CLEFIA [6], PRESENT [7], mCrypton [8], and TEA (Tiny Encryption Algorithm) [9]. The cryptographic algorithm can be implemented either in software or in hardware. The dedicated hardware implementation is used for those systems where physical safety and low-power consumption are important. Recently, some cases of low-power and small-area hardware implementation for the security of IoT, smart card, and NFC have been announced [10, 11].

In this paper, we designed a LEA crypto-processor for IoT applications, which supports three key lengths of 128, 192, 256 bits. The LEA crypto-processor was verified by FPGA implementation, and was fabricated as a test chip. After the LEA block cipher algorithm will be briefly explained in section II, the design of the LEA crypto-processor will be discussed in section III. Some functional verification and hardware implementation results are described in section IV, and conclusion will be followed in section V.

II. LIGHTWEIGHT BLOCK CIPHER ALGORITHM LEA [5]

The LEA is a symmetric key block cipher developed by NSRI (National Security Research Institute), which has a block size of 128 bits. The algorithmic structure of the LEA is similar to Feistel structure, and round function is based on ARX (Addition, Rotation, and XOR) operations. It is known that the ARX operations of 32 bits in round function make it suitable for software platform. Since the LEA does not use nonlinear substitution tables (S-box), it is also well suited to hardware implementation.

Encryption and decryption processes of the LEA are depicted in Figure 1, which consist of round key scheduling and round functions. The encryption (or decryption) is carried out by consecutive operations of 24/28/32 rounds depending on the key length of 128/192/256 bits. Round keys of 192 bits to be used in round transformation are expanded from master key of 128/192/256 bits.

a. Corresponding author; kwshin@kumoh.ac.kr

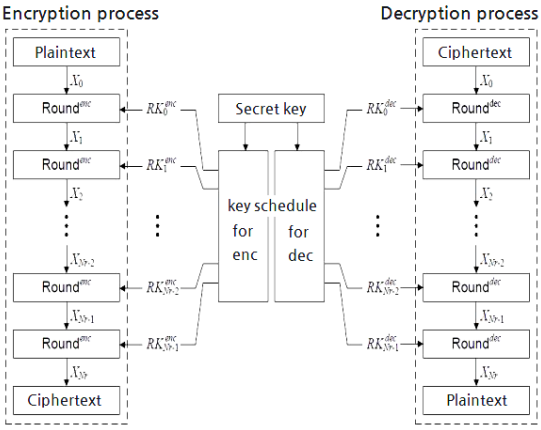


Fig. 1. Block cipher LEA

Decryption is the reverse process of encryption, which requires reversed key scheduling and inverse round functions. Modulo additions are used in encryption process, but modulo subtractions are used in decryption. The rotation operations for encryption and decryption are carried out in opposite directions.

A pseudo code for encryption process is shown in figure 2(a). Encryption process consists of key scheduling and encryption function. The encryption key scheduling $Keyschdule_i^{enc}$ generates round keys RK_i^{enc} $(0 \le i \le Nr-1)$ of 192 bits for Nr encryption rounds. The encryption function $Encrypt$ transforms a plaintext P of 128 bits to a ciphertext C of 128 bits using round keys RK_i^{enc} and round function $Round^{enc}$. Figure 2(b) shows a pseudo code for decryption process. Decryption process consists of key scheduling and decryption function. The decryption key scheduling $Keyschdule_i^{dec}$ generates round keys RK_i^{dec} $(0 \le i \le Nr-1)$ of 192 bits for Nr decryption rounds. The decryption function $Decrypt$ is the inverse function of $Encrypt$, which transforms a ciphertext C to a plaintext P using round keys RK_i^{dec} and round function $Round^{dec}$.

Encryption function : $C \leftarrow \text{Encrypt}(P, RK_0^{enc}, RK_1^{enc}, \cdots, RK_{Nr-1}^{enc})$
Input : 128-bit plaintext P, 192-bit round key $RK_0^{enc}, RK_1^{enc}, \cdots, RK_{Nr-1}^{enc}$
Output : 128-bit ciphertext C
1: $X_0 \leftarrow P$
2: for i = 0 to Nr-1 do
3: $X_{i+1} \leftarrow \text{Round}^{enc}(X_i, RK_i^{enc})$
4: end for
5: $C \leftarrow X_{Nr}$

(a)

Decryption function : $P \leftarrow \text{Decrypt}(C, RK_0^{dec}, RK_1^{dec}, \cdots, RK_{Nr-1}^{dec})$
Input : 128-bit ciphertext C, 192-bit round key $RK_0^{dec}, RK_1^{dec}, \cdots, RK_{Nr-1}^{dec}$
Output : 128-bit plaintext P
1: $X_0 \leftarrow C$
2: for i = 0 to Nr-1 do
3: $X_{i+1} \leftarrow \text{Round}^{dec}(X_i, RK_i^{dec})$
4: end for
5: $P \leftarrow X_{Nr}$

