社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

覃　　輝[†]　笹尾　　勤[†]　井口　幸洋[††]

[†] 九州工業大学情報工学部電子情報工学科　〒 820–8502 福岡県飯塚市大字川津 680–4
[††] 明治大学理工学部情報科学科　〒 214–8571 神奈川県川崎市多摩区東三田 1–1–1
E-mail: †qinhui@aries01.cse.kyutech.ac.jp, ††sasao@cse.kyutech.ac.jp, †††iguchi@cs.meiji.ac.jp

A Design of AES Encryption Circuit with 128-bit Keys Using Look-Up Table Ring

Hui QIN[†], Tsutomu SASAO[†], and Yukihiro IGUCHI[††]

[†] Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan
[††] Department of Computer Science, Meiji University
1–1–1, Higasimita, Tamaku, Kawasaki, Kanagawa, 214–8571 Japan
E-mail: †qinhui@aries01.cse.kyutech.ac.jp, ††sasao@cse.kyutech.ac.jp, †††iguchi@cs.meiji.ac.jp

Abstract This paper presents a partial-rolling architecture for an AES encryption processor. By using a look-up table ring and pipeline techniques, the proposed architecture reduces the amount of memory by 75% while maintaining the memory efficiency (i.e., throughput divided by the size of memory for core). We implemented two architectures: AES-4SM and AES-8SM on Altera Stratix FPGA. The AES-4SM achieved 5.61 G-bit/s throughput by using 20 memory blocks (M4Ks), and the AES-8SM achieved 10.49 G-bit/s throughput by using 40 M4Ks. We also implemented the unrolling architecture that achieves 20.48 G-bit/s throughput by using 80 M4Ks on the same FPGA. Compared with the unrolling implementation, the AES-4SM and the AES-8SM improved the memory efficiency by 10.9% and 18.2%, respectively, and reduced the amount of the memory blocks by 75% and 50%, respectively. The proposed implementation AES-8SM has an higher memory efficiency than both the fastest unrolling implementation and the fastest rolling implementation designed by other group. The proposed implementation fills gap between the rolling and unrolling implementations, and fits on less expensive FPGA.

Key words AES encryption, LUT ring, FPGA

1 Introduction

The Advanced Encryption Standard (AES) was accepted as a FIPS (Federal Information Processing Standards) standard in Nov. 2001 [1]. Since then, many AES implementations using ASICs or FPGAs have been reported. Some focus on the small chip area by using the rolling architecture in Figure 2(a) [2], [3], [4]. Others focus on high throughput by using the unrolling architecture in Fig. 2(b). To achieve a high throughput, partition of each round by inserting pipeline registers is necessary. However, this will increase the cycles (or stages) in the process of the whole AES rounds. For example, Saggese et al. [5] achieved 20.3 Gbps with 50 cycles, while Zambreno et al. [6] achieved 23.50 Gbps with 30 cycles.

In this paper, to achieve a high throughput with small area, we used the look-up table (LUT) ring architecture to perform partial-rolling in the round while adopting the pipeline technique. The rest of the paper is organized as follows: Section 2 explains the AES algorithm. Section 3 presents conventional AES implementation. Section 4 introduces the AES design based on LUT ring. Section 5 presents the AES implementation using an FPGA. Section 6 shows the experimental results. And finally, Section 7 concludes the paper.

2 AES Algorithm

The AES algorithm is based on arithmetic in a finite Galois field, GF(2^8), and is a symmetric block cipher that encrypts 128-bit plain text data with a 128-bit, 192-bit, or 256-bit cipher key [1]. In this paper, we focus on the AES encryption using a 128-bit key shown in Fig. 1 which requires 11 rounds (i.e., logic operations). Each round operates on a state, a 4×4 matrix of 8-bit values, and involves up to four basic transformations:

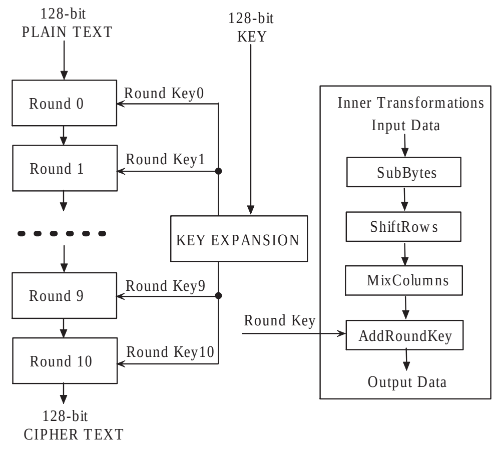


Fig. 1 Architecture of AES encryption with 128-bit key.

