A new compact dual-core architecture for AES encryption and decryption

Une nouvelle architecture compacte à double noyau pour le chiffrage et le déchiffrage de l'AES

Hua Li and Jianzhou Li*

This article presents a new compact architecture, consisting of two independent cores that process encryption and decryption simultaneously, for the Advanced Encryption Standard (AES) algorithm. The corresponding new compact key generation unit with 32-bit datapath is also explored to provide round keys on the fly for encryption and decryption. A novel way to implement ShiftRows/InvShiftRows, one of the key designs in the compact 32-bit architecture, is proposed. The new AES implementation requires only 16 629 gate equivalents on the $0.35\,\mu$m CMOS technology from CSMC Technologies Corporation, while providing encryption and decryption in parallel with 335 Mbits/s throughput.

Cet article présente une nouvelle architecture compacte, composée de deux noyaux de processus de cryptage et de décryptage simultanés, pour la norme avancée de chiffrage standard (*Advanced Encryption Standard*, AES). La nouvelle unité compacte correspondante de génération de clés avec un trajet de données de 32 bits est également étudiée pour prévoir des rondes de clés à la volée pour le chiffrage et le déchiffrage. Une nouvelle manière de mettre en oeuvre ShiftRows/InvShiftRows, une des conceptions principales dans l'architecture compacte de 32 bits, est proposée. L'exécution de la nouvelle implémentation AES nécessite seulement 16 629 équivalents de portes sur la technologie CMOS $0.35\,\mu$m de CSMC Technologies Corporation, tout en offrant le chiffrement et le déchiffrement en parallèle avec 335 Mbits/s de débit.

Keywords: AES; ASIC; compact architecture; dual cores

I Introduction

The Advanced Encryption Standard (AES) algorithm, developed by Joan Daemen and Vincent Rijmen and finalized in 2001 by the National Institute of Standards and Technology (NIST) [1]–[2], is in a state of constant improvement to accommodate hardware implementations. Many application-specific integrated circuit (ASIC) implementations [3]–[8] and field-programmable gate array (FPGA) implementations [9]–[14] of the AES have been presented to offer better performance. However, most of them focus on achieving higher speeds, which consumes substantial hardware resources. To our knowledge, only a very small portion of the literature concentrates on the compact AES architecture: [12] presents a very compact FPGA implementation of the AES algorithm, and [3] describes a highly regular and scalable AES architecture achieved by implementing ASICs.

In this paper, we propose a new compact dual-core architecture for the AES that can perform encryption and decryption simultaneously. We unroll one iterative round loop with a 32-bit datapath, which means that four clock cycles are required to complete one transformation round of a 128-bit block of data. We also present a new method for implementing ShiftRows/InvShiftRows, which are critical operations that directly affect the architecture of the compact 32-bit datapath design. In addition, a compact key generation unit that generates subkeys for encryption and decryption on the fly is proposed. The proposed architecture can be easily extended to 192- and 256-bit cipher key implementations. Our design is described in Verilog HDL and provides a high level of performance with only 16 629 gate equivalents on the $0.35\,\mu$m CMOS process from CSMC Technologies Corporation. It also provides flexibility for embedded system applications in which cipher keys are changed frequently to improve security, and in which security for both encryption and decryption is required; for example, real-time dual-duplex wireless communication.

*Hua Li and Jianzhou Li are with the Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta T1K 3M4, Canada. E-mail: hua.li@uleth.ca.

II Design of the AES architecture

II.A Primitive operations in AES

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

We present one complete round with the four operations in Fig. 1.

- The SubBytes operation is a nonlinear substitution that is performed on each byte of the state using S-box, which contains a permutation of all 256 possible 8-bit values. InvSubBytes uses the InvS-box.
- The ShiftRows/InvShiftRows operations are the cyclic shifting of each row of the state to the left for encryption and to the right for decryption over different numbers of bytes.
- The MixColumns operation treats each column of the state as a four-term polynomial over $GF(2^8)$ and transforms each column to a new one by multiplying it by a constant polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ modulo $x^4 + 1$. The InvMixColumns operation consists of a multiplication of each column by $b(x) = a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$ modulo $x^4 + 1$.