(b)

Fig. 2. Pseudo code for encryption and decryption of LEA (a) Encryption (b) Decryption

The i-th $(0 \le i \le Nr-1)$ round for encryption has operations as shown in figure 3(a). The state value $X_{i+1} = (X_{i+1}[0], X_{i+1}[1], X_{i+1}[2], X_{i+1}[3])$ is obtained by XOR operation of state value $X_i = (X_i[0], X_i[1], X_i[2], X_i[3])$ with round key $RK_i^{enc} = (RK_i^{enc}[0], RK_i^{enc}[1], RK_i^{enc}[2], RK_i^{enc}[3], RK_i^{enc}[4], RK_i^{enc}[5])$, modulo addition in 32 bits, bit rotation (ROL_9, ROR_5, ROR_3), and rotation in 32 bits. On the other hand, the i-th round for decryption process has operations as depicted in figure 3(b). The state values $X_{i+1} = (X_{i+1}[0], X_{i+1}[1], X_{i+1}[2], X_{i+1}[3])$ is obtained by XOR operation of state value $X_i = (X_i[0], X_i[1], X_i[2], X_i[3])$ with round key $RK_i^{dec}(RK_i^{dec}[0], RK_i^{dec}[1], RK_i^{dec}[2], RK_i^{dec}[3], RK_i^{dec}[4], RK_i^{dec}[5])$, modulo subtraction in 32 bits, bit rotation (ROR_9, ROL_5, ROL_3) and rotation in 32 bits. The key scheduling consists of AR (Addition, Rotation) operations and constant value generation that generates constant values for modulo operation. The number of constants generated depends on master key length. The AR operation is applied in opposite order depending on encryption or decryption.

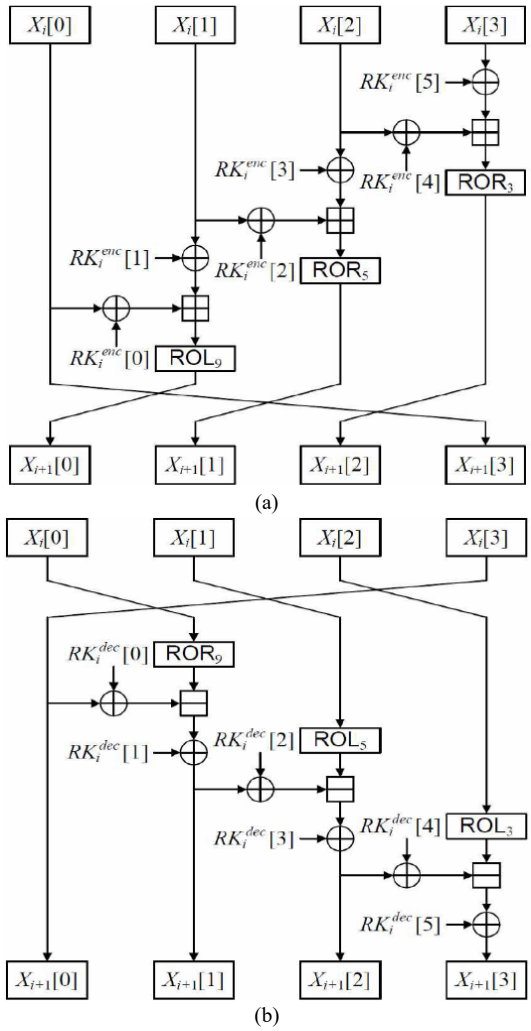


Fig. 3. Round functions for LEA encryption and decryption (a) Encryption (b) Decryption

Figure 4 shows pseudo code for the key scheduling of encryption process. The encryption key scheduling generates round keys of 192 bits RK_i^{enc}. It performs

modulo addition of master key with constant value in 32 bits, and bit rotation. The round keys for decryption process are generated in the same way as that of encryption except for the relationship of $RK_i^{dec} = RK_{Nr-i-1}^{enc}$.

Key schedule function for encryption of LEA-128 : $(RK_0^{enc}, \cdots, RK_{23}^{enc}) \leftarrow \text{KeySchedule}_{128}^{enc}(K)$

Input : 128-bit secret key K

Output : 192-bit encryption round key $RK_i^{enc}(0 \le i \le 23)$

$T \leftarrow K$
for i = 0 to 23 do
  $T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]))$
  $T[1] \leftarrow ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]))$
  $T[2] \leftarrow ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]))$
  $T[3] \leftarrow ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]))$
  $RK_i^{enc} \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1])$
end for

(a) LEA-128

Key schedule function for encryption of LEA-192 : $(RK_0^{enc}, \cdots, RK_{27}^{enc}) \leftarrow \text{KeySchedule}_{192}^{enc}(K)$

