AES Implementation for RFID Tags: *The Hardware and Software Approaches*

Thanapol Hongsongkiat
New Product Research Department
Silicon Craft Technology Co., Ltd.
Bangkok, Thailand
thanapol@sic.co.th

Prabhas Chongstitvatana, Ph.D.
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University
Bangkok, Thailand
prabhas@chula.ac.th

***Abstract*—This paper presents two different implementations of 128-bit AES cryptography for RFID tags: hardware module and software program. The hardware AES module has been designed with 0.13 um CMOS technology at 1.5 V. The size of the module is 7229 equivalent gates, and the maximum throughput is 130 Mbps. The power consumption at 125 kHz is 6 uA, which meets specification for low-power RFID tags at low frequency. The software version is implemented on a custom 8-bit microcontroller. The program is written in assembly language with our proprietary instruction set. It can process AES encryption in 6,012 instructions, which takes 20,601 clock cycles, using 2 kBytes of instruction memory and 320 Bytes of data memory. It is targeted to use in high-frequency RFID applications.**

Keywords—AES; ASIC; RFID; low-power; low-freuency;

I. Introduction

The need for strong encryption in RF tags has increased in recent years. The most popular encryption algorithm is the Advance Encryption Standard (AES), which is certified by the U.S. National Institute of Standards and Technology (NIST). RFID tags with AES encryption are being adopted as a secured encryption standard in the industry. The two frequency ranges in which AES encryption is widely used are low frequency range (LF, 120-150 kHz) and high frequency range (HF, 13.56 MHz). These two applications have different constraints in terms of encryption time. The LF applications require encryption to finish within a few hundred RF cycles, In HF applications, AES processing time can be as long as several thousand RF cycles.

This paper describes the implementation of 128-bit AES encryption in custom design RFID chips under the processing time constraint for the two frequency ranges. Another two important constraints of ASIC design are also taken into account: small area and low power consumption. For LF applications where the processing time is limited, a hardware module is proposed. For HF applications, a custom microcontroller unit with an encryption program is proposed in order to provide programmability. The software version for encryption also saves chip area since it does not require extra hardware.

Section II gives a short overview of AES encryption and decryption algorithm. In Section III and IV, our methods of hardware and software AES implementation are described. We also discuss the performance of both methods and compare the result with other existing designs.

II. AES Algorithm

AES is a block cipher that converts plain text input with length of 128, 192, or 256 bits into cipher text output with the same length. The input text is processed as a two-dimensional array of bytes illustrated in Fig 1, which is extracted from [1]. The intermediate data are called States. The States are processed by four transformations: SubByte, ShiftRows, MixColumn, and AddRoundKey. The AES encryption and decryption flow chart, as presented in [2], is shown in Fig 2.

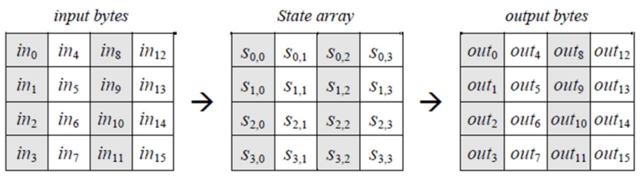
input bytes
in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
State array
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
output bytes
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}

