An FPGA Design of AES Encryption Circuit with 128-bit Keys

Hui QIN †
qinh_jp@yahoo.co.jp

Tsutomu SASAO †
sasao@cse.kyutech.ac.jp

Yukihiro IGUCHI ‡
iguchi@cs.meiji.ac.jp

† Dept. of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka, 820–8502, Japan
‡ Dept. of Computer Science, Meiji University, Kawasaki, Kanagawa, 214–8571, Japan

ABSTRACT

This paper addresses a pipelined partial rolling (PPR) architecture for the AES encryption. The key technique is the PPR architecture, which is suitable for FPGA implementation. Using the proposed architecture on the Altera Stratix EP1S20F780C5 FPGA, the AES-4SM achieves a throughput of 5.61 Gbps by using 20 M4Ks, and the AES-8SM achieves a throughput of 10.49 Gbps by using 40 M4Ks. Compared with the unrolling implementation that achieves a throughput of 20.48 Gbps by using 80 M4Ks on the same FPGA, implementations with the PPR architecture reduce the amount of memory up to 75% while increasing the memory efficiency (i.e., throughput divided by the size of memory for core) up to 9.6%. The PPR architecture fills the gap between unrolling and rolling architectures, and fits on less expensive FPGAs.

Categories and Subject Descriptors:
B.7.1 [INTEGRATED CIRCUITS]: Types and Design Styles – Algorithms implemented in hardware

General Terms: Design, Experimentation

Keywords: AES encryption, pipeline, FPGA

1. INTRODUCTION

The Advanced Encryption Standard (AES) [1] is based on arithmetic in a finite Galois field, GF(2^8), and it 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 as shown in Fig. 1. It requires 11 rounds (i.e., logic operations), in which the first round performs only the AddRoundKey transformation, the middle 9 rounds perform all the four transformations: SubBytes, ShiftRows, MixColumns and AddRoundKey, and the final round performs three transformations: SubBytes, ShiftRows, and AddRoundKey, omitting the MixColumns transformation. The round keys for each round are generated from the original 128-bit input key through the key expansion block. In general, 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. However, this method requires a large register or memory for the round keys, and it needs a preprocessing phase every time the key is changed. The second method is an online key generation algorithm, where the round keys are generated concurrently with the encryption process. Since the online key generation method allows the block cipher to work at full speed even if the key is changed, we adopted this method in this work.

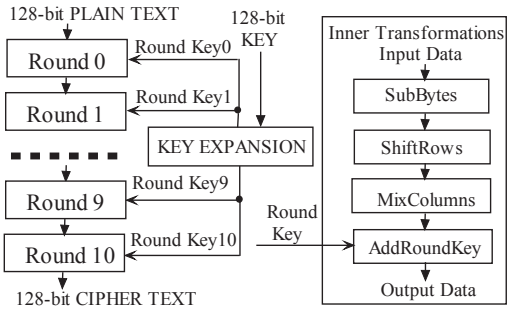
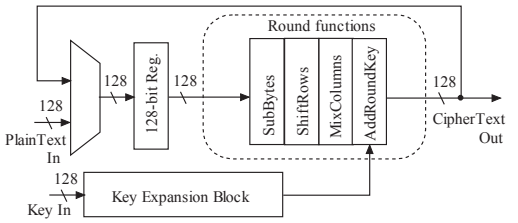


Figure 1: AES encryption with 128-bit key.

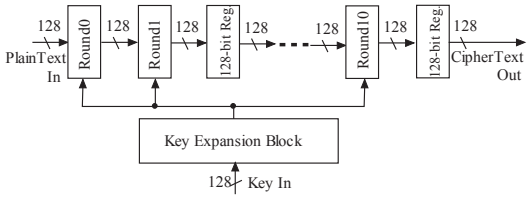
Since Nov. 2001, various AES implementations using ASICs or FPGAs have been reported. Some focus on the small chip area by using the rolling architecture in Figure 2(a) [2], [3], [4], and others focus on high throughput by using the unrolling architecture in Fig. 2(b). To achieve a high throughput, partition of each round by inserting pipeline registers is necessary. However, this will increase the cycles (or stages) in the AES round blocks. 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, we have developed a pipelined partial rolling (PPR) architecture to achieve a high throughput with small area. It performs partial-rolling in the round while adopting the pipeline technique. The rest of the paper is organized as follows: Section 2 presents the conventional AES architectures. Section 3 introduces the pipelined partial rolling (PPR) architecture. Section 4 presents the AES implementation using an FPGA. Section 5 shows the experimental results. And finally, Section 6 concludes the paper.

