Journal of Southeast University (English Edition) Vol. 26, No. 1, pp. 21 – 25 Mar. 2010 ISSN 1003—7985

Reconfigurable implementation of AES algorithm IP core based on pipeline structure

Li Bing Xia Kewei Liang Wenli

(School of Integrated Circuits, Southeast University, Nanjing 210096, China)

Abstract: In order to improve the data throughput of the advanced encryption standard (AES) IP core while reducing the hardware resource consumption and finally achieving a tradeoff between speed and area, a mixed pipeline architecture and reconfigurable technology for the design and implementation of the AES IP core is proposed. The encryption and decryption processes of the AES algorithm are achieved in the same process within the mixed pipeline structure. According to the finite field characterizations, the Sbox in the AES algorithm is optimized. ShiftRow and MixColumn, which are the main components in AES round transformation, are optimized with the reconfigurable technology. The design is implemented on the Xilinx Virtex2p xc2vp20-7 field programmable gate array (FPGA) device. It can achieve a data throughput above 2.58 Gbit/s, and it only requires 3 233 slices. Compared with other related designs of AES IP cores on the same device, the proposed design can achieve a tradeoff between speed and area, and obtain satisfactory results in both data throughput and hardware resource consumption.

Key words: advanced encryption standard (AES) algorithm; reconfigurable; pipeline; finite field; round transformation

The trend of modern signal processing systems development is towards all-digital and intelligent direction. Much higher demands are required on hardware systems. In the 1980s, Xilinx Company introduced the first field programmable gate array (FPGA). Since then, the configurable technology has gradually become the hot topic of academic research and industry interest. As it provides both the hardware efficiency and the software programmability, the reconfigurable technology is rapidly developing.

Reconfigurable hardware can achieve the most optimized design in both space and time[1]. The reconfigurable signal processing system can reuse hardware resources according to different application needs. The architecture is flexible enough to meet different specific application needs. The basic reconfigurable hardware is a reconfigurable large-scale integrated circuit. The configuration of these circuits, known as reconfigurable elements, can be designed in advance and stored in the memory of the system. Reconfigurable elements can be configured into a reconfigurable system for specific application needs.

The idea of the reconfigurable architecture to the AES encryption algorithm is introduced. It uses the same module to achieve the data encryption/decryption processing. The reconfigurable encryption circuit can well overcome the weaknesses of the traditional encryption system (Independent data encryption and decryption can cause recourse-consuming problems). The reconfigurable circuit with a pipeline structure is implemented, and the data throughput rate is greatly improved.

Received 2009-07-22.

Biography: Li Bing (1968—), male, doctor, associate professor, bernie_seu@seu.edu.cn.

Citation: Li Bing, Xia Kewei, Liang Wenli. Reconfigurable implementation of AES algorithm IP core based on pipeline structure[J]. Journal of Southeast University (English Edition), 2010, 26(1): 21 – 25.

1 Implement of AES with Pipeline Structure

In 1997, the National Institute of Standards and Technology (NIST) announced the initiation of an effort to develop the AES and made a formal call for algorithms on September 12, 1997[2]. After reviewing the results of this preliminary research, the algorithms such as MARS, RC6TM, Rijndael[3], Serpent and Twofish were selected as the finalists[4]. And further reviewing public analysis of the finalists, the NIST decided to propose Rijndael as the most advanced encryption standard (AES). It is expected to replace the DES and triple DES so as to fulfill the more strict data security requirements due to its enhanced security levels[5].

Besides, an ASIC solution of the AES is also required since it can be more secure and consumes less power than that implemented by software.

1.1 AES system

The input and output of the AES algorithm can be considered as an 8-bit one-dimensional byte-array. In encryption, the inputs are a plain text and a key; the output is a cipher text packet. In decryption, the inputs are a cipher text and a key; the output is a plain text. Each round of transformation in the AES is used in the middle results. These middle results are known as states. The texts and keys in this paper are all 128-bit, so the state of data is a 4×4 rectangular data array.