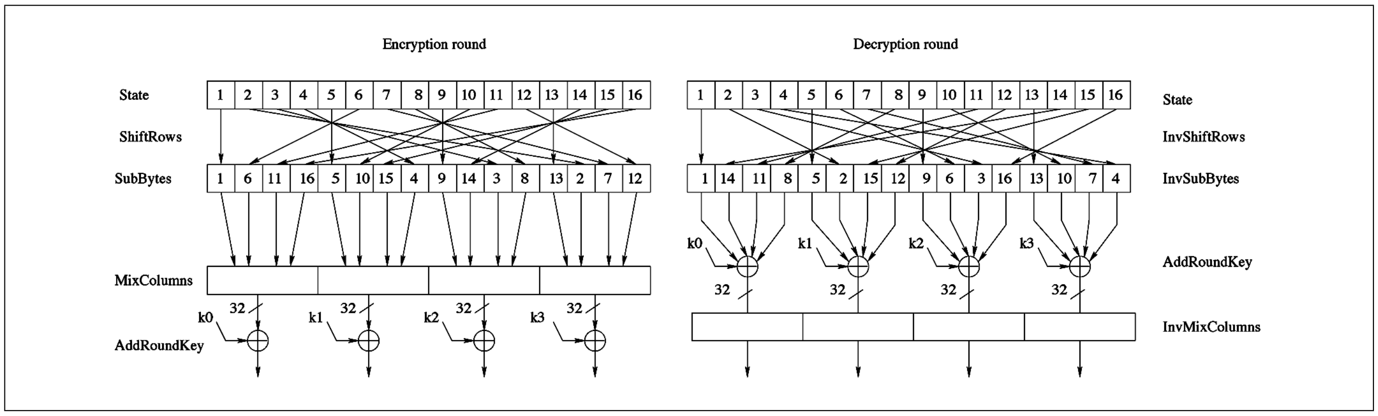

Figure 1: *One complete AES encryption/decryption round.*

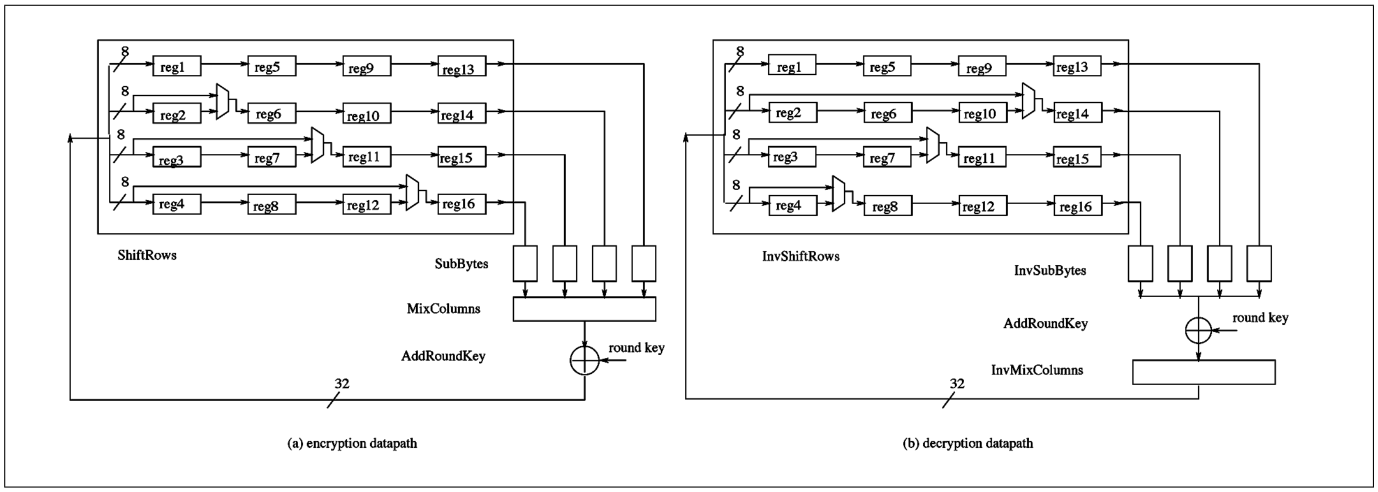

Figure 2: *Datapath of data unit for encryption and decryption.*

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

Fig. 1 features the parallelism in one round of 128-bit data length with resource sharing between encryption and decryption of the four operations and their respective inverses. S-box and InvS-box operate on the 16 8-bit groups of data independently, MixColumns and InvMixColumns require four 32-bit columns of data at the same time, whereas ShiftRows and InvShiftRows involve the whole 128 bits. A similar description is also given in [12]. The 32-bit datapath is chosen for its comparatively high speed and low area usage. It requires only a quarter of the resources needed for a one-round implementation with a 128-bit data width, although it takes four clock cycles to complete one round. From Fig. 1, we can see that each round for encryption and decryption looks very similar except that the sequence of the MixColumns and AddRoundKey operations is reversed. Therefore we can re-use the hardware resources of the encryption and decryption operations to a great extent. Fig. 1 also shows that the sequence of the ShiftRows and SubBytes opreations can be switched without changing the result of the round. We use this feature to develop a concise design.

II.B Data unit

Focusing on the compact AES architecture, we use the basic iterative architecture [10] with the 32-bit datapath. This architecture implements only a quarter of one normal round of 128-bit data width and re-uses the same resource in four consecutive clock cycles to complete one round of the encryption or decryption process. The data unit is shown in Fig. 2, which follows the structure shown in Fig. 1.