Fig. 1. AES Encryption Data Format

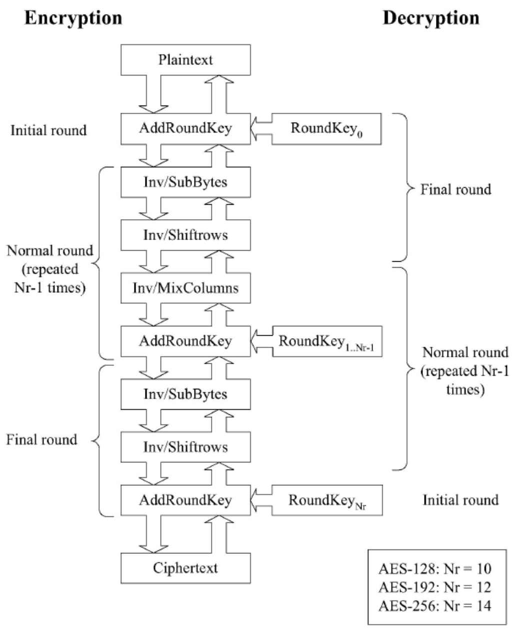

Fig. 2. AES Encryption and Decryption Flow Chart

A. *SuByte Transfomration*

The SubByte process substitutes each byte of the State with its corresponding S-box transformation. The SubByte transformation is the multiplicative inverse in the Galois Field GF(2^8) using the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ followed by an affine transformation. The InvSubByte transformation is the inverse affine transformation followed by the multiplicative inverse.

B. *ShiftRows Transformation*

The ShiftRow transformation rotates the last three rows of the State to the left. Row 1, 2, and 3 are rotated by 1, 2, and 3 bytes respectively. The InvShiftRow transformation rotates the rows to the right by the same amount of bytes.

C. *MixColumn Transformation*

The MixColumn transformation applies to each column of the State. It takes the four bytes of the column as a polynomial over GF(2^8), and multiply the polynomial with a fixed polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$, using modulo $x^4 + 1$. In case of InvMixColumn transformation, the column is multiplied with the polynomial $a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

D. *AddRoundKey Transformation*

The AddRoundKey transformation adds the State with the RoundKey. The addition is accomplished by bitwise XOR operation. The Key Expansion process takes the AES key as an input and expands it to a number of RoundKeys. Each RoundKey consists of four words. Each word is 4-byte long and is added to each byte of the State column.

The psudocode of the Key Expansion operation of AES128 encryption, as described in [1], is shown in Fig. 3.

```
RC[1..10] = ('01','02','04','08','10',
             '20','40','80','1B','36')
Rcon[i]   = (RC[i],'00','00','00')

for(i = 0; i < 4; i++)
  W[i] = (key[0], key[1], key[2], key[3])

for(i = 4; i < 44; i++) {
  temp = W[i - 1]
  if (i mod 4 == 0)
    temp = SubWord(RotWord(temp)) xor Rcon[i / 4]
  W[i] = W[i - 4] xor temp
}
```

Fig. 3. Psudocode for AES128 Key Expansion

Key[i] is one word of the input key, and 0 < i < 4

W[i] is one word of RoundKey, and 0 < i < 43

SubWord() applies S-Box transformation to a word.

RotWord() rotates left each word by one byte.

III. AES Hardware Implementation

The advantage of the hardware module over its software counterpart is that it can process encryption and decryption faster. The disadvantage is that the module occupies extra chip area. The data path of this hardware module relies on the architecture presented in [2]. This architecture is intuitive and modular, which allows easy customization and optimization.

A. *Top Module Block Diagram*

The top module architecture is shown in Fig 4. The AES_Top unit consists of three main units: Data Unit, Key Gen and Control.

B. *Data Units*

The structure of the Data Unit is shown in Figure 5. The Data Unit consists of sixteen Data Cells, four SBox, module, and one MCol module.

<u>Data Cell</u>

The Data Cell contains one byte of the. The Data Unit contains 16 Data Cells arranged as a 4x4 array. A Data Cell has three functions: load horizontal input, load vertical input, and add key to the State.

