Send Orders for Reprints to reprints@benthamscience.ae

2248 *The Open Cybernetics & Systemics Journal,* 2015, *9,* 2248-2253

Open Access

Performance Analysis of AES Algorithm on MPC Processor

Gao Haifeng[1], Han You[2,*], Song Yapeng[2] and Chen Shuizhong[1]

[1]*Department of Fire Control Electronics Research and Development, Luoyang Institute of Electro-optical Equipment, Luoyang, Henan, 471000, P.R. China;* [2]*School of Computer, Beijing Information Science and Technology University, Beijing, 100101, P.R. China*

Abstract: The Advanced Encryption Standard (AES) was widely accepted as the de facto standard in many security-related fields such as Embedded Development, Information Management, and Network Communication etc. In this paper, we firstly study the theory of AES algorithm and then conduct performance analysis of AES algorithm on MPC processor whose frequency is 800MHz according to the superscalar pipeline of E600 core. Test results show that the maximum encryption speed of AES algorithm is nearly 21MB/s under the circumstance of our experiment.

Keywords: AES Algorithm, performance analysis, MPC processor, superscalar pipeline.

1. INTRODUCTION

With the authorization of the National Security Agency (NSA) IBM developed a block cipher algorithm which became the first recognized and practical Data Encryption Standard (DES) later. On October 2, 2000, Rijndael algorithm was adopted by the National Institute of Standards and Technology (NIST) as a new generation of Advanced Encryption Standard (AES) [1]. With the appearance of differential and linear cryptanalysis, DES was replaced by AES gradually at the end of 20th century. As a kind of block iteration algorithm, AES has advantages of flexible design on key and block length, high safety and low memory usage etc. It could effectively resist brute-force attack, differential attack and linear cryptanalysis [2].

As we know, the MPC series processor is based on the core of "PowerPC" which is widely applied in the embedded field. In this paper, we mainly analyze the performance of AES algorithm under the circumstance of MPC processor and "VxWorks" which is an embedded real-time operating system. Through analyzing the feature of E600 superscalar pipeline and the execution time of dense memory access statement in source code of AES algorithm, we deduce the maximum encryption speed of AES algorithm under special testing environment.

2. PRINCIPLE OF AES ALGORITHM

In this section, we mainly analyze the principle of AES algorithm. According to the plaintext data encryption method, symmetric encryption algorithm can be divided into two categories: block cipher and stream cipher. Block cipher carves up the data into a fixed length, and the length of output cipher block is same with that of input plaintext block.

AES algorithm belongs to the block cipher algorithm. Its input block, output block and intermediate block in the process of encryption or decryption are 128bits. The key sizes of AES algorithm can be 128, 192 or 256 bits. The encryption and decryption diagram is shown in Fig. (**1**).

AES algorithm mainly includes four steps: SubBytes, ShiftRows, MixColumns, and AddRoundKey [3]. The SubBytes is a non-linear byte substitution which operates independently on each byte of "State" by S-box table [4].

1) SubBytes: This step use S-box to substitute the block one by one byte. The S-box is a matrix defined by AES. The high 4 bits of every byte in its "State" is set as row value, the low 4 bits as column value. Then take out the element of corresponding rank in S-box as the output. This step provides the non-linear transformation ability for AES's encryption.

2) ShiftRows: In this step, each row circularly moves an offset to the left. In the AES block of 128 bits, the first row of "State" remains unchanged, and the second row of the "State" shift left one byte circularly. Similarly, the third and fourth row shift left circularly 2 bytes and 3 bytes, respectively. After the ShiftRows, every column in the matrix is composed of elements of each column in the input matrix.

3) MixColumns: This step operates on every column independently, which make the 4 bytes of each column to combine with each other through linear transformation. Firstly, set the four elements of each column as a factor separately. It will become a polynomial in the finite field after merger. Then multiply this polynomial and s fixed polynomial. This step can also be considered as matrix addition and multiplication under the finite field [5]. The factor of matrix is based the maximum distance linear encoding between code words as well as the algorithm efficiency. These two steps of ShiftRows and MixColumns provide diffusibility for the cryptosystem.

