國立中興大學 范志鵬副教授 出席國際會議報告

2007 年 12 月 10 日

報告人姓名	國立中興大學 電機工程系 范志鵬 副教授
會議期間及地點	2007/11/28 - 20076/12/1 中國廈門
會議名稱	(中文) 2007年 智慧型信號處理與通訊系統國際會議, 中國 (英文) 2007 International Symposium on Intelligent Signal Processing and Communication Systems, China (ISPACS 2007)
發表論文題目 (共 2 篇)	1. (中文) 在 FPGA 上實現高輸出率的連續式與管線式 AES 處理模組 (英文) Implementations of High Throughput Sequential and Fully Pipelined AES Processors on FPGA 2. (中文) 應用於 IEEE 802.11i WEP/TKIP 的有效低延遲 RC4 架構設計 (英文) Efficient Low-Latency RC4 Architecture Designs for IEEE 802.11i WEP/TKIP

1. 參加會議經過
2. 與會心得
3. 建議
4. 參與此次會議的照片集
5. 在此會議發表的論文內容

出席國際會議報告

2007年 智慧型信號處理與通訊系統國際會議

中國廈門

Huaqiao University (華僑大學) 主辦

2007年11月28日 至 2007年12月1日

范志鵬 副教授
國立中興大學 電機工程系

1. 參加會議經過

此次本人所參與的會議為亞太地區國際電機電子學會所協辦之一年一度重要的信號處理與通訊系統設計相關會議,簡稱 ISPACS。ISPACS 為亞太地區年度信號處理與通訊系統設計的重要會議之一，每一年於 12 月初固定在亞太地區各國家及地區輪流舉辦，共有百篇左右的論文將於四天的議程中發表，而本人所參與的是 System on chip 與 FPGA realization 兩個 session 的 oral 論文發表。此次 oral 論文的發表共有二篇, 題目分別為

1. "Implementations of High Throughput Sequential and Fully Pipelined AES Processors on FPGA"，主要探討為在 FPGA 上實現高輸出率的連續式與管線式 AES 加解密處理模組。而研究動機乃為了提高 AES 硬體架構的輸出率，將管線式 AES 加解密處理模組利用管線式高速 CAM-based Subbytes 架構來加速，並且達到高位元輸出率的目的.
2. "Efficient Low-Latency RC4 Architecture Designs for IEEE 802.11i WEP/TKIP", 主要探討應用於 IEEE 802.11i WEP/TKIP 的有效低延遲 RC4 架構設計。而研究動機乃為了使 RC4 加解密模組具有低延遲的結果輸出，將應用單埠 128x16 位元或雙埠 256x8 位元的 SRAM 架構來做資料交換設計，以有效的降低計算延遲.

由研討會的議題觀察，使用多媒體視訊及音訊應用已是信號處理與通訊的一重要的研究與發展趨勢。而此次研討會所探討的重點為信號處理與通信系統設計方面的研究，重要的 Keynote 演講如下，包括先進多媒體視音訊處理的探討，多媒體導航系統，及系統晶片與系統整合設計:

Keynote Speech 1:
Immersive and Intelligent Collaboration Research
Presented by: Dr. B.H. Juang, Georgia Institute of Technology, USA

Keynote Speech 2:
What VisionCruiser Can Do For Digital Earth
Presented by: Dr. Deren Li, Wuhan University, China

Keynote Speech 3:
Automatic Video Segmentation and Tracking for Content-based Applications
Presented by: Dr. King Ngi Ngan, The Chinese University of Hong Kong, China

Keynote Speech 4:
Paradigm Shift in System LSI Design
Presented by: Dr. Satoshi Goto, Professor, Waseda University, Japan

而在論文發表期間，數位學者除了在現場不吝提出問題給予指教之外，更提出問題與我討論。而除了本身的論文發表之外，本人也多方參加了相關領域的 sessions，瞭解到了別人的研究進度與方向，並嘗試與參加會議之資深教授與專家請教，更進一步的瞭解目前學術界與工業界的研究與需求，以期自己所做的研究能與業界應用相結合。

2. 與會心得

此行最重要的目的，在於瞭解學術及業界專家的最新信號處理與通訊系統設

計研究方向，並且吸收更先進的觀念。而此研討會最大的功能，即是讓做相關領域不同國度的人有一個絕佳的機會能夠在一塊，大家互相交流彼此的研究心得，並多認識可能的合作伙伴或競爭者。在研究議題方面，Video signal processing，communication signal processing 與 VLSI signal processing，亦為目前電子電機及資訊界亟欲加速研究的方向，使得本人更加相信自己與研究生從事此一方面研究的重要性；另一方面，會議過程中有來自不同國家及地區的學者參與相關領域的報告，在聽取各方的論文報告之後，對於此一視訊處理與通訊領域的未來發展深感樂觀，使本人的收穫非常豐富。

在會議中，一些 VLSI 信號處理相關領域的學者與研究員都曾來與本人接觸與討論，可間接地證明此二篇論文在信號處理演算法與實作這個領域的研究上已有國際水準。

3. 建議

台灣所需之高科技人才依靠台灣自行培養者日益增加，因此如何使台灣高等教育與國際水準同步或超越，這是我們必須面對之嚴肅課題；大專院校應多鼓勵學校教授及自行培養的研究生多出國參加會議，可以幫助瞭解自己的研究在國際上的水平，並且增加科技的能見度及廣度。

4. 參與此次會議的照片集

2007 ISPACS 會議的開幕式

會議 Coffee break 時間

2007 ISPACS 會場之一

論文發表後

Keynote 的專家演講

論文發表後

5. 在此會議發表的論文

（共 2 篇，原文請參考下頁內容）

1. Chih-Peng Fan and Jun-Kui Hwang, “Implementations of High Throughput Sequential and Fully Pipelined AES Processors on FPGA,” 2007 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2007), Xiamen, Fujian China (EI), 11 2007

