A Rijndael Cryptoprocessor with On-the-fly Key Scheduler

Joon Hyoung Shim[1], Joo Yeon Bae[2], Yong Kyu Kang[2] and Jun Rim Choi[2]
[1] School of Information Security, Kyungpook National University,
1370 Sankyuk-Dong, Book-Gu, Daegu, Korea, 702-701
Tel. +82-53-940-8667, Fax. +82-53-950-5505
[2] School of Electrical Engineering, Kyungpook National University,
1370 Sankyuk-Dong, Book-Gu, Daegu, Korea, 702-701
Tel. +82-53-940-8667, Fax. +82-53-950-5505
e-mail : realface@palgong.knu.ac.kr

Abstract: We implemented a cryptoprocessor with a on-the-fly key scheduler which performs forward key scheduling for encryption and reverse key scheduling for decryption. This scheduler makes the fast generation of the key value and eliminates the memory for software key scheduler. The 128-bit Rijndael processor is implemented based on the proposed architecture using Verilog-HDL and targeted to Xilinx XCV1000E FPGA device. As a result, the 128-bit Rijndael operates at 38.8MHz with on-the-fly key scheduler and consumes 11 cycles for encryption and decryption resulting in a throughput of 451.5Mbps

1. Introduction

The National Institution of Standards and Technology (NIST) chose Rijndael as the Advanced Encryption Algorithm in October 2000 [1]. Generally, Rijndael has been implemented in software, but a software implementation cannot offer the physical security for the key. In this paper, Rijndael is implemented in hardware with on-the-fly key scheduler, which can make not only a forward scheduling for encryption but also a reverse scheduling for decryption. Therefore, it enhances the physical security and an outside attacker cannot easily modify it. In the first section, we describe Rijndael algorithm and how to generate the round keys for encryption and decryption respectively. In the second section, we propose the trade-off between a cost-effective architecture and fast processing speed for Rijndael. The on-the-fly key scheduler which generates round keys for encryption and decryption is described. Finally, we analyze the performance of the 128-bit Rijndael processor with on-the-fly key scheduler and compare with the other architectures in terms of processing time and chip area.

2. Rijndael Block Cipher

Rijndael is a symmetric block cipher. While other block ciphers have Fiestel structure, it has non-Feistel structure [3]. It supports key lengths of 128, 192, 256 bits and block sizes of 128, 192 or 256 bits. The number of round operation is determined by combination of key length and block size [1].

2.1 Rijndael encryption and decryption

Rijndael encryption is illustrated in Fig. 1. The algorithm consists of an initial round-key addition, the required number of round and a final round. Rijndael performs encryption by an iterative transformation called the "round transformation", of which inverse operation is used for decryption and the order of the round for decryption is reversed. The intermediate cipher result is known as the State. The State comprises of a rectangular array of bytes as 128-bit plaintext of 16-byte, B0, B1, B2, B3, ... B15, as shown in Fig. 2. Similarly, the key state is represented as a rectangular array of bytes as shown in Fig. 3 [4].

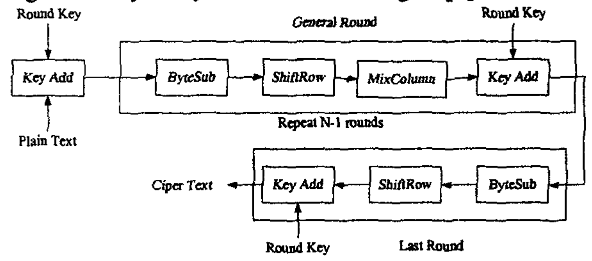

Fig. 1. Rijndael encryption

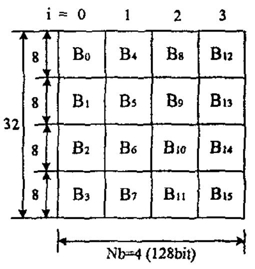

Fig. 2. Block State

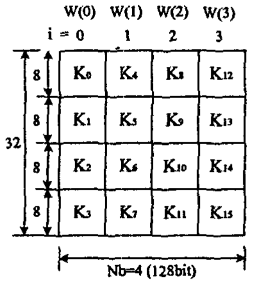

Fig. 3. Key State

Each round is composed of the following four operations:

1. The ByteSub transformation is a non-linear byte substitution, operating on each of the state bytes independently.
2. In ShiftRow, the rows of the state are cyclically shifted over different offsets. While elements in the first row are not shifted, elements in other row are shifted over different offsets, which depend on the block length.
3. In MixColumn, every column is transformed by multiplying it by a specific multiplication polynomial as described in [1].
4. Finally the round key is applied to the state by a simple bitwise XOR.