4) AddRoundKey: In every encryption cycle, a set of round keys will be produced by the expansion of main key.

1874-110X/15

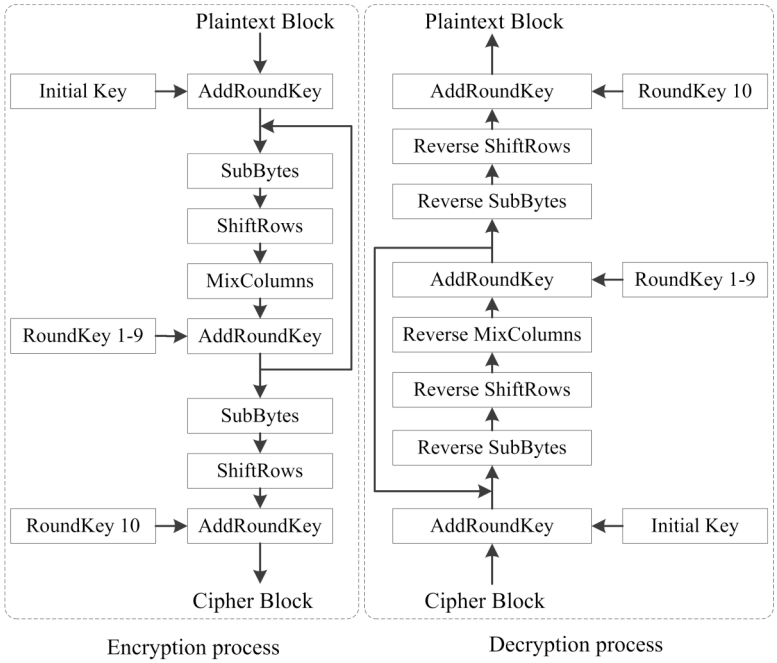

Fig. (1). Encryption and decryption diagram of AES algorithm.

This process is an operation of XOR between the four bytes of one specific column in "State" and one word of the round key. It has an impact on every bit in the "State" matrix in spite of the simple. The complexity of key expanding and other stages of AES can ensure the security of the algorithm.

3. E600 CORE OF MPC PROCESSOR

This section mainly studies the feature related to E600 core of MPC processor, which could contribute to the understanding of how the instruction executes. The execution of instruction in E600 core is generally divided into seven stages, namely fetch1, fetch2, decode/dispatch, issue, execute, complete and write-back, as shown in Fig. (**2**) [6]. The follows are the brief descriptions of the main stages.

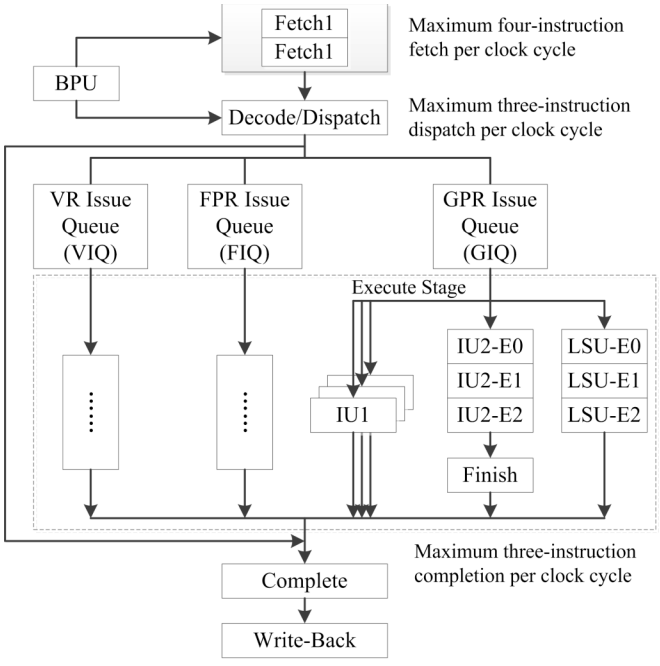

Fig. (2). E600 core superscalar pipeline diagram.