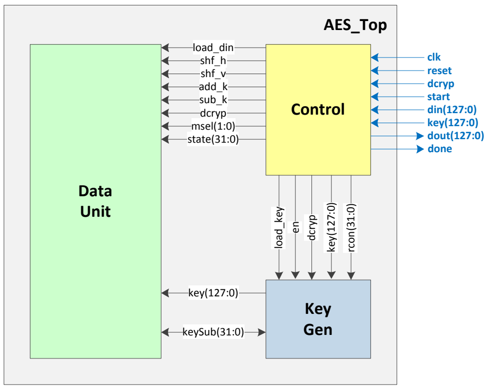

Fig. 4. Block Diagram of the AES Hardware Top Module

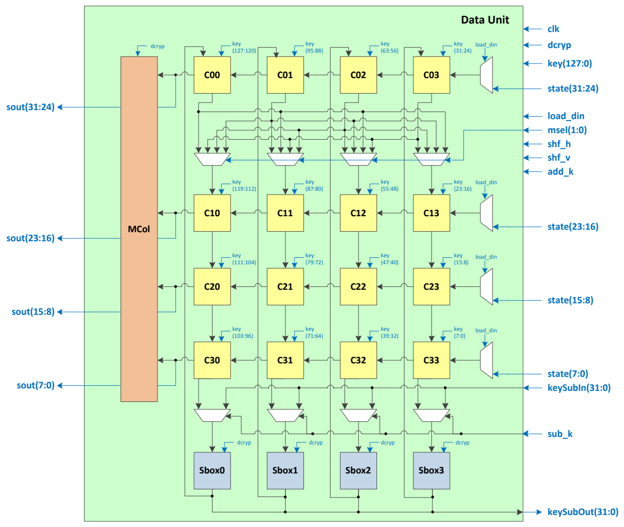

Fig. 5. Block Diagram of the Data Unit

SBox

SBox converts the input byte using multiplicative inversion and affine transformation. The input bytes are shifted down to the four SBoxes four bytes at a time during SubByte process. Therefore, the SubByte process takes four clock cycles to complete.

SBox can be implemented by storing pre-calculated data in a look-up table. However, the look-up version of the SBox is very big; the four SBoxes take about 46% of the AES module area. In order to reduce the size of the SBox, we implement it according to its mathematical definition: multiplicative inversion and affine transformation in GF(2^8).

The details of SBox implementation can be found in [4]. The overview of the module is shown in Fig 6. In case of encryption (INV = 0), the input goes into the multiplicative inversion module before going into affine transformation (AT) module. In case of decryption (INV = 1), the input goes into the inverse affine transformation (AT^{-1}) module before taking the multiplicative inverse.

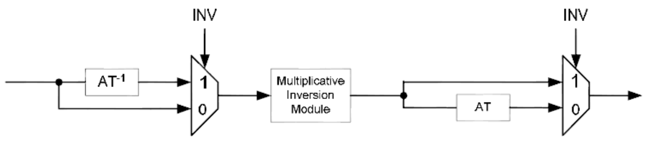

Fig. 6. Block Diagram of the SBox Module

Mix Column Module (MCol)

The Mix Column module is located on the leftmost side of the Data Cell row. During MixColumn transformation, the States are shifted to the left, passing through the MCol module one column at a time. Therefore, the MixColumn process takes four clock cycles to complete.

The MixColumn transformation involves multiplication in GF(2^8) with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. According to the finite field arithmetic property in [1], if S is an element in GF(2^8).

$$\begin{aligned} S \bullet \{02\} &= (S << 2) \oplus (S(7) \,\&\, \{1b\}) \\ &= \{\, S(6), S(5), S(4), (S(3) \oplus S(7)), \\ &\quad (S(2) \oplus S(7)), S(1), (S(0 \oplus S(7)), S(7) \,\} \end{aligned} \quad (1)$$

Equation (1) can be easily implemented with an 8-bit shift register and three XOR gates. We can expand this idea for the case of multiplication by four and eight.

$$S \bullet \{04\} = (S \bullet \{02\}) \bullet \{02\} \quad (2)$$

$$S \bullet \{08\} = (S \bullet \{04\}) \bullet \{02\} \quad (3)$$

We can decompose other multiplication in MixColumn transformation into several stages of multiply-by-2 and XOR. Every multiplication is complete within one clock cycle.