II.B.1 Implementation of ShiftRows/InvShiftRows operations

The ShiftRows and InvShiftRows operations involving the whole 128-bit operation become complicated in the 32-bit datapath implementation, due to the fact that the 128-bit block of data (or state) needs to be stored, and each byte in the current state should be kept until each row can perform a shift operation before the next state replaces its content. Several different ShiftRows designs have been presented for the compact architecture [12]. Reference [12] utilizes two 128-bit dual-port RAMs or a shift-register-based implementation to complete the ShiftRows. The other architectures for a folded implementation use one 128-bit register, one 96-bit register, and corresponding multiplexors instead of the straightforward implementation, which uses two 128-bit registers with simple wiring. In this paper, we propose a different way to implement ShiftRows that requires only 16 8-bit registers and some 8-bit multiplexors. We rewrite the ShiftRows and InvShiftRows processes as shown in Fig. 3. The block of data is shifted column by column, beginning with the most significant bytes. Let us first look at the ShiftRows:

1. In the first row, only sequential shifting occurs.
2. In the second row, the first-byte data in position 2 is delayed to the last position.
3. In the third row, the first two bytes in positions 3 and 7 are delayed until after the sequence of positions 11 and 15;
4. In the fourth row, the byte in position 16 shifts in last while shifting out first.

We add three multiplexors and the Clock Enable function to the registers in positions 2, 3, 7, 4, 8, and 12 to easily implement the above sequences (see Fig. 2). Consider the third row, for example: the Clock Enable signals of registers reg3 and reg7 are set for the first two clock cycles, and bytes 3 and 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