Input : 192-bit secret key K

Output : 192-bit encryption round key $RK_i^{enc}(0 \le i \le 27)$

$T \leftarrow K$
for i = 0 to 27 do
  $T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 6]))$
  $T[1] \leftarrow ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 6]))$
  $T[2] \leftarrow ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 6]))$
  $T[3] \leftarrow ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 6]))$
  $T[4] \leftarrow ROL_{13}(T[4] \boxplus ROL_{i+4}(\delta[i \bmod 6]))$
  $T[5] \leftarrow ROL_{17}(T[5] \boxplus ROL_{i+5}(\delta[i \bmod 6]))$
  $RK_i^{enc} \leftarrow (T[0], T[1], T[2], T[3], T[4], T[5])$
end for

(b) LEA-192

Key schedule function for encryption of LEA-256 : $(RK_0^{enc}, \cdots, RK_{31}^{enc}) \leftarrow \text{KeySchedule}_{256}^{enc}(K)$

Input : 256-bit secret key K

Output : 192-bit encryption round key $RK_i^{enc}(0 \le i \le 31)$

$T \leftarrow K$
for i = 0 to 31 do
  $T[6i \bmod 8] \leftarrow ROL_1(T[6i \bmod 8] \boxplus ROL_i(\delta[i \bmod 8]))$
  $T[6i+1 \bmod 8] \leftarrow ROL_3(T[6i+1 \bmod 8] \boxplus ROL_{i+1}(\delta[i \bmod 8]))$
  $T[6i+2 \bmod 8] \leftarrow ROL_6(T[6i+2 \bmod 8] \boxplus ROL_{i+2}(\delta[i \bmod 8]))$
  $T[6i+3 \bmod 8] \leftarrow ROL_{11}(T[6i+3 \bmod 8] \boxplus ROL_{i+3}(\delta[i \bmod 8]))$
  $T[6i+4 \bmod 8] \leftarrow ROL_{13}(T[6i+4 \bmod 8] \boxplus ROL_{i+4}(\delta[i \bmod 8]))$
  $T[6i+5 \bmod 8] \leftarrow ROL_{17}(T[6i+5 \bmod 8] \boxplus ROL_{i+5}(\delta[i \bmod 8]))$
  $RK_i^{enc} \leftarrow (T[6i \bmod 8], T[6i+1 \bmod 8], T[6i+2 \bmod 8], T[6i+3 \bmod 8], T[6i+4 \bmod 8], T[6i+5 \bmod 8])$
end for

(c) LEA-256

Fig. 4. Pseudo code for LEA encryption key schedule (a) LEA-128 (b) LEA-192 (c) LEA-256

III. DESIGN OF LEA CRYPTO-PROCESSOR[12]

A crypto-processor for LEA is designed, which supports three key lengths of 128/192/256 bits. The LEA processor consists of round block, key scheduler, and control block as shown in figure 5. The round block performs round transformations of 24/28/32 rounds depending on the key length of 128/192/256 bits. The key scheduler generates round keys of 192 bits that are used in round operation. The round block was designed with 32-bit data-path, so one round operation takes four clock cycles. To achieve area-efficient implementation, some design considerations were taken into account for sharing hardware resources of the key scheduler supporting three key lengths, as well as the round block performing encryption and decryption operations.

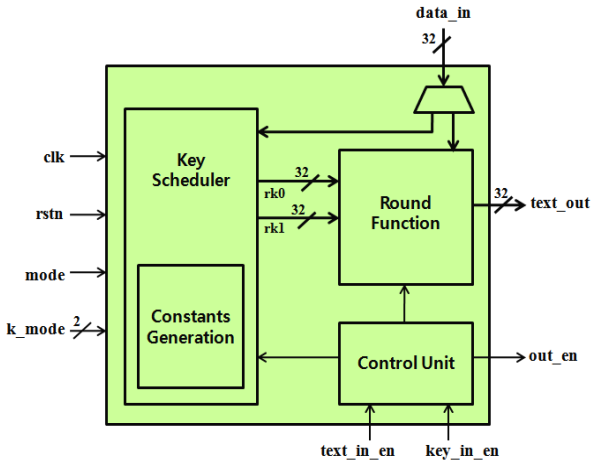


Fig. 5. LEA encryption/decryption processor

A. Round Block

The round block receives a plaintext (or ciphertext) of 128 bits and a round key of 192 bits generated by key scheduler, and performs round transformations. Figure 6 shows the round block that consists of four 32-bit registers($X0 \sim X3$), XORs, modulo adders/subtractors in 32 bits, rotators and multiplexors. The round block was designed to share hardware resources including registers, XORs and modulo adder/subtrator for encryption and decryption in order to achieve area-efficient implementation.

