A New Compact Architecture for AES with Optimized ShiftRows Operation

Hua Li and Jianzhou Li
Department of Mathematics and Computer Science
University of Lethbridge Canada T1K 3M4
huali@cs.uleth.ca, jianzhou.li@uleth.ca

***Abstract*— In this paper we present a new compact iterative architecture of 32-bit datapath for the AES block cipher. We propose a new way of implementing ShiftRows and InvShiftRows operation, which is realized by the simple enable function of registers and five 2-to-1 multiplexors of eight bits length. A new compact key generation unit of 32-bit datapath is also proposed to generate round keys on-the-fly for both encryption and decryption. The hardware resource is maximally shared for encryption and decryption. The implementation requires 9843 gate equivalents and provides a throughput of 247 Mbit/s on the CSMC's 0.35 μm CMOS technology. The comparison with the best previous work shows that it has better throughput and area performance parameters.**

Keywords: AES, Compact architecture, Cryptography, ASIC

I. Introduction

The Advanced Encryption Standard (AES) algorithm, developed by Joan Daemen and Vincent Rijmen, finalized in 2001 by the National Institute of Standards and Technology (NIST) [1], is in a state of constant improvement considering hardware implementations. Many ASIC [2], [3], [4], [5], [6], [7] and FPGA [8], [9], [10], [11], [12], [13], [14] implementation of the AES have been presented to achieve a better performance. However, most of them focus on the performance of higher speeds, which consume a lot of hardware resources. To the authors' knowledge, very little literature concentrates on the compact AES architecture. Chodowiec et al. [13] presented a very compact FPGA implementation of the AES algorithm, and Mangard et al. [2] described a highly regular and scalable AES architecture on ASIC.

In this paper, since we are interested in the compact architecture, an iterative 32-bit datapath is implemented. It utilizes the hardware resource sharing and integrates encryption and decryption functions together. Inv-/ShiftRows operation is a critical design that directly influences the overall architecture of a compact 32-bit datapath. Chodowiec et al. [13] present a very clever method to utilize two dedicated 128-bit dual-port RAMs or shift registers in FPGA to complete the ShiftRows/InvShiftRows. In this paper, we propose a new ASIC implementation of ShiftRows/InvShitRows, which only requires 16 8-bit registers and five additional 8-bit multiplexors. A new compact 32-bit datapath implementation of the key generation that can provide round keys on-the-fly for both encryption and decryption is also proposed. This provides flexibility for applications where the cipher keys are frequently changed to enhance the security. Moreover, the compact key generation unit for 128-bit cipher key can easily to extend to implement 192-, and 256-bit cipher key algorithms. Finally, a small size of 9843 gate equivalents is obtained for the whole implementation in our design using CSMC's 0.35 μm technology with 247Mbit/s throughput. It is 100 percent faster than the implementation in [2] while using slightly less equivalent gates. Our compact ASIC design provides more flexibility, a lower cost, and a higher throughput for the applications in the constrained embedded systems.

II. Design of the Architecture for AES

A. Primitive operations in AES

AES is a symmetric block cipher that performs four individual transformations, i.e., SubBytes, ShiftRows, MixColumns and AddRoundKey, on a 128-bit block of data for a certain number of repetitions for encryption, or InvSubBytes, InvShiftRows, InvMixColumns, and AddRoundKey for decryption. We only implement the 128-bit cipher key algorithm in this paper, which requires 10 rounds. All the intermediate results of the 128-bit block as well as the input and the output block are called states. The most intuitive way for AES operation is to picture each state as a 4×4 matrix of bytes which are filled in the matrix column by column, as shown in Fig. 1.

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Fig. 1. The sequence of bytes in each state

We capsule one complete round with the four operations in Fig. 2.