2. Jun-Dian Lee and Chih-Peng Fan, “Efficient Low-Latency RC4 Architecture Designs for IEEE 802.11i WEP/TKIP,” 2007 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2007), Xiamen, Fujian China (EI), 11 2007

Implementations of High Throughput Sequential and Fully Pipelined AES Processors on FPGA

Chih-Peng Fan* and Jun-Kui Hwang
Department of Electrical Engineering, National Chung Hsing University,
250 Kuo-Kuang Road, Tai-chung 402, Taiwan, R.O.C.
Email : cpfan@dragon.nchu.edu.tw *

ABSTRACT

In this paper, we use FPGA chips to realize the high-throughput 128 bits AES cipher processor by new high-speed and hardware sharing functional blocks. The AES functional caculations include four transformation stages, which are SubBytes, ShiftRows, MixColumns and AddRoundKey. The content-addressable memory(CAM) based scheme is used to realize the new proposed high-speed SubBytes block. The new hardware sharing architecture is applied to implement the proposed high-speed MixColumns block. Then the efficient low-cost AddRoundKey architecture is used for real-time key generations. The utilized FPGA tool is Xilinx ISE™ 7.1 with XST™ synthesizer. In our proposed sequential AES design, the operational frequency can reach 75.3MHz and the throughput can be up to 0.876Gbits/s. In our full pipelined AES design, the operational frequency can process 250MHz and the throughput can be up to 32Gbits/s. Both of the proposed sequential and full pipelined AES realizations achieve higher throughput than the other sequential and full pipelined designs, individually.

1. Introduction

National Institute of Standard and Technology(NIST) agreed on the block-based cipher system, data encryption standard (DES) in 1970. Since the DES cipher system was worn-out by violence attack methods, then NIST announced the new DES algorithm, which is called the Triple DES(3DES). The 3DES uses the same algorithm as the DES and increases the difficulty of illegal breaks. But the 3DES algorithm exists two disadvantages. First, the 3DES requires three times more execution cycles than the DES, so the execution efficiency of 3DES is not good enough. Second, both DES and 3DES only use a 64-bit length block data, the security and safety are not enough. Due to the two disadvantages of DES and 3DES, the NIST gave up the DES cipher systems and then asked for a new generation cipher system, which was called the advanced encryption standard(AES) in 1997. After serious elections, NIST chose the Rijndael algorithm for the final version of AES in 2001 [1].

In [2-10], several authors develop different AES architectures for high-speed and high-performance applications. In this paper, we use FPGA chips to realize the high-throughput 128 bits AES encryption/decryption processor with new high-speed and hardware sharing functional blocks. In our proposed sequential AES design, the operational frequency can reach 75.3MHz and the throughput can be up to 0.876Gbits/s. In our full pipelined AES design, the operational frequency can process 250MHz and the throughput can be up to 32Gbits/s. The rest of this paper is organized as follows. In Section 2, the AES algorithm is reviewed. The proposed high-speed and hardware sharing functional blocks in AES will be described and shown in Section 3. In Section 4, the proposed high-throughput sequential and full pipelined AES architectures will be discussed. The comparison and implementation results on FPGA chips will be shown in Section 5. Finally, a conclusion will be given.

2. Review of AES algorithm

In [1], the AES algorithm is clearly defined with key, functional blocks, and round numbers. The fixed length of plaintext is 128 bits, the lengths of keys are 128, 192 and 256 bits, and the execution round numbers have 10, 12 and 14. For the example of 128-bit key, during operations, the key must be segmented into 16 bytes, and the segmented 16 bytes will be mapped into a 4x4 matrix, which is called the state matrix. Each byte in the state matrix must be normalized under Galois Field (GF)(2^8) with the modulus of $x^8 + x^4 + x^3 + x + 1$. The AES operations include four transformation calculations, which are SubBytes, ShiftRows, MixColumns and AddRoundKey in order.

The SubBytes transformation performs non-linear bytes transformation under GF(2^8). The SubBytes process is divided into two steps. In the first step, 8-bit inverse multiplication in GF(2^8) is calculated, then in the second step, the affine transform is used to obtain the output results. Next, the ShiftRows transformation performs the fixed cyclic byte shift according to different row positions. In the 0^{th} row, this row does not act byte shift. In the 1^{th} row, this row acts one byte shift. In the 2^{th} row, this row acts two bytes shift. In the 3^{th} row, this row acts three bytes shift. The ShiftRows performs cyclic left shift during AES encryptions, and the ShiftRows performs cyclic right shift during AES decryptions.

The MixColumns transformation acts the row-by-row mapping operations. During encryptions, the row-by-row operation is based on the mapping polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ under constrains of GF(2^8) and x^4+1 modulus. During decryptions, the row-by-row operation is based on the mapping polynomial $a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$ under constrains of GF(2^8) and x^4+1 modulus. Finally, the AddRoundKey transformation performs the bit-by-bit XOR operations between outputs of MixColumns and the round key.

3. Implementation of the proposed high-performance AES hardware

The whole AES hardware blocks are composed of five operational modules, which are SubBytes, ShiftRows, MixColumns, AddRoundKey and Key expansion circuits. In our proposed design, the high-speed and high-throughput architectures are our design goals. In the following sections, we will focus on

the design of high performance architectures for the SubBytes, MixColumns and Key expansion operational modules.

3.1 High-speed implementations of SubBytes/InvSubBytes transformation

For high-speed realizations of the SubBytes and Inverse SubBytes (InvSubBytes) hardwares, we use the ROM-based concept to design this module. But the traditional ROM-based architecture can not reach very high-speed operations. Thus, we apply the content-addressable memory(CAM) [2] based architecture to realize the SubBytes/InvSubBytes circuits, which are shown in Figure 1. When we enable the SubBytes operation, then the registers a*i,* for *i*=1, 2, 3, …, 256, will output their 8 least significant bits to the inputs of compare(CMP) circuits. On the other hand, when we enable the InvSubBytes operation, then the registers a*i*, for *i*=1, 2, 3, …, 256, will output their 8 most significant bits to the inputs of CMP circuits. Figure 2 shows the implementation of CMP circuits. For further high-speed full-pipelined AES implementation, the proposed SubBytes and InvSubBytes scheme can be divided into three pipelined phases by adding two pipelined register arrays. This scheme will be shown in Section 4.2. In Table 1, the three-phase pipelined SubBytes/InvSubBytes module can achieve higher operational frequency than the traditional ROM-based scheme.