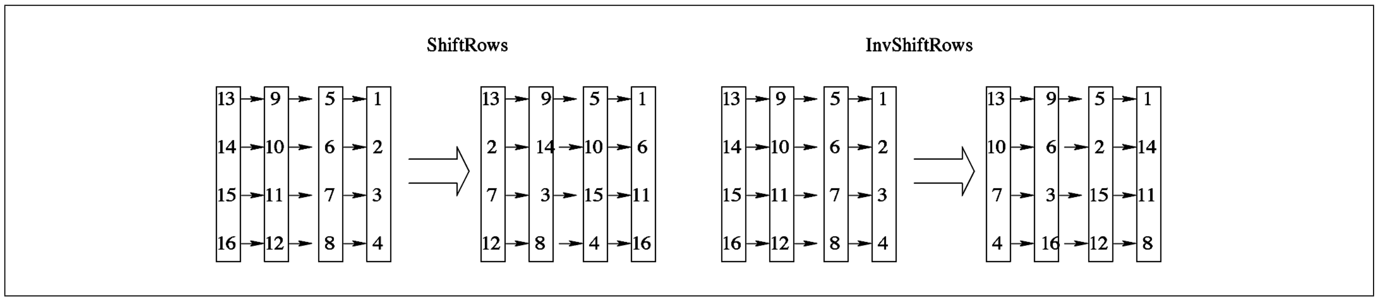

Figure 3: *ShiftRows/InvShiftRows transformation.*

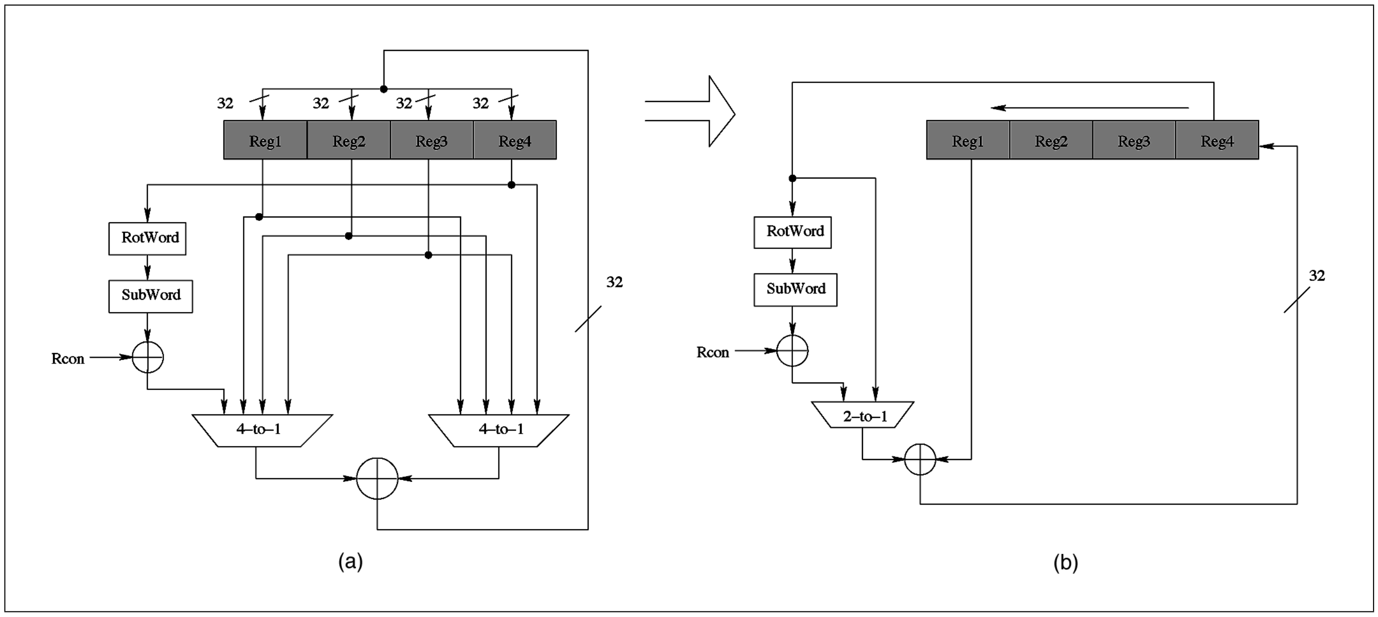

Figure 4: *The key generation unit for encryption.*

bytes 11 and 15 are shifted into reg11 and reg15 sequentially via the multiplexor. In this way, the ShiftRows operation is completed naturally. Thus we can share the two rows of 8-bit registers for encryption and decryption by adding only two 2-to-1 multiplexors before entering the ShiftRows/InvShiftRows operations and two after leaving. The detailed implementation of ShiftRows/InvShiftRows is presented in Fig. 2. Since the SubBytes/InvSubBytes operations have their own multiplexors to distinguish encryption and decryption, these multiplexors can be used to replace the two leaving the ShiftRows/InvShiftRows operations. As a result, only five additional 8-bit multiplexors are required to implement the ShiftRows/InvShiftRows operations besides the 16 8-bit registers. Our proposed design also requires fewer hardware resources in comparison to the designs in [3] and [7], which use the same 16 8-bit registers, but clearly more 8-bit multiplexors, to reselect the input and output of data for each register.

II.B.2 Implementation of SubBytes/InvSubBytes and MixColumns/InvMixColumns operations

We implement the S-box and InvS-box operations by using the efficient combinational circuit proposed in [6]. In this method, the finite-field arithmetic in $GF(2^8)$ is transformed into the equivalent arithmetic in $GF(2^4)$, which is more suitable for hardware implementation. In the implementation, both encryption and decryption share the common multiplicative inverse operation. Reference [12] implements the S-box/InvS-box operations by utilizing the dual-port RAMs provided by FPGAs. Even though the look-up table method can reduce the critical path, thereby providing a higher frequency compared with the method in [6] (which occupies almost half the critical path of the data unit), it is very difficult to implement by the ASIC technique.