Decryption is realized by applying reverse transformation of each round. Fig. 4 shows Rijndael decryption.

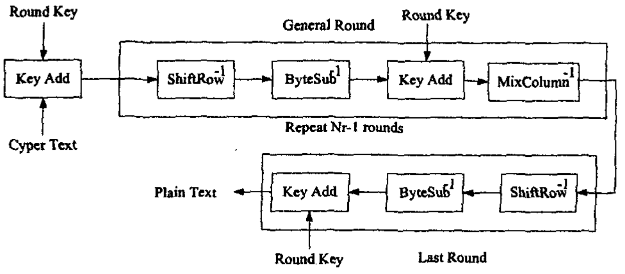

Fig. 4. Rijndael decryption

2.2 Key scheduling

Round keys are derived from the input initial key state(W(0),W(1),W(2),W(3)) using the following key scheduling process. Round key for the first round is the input cipher key itself. The round keys for each of the remaining rounds are generated from the previous round key. Key scheduling for encryption is illustrated in Fig. 5.

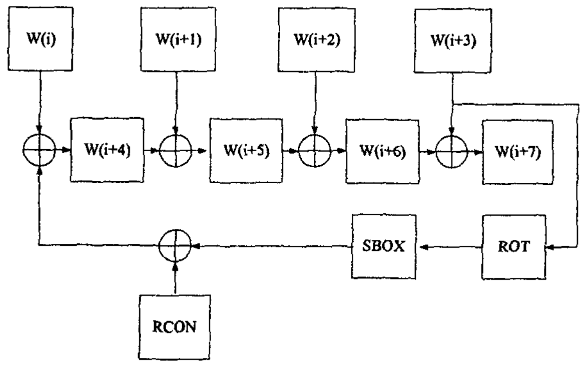

Fig. 5. Forward key scheduling for encryption

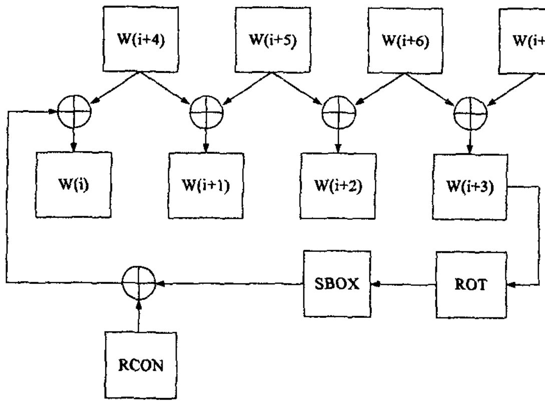

Fig. 6. Reverse key scheduling for decryption

Round key is partitioned into four words in key state: W(i), W(i+1), W(i+2), W(i+3). W(i) is the first column of key state in (i)th round, where i = 0,1,2 ... Nr, and Nr is the number of rounds. W(i+4) is obtained by XORing W(i) with the result of XORing W(i+3) with the corresponding RCON constant after being rotated by one byte yielding the address into S-box, where i = 0,4,8... (i < Nr). W(i+5),W(i+6),W(i+7) are derived by XORing the previous word with W(i+1), W(i+2), W(i+3) respectively. In this manner, (i+1)th round key is generated from (i)th round. Fig. 6 shows the reverse key scheduling for decryption. Similarly, (i)th round key is generated from (i+1)th round key, where i =Nr, Nr-1, Nr-2, ... 0. W(i+1), W(i+2), W(i+3) are obtained by XORing W(i+4) with W(i+5), W(i+5) with W(i+6), W(i+6) with W(i+6) respectively. But W(i) is obtained by XORing W(i+4) with the result of XORing W(i+3) with the corresponding RCON constant after being rotated by one byte yielding the address into Sbox

3. FPGA Implementation

3.1. Rijndael encryption and decryption

The cost-effective architecture is illustrated in the Fig. 7. The corresponding data paths for encryption & decryption are shown respectively. Control signals and seven multiplexers control data paths. In this paper, the 128-bit data bus is chosen for transformation operations, which are performed by sixteen 8-bit elements of a state.