The AES encryption algorithm has four main operations: SubBytes, ShiftRows, MixColumns and AddRoundKey. The number of the looping rounds is set to $N_r - 1$, in which N_r is specified according to the AES specification. The encryption flow chart is shown in Fig. 1(a). 10 rounds of encryption are required to achieve a 128-bit AES encryption. First, the plain text and the original key are taken as an XOR operation, and then 10-round transformations are done. However, in the last round transformation, the MixColumn operation is not required. The sequence of operations in decryption is basically the inverse of the operation sequence in encryption. The decryption flow chart is shown in Fig. 1(b).

1.2 Implement of pipeline structure

As shown in Fig. 1, There are a large number of hardware resources consumption in AES encryption and decryption, so we need to optimize the algorithm and reconfigure the

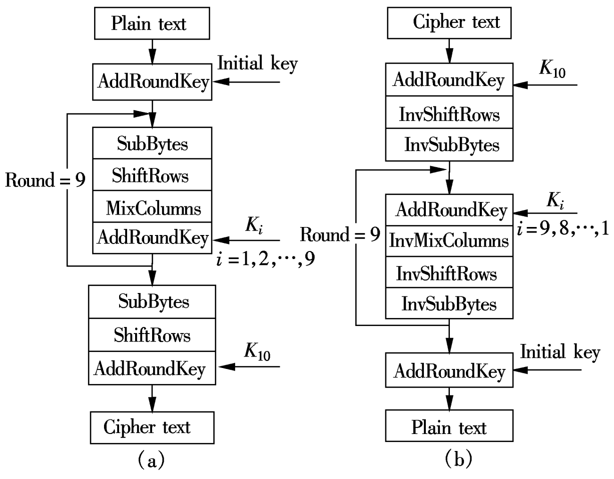

Fig. 1 AES encryption/decryption flow chart. (a) Encryption; (b) Decryption

circuit structure to reduce the hardware resources consumption. Two methods are used to optimize the algorithm:

1) ShiftRow does not change the value of byte, and it only shifts its position. InvShiftRow only replaces each byte in every state without changing its location. Therefore, ShiftRow and InvShiftRow can be exchanged.

2) Mathematically, the linear transformation A meets $A(x+k)=A(x)+A(k)$. InvMixColumn is also a linear transformation, so it meets InvMixColumn(state $\oplus$ roundkey) = InvMixColumn(state) $\oplus$ InvMixColumn(roundkey). Then AddRoundKey(state, roundkey) and InvMixColumn(state) can be exchanged to InvMixColumn(state) and AddRoundKey(state, roundkey).

After using the two optimized methods mentioned above, the decryption and encryption of the AES can approximately achieve the same process. Some inverse modules are used in decryption. The optimized AES encryption/decryption flow chart is shown in Fig. 2.