For MixColumns and InvMixColumns, we compare the methods used in [7] and [12] and implement the former in our design because it requires fewer hardware resources in the FPGA implementation. Both methods implement the more complicated InvMixColumns operation by sharing the simple MixColumns operation.

II.C Key generation unit

The key expansion expands the initial 128-bit cipher keys to generate the key schedule with a 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. The round keys can be generated on the fly with the data transformation, or they are precalculated and stored for later use [8], [12]. 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 is based on the fact that the cipher keys are changed frequently.

The straightforward implementation of the key generation for encryption is illustrated in Fig. 4(a). Every word (32 bits) of the next state is the XOR of the current word in the same position with its previous word neighbouring it. For words in the significant position Reg1, the neighbouring word is in position Reg4, and a transformation SubWord(Rot()) is applied, followed by an XOR with the round constant Rcon prior to the XOR. We further simplify the implementation as shown in Fig. 4(b). The two 4-to-1 multiplexors are removed, and the four 32-bit registers shift left in each clock cycle to complete the XOR of two neighbouring words. It takes 44 clock cycles to generate the round key words of length 11×4. The round keys for decryption can be similarly implemented in the compact mode (see Fig. 5). We add an additional 8-bit register to hold the significant byte for one more clock cycle on decryption. In addition, the two least significant bytes require an XOR before entering the RotWord function.