GLSVLSI'05, April 17–19, 2005, Chicago, Illinois, USA.



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



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

Figure 2: Existing AES architectures.

2. EXISTING AES ARCHITECTURE

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

The **rolling architecture shown** in Fig. 2(a) uses a feedback structure where the data are iteratively transformed by the round functions. This approach occupies small area, but achieves low throughput. Existing rolling implementations [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 architecture** shown in Fig. 2(b), the round blocks are pipelined and the inserted pipeline registers allows simultaneous operation of all 11 round blocks. Due to the pipeline, this approach achieves a high throughput, but requires large area. Existing unrolling implementations [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.

3. PIPELINED PARTIAL ROLLING (PPR) ARCHITECTURE

This section describes the AES design with a new type of memory-based architecture called **pipelined partial rolling** (**PPR**).

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 of the rolling implementation and the unrolling implementation use 16 S-Boxes. In the PPR, we use a special mechanism for the ShiftRows and the SubBytes to reduce the number of S-Boxes and areas for shifters and multiplexers. To realize the S-Box, we use a ROM 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.

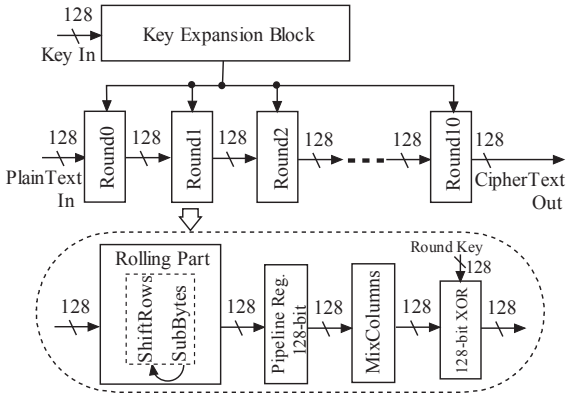


Figure 3: Architecture of PPR.

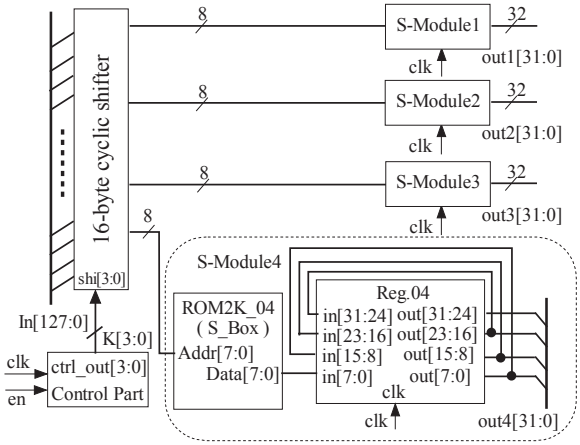


Figure 4: Rolling part: 4SM.

Figure 3 shows the architecture of the PPR. We used pipeline to increase the throughput. The key expansion block consists of 10 round key circuits that used to generate the round key for each round. A round unit consists of the following components:

Rolling Part: Performs the ShiftRows and the SubBytes transformations.

128-bit Pipeline 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.

The rolling part is the most important part in the round unit. In this paper, we show two designs 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-Module (SM)**. The 16-byte cyclic shifter have 16-byte (128-bit) inputs and 4-byte outputs. Let **F** be the input-to-output mapping function of the

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}

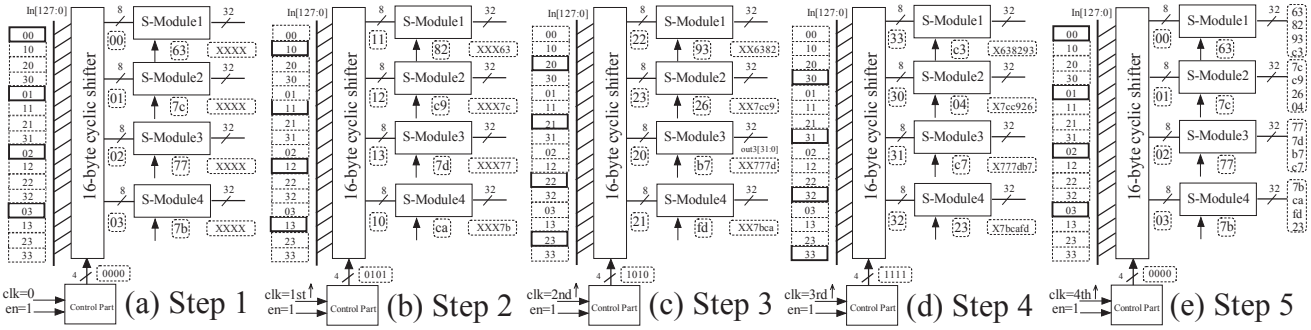


Figure 6: Operations of rolling part: 4SM.