The plaintext (or ciphertext) of 128 bits enters into the round block in 32 bits at a time, taking four clock cycles. The data entered in register $X0$ moves to the next register in the order of $X0 \rightarrow X1 \rightarrow X2 \rightarrow X3$. After completing plaintext (or ciphertext) input, round operation begins. Each round operation requires 3/3/4 clock cycles depending on the key length of 128/192/256 bits. The ciphertext (or plaintext) is obtained after repeating 24/28/32 round operations.

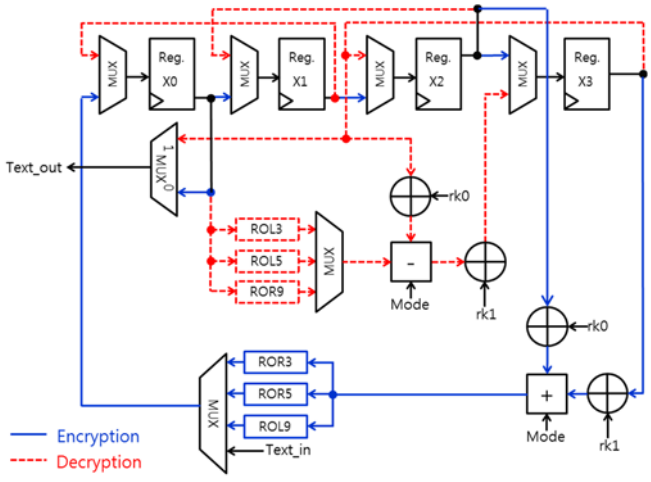


Fig. 6. Round block diagram

a) Round operation for encryption

The round operation for encryption includes round key additions (XOR operations) of rk0 with $X2$, and rk1 with $X3$, where rk0 and rk1 are round keys generated by key scheduler. These two results of XOR operations are added in

modulo 32, then the result is rotated depending on the order of clock cycles as following; at the first clock cycle, it is rotated to the right by 3 bits($ROR3$). At the second clock cycle, it is rotated to the right by 5 bits($ROR5$). At the third clock cycle, it is rotated to the left 9 bits($ROL9$). The rotated result is saved in the register $X0$. At the same time, data stored in the registers are shifted to the right in 32 bits. After shifts of $X0 \rightarrow X1 \rightarrow X2 \rightarrow X3$ are repeated for three clock cycles, one round operation is completed. As a result of these operations, data stored in registers $X0, X1, X2, X3$ are paired by $(X2, X3)$, $(X1, X2)$, $(X0, X1)$ to be used in the operation of next cycle.

b) Round operation for decryption

In decryption, round keys are used in reverse order of encryption. In addition, the order of operations in round function for decryption is $'Rotation \rightarrow Subtraction \rightarrow XOR'$, whereas that of the encryption process is $'XOR \rightarrow Addition \rightarrow Rotation'$. The direction of rotation for decryption is opposite to the encryption, and modulo addition is replaced by modulo subtraction. Moreover, the data stored in registers moves to the opposite direction, $X3 \rightarrow X2 \rightarrow X1 \rightarrow X0$. Similar to the encryption, a round operation takes three clock cycles, and the data in register $X0$ is rotated depending on the order of clock cycles as follows; at the first clock, data is rotated to the right by 9 bits($ROR9$), then it is rotated to the left by 5 bits($ROL5$) at the second clock, finally it is rotated to the left by 3 bits($ROL3$) at the third clock cycle.

B. Key Scheduler Block

The round keys of 192 bits used in the round operations are generated by key scheduling algorithm. Figure 7 shows the key scheduler proposed in this paper, which consists of eight registers of 32 bits; XOR operators, 32-bit modulo adders/subtractors, rotators. As shown in figure 7, the key scheduler has eight 32-bit registers $(T0 \sim T7)$. They are separated into two groups working in parallel, the registers with even index $(T0, T2, T4, T6)$ and the registers with odd index $(T1, T3, T5, T7)$.

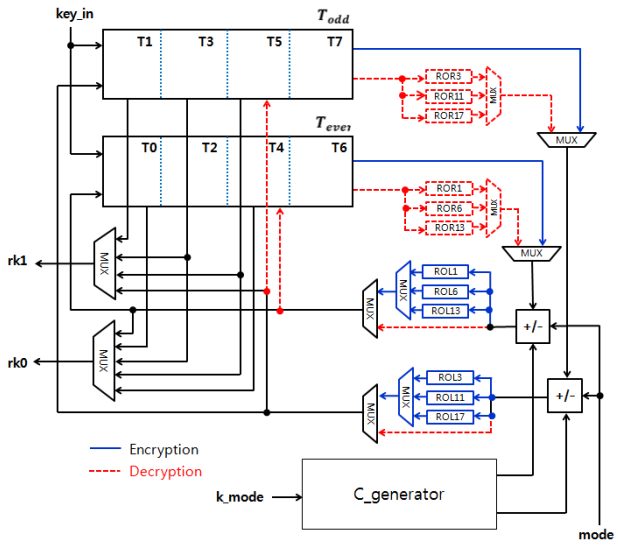


Fig. 7. Proposed key scheduler

The key scheduler also includes a constant generator for constant value δ to be used in key expansion. Depending on the key length of 128/192/256 bits and encryption/decryption modes, the key scheduler operates in different ways. Some multiplexers are controlled by signal k_mode that determines one of key scheduling of LEA-128, LEA-192, or LEA-256 depending on key length.