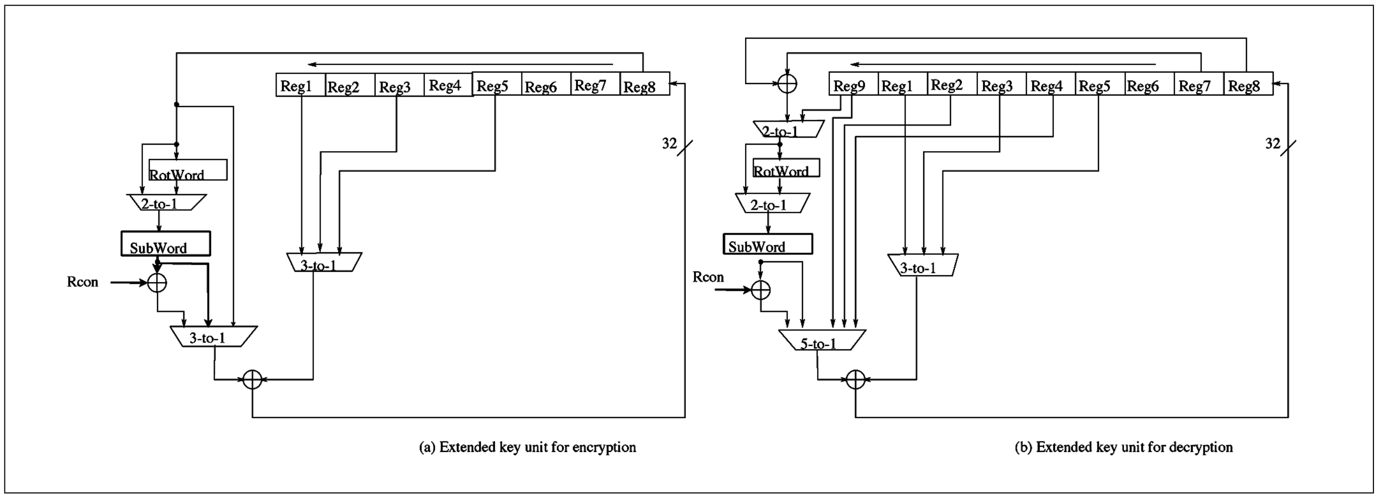

Figure 6: *The extended key generation unit for* 128-, 192-, *and* 256-*bit keys.*

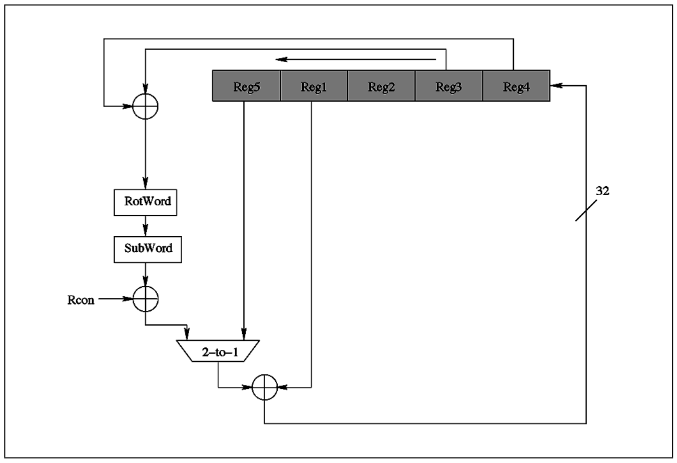

Figure 5: *The key generation unit for decryption.*

Our proposed key generation unit can be easily extended to process 192-bit keys and 256-bit keys as specified in the AES standard. The extended key generation unit is illustrated in Fig. 6.

III Implementations and performance comparison

We implement the dual-core AES on a $0.35\,\mu$m CMOS process from CSMC Technologies Corporation. This implementation contains 16 629 gate equivalents with a maximum frequency of 115 Mhz. Table 1 presents the implementation results.

To our knowledge, the best compact implementations of the AES algorithm in ASICs and FPGAs to date are implemented in [3] and [12], respectively. The design in [12], dedicated to FPGA devices, implements the ShiftRows/InvShiftRows operations by configuring look-up tables in the FPGA as 16-deep shift-registers. It realizes the key schedule by precomputing subkeys in advance and storing them in RAMs, and utilizes the look-up table method to complete the SubBytes. By contrast, in our design we generate the round keys for both encryption and decryption on the fly and require only 16 8-bit registers and a few simple 2-to-1 8-bit multiplexors to complete the ShiftRows/InvShiftRows operations for encryption or decryption.