1) Fetch: Up to 4 instructions can be fetched from the instruction-cache per clock cycle only when they stay in one cache block. The size of a single block in "L1" cache is 32B, and each instruction possesses 4B. Therefore, if there are 4 instructions and the first one occupies the 0-3 bytes in a block, the 4 instructions can be fetched in one time. If the first of them possesses the 4-7 bytes in a block, the first 3 instructions can be fetched in one time. The rest may be deduced by analogy. According to the description above, the fetch stage is possible to become the limited factor of the superscalar pipeline so that the pipeline can't complete 3 instructions in a clock cycle.

2) Dispatch: In this stage, each instruction will be decoded sufficiently and be sent to the VIQ (VR Issue Queue), FIQ (FPR Issue Queue), or GIQ (GPR Issue Queue). And up to 3 instructions can be sent to GIQ in a clock cycle. The limitations to the superscalar pipeline in this stage are described as follows: ①The instructions fetched at Fetch stage are stored in IQ (instruction queue), and only the lowest 3 instructions (IQ0, IQ1, IQ2) can be sent out in the order. ②Only when there are available spaces in the CQ (Complete Queue) can the instructions be dispatched. ③There are available spaces in VIQ, FIQ or GIQ. ④There are available rename registers. ⑤There can be only one memory access instruction among the instructions that are dispatched at a time.

3) Issue: In this stage, the core will read the source operands from the rename registers and register files. This stage also assigns and routes instructions to the proper execution unit. The GIQ can accommodate 6 instructions. Instructions can be issued out-of-order from the bottom three GIQ entries (GIQ2–GIQ0). An instruction in GIQ1 destined for an IU1

does not have to wait for an instruction in GIQ0 that is stalled behind a long-latency integer divide instruction in the IU2. In addition, only the bottom 3 instructions GIQ0, GIQ1 and GIQ2 can be issued.

4) Execute: The E600 core contains 3 IU1 (Integer Unit 1), 1 IU (Integer Unit 2) and 1 LSU (Load/Store Unit). IU1 execute all integer instructions except multiply, divide, and special-purpose register (SPR) instructions. IU2 executes miscellaneous instructions including condition register (CR) logical operations, integer multiplication and division instructions, and SPR instructions. *Via* the superscalar pipeline, most of the instructions can be executed completely in a clock cycle. Several multiplication and division operations need 2 or more clock cycles, for they can't make full use of the pipeline.

5) Complete: Retires an instruction from the 16-entry CQ when all instructions ahead of it have been completed, the instruction has finished execution and no interrupts are pending. Retires as many as the bottom three instructions (CQ0, CQ1, and CQ2) per clock cycle. In this stage, up to 3 rename registers can be changed per clock cycle, which is to say 3 instructions using 4 rename registers can't retire in a clock cycle.

4. PERFORMANCE ANALYSIS

In this section, we emphatically discuss the performance of AES algorithm when it is running on MPC processor. It is worth mentioning that the performance analysis of AES algorithm is based on the circumstance of "VxWorks" Operating System and MPC processor Hardware Platform with double "PowerPC". The MPC processor's frequency is 800MHz. In addition, the AES algorithm is written by C Language and considers the characteristic associated with the E600 core [6], for example 32 integral General Purpose Registers. In the following two fractions, we compute the executive time of dense memory access instruction in the source code of AES algorithm. Then we analyze the executive time of memory access instruction sequence according to the superscalar pipeline of E600 core.