- SubBytes - Replaces each byte of the data block with another byte by using an S-box lookup table. The contents of the S-box is the multiplicative inverse in GF (2^8), combined with an affine permutation over GF(2).
- ShiftRows - Cyclically shifts *i* bytes of the state in each row, where *i* is the row number.
- MixColumns - Transforms each column of the matrix by multiplying it with a constant GF(2^8) polynomial, by using the same four-row matrix organization.
- AddRoundKey - Adds the round key to the state using a bit-wise XOR operation.

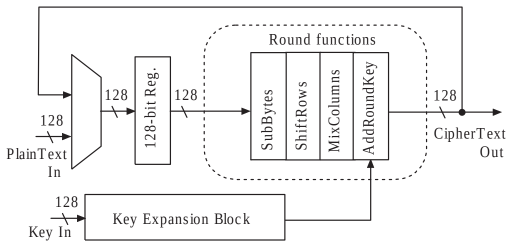
The first round performs only the AddRoundKey transformation. The middle 9 rounds perform all the four transformations, while the final round performs the ShiftRows, SubBytes, and AddRoundKey, omitting the MixColumns transformation.

The AES encryption processor is fed by a 128-bit input key. And then the round keys for each round are generated from the original input key through the key expansion block. Two methods exist to generate the round keys: In the first method, the round keys are stored in a register or memory, and then used for all incoming plain text data. In the second method, an online key generation, where the round keys are generated concurrently with the encryption process. In this paper, we use the online key generation method.

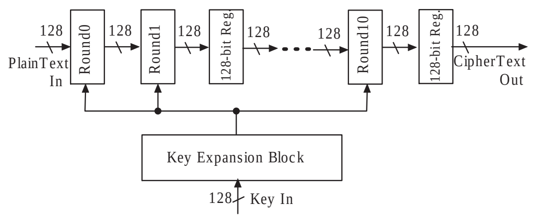
3 Conventional AES Implementation

Various methods exist to realize the AES encryption. Among them, the rolling implementation and the unrolling implementation shown in Fig. 2(a) and (b) are the two basic methods.

The rolling implementation shown in Fig. 2(a) uses a looping structure whereby the data are iteratively passed through the round functions (inner transformations per round). This approach occupies small area, but achieves low throughput. Existing designs [2], [3], [6], have the throughput of approximately 1 to 1.4 G-bit/s, and the size of the memory for the core is just 32 K bits. In the unrolling implementation shown in Fig. 2(b), the round blocks are pipelined and the registers are inserted to operate the round blocks simultaneously. Due to the pipeline, this approach achieves a high through-



(a) Rolling implementation of AES Encryption with 128-bit key.



(b) Unrolling implementation of AES Encryption with 128-bit key.

Fig. 2 Architecture of conventional AES implementations.

put, but requires large area. Existing designs [5], [6], [7], [8], have the throughput of approximately 10 to 23 G-bit/s, and the size of the memory for the core is up to 320 K bits.

4 AES Design Based on LUT Ring

In this section, we will show the AES encryption design based on LUT ring [9], [10].

First, consider the transformations in the AES round. Among the four inner transformations, the SubBytes requires the largest area and latency. It consists of 16 S-Boxes that is the most complicated function block in the entire circuit. For the SubBytes, both the rolling implementation and the unrolling implementation use 16 S-Boxes. In our design, we use an LUT ring to merge and implement the ShiftRows and the SubBytes to reduce the number of S-Boxes and area for shifters and multiplexers. To realize the S-Box, we use the table-lookup method that is one of the fastest methods. Since the multiplication over GF(2^8) in MixColumns uses a constant as one operand, and this constant multiplication can be simply converted into a bit-wise XOR operation, the matrix multiplication can be replaced by several XOR operations. Hence, the MixColumns can be realized as XOR operations. Also, the AddRoundKey operation uses an XOR operation to add the round key. Thus, we can easily implement the MixColumns and the AddRoundKey by using bit-wise XOR operations.