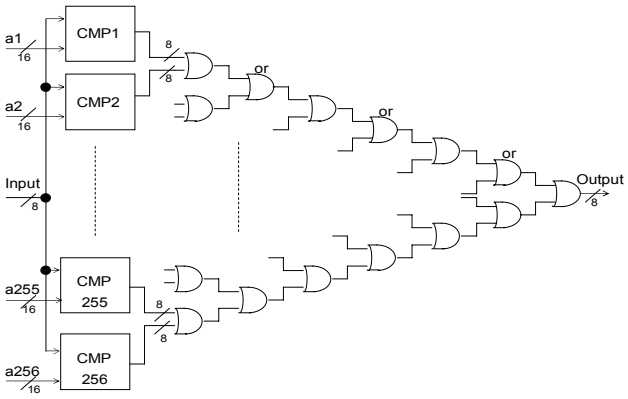

Figure 1 Proposed new realization for SubByte and InvSubBytes transformation

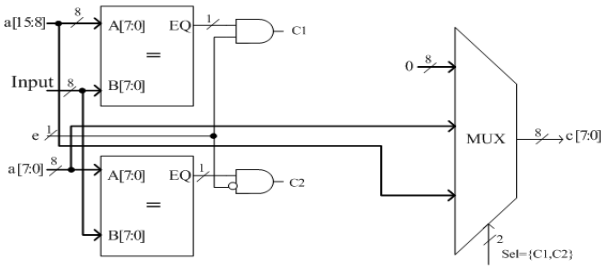

Figure 2 Proposed realization of CMP circuit

Table 1 Performance comparison of three SubBytes Architectures

Type	COEGEN (ROM)	LOGIC	Ours (3 stages pipelined)
Device	xc2v3000-6-fg676	xc2v3000-6-fg676	xc2v3000-6-fg676
# of Slices	132	248	771
Gate delay	5.986ns	8.761ns	4.796ns

3.2 Hardware sharing implementation of MixColumns / InvMixColumns transformation

The operations of MixColumns and Inverse MixColumns (InvMixColumns) transformations, which are realized in GF(2^8), are mapped into two different corresponding matrix polynomials. In order to design the hardware sharing architecture for both MixColumns and InvMixColumns transformations, we decompose the matrix operations of InvMixColumns in (1) and replace (1) with (2). In (2), the new matrix decomposition contains the same matrix operations of MixColumns, which is the first matrix term of (2). By using the common factors, we can design a high-speed hardware sharing circuits to implement these transformations.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}, \quad (1)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} + \begin{bmatrix} 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} + \begin{bmatrix} 04 & 00 & 04 & 00 \\ 00 & 04 & 00 & 04 \\ 04 & 00 & 04 & 00 \\ 00 & 04 & 00 & 04 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}. \quad (2)$$

Figure 3 shows the proposed hardware sharing circuits from (2). In Figure 3, the circuits can achieve the functions of MixColumns and InvMixColumn. The upper part circuit above dashed line in Figure 3 can carry out the operations of MixColumns function. Then the whole circuits in Figure 3 can implement the InvMixColumns function. The proposed hardware sharing circuits for MixColumns and InvMixColumn require 25 XOR gates, six xTime circuits and one x4Time circuits, where xTime means the operations of multiplication by {02} in GF(2^8), and x4Time means the operations of multiplication by {04} in GF(2^8).

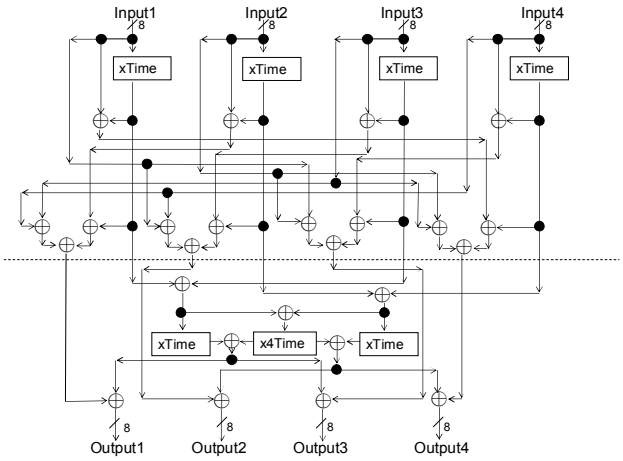

Figure 3 Circuit architectures of MixColumns and InvMixColumns

3.3 High-speed implementation of Key Expansions

We design a high-speed Key expansion circuit for real-time generations of 128-bit keys. The realized Key expansion circuits can generate keys for AES encryptions and decryptions. Since the decryption process is asymmetric, then the key expansion circuits for decryptions need to collocate the InvMixColumns circuits. In the operations of Key expansion, a 128-bit key will be segmented into four 32-bit data, and the four 32-bit data are stored into the corresponding a, b, c and d registers. Thus, the output of d register must be gone through the operations of ROT, S-box and RCON, where ROT means the shift of bytes, S-box is the same as SubBytes module, and RCON is a simple XOR operation. Figure 4 shows circuit architectures of sequential on-the-fly key expansions, and Figure 5 shows circuit architectures of non-sequential on-the-fly key expansions. According to the Key expansion circuits in [3], we reduce the hardware cost by the combination of the Multiplexer and S-box circuits. Since only 8 most significant bits (MSB) will change values in RCON, we just design a low-cost 8-bit XOR circuit for RCON outputs.