4.1. Execution Time of Single Array Access Statement

In the source code of AES algorithm, there are 84 times array operating statements similar to "A[i>>8]", 80 statements of "A[(i>>8) &0xff]", 40 "A[0]", 8 times Pointer Operation and 172 times Integer Arithmetic besides Shift and AND Operations. Through the auxiliary clock of E600 core, we measure the execution time of single array access statement and the accuracy of time is 20ns. Fig. (**3**) is used to measure the execution time of another code segment which is placed in the area "4".

1) The execution time of statement of "A[i>>3]": Put the code segment showed in Fig. (**4**) into the area "4" of Fig. (**3**). Then make 15 copies of the whole code segment, so it will be executed 15 times repeatedly. But the being tested code in the first two code segment is null, which is to measure the time difference between two consecutive calls of "sysMpc8641TmrValueGet(0)" function. In following 4 tests, the code structure is as same as this one. The test result shows that the time difference between two consecutive calls

```
1.  /*Enable Interrupt*/
2.  sysAuxClkEnable();
3.  gtccr = sysMpc8641TmrValueGet(0);
4.  /* Being tested code segment */
5.  gtccr2 = sysMpc8641TmrValueGet(0);
6.  gtccr &= 0x7fffffff;
7.  gtccr2 &= 0x7fffffff;
8.  /*Disable Interrupt*/
9.  sysAuxClkDisable();
10. time = (gtccr - gtccr2) * 20;
```

Fig. (3). Code segment used to measure time.

```
1.  for (i=8; i<2048; i+=8)
2.  {
3.      temp0 = s_box[i>>3];
4.      temp1 = dis_box[i >>3];
5.      temp2 = Te0[i>>3];
6.      temp3 = Te1[i>>3];
7.      temp4 = Te2[i>>3];
8.      temp5 = Te3[i>>3];
9.      temp6 = Td0[i>>3];
10.     temp7 = Td1[i>>3];
11.     temp8 = Td2[i>>3];
12.     temp9 = Td3[i>>3];
13. }
```

Fig. (4). Being tested code segment.

of "sysMpc8641TmrValueGet(0)" function is 80ns. Through the calculation of the test, we find that the average execution time is 31740ns which is the total time of 255*10=2550 "A[i>>3]"and "for loop" statements.

2) The execution time of "for loop" statement: Similar to the method of test above, the average execution time of "for loop" statements is 5190ns. So a "A[i>>3]" statement's running time equals (31740-5190)/(255*10)=10.4ns. As a clock cycle's time of the MPC processor is 1/800MHz=1.25ns, a "A[i>>3]" takes 10.4/1.25=8.32 clock cycles.

3) The execution time of "A[(i>>3) &0xff]": According to test result, the average execution time of total statements that include "A[(i>>3) &0xff]" and "for loop" is 35190ns. Therefore, the execution time of each "A[(i>>3) &0xff]" is (35190-5190)/(256*10)=11.7ns, namely 11.7/1.25=9.36 clock cycles.

4) The execution time of "A[0]": The test result shows that this average value of running time is 14580ns. Likewise, a "A[0]" statement's execution time is (14580-5190)/(256*10)=3.7ns with the exception of "for loop" statements, namely 3.7/1.25=2.96 clock cycles.

5) The study about Data Cache: On one hand, there are two data caches on the MPC processor, namely "L1" Cache with 32KB and "L2" Cache with 1MB. On the other hand, 10 unsigned integer arrays whose total size equals 4*256*10=10KB are declared in the source code of AES

algorithm. Because of "static", these arrays should be allocated memory space before program execution. But we don't know whether this 10KB data enter into Data Cache or Memory before running. At the first, we assume that the 10KB data would be placed into the Data Cache from Memory until being used. However, these 4 tests above couldn't verify this assumption because that the difference of execution time between the first "for loop" and others is too small to match the difference of access time between the memory and L1 cache.