Fig. 3 shows the architecture of the proposed method. We used pipeline to increase the throughput. A round unit consists of the following components:

Rolling Part: Performs the ShiftRows and the SubBytes using LUT ring.

128-bit Reg.: Stores the states of each round.

MixColumns: Performs the MixColumns using several bit-wise XOR operations.

128-bit XOR: Performs the AddRoundKey using 128-bit bit-wise XOR operations.

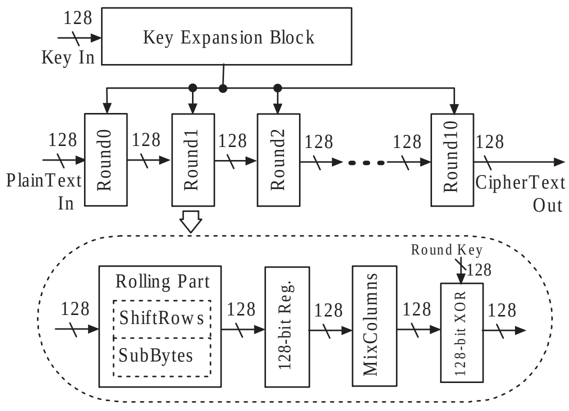


Fig. 3 Architecture of the proposed method.

The rolling part is the most important part in the round unit. In this paper, we show two architectures for the rolling part: The 4SM and the 8SM. The 4SM shown in Fig. 4 consists of a 16-byte cyclic shifter and four copies of S-Modules or SM. The 16-byte cyclic shifter have 16-byte (128-bit) inputs and 4-byte outputs. Let **F** be the outputs of the cyclic shifter, then

$$\mathbf{F}(X_0, X_1, \ldots, X_{15}, K) = (X_{K(mod16)}, X_{K+4(mod16)}, X_{K+8(mod16)}, X_{K+12(mod16)}),$$

where K can be 0, 5, 10 and 15, and represented by the 4-bit control signals, and X_i denotes byte data. For example,

$$\mathbf{F}(X_0, X_1, \ldots, X_{15}, 5) = (X_5, X_9, X_{13}, X_1).$$

Each S-Module consists of a 2K-bit ROM and a 32-bit feed-back register, where the ROM stores the table for the S-Box, and the feed-back register stores the outputs of the S-Box. The 8SM shown in Fig. 5 consists of a 16-byte to 8-byte selector and eight copies of S-Modules. In front of the selector, the permutation network is used to arrange the input data in the required order. Table 1 shows the permutation, where both inputs and outputs are 16-byte data. When the output Sel is 0, the upper 8 bytes are selected, on the other hand when the output Sel is 1, the lower 8 bytes are selected. Each S-Module consists of a 2K-bit ROM and a 16-bit feed-back register, where the ROM stores the table for the S-Box, and the feed-back register stores the outputs of the S-Box. In the round operation, the S-Modules of the 4SM are used four times, while the S-Modules of the 8SM are used twice.

For the 4SM, we can also adopt the architecture of the 8SM. That is to use permutation network and selector instead of the 16-byte cyclic shifter. In our FPGA implementations, using a 16-byte cyclic shifter produces slightly higher throughput than the permutation and a selector, while the areas for the two implementations are almost the same. Both of the 4SM and the 8SM, the whole circuits of the rolling parts form the special LUT rings [9], [10].

〔Example 1〕 In this part, we will illustrate the operations for the 4SM. Let the four-valued representation of the 128-bit original data be 00 10 20 30 01 11 21 31 02 12 22 32 03 13 23 33. After SubBytes and ShiftRows, the output data become 63 82 93 c3 7c c9 26 04 77 7d b7 c7 7b ca fd 23. Figure 6 shows the operations of the 4SM.