$$\begin{aligned} S \bullet \{03\} &= (S \bullet \{02\}) \oplus S \\ S \bullet \{09\} &= (S \bullet \{08\}) \oplus S \\ S \bullet \{0b\} &= (S \bullet \{08\}) \oplus (S \bullet \{02\}) \oplus S \\ S \bullet \{0d\} &= (S \bullet \{08\}) \oplus (S \bullet \{04\}) \oplus S \\ S \bullet \{0e\} &= (S \bullet \{08\}) \oplus (S \bullet \{04\}) \oplus (S \bullet \{02\}) \end{aligned} \quad (4)$$

It should be noted that in [2] there are sixteen Mix Column modules located inside every Data Cell. In our design we separate the MCol from the Data Cells. This substantially reduces the size of the whole module with the price of longer processing time.

C. Key Generator

Before starting AddRoundKey transformation, the controller must load the AES key into the generator. Then enable it to produce RoundKey. Key generation takes one clock cycle for each round.

In case of decryption, the Controller must load the AES key, enable the Key Generator to run forward until the last RoundKey is generated, after which it calculates the next RoundKey in reverse order. This forward calculation process takes 10 additional clock cycles.

The structures of the Key Generator in case of encryption and decryption are shown separately in Fig 7 and 8, although they are actually implemented with a single hardware module.

D. Controller

The Controller is a 6-state Finite State Machine. The operation of each state is explained in Table I. The state diagram during encryption and decryption are provided in Fig 9 and Fig 10. Each normal AES round takes nine clock cycles to complete, while the first round and the last round takes one and six clock cycles respectively. The total encryption time is 93 clock cycles, while the decryption time is 103 clock cycles.

TABLE I. OPERATING STATES OF THE CONTROLLER

State	Description	Processing Time (clk)
IDLE	Wait for Start signal	-
LOAD	Load input data into Data Unit, and load AES key into Key Generator. During decryption, it needs 10 extra clock to generate the last RoundKey	Encrypt: 4 Decrypt: 14
ADD_RKEY	Perform AddRoundKey operation	1
SUBBYTE_ SHFROW	Perform SubByte and ShiftRow by shifting the State down.	4
MIXCOL	Perform MixColumn operation by shifting the State to the left.	4
DONE	Generate done signal to indicate end of operation	-