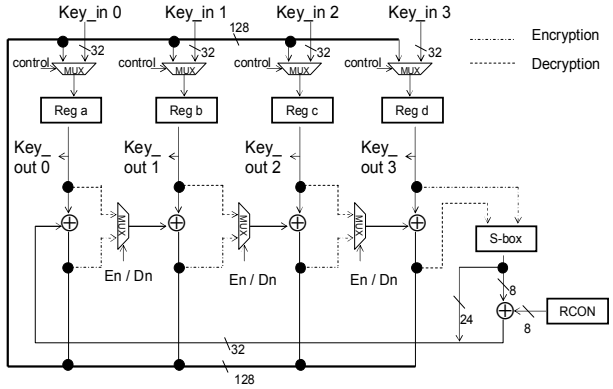

Figure 4 Circuit architectures of sequential on-the-fly key expansions

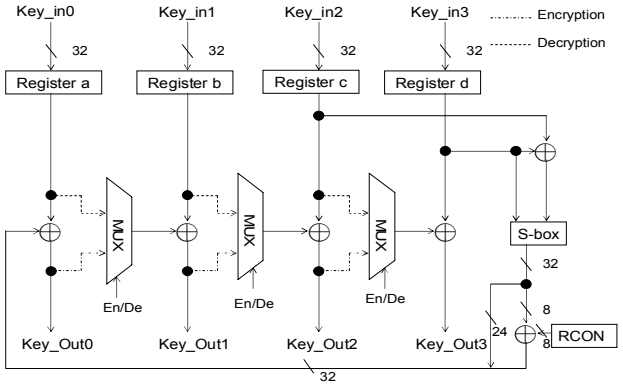

Figure 5 Circuit architectures of non-sequential on-the-fly key expansions

4. Architectures of the proposed AES processor

For the design of the proposed AES processor, two different architectures, which include sequential and full pipelined schemes, are utilized for implementations. The detailed descriptions about the proposed two architectures are discussed in the following.

4.1 Sequential Architecture

Figure 6 shows the hardware architecture of the proposed sequential AES processor. In Figure 6, this design includes one AES function block and one key expansion circuit. In the AES internal function blocks, the operational circuits are not pipelined. Two registers are added at input and output interfaces for buffers of inter-round computations. The proposed sequential AES architecture can generate 128-bit encrypted or decrypted data every 11 clock cycles.

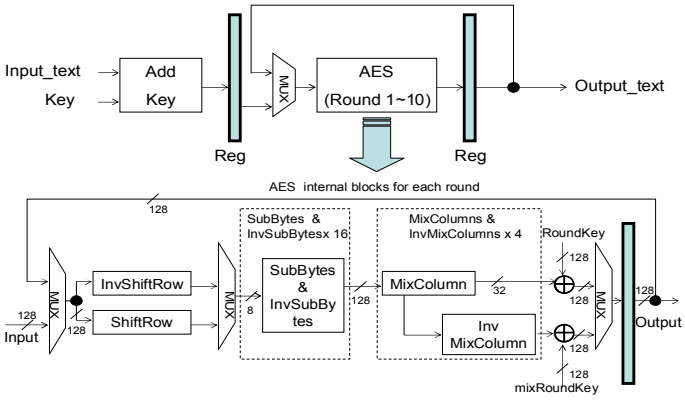

Figure 6 Hardware architecture of the proposed sequential AES processor

4.2 Fully pipelining Architecture

Figure 7 shows the architecture of proposed full pipelined AES processor. In Figure 7, this architecture is composed of ten AES functional blocks and key expansion circuits. The inter-pipelined and outer-pipelined schemes are utilized for implementations. In the inter-pipelined scheme, the register arrays are assigned among the operational circuits of SubByte, ShiftRows, MixColumns and AddRoundKey. The SubBytes block is further divided into three pipelined phases. Therefore, the AESi functional blocks, for i=1, 2, 3, …, 10 are segmented into five inter-pipelined phases. In the outer pipelined scheme, several register arrays are added between each AES round computation. Thus, the latency delay of the proposed full-pipelined AES processor is 51 clock cycles. The throughput of this architecture is 128 bits per clock cycle.

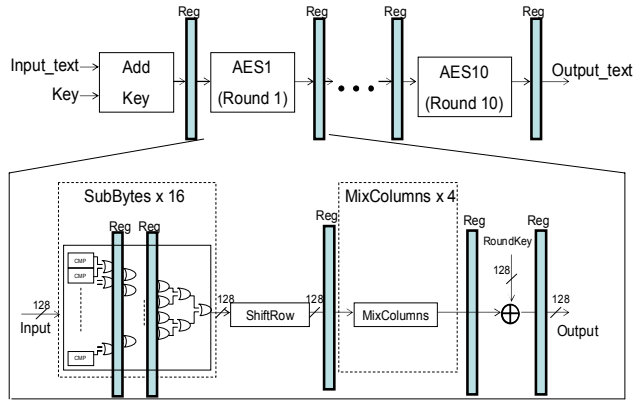

Figure 7 Hardware architecture of the proposed full pipelined AES processor

5. Performance and Comparison of AES with FPGA

We use the Xilinx ISE™ 7.1 FPGA design tool to implement our proposed AES processor. Table 2 shows the realization results of our AES processors after placement and route with Xilinx FPGA design tool. Table 3 shows the comparison of different sequential AES designs. From Table 3, our proposed sequential AES processor achieves more output throughput (0.876Gbits/s) than the other sequential AES designs on FPGA. Table 4 shows the comparison of different full-pipelined AES designs. From Table 4, our proposed full-pipelined AES processor also achieves more output throughput (32Gbits/s) than the other full pipelined AES designs on FPGA.