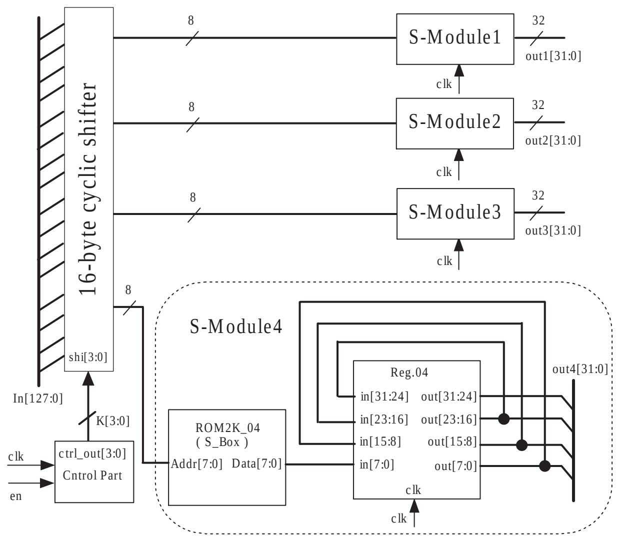


Fig. 4 Architecture of the rolling part: 4SM.

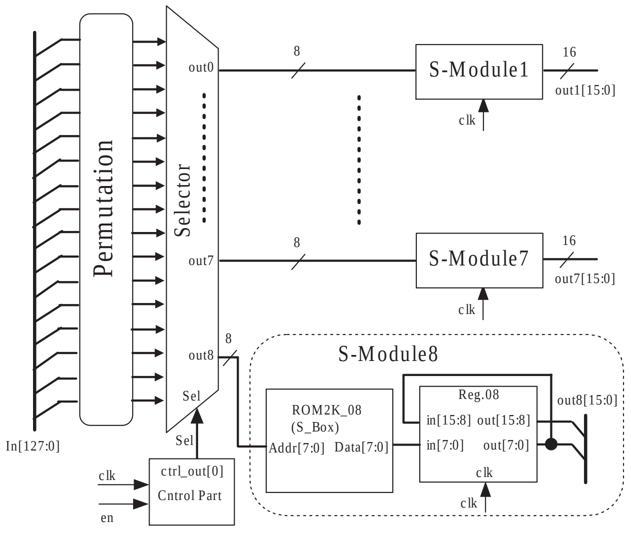


Fig. 5 Architecture of the rolling part: 8SM.

In Step 1 (Fig. 6(a)), the outputs of the control part are 0000, and then the outputs of the 16-byte cyclic shifter become 00,01,02,03. By using the S-Boxes, the outputs of the ROMs become 63,7c,77,7b, respectively.

In Step 2 (Fig. 6(b)), when a positive clock is applied, the outputs of the control part become 0101. At the same time, the previous outputs of ROMs (63,7c,77,7b) are stored in the registers, and also sent to the output terminals. And then the outputs of the 16-byte cyclic shifter become 11,12,13,10. By using the S-Boxes, the outputs of the ROMs become 82,c9,7d,ca, respectively.

In Step 3 (Fig. 6(c)), when a positive clock is applied, the outputs of the control part become 1010. At the same time, the previous outputs of ROMs (82,c9,7d,ca) are stored in the registers, and also sent to the output terminals. And then the outputs of the 16-byte cyclic shifter become 22,23,20,21. By using the S-Boxes, the outputs of the ROMs become 93,26,b7,fd, respectively.

In Step 4 (Fig. 6(d)), when a positive clock is applied, the out-

Table 1 Relationship between the inputs and the outputs of the permutation network in 8SM.

Inputs	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
Outputs	X_0	X_{10}	X_4	X_{14}	X_8	X_2	X_{12}	X_6	X_5	X_{15}	X_9	X_3	X_{13}	X_7	X_1	X_{11}