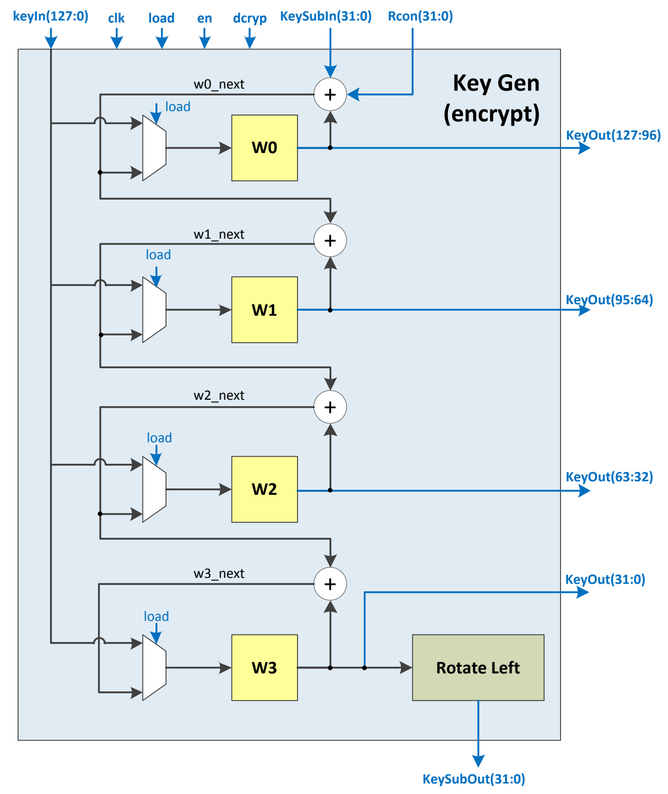

Fig. 7. Block Diagram of the Key Generator during Encryption

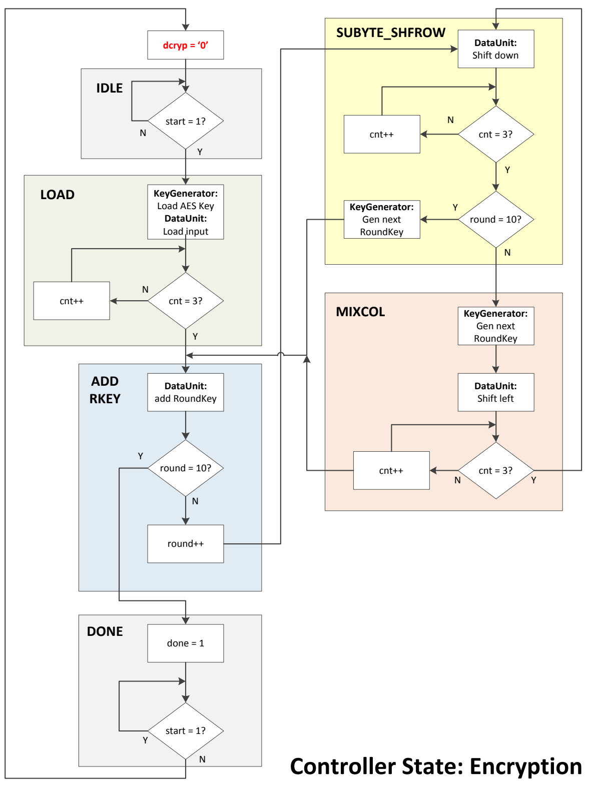

Fig. 9. State Diagram of the Controller during Encryption

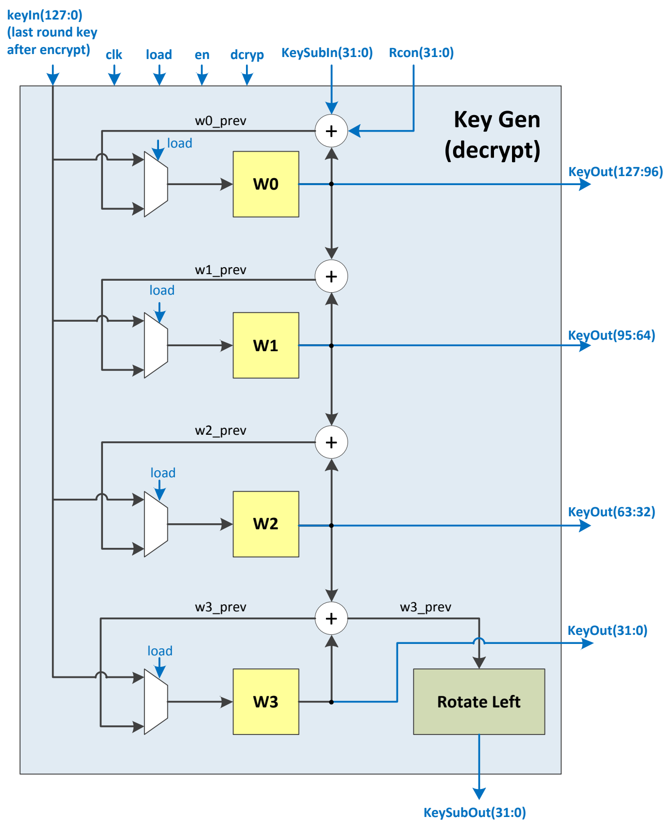

Fig. 8. Block Diagram of the Key Generator during Decryption

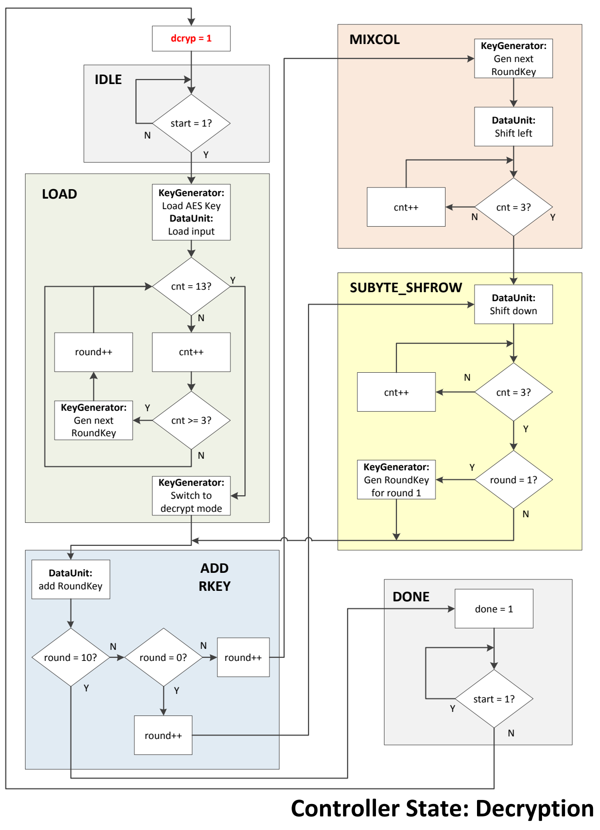

Fig. 10. State Diagram of the Controller during Decryption

E. AES Hardware Characteristic

The module AES_top is synthesized with the digital synthesis tool with 0.13um CMOS technology. The power consumption is measured by exporting the synthesized design to a netlist and simulates it using a netlist simulator. The supply voltage is 1.5V and the clock frequency is 125 kHz. The details are presented in Table II and III.

F. Performance Comparison

The characteristic of our AES module is given in Table IV in comparison with reference design in [2] and [3].

Our design is about 30% smaller than the size of [2] while yielding about as fast throughput. The longer processing time results from the fact that we only have one Mix Column module, in contrast to 16 modules in [2]. Therefore, our design requires four clock cycles while the design of [2] requires only one clock cycle for MixColumn. Another reason for longer processing time is the lack of Key Cache unit presented in [2].

Although our processing time is longer, the throughput is about the same as [2]. This implies that our maximum clock speed is faster. This is because the design of [2] is implemented on larger technology node. However, the maximum speed of our design can be increased by applying pipeline states in the DataUnit to increase throughput.