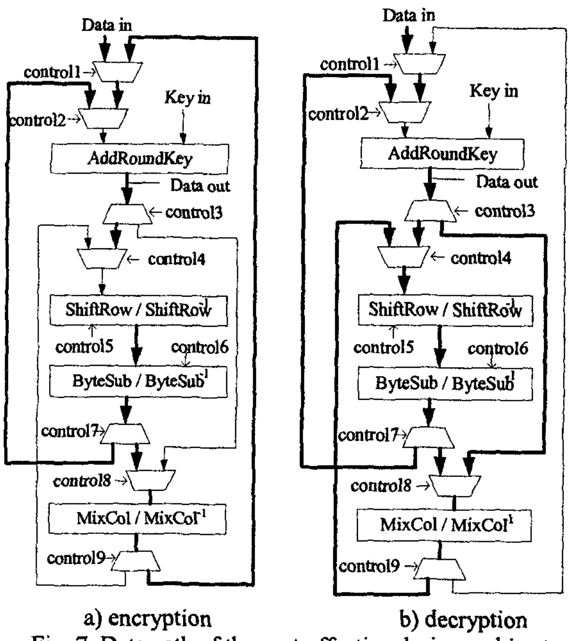

Fig .7. Data path of the cost-effective design architecture

In the AddRoundKey, a round key is applied to the state by a simple bitwise XOR. There are 11 key additions during operation for Nr=10(Nb=4, Nk=4). For encryption, first key addition involves XORing of input key with the plaintext. The second through tenth key addition involves XORing of the round key with the MixColumn output for encryption and the inverse of the ByteSub output for decryption. The final key addition involves XORing of the final round key with the output of the ByteSub for encryption and the inverse of ByteSub for decryption respectively.

The ShiftRow/ShiftRow-1 and ByteSub/ByteSub-1 operations can be exchangeable. So we can reuse these 2 encryption blocks in a decryption. This approach reduces the hardware requirement.

The ShiftRow/ShiftRow-1 operation is implemented using mux and wiring as shown in Fig 8. Odd number lines need mux to separate encryption and decryption. In the odd number lines, dotted lines are for decryption and solid lines are for encryption. But even number lines don't need mux because encryption path and decryption path are equal.

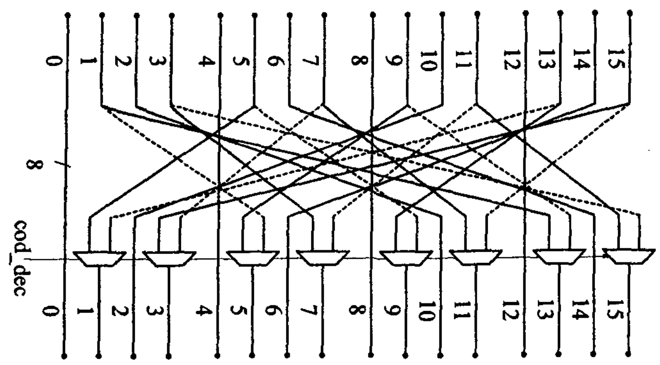

Fig. 8. ShiftRow/ the inverse of ShiftRow

For the ByteSub/ByteSub-1, the Sbox/Sbox-1 is implemented by 256x8 ROM, of which the input is an 8-bit address and the output is an 8-bit data. For processing 128-bit data, we use sixteen asynchronous look-up table ROMs for encryption and decryption. ByteSub/ByteSub-1 operation is illustrated in Fig.9.