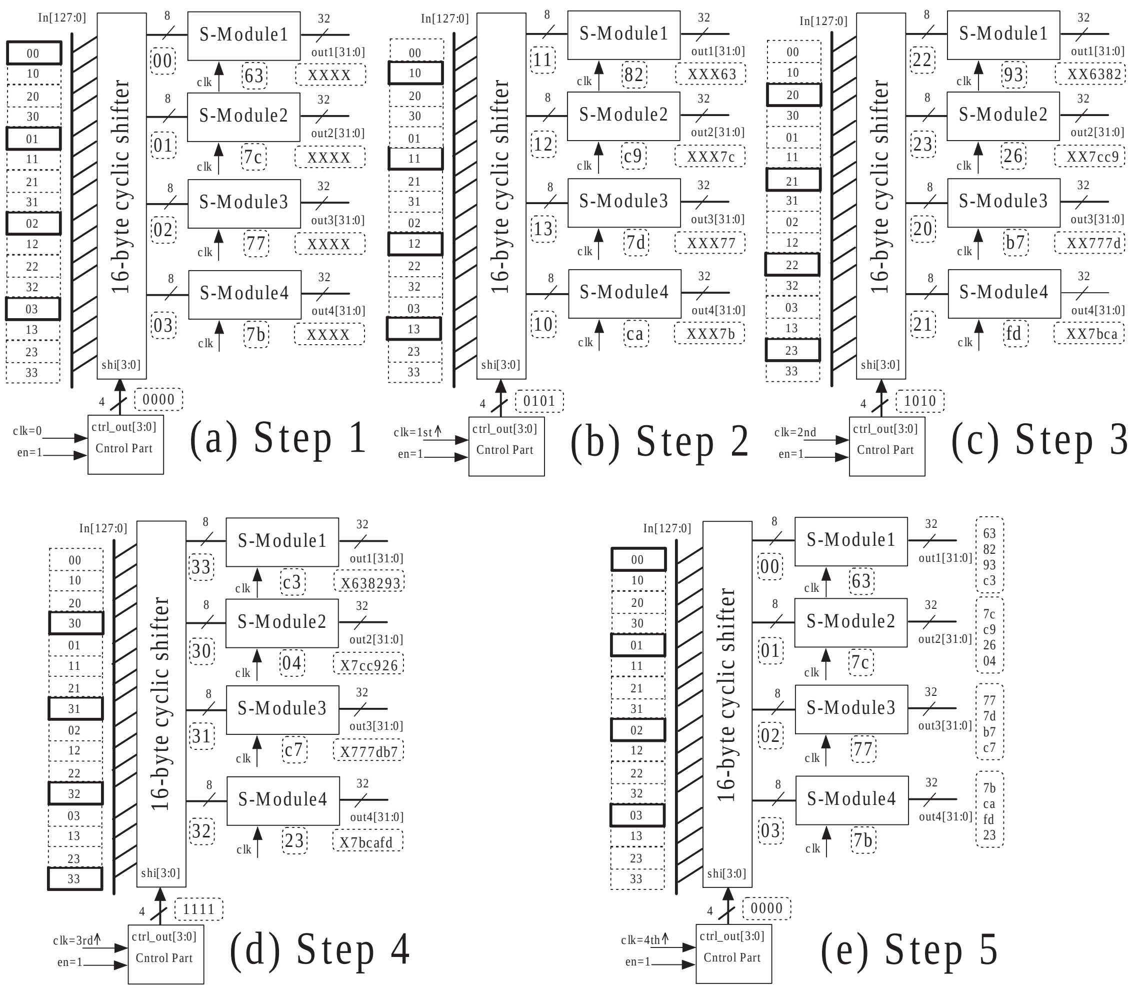


Fig. 6 Operations of rolling part: 4SM.

puts of the control part become 1111. At the same time, the previous outputs of ROMs (93,26,b7,fd) are stored in the registers, and also they are sent to the output terminals. And then the outputs of the 16-byte cyclic shifter become 33,30,31,32. By using the S-Boxes, the outputs of the ROMs become c3,04,c7,23, respectively.

In Step 5 (Fig. 6(e)), when a positive clock is applied, the outputs of the control part become 0000. At the same time, the previous outputs of ROMs (c3,04,c7,23) are stored in the registers, and also sent to the output terminals. And then the outputs of the 16-byte cyclic shifter become 00,01,02,03. In this way, the rolling part implements the SubBytes and ShiftRows transformations. (End of Example)

5 FPGA Implementation

We use the Altera Stratix FPGA to implement the AES encryption circuit based on LUT ring. The Altera Stratix FPGAs offer special RAM blocks called M4K that can store 4096 bits. The M4K can be configured at ratios between 4096×1 to 256×16, and may have dual-port functionality. The M4Ks are also suitable for implementing synchronous ROMs.