Comparing to [3], our design is about 30% bigger. The processing time is also 50% slower, resulting in slower throughput. This is because the design in [3] uses the optimized data path where the States pass the S-Box and MixColumn module in five clock cycles, while our design takes nine clock cycles.

TABLE II. AREA OF THE AES HARDWARE MODULE

Module Name	Area (um^2)	No. of Gates	%
Data Unit Total	(23,784)	(4,664)	(64.5%)
SBox x4	8,448	1,656	22.9%
MixColumn	5,226	1,025	14.2%
DataCell x16	8,384	1,644	22.7%
Other logic gates	1726	338	4.7%
Key Generator	(9,363)	(1,836)	(25.4%)
Controller and other logic gates	(3,722)	(730)	(10.1%)
Total	36,869	7,229	100.0%

TABLE III. POWER CONSUMPTION OF THE AES HARDWARE MODULE

Avg. Switching Power (at 1.5V, 125 kHz)	6.0 uA
Avg. Static Power	900 nA

TABLE IV. COMPARISON OF DIFFERENT AES MODULES

Design	Gate Equivalent	Processing Time (clk)	Through put(Mbit/s)	Tech. Node
Mangard [2]	10,799	64	128	0.60
Our design	7,229	103	130	0.13
Satoh [3]	5,398	54	311	0.11

IV. AES SOFTWARE IMPLEMENTATION

We now explore the implementation of AES encryption by a software program of a custom microcontroller unit. The advantage of using software version is that we can save chip area since the program resides in the instruction memory and no extra hardware is required. The disadvantage is that it takes much longer processing time than the hardware version.