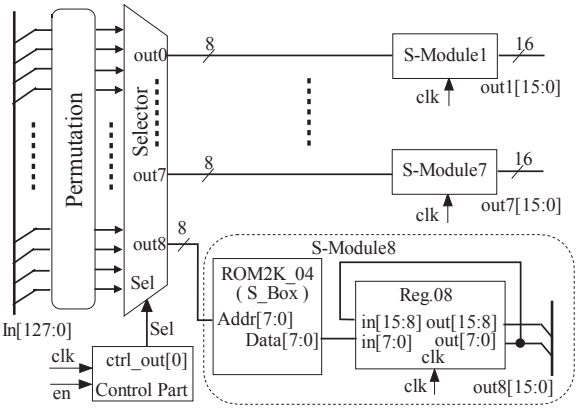


Figure 5: Rolling part: 8SM.

cyclic shifter, then

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

where K is 0, 5, 10 or 15. K is represented by the 4-bit output signals from the control part, and X_i denotes byte data. For example,

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

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-Module. 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 implementation, Quartus II 4.1 simulator shows that a 16-byte cyclic shifter produces higher memory efficiency than the permutation and a selector, while the areas for the two implementations are the same.

EXAMPLE 1. *This part illustrates the operations for the 4SM. Let the hexadecimal 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. Figures 6 (a) - (b) show the operations of the 4SM.*

In Step 1, *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, *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, *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, *when a positive clock is applied, the outputs 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.*

Table 2: Comparison of AES-4SM, AES-8SM, UNROLLING and the published works

Design	Device	LEs / Slices	Memory for key	Memory for core	Cycles	f_{clk} (MHz)	Th. (Gbps)	Eff ($\frac{\text{Mbps}}{\text{K-bit}}$)	Design Rule
AES-4SM	1S20C5	13368 LEs	0	80 K-bit (20 M4Ks)	20	43.86	5.61	70.13	0.13 μm
AES-8SM	1S20C5	12827 LEs	0	160 K-bit (40 M4Ks)	20	82.00	10.49	65.56	0.13 μm
UNROLLING	1S20C5	12560 LEs	0	320 K-bit (80 M4Ks)	20	160.05	20.48	64.00	0.13 μm
AES-4SM	1S10C5	5142 LEs	80 K-bit (20 M4Ks)	80 K-bit (20 M4Ks)	20	39.68	5.08	63.50	0.13 μm
UNROLLING	1S10C5	3886 LEs	oversize	oversize					
Standaert et al. [7](unrolling)	XCV 3200E-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)	XVE 2000-8	5810 Slices	80 K-bit (20 BRAMs)	320 K-bit (80 BRAMs)	50		20.30	63.44	0.18 μm
UF10-PP3B [6] (unrolling)	XC2V 4000	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)	XC2V 4000	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 -C5	1023 LEs	8 K-bit (2 M4Ks)	32 K-bit (8 M4Ks)	10		1.40	43.75	0.13 μm

1S20C5: Altera Stratix EP1S20F780C5; 1S10C5: Altera Stratix EP1S10F780C5
Slice: Contains two 4-input look-up tables; BRAM: Block Selected RAM (4 K-bit)

By using the S-Boxes, the outputs of the ROMs become c3, 04, c7, 23, respectively.

In Step 5*, 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)*

4. FPGA IMPLEMENTATION

We use the Altera Stratix FPGA to implement the AES encryption circuit. 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 3, in the round, the MixColumns and the AddRoundKey are realized as a network of XOR gates where each gate requires one Logic Element (LE). In the rolling part, the 16-byte cyclic shifter for the 4SM and selector for the 8SM are implemented by LEs. To implement 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 that contains 16 S-Boxes. As for the key expansion block, each round key circuit consists of 4 S-Boxes and 6 XOR gates where the S-box is realized by 208 LEs since this realization has a slightly higher throughput than using M4Ks for the S-Box of the key, as can be seen from the experimental results in later section.