As mentioned in Section 4, in the round, the MixColumns and the AddRoundKey can be realized as a network of XOR gates where each gate can be implemented by one Logic Element (LE). In the rolling part, the 16-byte cyclic shifter for the 4SM and selector for the 8SM can be implemented by LEs. For implementing S-Boxes, we used the M4K. Each M4K is configured as a dual-port synchronous 256×8-bit words ROM to implement two separate S-Boxes. The values in the look-up tables for S-Boxes are loaded into the M4Ks at the configuration time. Since an M4K implements two separate S-Boxes, 8 copies of the M4K are sufficient for each SubBytes (16 S-Boxes).

We also designed the unrolling implementation with the same FPGA for comparison. In this design, the SubBytes was implemented by M4Ks, the MixColumns and the AddRoundKey were implemented by XOR gates, while the ShiftRows was simply realized by hardwiring.

6 Implementation Results and Comparisons

We evaluated the performance of the proposed implementations

Table 2 Comparison of AES-4SM, AES-8SM and the UNROLLING.

Design	Device	LEs	Memory for key	Memory for Core	f_{clk}	Cycles	Throughput Gbps)	$Eff\,(\frac{\text{Mbps}}{\text{K-bit}})$
AES-4SM	EP1S20F780C5 (Stratix)	13368	0	80 K-bit (20 M4Ks)	44.39	20	5.68	71.00
AES-8SM	EP1S20F780C5 (Stratix)	12827	0	160 K-bit (40 M4Ks)	94.53	20	12.10	75.63
UNROLLING	EP1S20F780C5 (Stratix)	12560	0	320 K-bit (80 M4Ks)	160.05	20	20.48	64.00
AES-4SM	EP1S10F780C5 (Stratix)	5142	80 K-bit (20 M4Ks)	80 K-bit (20 M4Ks)	39.68	20	5.08	63.50
AES-8SM	EP1S10F780C5	4616	80 K-bit (20 M4Ks)	160 K-bit (40 M4Ks)	88.28	20	11.30	70.63
UNROLLING	EP1S10F780C5	3886	oversize	oversize				
AES-4SM	EP2C35F672C7 (Cyclone II)	13354	0	80 K-bit (20 M4Ks)	38.23	20	4.89	61.13
AES-8SM	EP2C35F672C7 (Cyclone II)	12823	0	160 K-bit (40 M4Ks)	82.99	20	10.62	66.38

AES-4SM and AES-8SM, and compared them with UNROLLING (designed by us) and other published works.

6.1 Comparison with the Unrolling Implementation

To compare the performance of different architectures, we designed both the proposed implementations and the unrolling implementation (UNROLLING) on the same FPGA. We used Verilog HDL to design the AES encryption with 128-bit key. In this experiment, we only used Quartus II 4.1 tool for synthesis, place & route and timing analysis.

Table 2 compares the AES-4SM, the AES-8SM and the UNROLLING. The AES-4SM is implemented with rolling part 4SM, and the AES-8SM is implemented with rolling part 8SM. For each implementation, we used Quartus II 4.1 to obtain the maximum possible clock rate (f_{clk}), the number of utilized logic elements (LEs) and the number of M4K blocks. In Table 2, the column "Device" denotes the FPGA device used, the column "Memory for key" denotes the amount of memory utilized for the key expansion and "Memory for core"denotes the amount of memory utilized for the core, where one M4K is equivalent to 4096 bits. The column "Cycles" denotes the number of clock cycles for the process of the whole AES rounds. The column "Throughput" denotes the maximum **throughput** calculated by:

$$\text{Throughput} = 128 \cdot f_{clk}.$$

The final column "Eff" shows the **memory efficiency** that is the throughput (Mbps) per memory for the core (K bits) calculated by:

$$Eff = \frac{\text{Throughput}}{\text{Memory for core}}. \quad (1)$$

The Altera Stratix Device [11] EP1S10F780C5 has the minimum devices in the family, containing 10570 LEs and 60 M4Ks. The EP1S20F780C5 is larger than the EP1S10F780C5 and contains 18460 LEs and 82 M4Ks. For the EP1S20F780C5, as shown in Table 2, the AES-8SM design has the highest memory efficiency while its throughput is lower than UNROLLING. The efficiency of AES-8SM is higher than AES-4SM, since the selector in the AES-8SM used one control signal while the cyclic shifter in the AES-4SM used four control signals.