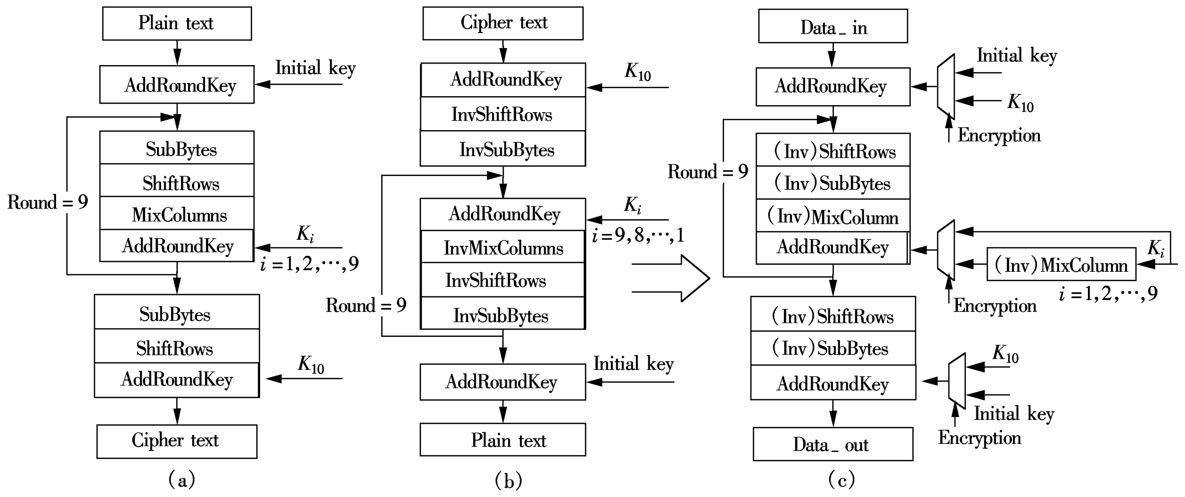

Fig. 2 Optimized AES encryption/decryption flow chart. (a) Encryption; (b) Decryption; (c) Encryption/Decryption

In Fig. 2, a hybrid pipeline structure to implement the AES algorithm circuit is proposed. A register is used in each round operation. In this structure, each round of calculations can be completed in one clock cycle. The output of each round operation is the output of a register, and one set of data processing can be done in less than 10 clock cycles. Therefore, the hybrid pipeline structure can accelerate the data computational speed and improve data throughput.

2 Reconfigurable Design of Round Transformation

The pipeline architecture can improve the clock frequency due to the fact that it can be copied into the process circuits as a module when required, and this is a method by which we maybe cost more circuit-area in exchange for higher circuit-speed. Therefore, reducing the module size is a key problem. We find that the architecture can be reconfigured in each round transformation and can obtain a balance between size and speed of the circuit. Since our design allows sharing of resources by using the reconfigurable structure, the following proposed modules of the AES algorithm obtain a better balance in the result, including Sbox, ShiftRow and MixColumn.

2.1 Sbox

Sbox is a reversible nonlinear transformation, and it is also a key module in the AES. Sbox is used in subbytes and expandkey operations. Since the optimization of Sbox can significantly reduce the consumption of hardware, we optimize the Sbox module based on the theory of the finite field GF(2^8) and then carry out the reconfigurable design of Sbox. The Sbox architecture can be reconfigured and reused as the core modules in Sbox. This architecture can be implemented by combination logic. The area of the circuit is optimized by these improvements.

2.1.1 Generation principles of Sbox

In the AES algorithm, the Sbox structure is generated on the principle of affine theories. The operation of Sbox has two steps: the multiplicative inverse function on the field GF(2^8) and the affine function. In order to reduce the complexity of combinational logic, we make an isomorphic map on the finite field GF(2^8) to the domain GF(2^4), and reverse the multiplicative inverse function into a computing operation on the finite field GF(2^4). The Sbox structure is shown in Fig. 3.

As shown in Fig. 3, Sbox and InvSbox use the same in-

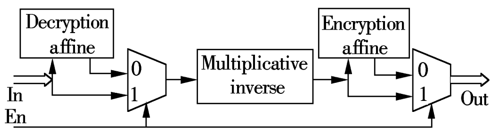

Fig. 3 Sbox structure

verse circuit. When encrypting, the control signal En is set to 1. We first do multiplicative inverse operation checks, and then encrypt the affine transformation. The process of decryption is contrary to the encryption process, and the control signal En is reset to 0. The encryption affine transformation is calculated by

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad (1)$$

And the decryption affine transformation is calculated by

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2)$$

2.1.2 Multiplicative inverse circuit

The multiplicative inverse module is a key component in Sbox. The inverse multiplication circuit is designed as follows:

1) Make the isomorphic element map on the finite field GF(2^8) to the domain GF(2^4);

2) Obtain the inverse elements on the finite field GF(2^4), and then obtain the isomorphic inverse elements on GF(2^8).