In this test, the step length of "for loop" statement is changed from 1 to 8, which will makes execution time of the first "for loop" to be far greater than that of others if the 10

Table 1. Instruction pipeline of "A[i>>3]".

Clock Cycle	1	2	3	4	5	6	7	8	9	10
srwi	D	I	E1	E2	C					
mulhw	D	I	-	E0	E0	E1	F	C		
sub	D	I	E	-	-	-	-	C		
lwzux		D	I	-	-	-	E1	E2	E3	C

Table 2. Instruction pipeline of multiple "A[i>>3]".

Clock Cycle	1	2	3	4	5	6	7	8	9	10
srwi r11,r10,3	D	I	E1	E2	C					
mulhw r11,r11,4	D	I	-	E0	E0	E1	F	C		
sub r0,**,**	D	I	E	-	-	-	-	C		
lwzux r0, r11, r0		D	I	-	-	-	E1	E2	E3	C
srwi r12,r10,3		D	I	E1	E2	-	-	-	-	C
mulhw r12,r12,4		D	I	-	-	E0	E0	E1	F	-
sub r1,**,**			D	I	E	-	-	-	-	-
lwzux r1, r12, r1			D	-	-	-	I	-	E1	E2
srwi r13,r10,3			D	I	E1	E2	-	-	-	-
mulhw r13,r13,4				D	-	I	-	E0	E0	E1
sub r2,**,**				D	I	E	-	-	-	-
lwzux r2, r13, r2				D	-	-	-	-	I	-
Clock cycle	11	12	13	14	15	16	17	18	19	20
srwi r11,r10,3										
mulhw r11,r11,4										
sub r0,**,**										
lwzux r0, r11, r0										
srwi r12,r10,3										
mulhw r12,r12,4	C									
sub r1,**,**	C									
lwzux r1, r12, r1	E3	C								
srwi r13,r10,3	-	C								
mulhw r13,r13,4	F	-	C							
sub r2,**,**	-	-	C							
lwzux r2, r13, r2	E1	E2	E3	C						

arrays are stored in the Memory or "L2" Cache. The first running of "for loop" will transfer the data of 10 arrays from Memory to "L1" Cache. When the "for loop" statement is executed again, all memory access operations will hit "L1" Cache. However, the test result doesn't support this assumption. It shows that the execution time of each statement in first "for loop" is larger than the second for 1080/[(256/8)*10]=3.4ns, namely 3.4/1.25=2.72 clock cycles. In addition, the time difference between "L1" Cache's being hit and "L2" Cache is 13-1=12 clock cycles. "12 > 2.72" indicates that most of memory access instructions hit the "L1" Cache on the first visit of 10 arrays. Therefore, we can conclude that most data of the 10 "static" arrays may have been written into "L1" Cache before the running of source code.

4.2. Analysis of Execution Time based on Superscalar Pipeline

In this fraction, we analyze the execution time of memory access instruction of AES algorithm basing on superscalar pipeline of E600 core.

Let's take the memory access statement of "A[i>>3]" for example. When this array access statement is compiled, it will generate the following 4 assembler instructions [6, 7]: ① "srwi r11, r10, 3"; ② "mulhw r11, r11, 4"; ③ "sub r0, /*SPR*/, /*constant*/"; ④ "lwzux r0, r11, r0".

According to literature [7], the execution of multiply instruction "mulhw" and memory access instruction "lwzux" take 3 clock cycles, respectively. Shift instruction "srwi" takes 2 clock cycles and only needs one to produce available data. It spends 1 clock cycle on executing the integer arithmetic instruction "sub".

Since these three instructions "①②④"have relation to Register "r11", it takes 7 clock cycles to execute these four instructions. As shown in Table **1**, the execution of "A[i>>3]" begins at clock cycle 3 and ends at clock cycle 9 according to the instruction pipeline.

Table **1** shows that it will spend only 7 clock cycles on running the statement "A[i>>3]" even if instructions are executed sequentially, which is close to the time of 8.32 clock cycles in first test above.