Compared with the UNROLLING, LEs utilized for the AES-4SM and the AES-8SM are increased by 6% and 2%, but the amounts of memory utilized for the AES-4SM and the AES-8SM are reduced by 75% and 50%, respectively, and the memory efficiencies for the AES-4SM and the AES-8SM are improved by 10.9% and 18.2%, respectively.

It is interesting to find that the AES-4SM can also be implemented on the EP1S10F780C5 by utilizing the M4Ks for the key expansion. Note that in this method, the throughput is lower than that on the EP1S20F780C5. Also, note that the EP1S10F780C5 is too small for the UNROLLING, since it requires 100 copies of the M4K to utilize the memory for both the key and the core that exceeds the maximum number of the M4Ks of the EP1S10F780C5. However, the AES-8SM can also be implemented on the EP1S10F780C5 (not shown in Table 2), since it requires 60 M4Ks. We also implemented the AES-4SM and the AES-8SM on CycloneII EP2C35 FPGA. The throughputs are 4.89 Gbps and 10.62 Gbps, respectively. Hence, the AES-4SM and the AES-8SM are suitable for the smaller FPGAs. In this regard, the AES-4SM and the AES-8SM fill gap between the rolling implementation and the unrolling implementation.

6.2 Comparison with the published works

Direct comparison among various FPGA implementations of the AES algorithms is difficult, since FPGA target devices are often different. However, many AES implementations have provided the maximum throughputs and the amount of the memory utilized for the core. Thus, we can compare the memory efficiency defined in Equation 1.

Table 3 compares the AES-4SM, the AES-8SM and the published works. The top two rows show our implementations. The middle three rows show the unrolling implementations, and the last two rows show the rolling implementations.

Compared with the unrolling implementations, the memory efficiency of the AES-8SM is higher than the fastest

Table 3 Comparison of AES-4SM and AES-8SM with the published works.

Design	Device	$\frac{\text{LEs}}{\text{Slices}}$	Memory for key	Memory for core	Cycles	Throughput (Gbps)	$Eff(\frac{\text{Mbps}}{\text{K-bit}})$	Design Rule
AES-4SM	EP1S20F780C5 (Stratix)	13368 LEs	0	80 K-bit (20 M4Ks)	20	5.68	71.00	0.13 μm
AES-8SM	EP1S20F780C5 (Stratix)	12827 LEs	0	160 K-bit (40 M4Ks)	20	12.10	75.63	0.13 μm
Standaert et al. [7](unrolling)	XCV3200E-8	2784 Slices	80 K-bit (20 BRAMs)	320 K-bit (80 BRAMs)	21	11.77	36.78	0.18 μm
Saggese et al. [5](unrolling)	XVE2000-8	5810 Slices	80 K-bit (20 BRAMs)	320 K-bit (80 BRAMs)	50	20.30	63.44	0.18 μm
UF10-PP3B [6] (unrolling)	XC2V4000	5142 Slices	80 K-bit (20 BRAMs)	320 K-bit (80 BRAMs)	30	23.50	73.44	0.12 μm / 0.15 μm
UF1-PP0B [6] (rolling)	XC2V4000	387 Slices	8 K-bit (2 BRAMs)	32 K-bit (8 BRAMs)	10	1.41	44.06	0.12 μm / 0.15 μm
Helion [2] (rolling)	Stratix-5	1023 LEs	8 K-bit (2 M4Ks)	32 K-bit (8 M4Ks)	10	1.40	43.75	0.13 μm

Slice: Contains two 4-input look-up tables; BRAM: Block SelectedRAM (4 K-bit)

implementations(UF10-PP3B) while the memory efficiency of the AES-4SM is very close to it. Note that the number of cycles for UF10-PP3B is 30. And the amounts of the memory utilized for the core of the AES-4SM and the AES-8SM are reduced by 75% and 50%, respectively.