- SubBytes operation is a nonlinear substitution that performs on each byte of the state using S-box which contains a permutation of all possible 256 8-bit values. InvSubBytes uses the inverse S-box.
- The Inv-/ShiftRows operation is the cyclic shifting of each row of the state to the left on encryption or to the right on decryption over different numbers of bytes.
- MixColumns operation treats each column of the state as a four-term polynomial over $GF(2^8)$ and transforms each

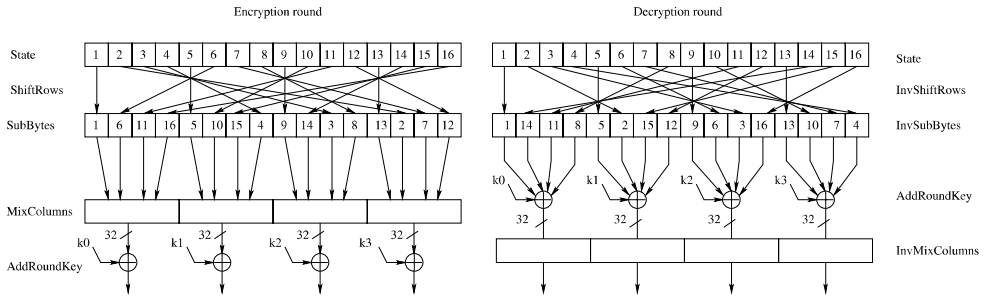

Fig. 2. One AES encryption/decryption round

column to a new one by multiplying it with a constant polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ modulo $x^4 + 1$. The inverse MixColumns operation is a multiplication of each columns with $b(x) = a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$ modulo $x^4 + 1$.

- The AddRoundKey operation is only a simple bitwise XOR of the current state with the round key that is generated by the key expansion. XOR operation is its own inverse.

Fig. 2 features the parallelism and resource sharing of the four operations. The Inv-/S-box operates on the 16 eight bits of data independently; Inv-/MixColumns requires four 32-bit columns of data at the same time; whereas Inv-/ShiftRows involves the whole 128 bits. Similar description is also given in [13]. The datapath of 32 bits is chosen considering the comparatively high speed and low area that uses only about 1/4 resource of one round implementation with a 128-bit data width. It takes four clock cycles to complete one round. From Fig. 2, it can be seen that each round for encryption or decryption looks very similar except that the sequence of the MixColumns and AddRoundKey is different with that of the AddRoundKey and InvMixColumns. So we can re-use the hardware resources of encryption and decryption operations as more as possible. Fig. 2 also shows that the sequence of the ShiftRows and SubBytes can be switched without changing the result. We utilize this feature to make the design concise-looking.

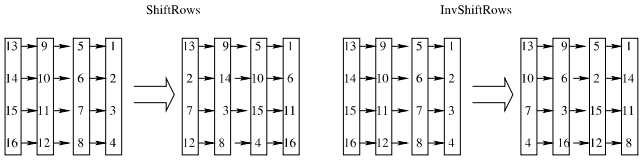

Fig. 3. Inv/-ShiftRows transformation

B. Data Unit

Focusing on the compact AES architecture, we use the basic iterative architecture [9] with a 32-bit datapath. It implements only 1/4 of one round of the original 128-bit data width and re-uses the same resource iteratively four times to complete one round, and 44 times to complete the whole encryption or decryption process. The data unit is shown in Fig. 4, which follows the structure shown in Fig. 2.