We select reducible polynomial $p(x) = x^2 + x + 8$ and primitive polynomial $q(x) = x^4 + x + 1$ to build a polynomial $bx + c$ on GF(2^4). The isomorphic map between GF(2^4) and GF(2^8) is implemented by

$$\{b[3:0],\ c[3:0]\} = \boldsymbol{T}x \quad (3)$$

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

The inverse formula of $bx + c$ is

$$(bx + c)^{-1} = b(8b^2 \oplus bc \oplus c^2)^{-1}x + (c \oplus b)(8b^2 \oplus bc \oplus c^2)^{-1} \quad (4)$$

So we can obtain the elements p and q which are the inverses of b and c from Eq. (4),

$$p = b(8b^2 \oplus bc \oplus c^2)^{-1}$$
$$q = (c \oplus b)(8b^2 \oplus bc \oplus c^2)^{-1}$$

Then we can make isomorphic maps p and q to the elements on GF(2^8) by

$$Y = \boldsymbol{T}^{-1}\{p[3:0],\ q[3:0]\} \quad (5)$$

$$\boldsymbol{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

2.2 ShiftRow

ShiftRow is a shifting operation and it just rotates the bytes in the state matrix. The shift law of ShiftRow and InvShiftRow is shown in Fig. 4. From Fig. 4, it can be seen that after the ShiftRow and InvShiftRow operations are done, and row 0 and row 2 are the same, and row 1 and row 3 are different. Therefore, row 0 and row 2 in ShiftRow and InvShiftRow can be reconfigured. Fig. 5 shows the realization of the process.

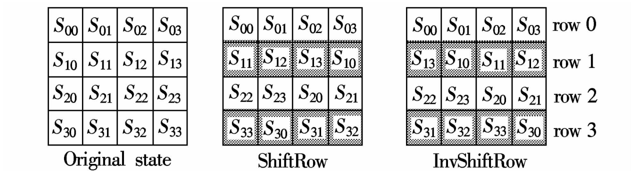
S_{00} S_{01} S_{02} S_{03}
S_{10} S_{11} S_{12} S_{13}
S_{20} S_{21} S_{22} S_{23}
S_{30} S_{31} S_{32} S_{33}
Original state
S_{00} S_{01} S_{02} S_{03}
S_{11} S_{12} S_{13} S_{10}
S_{22} S_{23} S_{20} S_{21}
S_{33} S_{30} S_{31} S_{32}
ShiftRow
S_{00} S_{01} S_{02} S_{03} row 0
S_{13} S_{10} S_{11} S_{12} row 1
S_{22} S_{23} S_{20} S_{21} row 2
S_{31} S_{32} S_{33} S_{30} row 3
InvShiftRow

Fig. 4 Row exchange in ShiftRow and InvShiftRow

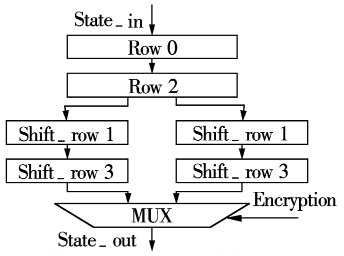

Fig. 5 ShiftRow and InvShiftRow structure

2.3 MixColumn

The role of MixColumn is to replace each column in states. It uses the value of the status byte to a mathematical domain plus and the domain multiply, and then each byte is replaced by the operation results. The realization of the process is shown in Fig. 6.

MixColumn can be transformed by

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \otimes \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 2(a_0 + a_1) + a_1 + a_2 + a_3 \\ 2(a_1 + a_2) + a_0 + a_2 + a_3 \\ 2(a_2 + a_3) + a_0 + a_1 + a_3 \\ 2(a_3 + a_0) + a_0 + a_1 + a_2 \end{bmatrix} \quad (6)$$