The comparison with alternative ASIC implementations is shown in Table 2. Our design is simpler for both the data unit and the corresponding key generation unit. The throughput/area parameter shows that our design has additional advantages. When calculating the

Table 1
Performance of the proposed compact AES ASIC implementation

Parameter	Result
Gate equivalents	16 629
Frequency (MHz)	115
Clock cycles	44
Throughput (Mbits/s)	335

throughput/area parameter, we use the sum of throughputs for both encryption and decryption (that is, $335 \cdot 2$) as the numerator, since our proposed dual cores can process both encryption and decryption simultaneously.

IV Conclusion

In this paper, we implemented a new compact 32-bit dual-core architecture for encryption and decryption of the AES. The architecture is based mainly on new optimized ShiftRows and InvShiftRows operations which avoid complex multiplexors and reduce the critical path of the design. Also, we simplified the corresponding compact key generation units in a way that allows easy processing of the 192-bit and 256-bit cipher keys required by the AES standard. Our design requires only 16 629 gate equivalents and provides a throughput of 335 Mbits/s on a $0.35\,\mu$m CMOS process. The compact implementation can perform encryption and decryption in parallel, and it is most suitable for applications such as real-time dual-duplex wireless communication.

Acknowledgements

The authors would like to acknowledge the work of Melody F. Wang and the reviewers' comments. This research work was supported by funding from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] National Institute of Standards and Technology, *Federal Information Processing Standard 197: The Advanced Encryption Standard (AES)*, 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

Table 2

Performance comparison with other compact cores

Implementation	Throughput (Mbits/s)	Area (gate equivalents)	Throughput/area (Kbits/gate)	Frequency (MHz)	Clock cycles
Proposed architecture	335	16 629	40.29	115	44
Reference [3] standard	128	10 799	11.80	64	64
Reference [3] high performance	241	15 493	15.50	64	34

[2] W. Stallings, *Cryptography and Network Security*, 3rd ed., Upper Saddle River, N.J.: Prentice Hall, 2003.

[3] S. Mangard, M. Aigner, and S. Dominikus, "A highly regular and scalable AES hardware architecture," *IEEE Trans. Comput.*, vol. 52, no. 4, Apr. 2003, pp. 483–491.

[4] Nam Sung Kim, T. Mudge, and R. Brown, "A 2.3 Gb/s fully integrated and synthesizable AES Rijndael core," in *Proc. IEEE Conf. Custom Integrated Circuits*, San Jose, Calif., Sept. 21–24, 2003, pp. 193–196.

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

[6] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES Sboxes," *Lecture Notes in Comput. Sci.*, vol. 2271, 2002, pp. 67–78.

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

[8] F.K. Gurkaynak et al., "A 2 Gb/s balanced AES crypto-chip implementation," in *Proc. ACM Great Lakes Symp. VLSI (GLSVLSI '04)*, Boston, Mass., Apr. 2004, pp. 39–44.

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

[10] P. Chodowiec, P. Khuon, and K. Gaj, "Fast implementations of secret key block ciphers using mixed inner- and outer-round pipelining," in *Proc. Int. Symp. Field Programmable Gate Arrays*, Feb. 2001, pp. 94–102.

[11] M. McLoone and J. McCanny, "High performance single-chip FPGA Rijndael algorithm implementation," in *Proc. 3rd Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 2001)*, May 14–16, 2001, pp. 65–76.

[12] P. Chodowiec and K. Gaj, "Very compact FPGA implementation of the AES algorithm," in *Proc. 5th Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 2003)*, 2003, pp. 319–333.

[13] M. Mcloone and J.V. McCanny, "Generic architecture and semiconductor intellectual property cores for advanced encryption standard cryptography," in *IEE Proc. Computers and Digital Techniques*, vol. 150, no. 4, July 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 *Proc. 5th Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 2003)*, 2003, pp. 334–350.

Hua Li is an assistant professor in the Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta, Canada. His research areas include reconfigurable computing, network and information security, image processing, and neural networks. He is a member of IEEE and CMC.

Jianzhou Li was an M.S. candidate in the Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta, Canada.