The key scheduler was designed to share hardware resources for encryption and decryption, as well as for key length in order to minimize hardware.

The master key enters into the register $T0$ and $T1$ alternately in 32 bits at a time. The data in registers shift as follows; $T0 \rightarrow T2 \rightarrow T4 \rightarrow T6$ and $T1 \rightarrow T3 \rightarrow T5 \rightarrow T7$. The key scheduling of LEA-128 (i.e., key length is 128-bit) requires four 32-bit registers, and it takes four clock cycles per round. We devised an efficient scheme to reduce one clock cycle to generate round key. The idea is based on the register arrangement in the key scheduler.

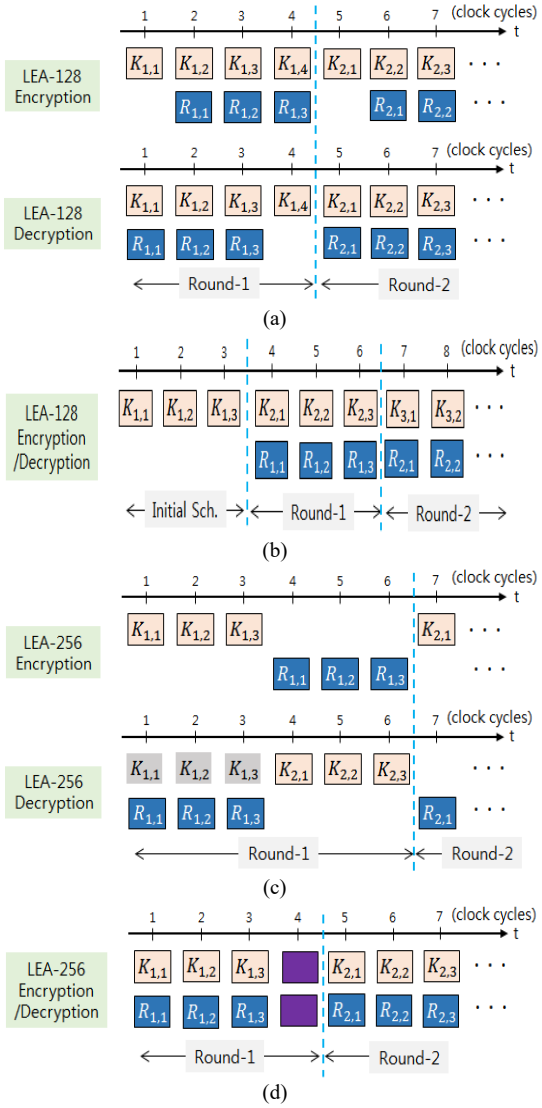


Fig. 8. Timing relations of key scheduling and round transformation (a) key scheduling of LEA-128 in [13] (b) proposed key scheduling of LEA-128 (c) key scheduling of LEA-256 in [13] (d) proposed key scheduling of LEA-256

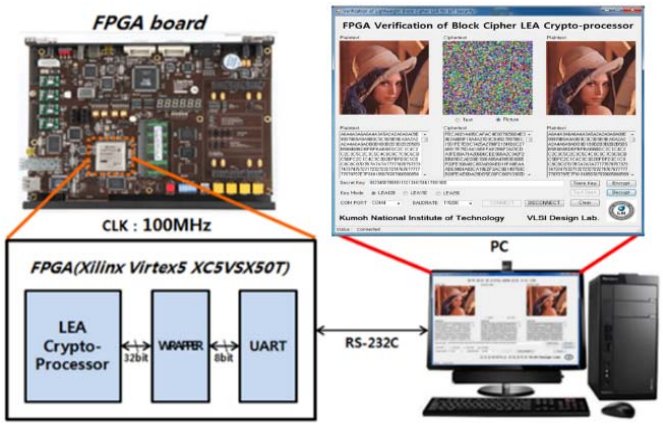
Since the key scheduler in our design has six 32-bit registers, the intermediate keys generated previously round can be saved in unused parts of the registers, and they can be outputted by using multiplexer. This key scheduling requires three clock cycles per round rather than four clock cycles per round in [13], resulting in the clock cycle reduction.