InvMixColumn can be realized by

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} e & b & d & 9 \\ 9 & e & b & d \\ d & 9 & e & b \\ b & d & 9 & e \end{bmatrix} \otimes \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 8(a_0+a_1+a_2+a_3)+4(a_0+a_2)+2(a_0+a_1)+a_1+a_2+a_3 \\ 8(a_0+a_1+a_2+a_3)+4(a_1+a_3)+2(a_1+a_2)+a_0+a_2+a_3 \\ 8(a_0+a_1+a_2+a_3)+4(a_0+a_2)+2(a_2+a_3)+a_0+a_1+a_3 \\ 8(a_0+a_1+a_2+a_3)+4(a_1+a_3)+2(a_3+a_0)+a_0+a_1+a_2 \end{bmatrix} = \begin{bmatrix} 8(a_0+a_1+a_2+a_3)+4(a_0+a_2)+b_0 \\ 8(a_0+a_1+a_2+a_3)+4(a_1+a_3)+b_1 \\ 8(a_0+a_1+a_2+a_3)+4(a_0+a_2)+b_2 \\ 8(a_0+a_1+a_2+a_3)+4(a_1+a_3)+b_3 \end{bmatrix} \tag{7}$$

where b_0, b_1, b_2 and b_3 are as the same as those in Eq. (6). It can be compacted as the structure of MixColumn. InvMixColumn is shown in Fig. 7. From the circuit view (see Fig. 8), some parts of the logic circuit are reconfigured, which can save a few gates so that the circuit area is reduced. In Fig. 8, Xtime block can achieve an AND operation between input data and the immediate hex number 0x02.

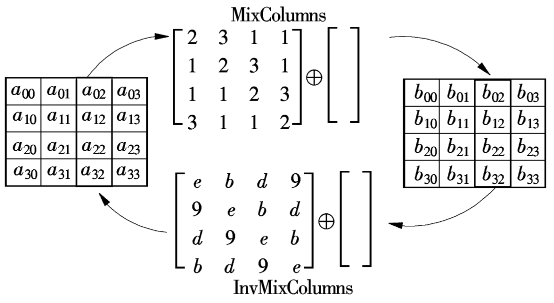

Fig. 6 Realization of MixColumn and InvMixColumn

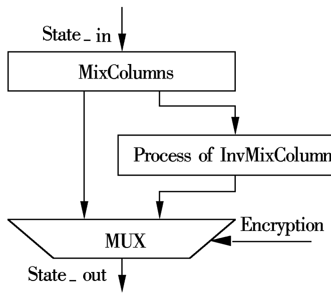

Fig. 7 Optimized MixColumn and InvMixColumn structure

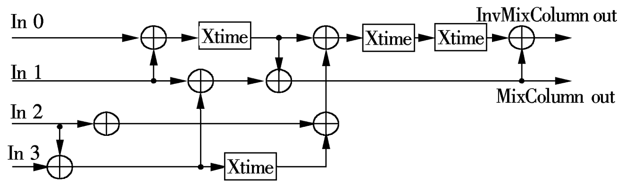

Fig. 8 MixColumn and InvMixColumn circuit realization

3 Synthesis Results

In the AES algorithm, the data throughput and the circuit area are two important parameters used to evaluate the hardware implementation:

1) Data throughput can be expressed as throughput = 128 / (average clock cycles in processing a data block × clock cycle value);

2) Circuit area for FPGA refers to the basic logic unit consumed or the number of configurable logic blocks.

The proposed design is implemented in Verilog HDL language and correctly simulated through the Modelsim tool. We realize the AES algorithm by the integrated RTL-level code on the Xilinx Virtex2p xc2vp20-7 FPGA device. The clock frequency is 164 MHz; the slack is 6.23 ns in the worst situation; the data throughput is 2.58 Gbit/s, and the logical gates consumed are 3 233 slices.

Tab. 1 lists some models of the AES algorithm based on the same Xilinx Virtex2p xc2vp20-7 device, and shows different designs with the synthesized results.

Ref. [4] does not use the pipeline structure to implement the AES algorithm and its operation frequency is too low; Ref. [5] improves the data throughput, but the realization of Sbox still uses the look-up table form, and the structure occupies a large number of hardware resources; Ref. [6] uses the pipeline structure and has the data throughput improved, but it is only implemented in the encryption process; Ref. [7] uses fewer hardware resources, but its calculation speed is too low; Ref. [8] can support three length keys, but there is no optimization in the encryption or decryption process, and it also occupies a large number of hardware resources. Compared with all the designs mentioned above, the proposed design can obtain more satisfactory results in terms of both data throughput and circuit area.