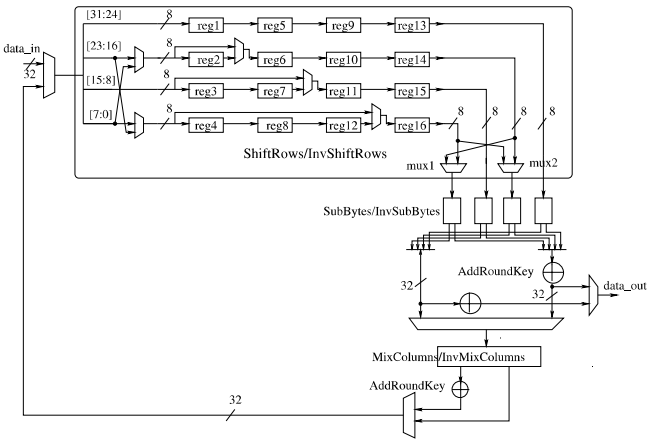

Fig. 4. Datapath Unit

1) Implementation of ShiftRows and InvShiftRows: The ShiftRows and InvShiftRows operation becomes complicated in the 32-bit datapath implementation since it involves the whole 128-bit operation. So at least 32 8-bit registers are required to hold the current 128-bit block of data (or state) in order to prevent the replacement by next state before the the ShiftRows or InvShiftRows operation is completed. The straightforward implementation of the ShiftRows operation is to use two 128-bit registers with simple wiring as shown in Fig. 2. This is further simplified by the architecture that uses one 128-bit register, one 96-bit register, and corresponding multiplexors as pointed out in [13]. Chodowiec et al. [13] propose to utilize two 128-bit dual-port RAMs or shift registers to complete the ShiftRows and InvShiftRows in dedicated FPGA devices. In this paper, we propose a different way to implement Inv-/ShiftRows, which only requires 16 8-bit registers and a few 8-bit multiplexors. This design can be easily implemented in ASIC. We re-write the processes of ShiftRows and InvShiftRows in Fig. 3. The block of data is shifted column by column. Let us take a look at the ShiftRows as an example.

1) In the first row, it is just a sequential shifting in the sequence of 1, 5, 9, 13;
2) In the second row, the first byte 2 is delayed to the last one;

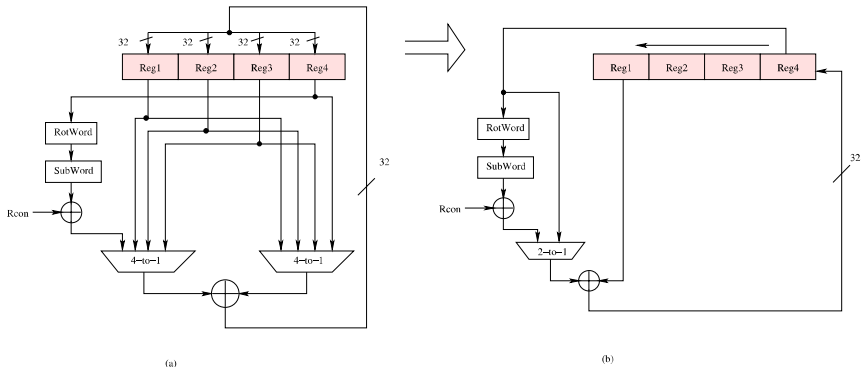

Fig. 5. The round key generation for encryption

3) In the third row, the first two bytes 3 and 7 are delayed behind bytes 11 and 15;
4) In the fourth row, the byte 16 shifts in last while shifts out first. The other three bytes still keep the same sequence.

We add three multiplexors and use the clock enable function of registers in position 2, 3, 7, 4, 8, and 12 to easily implement the above sequences (see Fig. 4). Take the third row for an instance, the clock enable signals of registers reg3 and reg7 are enabled for the first two clock cycles, so byte 3, and byte 7 are sequentially shifted into reg3 and reg7; in the next two clock cycles, the clock enable signals are disabled, and the data in reg3 and reg7 are unchanged, while the byte 11 and 15 are shifted into reg11 and reg15 sequentially via a multiplexor. In this way, the ShiftRows operation is completed naturally. Comparing the operations of ShiftRows and InvShiftRows, we further find that the hardware structure in the second row for ShiftRows is the same as that in the fourth row for InvShiftRows. Similarly, the hardware structure required in the fourth row for ShiftRows is the same as the second row for InvShiftRows. So we can share the two rows of 8-bit registers for both encryption and decryption by only adding two 2-to-1 multiplexors to control which row should enter according to encryption or decryption and two multiplexors for leaving. Since the SubBytes/InvSubBytes operation has its own multiplexors to distinguish encryption and decryption, these multiplexors can be used to replace the two multiplexors mux1 and mux2 leaving the ShiftRows/InvShiftRows operation. This means only five additional 8-bit multiplexors are required to implement ShiftRows/InvShiftRows operation besides the 16 8-bit registers. Our proposed design also requires less hardware resources compared with those in [6], [2], which use the same 16 8-bit registers but clearly more 8-bit multiplexors to re-select the input data for each register.