The microcontroller is 8-bit, with 1024 kByte data memory, 8 kByte instruction memory, and 16 bytes of general-purpose registers. The instruction set consists of 49 instructions. To support AES, we add an instruction for 8-bit multiplication (*mul*). The microcontroller supports four interrupt requests, including a wake-up interrupt. It is designed for RFID applications, which require small area and low power. It has 3.42 CPI (cycle per instruction) data path.

The AES encryption program is written in assembly language, and assembled with our customized assembler. The assembled program is simulated by a cycle-accurate simulator to measure its characteristic. The result is shown in Table V.

We compare this implementation with a commercial soft-core processor, MicroBlaze [5], which is a simple 32-bit processor. It executes the AES benchmark in 43,500 clocks [6]. Our implementation with specialized instruction has huge advantage, being twice as fast with 8-bit data path.

The AES processing time of our microcontroller is in the acceptable range for HF applications. Nevertheless, we suggest the following optimization to further improve the execution speed.

1) Modify the microcontroller so that it executes every instruction in one clock cycle. This would reduce the encryption time by three times.
2) Introduce a new instruction, *mix*, to execute the MixColumn transformation in one instruction. This requires an additional hardware module, which is the MCol module designed in Section III. This will shorten the program by approximately 1,800 instructions. Assuming the *mix* instruction takes one clock, the speed up of execution time will be significant (around 30%).

TABLE V. CHARACTERISTIC OF THE AES PROGRAM

Total Instruction Executed	6012 instructions
Total Processing Time	20601 clock cycles
Total RAM Usage	320 bytes
Total Program Size	1024 instructions (2 kByte)

V. CONCLUSION

Two methods of AES cryptography implementations have been explored: by hardware and software. The hardware AES module size is 30% smaller than the design in [2] with comparable processing time. However, the design can be optimized toward [3]. The processing time of the module is fast enough for low-frequency RFID applications. The power consumption and area are small enough for ASIC implementation. The module can be implemented as a co-processor unit in a chip, or it can be integrated into a custom microcontroller as a peripheral module.

The software AES program is implemented based on our proprietary instruction set. The resource usage of the program can fit in the microcontroller. The processing time also meet our specification for high-frequency RFID applications. Two methods to speed up the microcontroller processing time are also proposed.

ACKNOWLEDGMENT

We would like to express our appreciation to Silicon Craft Technology for continuing support and providing valuable information to the project.

REFERENCES

[1] National Institute of Standards and Technology (NIST) , “The Advance Encryption Standard (AES)”, Federal Information Processing Standard Publication 197, November 26, 2001

[2] S. Mangard, M. Aigner, and S. Dominikus, “A highly regular and scalable AES hardware architecture” IEEE Transactions on Computers, 52(4):483–491, April 2003.

[3] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, “A compact Rijndael hardware architecture with S-Box optimization” Proc. Advances in Cryptology—ASIACRYPT 2001, pp. 239-254, 2001.

[4] Edwin NC Mui, “Practical implementation of Rijndael S-Box using combinational logic”, Texco Enterprise Ptd. Ltd.

[5] Xilinx, MicroBlaze Processor Reference Guide, 2008. http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[6] Satayavibul, C. and Chongstitvatana, P., "An embedded processor with instruction packing", Electrical Engineering, Electronics, Computer, Telecommunications and Information Technology (ECTI) International Conference, Chiang Rai, Thailand, 9-12 May 2007, pp.1135-1138.