Table 2 Implementation results of our AES architectures

Design	FPGA Device	Encryption /Decryption	Number of Slices	Latency (cycles)	Freq.	Throughput
Our Sequential AES	XC2V3000	Enc/Dec	7617	11	75.3 MHz	0.876 Gbits/s
Our Fully pipelined AES	XC4VLX200	Enc	86806	51	250 MHz	32 Gbits/s

Table 3 Comparison of sequential AES Designs on FPGA

Design	Device	Encryption /Decryption	Frequency	Throughput
R. Sever et al [4]	XC2V8000bf957-5	Enc/Dec	65 MHz	832 Mbits/s
N. Sklavos & et al [5]	XCV300BG432	Enc/Dec	22 MHz	259 Mbits/s
Wang et al [6]	XC1000e-8	Enc/Dec	76 MHz	463 Mbits/s
N.A Saqib et al [7]	XCV812	Enc	22.41 MHz	259 Mbits/s
C. Chitu et al [8]	XC2V1000-4	Enc/Dec	75 MHz	739 Mbits/s
Our proposed	XC2V3000fg676-6	Enc/Dec	75.3 MHz	876 Mbits/s

Table 4 Comparison of fully pipelining AES designs on FPGA

Design	Device	Encryption /Decryption	Frequency	Throughput
Kotturi et al [9]	XC2VP70-7	Enc	232.6MHz	29.77Gbits/s
Zhang et al [11]	XCV1000 e-8	Enc	168.4MHz	21.56Gbits/s
Hodjat et al with 7 stages in round [10]	XC2VP20-7	Enc	169.1MHz	21.64Gbits/s
Hodjat et al with 4 stages in round [10]	XC2VP20-7	Enc	168.3MHz	21.54Gbits/s
Our proposed	XC4VLX200	Enc	250MHz	32Gbits/s

6. Conclusion

The FPGA-based high-throughput 128 bits AES cipher processor by using new high-speed and hardware sharing functional blocks is developed in this paper. The AES functional blocks include SubBytes, ShiftRows, MixColumns and AddRoundKey transformations. The content-addressable memory(CAM) based scheme is used to realize the new proposed high-speed SubBytes block. The new hardware sharing architecture is applied to implement the proposed high-speed MixColumns block. Next, the efficient low-cost AddRoundKey architecture is used for real-time key generations. The Xilinx ISE™ 7.1 with XST™ synthesizer is our design tool. In our proposed sequential AES design with both encryption and decryption, the operational frequency can reach 75.3MHz and the throughput can be up to 0.876Gbits/s. In our full pipelined AES design at encryption mode, the operational frequency can process 250MHz and the throughput can be up to 32Gbits/s. Both of the proposed sequential and full pipelined AES realizations achieve high throughput requirements and can be suitably used for the 802.11i CCMP applications.

ACKNOWLEDGMENT

This work was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC96-2220-E-005-006.

REFERENCES

[1] "Advanced Encryption Standard (AES)" Federal Information Processing Standards Publication 197, Nov. 26, 2001.

[2] H. Li "A New CAM Based S/S^{-1}-Box Look-up Table in AES", IEEE Symposium on Circuits and Systems, vol. 5, pp.4634 - 4636, May 23-36, 2005.

[3] Joon Hyoung Shim, Dae Won Kim, Young Kyu Kang, Taek Won Kwon, Jun Rim Choi, "A Rijndael cryptoprocessor using shared on-the-fly key scheduler", IEEE Asia-Pacific Conference on ASIC, pp. 89-92, Aug. 6-8, 2002.

[4] R. Sever, A.N. Ismailglu, Y.C. Tekmen, M. Askar and B. Okcan, "A high speed FPGA implementation of the Rijndael algorithm", Euromicro Symposium on Digital System Design, pp. 358-362, Aug. 31- Sept. 3, 2004.

[5] N. Sklavos and O. Koufopavlou, "Architectures and VLSI implementations of the AES-Proposal Rijndael", IEEE Transactions on Computers, vol. 51, issue 12, pp. 1454-1459, Dec. 2002.

[6] Shuenn-Shyang Wang and Wan-Sheng Ni, "An efficient FPGA implementation of advanced encryption standard algorithm", International Symposium on Circuits and Systems, vol. 2, pp. II-597-600, May 23-26, 2004.

[7] N.A Saqib, F. Rodriguez-Henriquez and A. Diaz-Perez, "AES algorithm implementation - an efficient approach for sequential and pipeline architectures", the Fourth Mexican International Conference on Computer Science, pp.126-130, Sept. 8-12, 2003.

[8] C. Chitu, D. Chien, C. Chien, I. Verbauwhede and F. Chang, "A hardware implementation in FPGA of the Rijndael algorithm", The 45th Midwest Symposium on Circuits and Systems, vol. 1, pp. I-507-10, Aug. 4-7, 2002

[9] D. Kotturi, Seong-Moo Yoo, J. Blizzard, "AES crypto chip utilizing high-speed parallel pipelined architecture", IEEE International Symposium on Circuits and Systems, vol. 5, pp.4653-4656, May 23-26, 2005.

[10] A. Hodjat, I. Verbauwhede, "A 21.54 Gbits/s fully pipelined AES processor on FPGA", 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines , pp. 308-309, April 20-23, 2004.

[11] Xinmiao Zhang; K.K. Parhi, "High-speed VLSI architectures for the AES algorithm", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, issue 9, pp. 957-967, Sept. 2004.

Efficient Low-Latency RC4 Architecture Designs for IEEE 802.11i WEP/TKIP

Jun-Dian Lee and Chih-Peng Fan*
Department of Electrical Engineering, National Chung Hsing University,
250 Kuo-Kuang Road, Tai-chung 402, Taiwan, R.O.C.
Email : cpfan@dragon.nchu.edu.tw *

ABSTRACT

In this paper, novel low-latency RC4 implementations with cell-based VLSI design flow are proposed for IEEE 802.11i WEP/TKIP. The RC4 stream cipher is used in the security protocol WEP in IEEE 802.11b wireless network, and is also used in the TKIP of wireless network IEEE 802.11i cryptography. The major process of RC4 algorithm is to shuffle the memory continuously. For quick memory shuffling, we investigate two different memory shuffling architectures to design the RC4. By using single-port 128x16 memory design, this architecture reduces 25% shuffling latency, compared with the conventional single-port 256x8 architecture. By using dual-port 256x8 memory design, this architecture achieves less latency and less power consumption at the same time. Both of the proposed architectures can reduce much latency in comparison with the conventional single-port 256x8 memory design.