Tab. 1 Different synthesize results of AES

Design	Architecture	Areas CLB slices	Clock frequency/MHz	Throughput/(Mbit·s^{-1})	Mode support
Ref. [4]	Sequential	4 189	65	1 190	Encryption/decryption
Ref. [5]	Sequential	7 301	148	1 722	Decryption
		6 766	194	2 257	Encryption
Ref. [6]	Pipeline	9 446	168	21 640	Encryption
Ref. [7]	Sequential	928 + 8BRAM	122	1 392	Encryption/decryption
Ref. [8]	Sequential	3 168 + 8BRAM	156	1 995	Encryption/decryption
This paper	Pipeline	3 233	164	2 580	Encryption/decryption

4 Conclusion

In this paper, we use the reconfigurable structure to optimize the AES encryption algorithm. Compared with other related designs, the proposed design optimizes the circuit area well and the data throughput is significantly improved. However, the inadequacy of this design does not support different lengths of keys, and in future we will make further studies on it and improve the circuit design of the AES algorithm.

References

[1] Fischer V, Drutarovsky M. Two methods of Rijndael implementation in reconfigurable hardware [C]//*The Third International Workshop on Cryptographic Hardware and Embedded Systems*. Paris, France, 2001, **2162**: 77 − 92.
[2] National Institute of Standards and Technology(NIST). Advanced encryption standard(AES) (FIPS PUB 197) [S]. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2001.
[3] Daemen J, Rijmen V. *The design of Rijndael AES*: *the advanced encryption standard* [M]. Berlin, Germany: Springer-Verlag, 2002.
[4] Sever R, Ismailglu A N, Tekmen Y C, et al. A high speed FPGA implementation of the Rijndael algorithm [C]//*Euromicro Symposium on Digital System Design*, *Architectures*, *Methods and Tools*. Rennes, France, 2004: 358 − 362.
[5] Sivakumar C, Velmurugan A. high speed VLSI design CCMP AES cipher for WLAN(IEEE 802. 11i) [C]//*International Conference on Signal Processing*, *Communications and Networking*. Chennai, India, 2007: 398 − 403.
[6] Hodjat A, Verbauwhede I. A 21. 54 Gbits/s fully pipelined AES processor on FPGA [C]//*Proceedings of the* 12*th Annual IEEE Symp on Field-Programmable Custom Computing Machines*. Napa, CA, USA, 2004: 308 − 309.
[7] CAST Inc. AES128-P Programmable advanced encryption standard core [EB/OL]. (2005-01-10)[2009-07-10]. http://www.cast-inc.com/cores/aes-p/index.shtml.
[8] Helion Technology Limited Company. High performance AES (Rijndael) cores for Xilinx FPGA[EB/OL]. (2005-02-23) [2009-07-10]. http://www.heliontech.com/aes.htm.

基于流水线结构的可重构 AES 算法 IP 核的硬件实现

李　冰　夏克维　梁文丽

(东南大学集成电路学院, 南京 210096)

摘要: 为了提高 AES 算法中 IP 核数据的吞吐量并同时减小硬件资源的占用, 以达到速度和面积的折中实现, 采用混合流水线结构和可重构技术完成了 IP 核的设计. 该设计包括在同一个混合流水线结构的流程中实现了 AES 算法的加密和解密过程; 根据有限域的性质, 对 AES 算法中的 Sbox 盒进行了优化; 结合可重构技术, 完成了对 AES 轮变换的主要构件 ShiftRow 和 MixColumn 的优化. 本设计在 Xilinx Virtex2p xc2vp20-7 FPGA 器件上完成, 其数据吞吐量达到 2. 58 Gbit/s, 所需组合逻辑仅为 3 233 块, 通过与同型号器件上的其他设计进行对比, 实现了速度和面积的折中, 在吞吐量和面积上都得到了比较理想的结果.

关键词: AES 算法; 可重构; 流水线; 有限域; 轮变换
中图分类号: TN911. 21