The similar idea was applied to the key scheduling of LEA-256 (i.e., key length is 256-bit). It requires four clock cycles per round to generate round key. Three clock cycles are used for key scheduling and one clock cycle is used for shift & save operation.

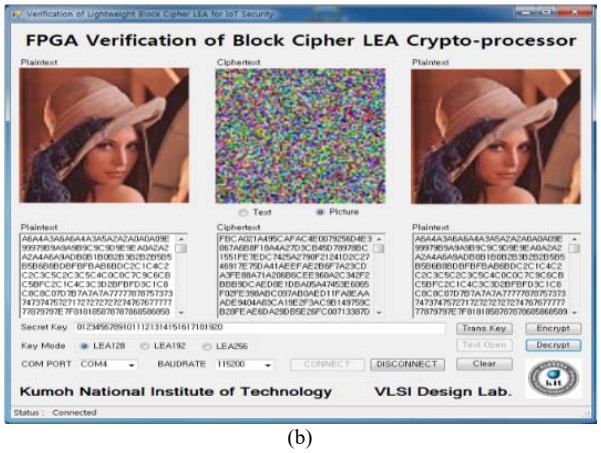
Figure 9 shows a comparison of timing relationships for round transformation and key scheduling between our design and the design of [13]. In the LEA-128 mode shown in Figure 9-(a) and (b), our design requires three clock cycles per round as compared to four clock cycles per round in [13]. Consequently, the performance is improved by 20% in encryption mode and 23% in decryption mode. In the case of LEA-256 mode shown in Figure 9-(c) and (d), our design requires four clock cycles per round as compared to six clock cycles per round in [13]. Consequently, the performance is improved by 33% and 32% in encryption mode and decryption mode, respectively.

IV. FUNCTIONAL VERIFICATION AND CHIP TEST RESULTS

The LEA crypto-processor was designed in Verilog HDL, and was verified by FPGA implementation. Figure 9-(a) shows the FPGA verification set-up that consists of FPGA board, UART interface, and GUI software. The Xilinx Virtex5 XC5VSX-50T FPGA device is used. Master key and plaintext (or ciphertext) are sent from PC to FPGA through RS232C, and test results of ciphertext (or plaintext) are sent from FPGA to PC. Figure 9-(b) shows the FPGA verification results. The left Lena image is the original data used as test vector. The center part shows the cipher-image obtained from the encryption of the original Lena image. The right part shows the decryption result of the cipher-image. The original Lena image was restored from the decryption of cipher-image. The FPGA verification results confirm that the LEA crypto-processor functions correctly.



(a)



(b)

Fig. 9. FPGA verification results of LEA crypto-processor (a) FPGA verification set-up (b) FPGA verification results

Figure 10 shows the design flow of the LEA processor which was fabricated in a 0.18-um CMOS process. The LEA processor modeled in Verilog HDL was verified by functional simulation using Modelsim, and then it was synthesized to gate-level by using a 0.18um standard cell library. Formality check was carried out between RTL modeling and the netlist extracted from synthesis. For the sign-off of front-end design, the pre-layout STA (static timing analysis) using PrimeTime and pre-layout timing simulation using VCS were carried out. The layout of the chip was designed by auto P&R (place & route) using Astro, and then equivalence between gate-level netlist and layout was verified by formality check. In addition, post-layout STA and timing simulation were performed with the back-annotation of the parasitic RC effects extracted from