1. INTRODUCTION

RC4 stream cipher [1, 2, 4, 5] is one of the symmetric encryption methods. The symmetric encryption indicates that the transmitter and the receiver share a same key. By using the same key, we can encrypt or decrypt data to achieve confidentiality. In theory, if nobody can discover the key, we can communicate and transmit data securely. Although RC4 is a symmetric cipher, the key which is used in RC4 is not like DES (Data Encryption Standard) or AES (Advanced Encryption Standard) [5], which encrypts or decrypts data with keys directly. The key used in RC4 simply permutes a 256-byte array. Once the permutation procedure is finished, the key is never used again. After that, a byte stream is selected from the 256-byte array in a systematic fashion, and it is used to encrypt or decrypt data.

RC4 uses a variable-length key K, where the key length is from 1 to 256 bytes, and a 256-byte array S is used in RC4. In initialization process, let S[i] be i, where i is from 0 to 255, and then we put the variable-length key in a 256-byte array T. If the length of the key K is 256 bytes, then T stores this key. Otherwise, if the length of a key K is L bytes (L < 256), the first L elements of T are copied from K and then K is repeated as many times as possible to fill out T. We use C-language to describe the initialization process as follows.

```
// initial memory
  for (i = 0; i <= 255; i++) {
       S[i] = i;
       T[i] = K[i % L]; }
```

Next, we use T to do initial permutation of S. Thus, the elements in S are still the values from 0 to 255. We use C-language codes to describe this process as follows.

```
// key shuffling
  j = 0;
  for (i = 0; i <= 255; i++) {
       R1 = S[i];
       j = (j + R1 + T[i]) % 256;
        R2 = S[j];
        S[j] = R1;
        S[i] = R2; }
```

Finally, we choose 1-byte value from the permuted S, and at the same time swap some two bytes in the S. For the sake of clarity, the corresponding C-language codes are described as follows.

```
// stream generation
  i = 0, j = 0;
  while (true) {
      i = (i + 1) mod 256;
      R1 = S[i];
      j = (j + R1) % 256;
      R2 = S[j];
      S[j] = R1;
      S[i] = R2;
      t = (R1 + R2) % 256;
      stream = S[t]; }
```

In [4, 6, 7, 8], the authors develop several FPGA or CPLD based RC4 architecture designs. In [4], a FPGA-based implementation of the RC4 stream cipher provides high data throughput using 8-bit word and variable key length. The FPGA-based RC4 schemes in [4, 8] reduce read/write access cycles for shuffling process. In this paper, the cell-based logic and standard library based SRAM [3] are applied to design and implement our proposed new low-latency RC4 architecture.

The paper is organized as follows. Section 2 describes the motivation of RC4 stream cipher design. In Section 3 the proposed low-latency architecture is presented and analyzed in details. The cell-based VLSI implementation and comparison results for RC4 are shown and discussed in Section 4. Finally, a conclusion is given in Section 5.

2. MOTIVATION OF INVESTIGATION

RC4 stream cipher has been used in the WEP (Wired Equivalent Privacy) algorithm of IEEE 802.11b wireless network [4, 5]. A new generational cipher standard of wireless network - IEEE 802.11i [1], in addition to using WEP algorithm, also constitutes the TKIP (Temporal Key Integrity Protocol) algorithm to resolve the known weakness of WEP. The main core of TKIP algorithm still uses the RC4

algorithm. Therefore, how to implement RC4 algorithm effectively is worth developing.

The implementation of the 256-byte S array in RC4 uses a 256x8 memory, while the implementation of T array can use a 128-bit register because the maximum key length used in TKIP and WEP is 128 bits. Thus, we do not use another 256x8 memory to implement the T array. The major process of RC4 algorithm is to shuffle the S array (i.e. 256x8 memory), and it will cost at least two read and two write cycles to swap two byte data, and only one key is applied for transferring a string of data every time in spite of data length. That is, there is a regular execution cycle for the shuffling of S array, i.e. at least 256 x 4 = 1,024 cycles. If we can reduce the execution cycle of this phase as much as possible, then we can reduce the latency and improve the throughput of RC4. So we focus on the low-latency permutation of S array in this paper.

3. RC4 ARCHITECTURE DESIGNS

3.1 Traditional Hardware Architecture

As the description in the foregoing section, when we implement the shuffling permutation of S array by using a traditional single-port 256x8 (SP 256x8) SRAM, it will cost at least two read and two write access cycles every time, and then the total operations need 1,024 read/write access cycles.

3.2 Proposed Low Latency Design with Dual-port 256x8 SRAM

In addition to using the SP 256x8 memory described in Section 3.1, we can implement RC4 hardware with a dual-port 256x8 (DP 256x8) SRAM. Due to the position j which will swap with current position i every time which is related to the data within position i (i.e. j = (j + S[i] + T[i]) % 256), we still need two read access cycles, but we only need one write access cycle every time, and then the total memory access cycles are 768 (i.e. 256x3) cycles. Figure 1 shows the shuffling process by using dual-port 256x8 SRAM to permute memory data.

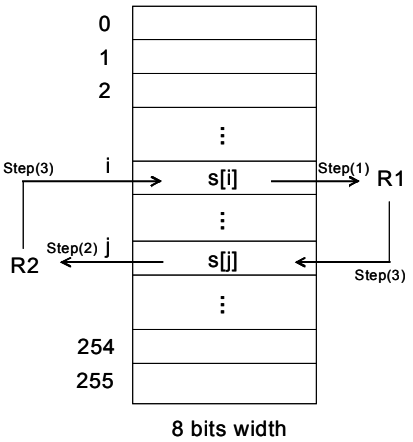

Figure 1 Shuffling process with dual-port 256x8 bits memory

In Section 4, we will find that the proposed RC4 architecture with DP 256x8 SRAM has lower power consumptions than the other architectures, but it requires more area cost for realizations.