2) Implementation the Inv-/SubBytes and Inv-/MixColumns operations: We implement S-box and inverse S-box using the efficient combinational circuit proposed by Wolkerstorfer et al. [5]. In this method, the finite field arithmetic in $GF(2^8)$ is transformed into the equivalent one in $GF(2^4)$ which is more suitable for hardware implementation. In the implementation, both the encryption and decryption share the common multiplicative inverse operation. Chodowiec et al. [13] implements the Inv-/S-box operation by look-up table method utilizing the dedicated dual-port RAMs provided by FPGA. Even though the look-up table method can reduce the critical path, therefore getting higher frequency compared with combinational circuit method in which the Inv-/SubBytes occupies almost a half of the critical path, ROM is much difficult to implement on ASIC technique.

For MixColumns and InvMixColumns, we compare the methods used in [13] and [6]. Both the methods implement the more complicated InvMixColumns operation by sharing the simple MixColumns operation. We implement the former one in our design which requires less hardware resource after comparison.

C. Key Generation Unit

The key generation expands the initial 128-bit cipher keys to generate the key schedule with the length of $11 \times 4 \times 32$ bits. The 128-bit round keys are derived iteratively from the cipher key for encryption, and they are used in reverse order for decryption. Two methods for the key expansion are commonly used, i.e., the round keys can be generated on-the-fly with the data transformation, or they are pre-calculated and stored for later use [7], [13]. In this paper, the round keys applied to the data transformation for encryption or decryption are calculated on-the-fly. The agility of the key expansion deal with the situation that the cipher keys are changed frequently.

The straightforward implementation of the key generation for encryption is illustrated in Fig. 5 (a). Every word (32 bits) of the next state is the XOR of the current word in the same position with its left neighboring word. For example, the word in Reg2 is calculated as $w'[reg2] = w[reg2] \oplus w[reg1]$. For words in the position Reg1, its neighboring word is in position Reg4. A transformation SubWord(RotWord()) is applied to the word in position Reg4 prior the XOR, followed by an XOR with the round constant Rcon. We further simplify the implementation as shown in Fig. 5 (b). The two 4-to-1 multiplexors are removed. The four 32-bit registers shift left in each clock cycle to complete the XOR of two neighboring

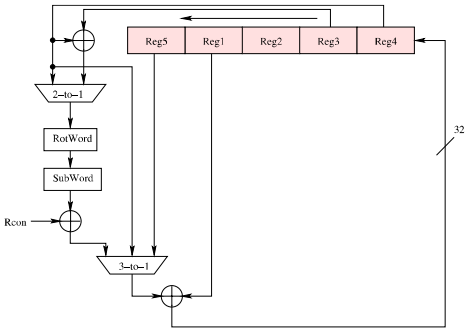

Fig. 6. The final key generation unit for both encryption and decryption

words. It takes 44 clock cycles to generate the round key words of $11 \times 4 \times 32$ bits in length. Similarly, the round keys for decryption can also be implemented in the compact mode. Fig. 6 shows the final datapath of the key generation integration for both encryption and decryption. We add an additional 8-bit register Reg5 to hold the most significant byte for one more clock cycle on decryption. The least significant two bytes from Reg3 and Reg4 are required XOR before entering the RotWord function. Chodowiec et al. [13] precomputed all the round keys using a similar implementation to Fig. 5 (b) and stored in a Block RAM for key schedule, while our compact key schedule architecture of 32-bit datapath further expand to generate round keys for both encryption and decryption simultaneously. Finally, our proposed compact key generation unit can be easily extended to process 192-bit and 256-bit cipher key that are specified in the AES standard.

III. Performance evaluation and comparison

Our design was implemented on a 0.35-μm CMOS process using the CSMC library, which requires 9843 gate equivalents and obtains the maximum frequency of 85Mhz.