In the first test, there are 10 parallel "A[i>>3]" statements in the body of "for loop", which means the four assembler instructions above will be executed 10 times repeatedly after compiling. Literature [6, 8] point out that E600 core could reduce the "Data Correlation" conflict between two instructions by adjusting the use of 32 integral General Purpose Registers. According to the restrictive condition mentioned above of instruction execution on E600 core, we could deduce the instruction pipeline of multiple "A[i>>3]" statements. The instruction pipeline is shown in Table **2**. From the beginning of clock cycle 8, a "A[i>>3]" statement will be executed every 2 or 3 clock cycles.

Without the optimization of compiling, a "A[i>>3]" statement's execution time is 8.32 clock cycles. However, it takes only 2~3 clock cycles with the full use of superscalar pipeline of E600 core, which explains the reason that speed of AES algorithm with GCC-O2 optimization [9] is nearly three times of that without optimization.

In summary, these test results in fraction A have referenced significance to analyze the execution time of AES algorithm although it is hard to deduce a C Language statement's running time accurately. Compared to the little integer arithmetic operations in AES algorithm, the memory access related statement is as dense as that in the 5 being tested code segments above. Therefore, the measured average execution time of code segment could represent the average running time of AES algorithm. In addition, we could deduce the relation between the execution time of statement with GCC-O2 optimization and that without optimization.

CONCLUSION

In this paper, we have studied the algorithm of AES firstly and then analyzed the characteristic which can influence the execution speed of AES algorithm in E600 core superscalar pipeline. By testing the execution time of main memory access statements in AES algorithm and writing the pipeline space-time diagram, we deduced that the maximum encryption speed of AES algorithm can reach is nearly 21MB/s under the condition of our experiment.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

Declared none.

REFERENCES

[1] K. Rahimunnisa, "FPGA implementation of AES algorithm for high throughput using folded parallel architecture," *Security and Communication Networks*, vol. 7, no. 11, pp. 2225-2236, 2014.

[2] A. Bogdanov, and V. Rijmen, "Linear hulls with correlation zero and linear cryptanalysis of block ciphers," *Designs, Codes and Cryptography*, vol. 70, no. 3, pp. 369-383, 2014.

[3] T. Hoang, and V. L. Nguyen, "An efficient FPGA implementation of the Advanced Encryption Standard algorithm, " In: *Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on*, Ho Chi Minh City, 2012, pp. 1-4.

[4] H. S. Deshpande, K. J. Karande, and A. O. Mulani, "Efficient implementation of AES algorithm on FPGA," In: *Communications and Signal Processing (ICCSP), 2014 International Conference on*, Melmaruvathur, 2014, pp. 1895-1899.

[5] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced en-cryption standard using a single fault," In: *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, Ardagna, Ed. Berlin Heidelberg: Springer, 2011, pp. 224-233.

[6] Freescale Semiconductor, Inc, "*e600 PowerPC Core Reference Manual*," 2006. [Online]. Available: http://cache.freescale.com/files/32bit/doc/ref_manual/E600CORERM.pdf. [Accessed: Apr. 7, 2015].

[7] Freescale Semiconductor, Inc, "*MPCxxx Instruction Set*," 1997. [Online]. Available: http://cache.freescale.com/files/product/doc/MPC82XINSET.pdf. [Accessed: Apr. 7, 2015].

[8] Freescale Semiconductor, Inc, "*MPC8641D Integrated Host Processor Family Reference Manual*," 2008. [Online]. Available: http://cache.freescale.com/files/32bit/doc/ref_manual/MPC8641DRM.pdf. [Accessed: Apr. 7, 2015].

[9] M. T. Jones, "Optimization in GCC," *Linux Journal*, vol. 11, no. 131, p. 11, 2005.

Received: June 10, 2015 Revised: July 29, 2015 Accepted: August 15, 2015