3.3 Proposed Low Latency Design with Single-port 128x16 SRAM

The design concept of the proposed hardware architecture is similar to that of the architecture in [2]. This architecture uses one single-port 128x16 (SP 128x16) SRAM to express the S array. We keep two 1-byte values within one position instead of keeping one 1-byte value within one position. By using C-language codes, we describe the initialization of memory as follows.

```
// initial memory
for (i = 0; i <= 127; i++) {
        S[i][0] = 2*i;
        S[i][1] = 2*i + 1; }
```

Table 1 shows the write cycles which are needed to accomplish the initialization process of memory when we uses three different memory architectures. Figure 2 shows the shuffling process by using single-port 128x16 SRAM to permute memory data.

Table 1 Comparison of write cycles in three different memory initializations

Memory	# of write cycles
Single-port(SP) 256×8 [5]	256
Dual-port(DP) 256×8	128
Single-port(SP) 128×16	128

[Note] : Since the dual-port 256x8 memory can write two-byte data every time, it halves the write cycles.

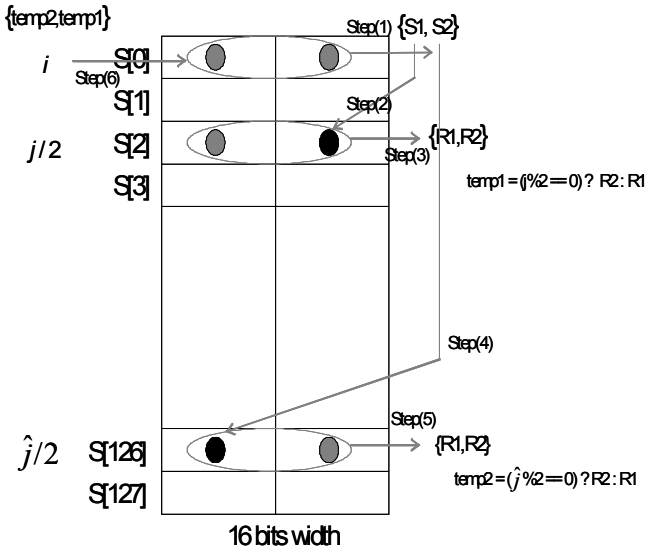

Figure 2 Shuffling process with 128x16 bits memory

The permutation process can be roughly described as follows.

1. Take two bytes data out in position i.

2. Use the first byte data in position i to compute the position j that is swapped. Divide the position j by 2 to get the correct position j/2.
3. Take two bytes data out in position j/2, write the first byte data in position i to appropriate location, and keep the data which are written back in the first byte of position i.
4. Use the second byte data in position i to compute the new position $\hat{j}$ again, and then divide the position j by 2 to get the correct position $\hat{j}/2$.
5. Take two bytes data out in position $\hat{j}/2$, write the second byte data in position i to appropriate location, and keep the data which are written back in the second byte of position i.
6. Write the data at step 3 and step 5 to position i.

The permutation process descriptions using C-language codes are shown in the following.

```
for (i = 0; i <= 127; i++) {
   // read all from memory
  S_highbyte = S[i][0];
  S_lowbyte = S[i][1];
    // address computation
  j = (j + S_highbyte + K[(2*i)%16]) % 256;
// read from memory
  R1 = S[j/2][0];
  R2 = S[j/2][1];
  temp1 = (j % 2 == 0)? R1: R2;    // backup
// write to memory
  if (j % 2 == 0)
       S[j/2][0] = S_highbyte;
  else
       S[j/2][1] = S_highbyte;
// update
     if (j / 2 == i) {
       if (j % 2 == 1)
       S_lowbyte = R1;    }
// address computation
     j = (j + S_lowbyte + K[(2*i+1)%16]) % 256;
// read from memory
       if (j / 2 == i) {
       R1 = temp1;
       R2 = S[i][1];    }
     else {
       R1 = S[j/2][0];
       R2 = S[j/2][1]; }
      temp2 = (j % 2 == 0)? R1: R2;    // backup
// write to memory
      if (j % 2 == 0)
        S[j/2][0] = S_lowbyte;
      else
        S[j/2][1] = S_lowbyte;
// update
      if (j / 2 == i) {
            if (j % 2 == 0)
        emp1 = S_lowbyte;    }
   // write all to memory
      S[i][0] = temp1;
      S[i][1] = temp2;       }
```

As mentioned above, the permutation process needs three read and three write access cycles every time, and then the total access cycles are 128 x 6 (i.e. 768) cycles, which are the same accesses as using dual-port 256x8 SRAM to implement memory permutation. The FPGA-based RC4 schemes in [4, 8] also need the same read/write access cycles (i.e. 768) for shuffling process. By using 16-bit memory to implement data permutations, we reduce the access times of memory. But we still need to design the memory permutation carefully, since there are some factors which lead the permutation not to be easy as above described.

As described in [2], this method can change further to use a single-port 64x32 SRAM to implement S array. But the complexity of using a single-port 64x32 SRAM is larger than that of using 16-bit memory, so the implementation of control circuits is difficult, and the reduction of memory access times (from 16-bit to 32-bit) is just only 20% (it's 25% from 8-bit to 16-bit). Therefore, we only use 16-bit memory to implement data permutations.

In Section 4, we will find that the proposed RC4 architecture with SP 128x16 SRAM achieves the same latency as the proposed architecture in Section 3.2, but it pays less area cost than the architecture with DP 256x8 SRAM.

4. VLSI IMPLEMENTATION RESULTS AND COMPARISON

We use cell-based design flow to realize the RC4 circuits, and adopt Artisan Standard Library SRAM Generator 0.18um process to generate memories. Figure 3 shows the hardware block diagram of RC4 processor. The comparison of permutation access cycles is shown in Table 2 when we use three different memory architectures to implement under the same constraints.