To our knowledge, the best compact implementations of AES algorithm in ASIC and FPGA are [2] and [13] respectively. However, the FPGA implementation is not the ultimate choice in the resource constrained embedded system such as wireless communication. Here we only compare our performance with [2] in Table I, where both use the ASIC technique. From the table we can see that our design shows a better performance, which is 100 percent faster than the implementation in [2] while using slightly less equivalent gates. Even if different processes are used in the two papers, our performance still has advantage if evaluated in the same frequency. This is mainly due to that our design has a shorter critical path. The data unit in [2] is essentially the similar 32-bit datapath design. And both design use combinational circuit proposed in [5] to implement Inv-/Sbox. However, the Inv-/ShiftRows operation in [2] are implemented via four 4-to-1 8-bit multiplexors. And it requires extra 16 3-to-1 8-bit multiplexors to re-select the input data of 16 8-bit cells. Our design only use the enable function of registers and five additional 2-to-1 8-bit registers to complete the same operation. Besides that, our key generation unit is more compact which is of only 32-bit datapath, whereas it is a 128-bit operation in [2].

TABLE I
Performance comparison with other compact cores

Parameters	Ours	Mangard et al. [2]
Throughput (Mbit/s)	247	128
Clock cycles	44	64
Freq. (MHz)	85	64
Area (gate equivalents)	9843	10799
Process	0.35 μm	0.6 μm

IV. Conclusion

A new compact 32-bit architecture for AES with optimized Inv-/ShiftRows operation is presented in this paper. The novel compact on-the-fly key schedule for both encryption and decryption is also proposed. The ASIC implementation result shows that it has better performance of throughput and area parameters. Moreover, the design is flexible for applications in the resource constrained embedded system.

References

[1] NIST, "Federal information processing standard 197: The Advanced Encryption Standard (AES)," 2001. available: http://csrc.nist.gov/publications/fips/fips197/fips197.pdf.

[2] S. Mangard, M. Aigner, and S. Dominikus, "A highly regular and scalable AES hardware architecture," *IEEE Transactions on Computers*, vol. 52, pp. 483–491, 2003.

[3] N. Kim, T. Mudge, and R. Brown, "A 2.3Gb/s fully integrated and synthesizable AES Rijndael core," in *Custom Integrated Circuits Conference (CICC)*, (San Jose, CA), Sept. 2003.

[4] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and performance testing of a 2.29 Gb/s Rijndael processor," *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 569–572, Mar. 2003.

[5] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES SBoxes," in *CT-RSA 2002, LNCS2271*, pp. 67–78, 2002.

[6] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-Box optimization," in *Advances in Cryptology, ASIACRYPT*, pp. 239–254, 2001.

[7] F. K. Gurkayna, A. Burg, N. Felber, W. Fichtner, D. Gasser, and et al., "A 2 Gb/s balanced AES crypto-chip implementation," in *GLSVLSI'04*, Apr. 2004.

[8] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transation of VLSI Systems*, vol. 9, pp. 545–557, Aug. 2001.

[9] P. Cholowiec, P. Khuon, and K. Gaj, "Fast implementations of secret key block ciphers using mixed inner- and outer-round pipelining," in *Proceedings of the International Symposium on Field Programmable Gate Arrays*, Feb. 2001.

[10] M. McLoone and J. McCanny, "High performance single-chip FPGA Rijndael algorithm implementation," in *Cryptographic Hardware and Embedded Systems CHES 2001, LNCS 2162*, pp. 65–76, Springer-Verlag, 2001.

[11] N. Weaver and J. Wawrzynek, "High performance, compact AES implementations in Xilinx FPGA," http://www.cs.berkeley.edu/∼nweaver/sfra/rijndael.pdf.

[12] M. Mcloone and J. McCanny, "Generic architecture and semiconductor intellectual property cores for advanced encryption standard cryptography," *IEE Proceedings of Computer Digit Tech.*, vol. 150, July 2003.

[13] P. Chodowiec and K. Gaj, "Very compact FPGA implementation of the AES algorithm," in *Cryptographic Hardware and Embedded Systems-CHES 2003, LNCS 2779*, pp. 319–333, Springer-Verlag, 2003.

[14] 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 *Cryptographic Hardware and Embedded Systems-CHES 2003, LNCS 2779*, pp. 334–350, Springer-Verlag, 2003.