Compared with the rolling implementations, the throughput of the AES-4SM is 4.01 times higher than the fastest rolling implementation (UF1-PP0B), and the memory efficiency of the AES-4SM is 1.61 times higher than UF1-PP0B , but the amount of the memory utilized for the core is 2.5 times larger than them. The throughput of the AES-8SM is 8.58 times higher than the fastest rolling implementation (UF1-PP0B), and the memory efficiency of the AES-4SM is 1.72 times higher than UF1-PP0B , but the amount of the memory utilized for the core is 5 times larger than them.

The AES-4SM and AES-8SM have much higher throughput than the software implementations. An AES encryption in a feedback mode achieves 1.538 Gbps on a 3.2 GHz Pentium4 processor [12] and a 640Mbps on a 1 GHz embedded processor [13].

7 Conclusions

In this paper, we have shown two partial-rolling architectures (AES-4SM and AES-8SM) of the AES encryption, and implemented them on Altera Stratix EP1S20F780C5 FPGA. The key technique is LUT ring, which is quite suitable for FPGA implementation. The AES-4SM achieved 5.61 Gbps throughput by using 20 M4Ks, and the AES-8SM achieved 10.49 Gbps throughput by using 40 M4Ks. Compared with the unrolling implementation on the same FPGA, the amounts of the memory were reduced by 75% and 50%, and the memory efficiencies were improved by 10.9% and 18.2%, respectively. The AES-4SM fills gap between the rolling and unrolling implementations, and fits on less expensive FPGA devices such as Cyclone II.

Acknowledgments

This research is partly supported by JSPS, the Grant in Aid for Scientific Research, and MEXT, the Kitakyushu area innovative cluster project.

References

[1] National Institute of Standards and Technology (NIST), *Advanced Encryption Standard (AES),* Federal Information Processing Standards Publications 197 (FIPS197), Nov. 2001.

[2] HELION Technology Limited,
"High performance AES (Rijndael) cores for Altera FPGA," available at http://www.heliontech.com/core2.htm.

[3] Amphion Semiconductor,
"CS5210-40: High performance AES encryption cores," 2003, available at http://www.amphion.com/cs5210.htm.

[4] N. Pramstaller and J. Wolkerstorfer, "A universal and efficient AES co-processor for field programmable logic arrays," *FPL 2004,* LNCS3203, pp. 565-574, 2004.

[5] G. P. Saggese, A. Mazzeo, N. Mazzocca and A. G. M. Strollo, "An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm," *FPL 2003,* LNCS 2778, pp. 292-302, 2003.

[6] J. Zambreno, D. Nguyen and A. N. Choudhary, "Exploring area/delay tradeoffs in an AES FPGA implementation," *FPL 2004,* LNCS3203, pp. 575-585, 2004.

[7] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater and J.-D. Legat, "Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs," *in the proceedings of CHES 2003,* Lecture Notes in Computer Science, vol. 2523, pp. 334–350, Cologne, Germany, September 2003, Springer-Verlag.

[8] F. Charot, and E. Yahya, and C. Wagner, "Efficient modular-pipelined AES implementation in counter mode on ALTERA FPGA," *FPL 2003,* pp. 282-291, Lisbon, Portugal, 2003.

[9] H. Qin, T. Sasao, M. Matsuura, S. Nagayama, K. Nakamura and Y. Iguchi "A realization of multiple-output functions by a look-up table ring," *IEICE Transactions on Fundamentals of Electronics,* Vol.E87-A, pp. 3141-3150, Dec. 2004.

[10] T. Sasao, M. Kusano, and M. Matsuura, "Optimization methods in look-up table rings," *International Workshop on Logic and Synthesis (IWLS-2004),* June 2-4, Temecula, California, U.S.A. .pp. 431-437.

[11] http://www.altera.com

[12] H. Lipmaa, "AES implementation speed comparison," available at http://www.tsc.hut.fi./ aes/rijndael.html,2003.

[13] K. Nadehara, M. Ikekawa, and I. Kuroda "Extended instructions for the AES cryptography and their efficient implementation," *IEEE Workshop on Signal Processing System (SIPS'04),* Oct. 13-15, 2004, FA-1.3.