Table 2 Comparison of total numbers of read/write accesses in shuffling process when using different RAM modules

Memory architecture	# of read/write accesses in permutation
Single-port 256×8 [5]	1,024 (i.e. 256 × 4)
256x8 RAM [2]	1282
128×16 RAM [2]	896
256x3 RAM with FPGA [4]	768
Block RAM with FPGA [8]	768
Proposed Dual-port 256×8	768 (i.e. 256 × 3)
Proposed Single-port 128×16	768 (i.e. 128 × 6)

[Note] : For SP 256x8：2 read cycles + 2 write cycles = 4 cycles；for DP 256x8：2 read cycles + 1 write cycle = 3 cycles；for SP 128x16：3 read cycle + 3 write cycle = 6 cycles

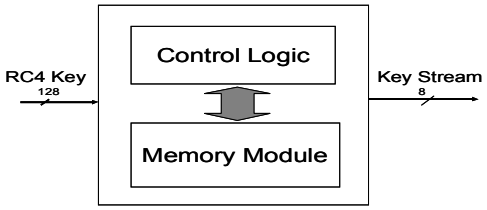

Figure 3 Block diagram of RC4 processor

The results in Table 2 show that the architecture with single-port 128x16 memory has the same permutation access cycles as that with dual-port 256x8 memory. Table 3 shows the area comparison of RC4 hardware which does not include the memory size. We use Synopsys Design Complier™ to synthesize three memory architectures by TSMC 0.18μm CMOS technologies.

Table 3-1 Area comparison of RC4 control logic by using Design Compiler™ to synthesis (Not including memory size)

RC4 Architectures	Logic Area (gate count)
Using single-port 256×8 [5]	2,457
Using proposed Dual-port 256×8	2,630
Using proposed Single-port 128×16	3,569

Table 3-2 Memory size comparison of RC4 by using Design Compiler™ to synthesis

RC4 Architectures	Memory Size (gate count)
Single-port 256×8 [5]	10,375
Proposed Dual-port 256×8	18,635
Proposed Single-port 128×16	12,296

From Tables 3-1, the order of pure logic area size in three different RC4 schemes is SP 256×8 < DP 256×8 < SP 128×16. But in Table 3-2, since the dual-port memory is larger than the other memories, the complete RC4 architecture with DP 256x8 memory consumes larger area than the other two architectures.

Table 4 Comparison of average power consumptions by using PrimePower™

RC4 Architectures	Power consumption
Using single-port 256×8 [5]	1.00925mW
Using proposed Dual-port 256×8	0.9612mW
Using proposed Single-port 128×16	1.42575mW

[Note]: Power consumptions are compared at the same frequency (50MHz), the same constraints, the same key input and the same plaintext input.

Table 4 shows the comparison of the average dynamic power consumption by using Synopsys PrimePower™ to estimate three different RC4 architectures. From Table 4, the RC4 design by using DP 256x8 SRAM needs less power consumption than the other designs due to the reduction of memory access times, and the memory control circuit is as easy as that by using conventional SP 256x8 memory architecture. However, when we use SP 128x16 memory for implementations, the control circuit of SP 128x16 memory will result in the additional power consumption which is greater than the power reduction from reducing memory accesses since it is complex. Therefore, the SP 128x16 architecture consumes the largest power.

5. CONCLUSION

We propose efficient low-latency RC4 architectures for IEEE 802.11i WEP/TKIP by using cell-based VLSI realization in this paper. For quick memory shuffling and permutation, we develop two different memory shuffling architectures to design the RC4 hardware. By comparing the RC4 architectures with three different memories, the latency comparison is SP128×16 = DP256×8 < SP256×8 and the proposed RC4 with DP256×8 SRAM requires less power consumptions than the other architectures. By using single-port 128x16 memory design, this architecture reduces 25% shuffling latency, compared with the conventional single-port 256x8 architecture. By using DP 256x8 memory design, this architecture pays larger area but achieves less power consumption and less latency simultaneously. Both of the proposed architectures can reduce much latency in comparison with the conventional single-port 256x8 memory design.

Acknowledgement

This work was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC96-2220-E-005-006.

References

[1] "IEEE Standard for Information technology— Telecommunications and information exchange between systems— Local and metropolitan area networks— Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 6: Medium Access Control (MAC) Security Enhancements," IEEE Std. 802.11i, 2004.

[2] Yukio Mitsuyama, Motoki Kimura, Takao Onoye, and Isao Shirakawa, "Architecture of IEEE802.11i Cipher Algorithm for Embedded Systems," IEICE Trans. on Fundamentals, vol. E88-A, no.4, pp. 899-905, April 2005.

[3] Artisan Standard Library SRAM Generator User Manual, pp. 3-1~3-39, Artisan Components, Inc., 2003.

[4] P. Kitsos, G. Kostopoulos, N. Sklavos and O. Koufopavlou, "Hardware Implementation of the RC4 Stream Cipher," Proceeding of 46th IEEE Midwest Symposium on Circuits & Systems, December 27-30, Cairo, Egypt, 2003.

[5] Stallings, "Cryptography and Network Security, Principles and Practices", Third Edition, Prentice Hall, 2003.

[6] P. D. Kundarewich, S. J.E. Wilton, A. J. Hu, "A CPLD- Based RC-4 Cracking System", The 1999 Canadian Conference on Electrical and Computer Engineering, May 1999.

[7] P. Hamalainen, M. Hannikainen, T. Hamalainen and J. Saarinen, "Hardware Implementation of the Improved WEP and RC4 Encryption Algorithms for Wireless Terminals", The European Signal Processing Conference (EUSIPCO'2000), September 5-8, Tampere, Finland, pp. 2289-2292, 2000.

[8] K.H Tsoi, K.H Lee and P.H.W Leong, "A Massively Parallel RC4 Key Search Engine", Proc. of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02), September 22-24, Napa, California, pp. 13-21, 2002.