We also implemented the unrolling architecture with the same FPGA for comparison. In this case, the SubBytes was implemented by M4Ks, the MixColumns and the AddRoundKey were implemented by XOR gates, while the ShiftRows was simply realized by hardwiring. For the part of the rolling implementation, we compared with the work of Helion [2], which achieves 1.4 Gbps throughput on the Altera Stratix-c5 FPGA.

5. PERFORMANCE AND COMPARISONS

We evaluated the performance of the AES-4SM and the AES-8SM, and compared with the unrolling implementation called UNROLLING (designed by us) and other published works.

The AES-4SM is implemented with rolling part 4SM, while the AES-8SM is implemented with rolling part 8SM. To compare the performance of different architectures, we implemented the AES-4SM, the AES-8SM and the UNROLLING on the same FPGA. For each implementation, first we designed the circuit by Verilog HDL, and then used Quartus II 4.1 for synthesis, place & route and timing analysis. Finally, we used Quartus II 4.1 simulator to test the logical operation and to do the worst-case timing analysis for the design in the target FPGA. The maximum clock rate (f_{clk}) was obtained by Quartus II 4.1 simulator.

In Table 2, the upper five rows show our implementations AES-4SM, AES-8SM and UNROLLING, and the lower five rows show the published works. The column "Device" denotes the FPGA 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 "Th." denotes the maximum **throughput** calculated by:

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

The column "Eff" shows the **memory efficiency** calcu-

lated by:

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

In the Altera Stratix Device family [9], EP1S10F780C5 has the minimum devices, 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-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. Compared with the UNROLLING that achieved 20.48 G-bit/s throughput on the same FPGA, 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 9.6% and 2.4%, respectively.

The AES-4SM can also be implemented on the EP1S10F780C5 by utilizing the M4Ks for the key expansion, but the throughput is slightly lower than that on the EP1S20F780C5. Note that the EP1S10F780C5 is too small for the UNROLLING. This is because in order to utilize both of the memory for key and the memory for core, it requires 100 copies of the M4K, which 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 only requires 60 M4Ks. We also implemented the AES-4SM and the AES-8SM on Cyclone II EP2C35 FPGA. The throughputs are 4.57 Gbps and 8.00 Gbps, respectively (not shown in Table 2). 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 the gap between the rolling implementation and the unrolling implementation.

Direct comparison among various FPGA implementations of the AES algorithms is difficult, since FPGA target devices are usually 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 (1).

Compared with the published unrolling implementations signified with "(unrolling)" in the column "Design", the memory efficiency of the AES-4SM is very close to the fastest implementation (UF10-PP3B). Note that the number of cycles for UF10-PP3B is 30. Besides, 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 published rolling implementations signified with "(rolling)" in the column "Design", both the throughputs and the memory efficiencies of the proposed implementations are much higher than the fastest rolling implementation (UF1-PP0B).

Both of the AES-4SM and the AES-8SM have much higher throughput than the software implementations. An AES rolling implementation achieves 1.538 Gbps on a 3.2 GHz Pentium4 processor [10] and a 640Mbps on a 1 GHz embedded processor [11].

Table 3 summarizes the features of the three different architectures: unrolling, PPR, and rolling. In the column "Memory efficiency", let the value of Helion [2] be 1, since it is approximately the same as the value of UF1-PP0B [6]. We can see that the PPR architecture offers a high memory efficiency and a medium to high throughput using medium memory area.

Table 3: Comparison of the different architectures

Architecture	Memory area	Throughput	Memory efficiency
Unrolling	Large	High	0.84∼1.68
PPR	Medium	Medium / High	1.50∼1.60
Rolling	Small	Low	1

6. CONCLUSIONS

In this paper, we presented the pipelined partial rolling (PPR) architecture for an AES encryption processor. We implemented two different designs: AES-4SM and AES-8SM on Altera Stratix EP1S20F780C5 FPGA using the PPR architecture. The AES-4SM achieves a throughput of 5.61 Gbps by using 20 M4Ks, and the AES-8SM achieves a throughput of 10.49 Gbps by using 40 M4Ks. Compared with the unrolling implementation that achieves a throughput of 20.48 Gbps by using 80 M4Ks on the same FPGA, the AES-4SM and the AES-8SM improve the memory efficiency by 9.6% and 2.4%, respectively, and reduce the amount of the memory by 75% and 50%, respectively. The PPR architecture fills the gap between unrolling and rolling architectures, and fits on less expensive FPGAs.

Acknowledgments

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

7. 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] http://www.altera.com

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

[11] 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.