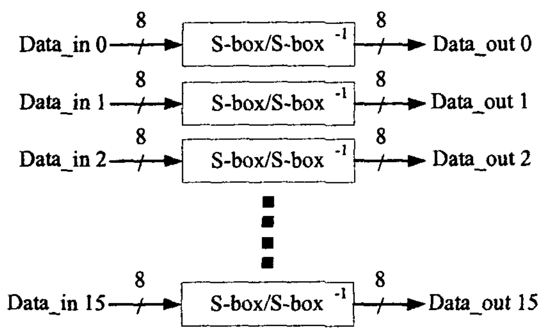

Fig. 9. ByteSub/ the inverse of ByteSub

Unlike ShiftRow and ByteSub operation, the MixColumn/MixColumn-1 needs 4-byte input to compute 1-byte output as shown in Fig. 10 and Fig 11. So it needs 4 MixColumn/Mixcolumn-1s to process 128-bit data. Four inputs a3, a2, a1, a0 are multiplied with fixed constants, which are from the following fixed polynomials (1), (2) :

$c(x)='03'x^3+'01'x^2+'01'x+'02'$ for encryption. (1)

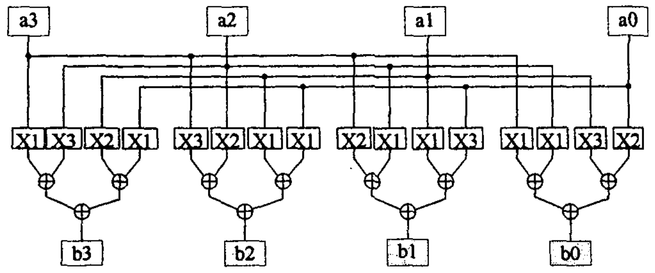

Fig. 10. MixColumn

$c(x)='0B'x^3+'0D'x^2+'09'x+'0E'$ for decryption. (2)

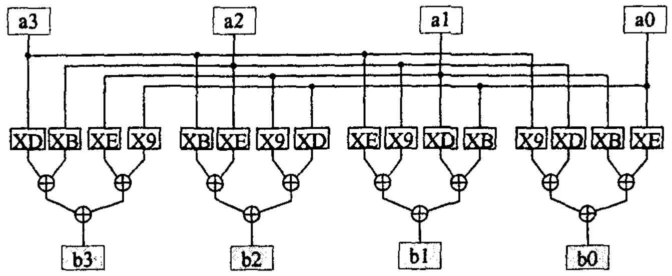

Fig. 11. The inverse of MixColumn

3.2. On-the-fly key scheduler

The on-the-fly key scheduler is designed to schedule key values reverse and forward for encryption and decryption. The forward key scheduling and the reverse key scheduling are illustrated. Fig. 12 shows the top block of on-the-fly key scheduler for forward and reverse scheduling.

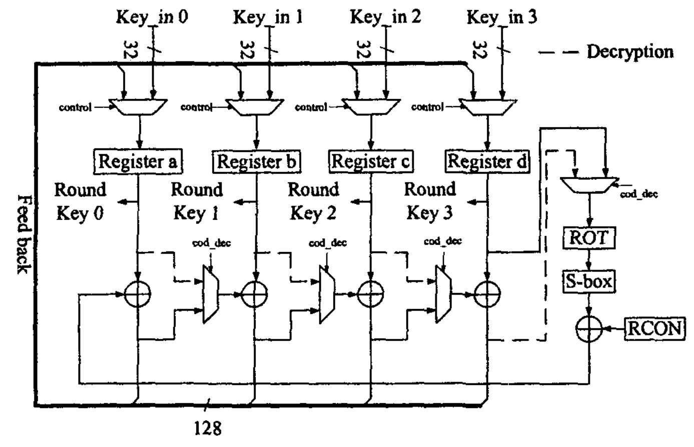

Fig. 12. On-the-fly key scheduler for forward & reverse scheduling.

Four 32-bit keys are stored in register a, b, c and d. For forward scheduling, the word in the register d goes through ROT, S-box and XOR with RCON. And it generates the next four 32-bit words. For reverse scheduling, the last round key should be generated with forward scheduling at the first time. With the last round key, we can make reverse round key. The current keys are in the registers a, b, c and d. They generate the next under 96-bit key and 32-bit key from register c, d goes through ROT, S-box and XOR with RCON. With this 32-bit value, current 32-bit key in the "register a" generate the next upper 32-bit key. It is illustrated as the dotted line in the Fig.12.

3.3. I/O Implementation

Fig. 13 shows Rijndael top block with interface I/O block for 32-bit data bus. It has the function of updating key value. Initial round key is stored to key buffer and the stored initial round key is used repeatedly for encryption.