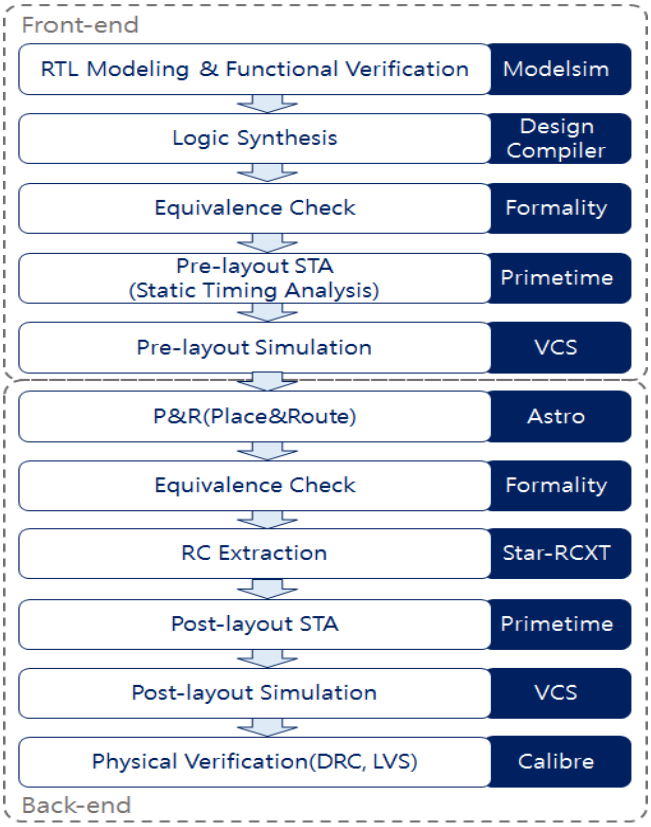


Fig. 10. Design flow of the LEA crypto-processor

the layout using Star-RCXT. Finally, layout verification was done by DRC (design rule check) and LVS (layout versus schematic) using Calibre. Figure 11 shows the chip layout. The maximum clock frequency estimated by STA is 179 MHz. At the maximum frequency, the estimated throughputs are about 286.4/257.4/173.6 Mbps for encryption, and about 297.6/ 257.4/172.3 Mbps for decryption depending on the key length of 128/192/256 bits.

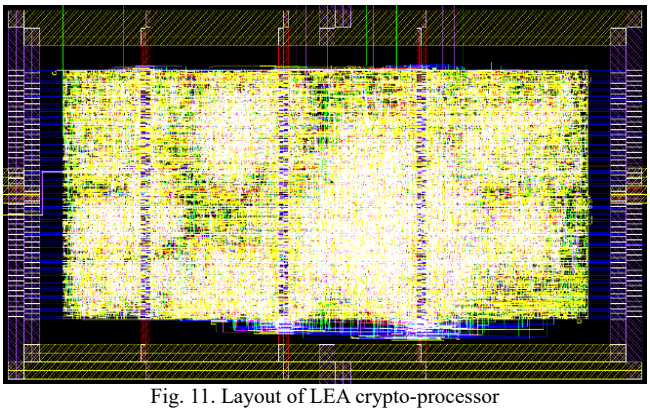


Fig. 11. Layout of LEA crypto-processor

Figure 12 shows the test set-up, which consists of test board and GUI software on PC. The test board contains a socket where a test chip is mounted, a FPGA device of Xilinx Spartan3 XC3S1000, and some switches. In order to send test vectors from PC to the test board, and to receive test responses from the test board, UART and wrapper circuits are implemented in the FPGA device on the test board.

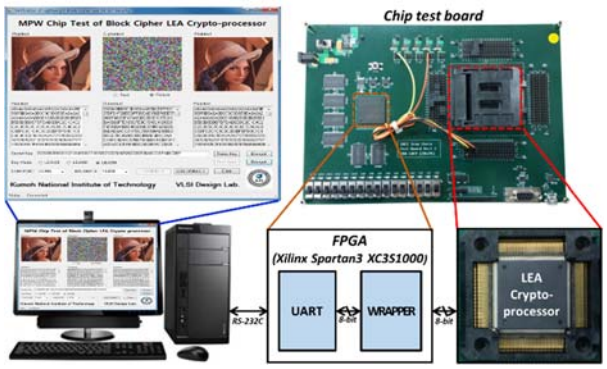


Fig. 12. MPW chip test result of LEA core

GUI software on PC displays the test vectors and test responses. Figure 12 shows that the test results of the LEA crypto-processor. Test vector (Lena image) are sent to the wrapper implemented in the FPGA, then are moved into the test chip (LEA crypto-processor). The test chip encrypts the test vectors, and sends cipher-image to the wrapper. Then the cipher-image stored in the wrapper is sent back to PC via UART. As can be seen in the center part of the display, the Lena image was converted into random values by encryption. The right part of the display shows the decryption result of the cipher-image. The original Lena image was reconstructed by decryption. The test results show that the LEA crypto-processor functions correctly. All of the 22 chips tested works correctly, resulting 100% yield.

Table I compares our design with the LEA processor of [13]. The LEA crypto-processor designed in this paper supports three key lengths, but the LEA processor of [13] supports only one key length of 128 bits. Therefore, direct comparison of hardware complexity is unfair. The estimated throughput of our LEA processor is about 99~177 Mbps when operating with 100 MHz clock, which means about 30 percent higher than that of the LEA processor of [13].

V. CONCLUSIONS

A LEA crypto-processor design supporting three key lengths of 128/192/256 bits is discussed. Some design optimizations were considered to achieve area-efficient implementation, which share hardware resources for encryption and decryption, as well as key expansion for three key lengths. In addition, a new key scheduling scheme was devised to reduce the number of clock cycles required per round, which results in performance increase by 20~30%.