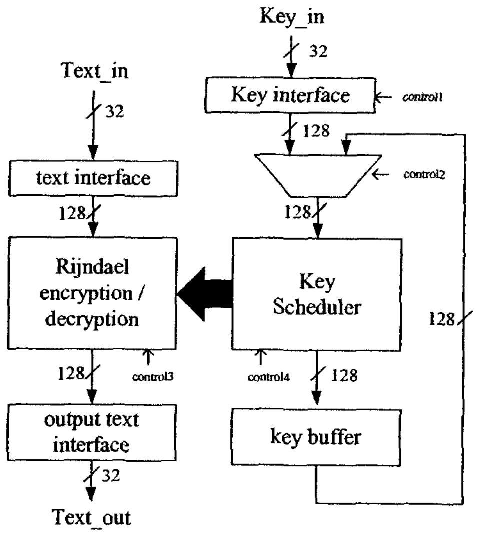

Fig 13. Rijndael Top Block

For decryption, the last round key is the start key of operation. First, the last round key value for encryption is generated. The last round key for encryption is used as the initial round key for decryption. Similarly it is reused for the repeated decryption.

4. Performance and Comparison

Table 1 shows the performance evaluation for other architectures for the encryptor only in terms of processing and hardware cost. The architecture types (loop unrolling (LU), full or partial pipelining (PP), and partial pipelining with sub-pipelining (SP)) are listed along with the number of stages and sub-pipelining stages; LU-1 implies a loop unrolling architecture with one round, while SP-1-1 implies pipelined architecture with one stages and one sub-pipelining and PP-2 implies partial pipelining with two stage [2].

Table 1. Performance evaluation for the encryptor

Architecture	Slices	Clock Frequency (MHz)	Latency (cycles)	Throughput (Mbit/s)
LU-1[2]	3,488	24.9	11	290.1
PP-2[2]	5,275	24.7	5.5	575.3
SP-1-1[2]	3,540	40.0	10.5	487.7
* Our Design	2,580	38.8	11	451.5

The core performance of our design architecture of Rijndael is shown in table1. It needs fewer slices than another architecture but has 451.5Mbps throughput. When it uses input and output interface, the total throughput is depend on the interface block. And decryption requires more cycles than encryption because it needs pre-scheduling to generate the last key value.

Fig. 14. Verification of Rijndael crypto-processor

Fig. 14 shows the setup for verification of Rijndael cypto-processor on the board level test. First, Verilog HDL simulation of the Rijndael was compared to the test vectors provided in the AES submission package [1]. The simulation results are verified. It is targeted to Xilinx XCV1000E FPGA series, and then the board level test was completed.

5. Conclusion

In this paper, we implemented a cost-effective Rijndael crypto-processor for encryption and decryption. We suggest the on-the-fly key scheduler, which performs forward key scheduling for encryption and reverse key scheduling for decryption. It supports the fast generation of key values. As a result, the memory elimination of software key scheduler provides a small chip area and a high physical security. We perform the encryption and decryption of Rijndael on a chip while scheduling the round keys. This architecture has 16-byte-oriented operation internally. It supports up to 451.5Mbps for encryption and decryption when it is targeted to Xilinx's Vertex XCV1000E.

6. Acknowledgement

This work is supported by IC Design Education Center (IDEC) in Korea.

References

[1] T.A. Jones, "Writing a good paper," *IEEE Trans. on General Writing*, Vol. 1, no. 2, pp.1-10, May 2002.
[2] K. Hwang, *Computer Arithmetic*, John Wiley, 1997.

[1] Joan Daemen and Vincent Rijmen, "The Rijndael Block Ciper," *AES Proposal*, ver.2, March 1999
[2] Adam J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, " An FPGA-Based Performance Evaluation of the AES Block Ciper Candiate Algorithm Finalists", *IEEE Transactions on VLSI System*, Vol.9, NO. 4, August 2001.
[3] Sanchez-Avila, C., Sanchez-Reillo, R, "The rijndael block ciper(AES proposal): a comparison with DES", *Security Technology, 2001 IEEE 35th International Carnahan Conference on, Oct 2001*, pp. 229-234
[4] Maire McLoone, John V McCanny, "Rijndael FPGA Implementation Utilizing Look-Up Tables", *Signal Processing Systems, 2001 IEEE Workshop on, 2001* pp. 349-360