TABLE I.
Comparison of LEA cores

		[12]		This work		
		cycles/round	Throughput @ 100 MHz (Mbps)	cycles/round	Throughput @ 100 MHz (Mbps)	
			LEA core		LEA core	Including I/O
LEA-128	Enc	4	133	3	170	160
	Dec	4	133	3	177	166
LEA-192	Enc	3	152	3	152	143
	Dec	3	152	3	152	143
LEA-256	Enc	6	66	4	100	97
	Dec	6	66	4	99	96

The LEA crypto-processor was fabricated in a 0.18-um CMOS process. The chip test results show that the encryption and decryption for the key length of 128/192/256 bits work correctly. The throughputs, estimated from simulation for a clock frequency of 179 MHz, are about 286.4/257.4/173.6 Mbps for encryption and about 297.6/257.4/172.3 Mbps for decryption depending on the key length. The LEA crypto-processor can be used as an IP in security IC for IoT and mobile devices that require low power and small-area.

ACKNOWLEDGMENT

This work was supported by the Industrial Core Techno-logy Development Program (10049009, Development of Main IPs for IoT and Image Based Security Low-Power SoC) funded by the Ministry of Trade, Industry & Energy.

The authors are thankful to IDEC for supporting EDA software.

REFERENCES

[1] Dong-hui Kim et al, "Security for IoT Service", Journal of Korea Institute of Communication and Information Services, vol. 30, no. 8, pp.53, July 2013.

[2] C. Lu, "Overview of Security and Privacy Issues in the Internet of Things", http://www.cse.wustl.edu/~jain/cse57414/ftp/security/

[3] T. Eisenbarth, C. Paar, A. Poschmann, S. Kumar and L. Uhsadel, "A Survey of Lightweight Cryptography Implementations, " IEEE Design & Test of Computers, vol.24, no. 6, pp. 522-533, 2007.

[4] Korea Internet & Security Agency, "HIGHT Algorithm Specification", 2009.

[5] Telecommunications Technology Association, "128-Bit Block Cipher LEA", TTA Standard, TTAK.KO-12.0223, 2013.

[6] T. Akishita and H. Hiwatari, "Very Compact Hardware Implementations of the Block Cipher CLEFIA", in Selected Areas in Cryptography—SAC 2011, ser. LNCS, vol. 7118, pp. 278-292, Springer-Verlag, 2012.

[7] A. Bogdanov et al., "PRESENT: An Ultra-Lightweight Block Cipher", Proc. Workshop Cryptographic Hardware and Embedded Systems (CHES 07), LNCS 4727, Springer, pp. 450-466, 2007.

[8] C.H. Lim and T. Korkishko, "mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors," Proc. of Information Security Applications, LNCS, vol. 3786, Aug. 2005, pp. 243-258.

[9] D. Wheeler and R. Needham, "TEA, a Tiny Encryption Algorithm", Proc. of the Second International Workshop on Fast Software Encryption, pp. 97-110, 1995.

[10] N. Hanley and M. O′Neill, "Hardware Comparison of the ISO/IEC 29192-2 Block Ciphers", 2012 IEEE Computer Society Annual Symposium on VLSI, pp. 57-66, 2012.

[11] S.S.M. AlDabbagh and I.A. Shaikhli, "Lightweight Block Cipher: a Comparative Study", Journal of Advanced Computer Science and Technology Research Vol.2 No.4, pp. 159-165, Nov., 2012.

[12] Miji Sung and Kyungwook Shin, "An Efficient Hardware Implementation of Lightweight Block Cipher LEA-128/192/256 for IoT Security Applications", Journal of Korea Institute of Information and Commu nication Engineering, vol. 19, No. 7, pp.1608-1616, 2015.

[13] Donggeon Lee et al, "Efficient Hardware Imple-mentation of the Lightweight Block Encryption Algorithm LEA", Sensors, pp. 982-983, 2014.

Mi-Ji Sung received the B.S. degree in electronic engineering from Kumoh National Institute of Technology, Gumi, Korea, in 2015 and is currently working toward the M.S. degree.

Her main interests are semiconductor IP design for communications and signal process, semiconductor IP design for information security.

Gi-Chur Bae received the B.S. degree in electronic engineering from Kumoh National Institute of Technology, Gumi, Korea, in 2015 and is currently working toward the M.S. degree.

His main interests are semiconductor IP design for communications and signal process, semiconductor IP design for information security.

Kyung-Wook Shin received the B.S. degree in electronic engineering from Korea Aerospace University, Goyang, Korea, 1984. He received the M.S. degree and the Ph.D. degree, both in electronic engineering, from Yonsei University, Seoul, Korea, in 1990 and 1990. From 1990 to 1991, He worked at Electronics and Telecommunications Research Institute (ETRI). He is currently a professor of school of electronic engineering at Kumoh National Institute of Technology since 1991. He spent a sabbatical year as visiting professor at University of Illinois at Urbana-Champaign during 1995-1996, and at University of California at San Diego during 2003-2004 and at Georgia Institute of Technology during 2013-2014.

His research interests are SoC design for communications and signal processing, SoC design for information security, and semiconductor IP design.