International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 4
ISSN: 2088-5334

# A Fast AES Hardware Security Module for Internet of Things Applications

Minyoung Lee[a,1], HyunSeo Lim[a,2], Yoojeong Yang[a,3], and Sunhee Kim[a,4]

[a]*Department of System Semiconductor Engineering, Sangmyung University, Dongnam-gu, Cheonan, 31066, Korea*
*E-mail: [1]lmo9793@naver.com; [2]hihi981025@naver.com; [3]youjung4680@naver.com; [4]happyshkim@smu.ac.kr*

***Abstract*— As the Internet of Things is used in various fields, Internet of Things security has become important. Since most Internet of Things devices is implemented as embedded systems, they provide a software-implemented encryption algorithm. Most embedded systems use relatively low-performance CPUs and the software processes data serially, making it difficult to process complex security algorithms in real-time. Therefore, it is necessary to have a stable encryption module with less impact on system performance. In this paper, we designed an AES hardware security module. Because it is implemented with dedicated hardware, it can process a strong encryption algorithm in real time without affecting the performance of the system. The proposed AES hardware module is designed using Verilog-HDL, tested in ModelSim and implemented in Altera FPGA CycloneIV. The designed AES hardware adopts parallel processing technique and pipeline structure considering the computational complexity and processing order of the algorithm. As a result, it is faster than AES modules implemented in software. In addition, its latency was reduced to about 280 ns, which is about 16 % of the latency of the previous AES hardware module. Not only did the performance improve, but the number of logic elements and registers also decreased to 83.6% and 92.8%, respectively. The proposed AES hardware module is verified by applying it to a door lock system and is expected to be applied to various Internet of Things devices.**

***Keywords*— AES; encryption; internet of things; security.**

## I. Introduction

The Internet of Things means that things are sent and received directly to the Internet in real-time. Recently, the Internet of Things has been used in various fields and is rapidly developing [1], [2]. Smart home systems, healthcare services, sensor-embedded equipment, and storage containers are typical examples [3]-[6]. As the Internet of Things is used in many fields, there are many security problems [7]-[9]. In 2017, a smart toy for children, CloudPets leaked user information such as email accounts and voice message data [10]. Besides, as smart homes are activated a lot, CCTVs and door locks in the house are often hacked [11]. In order to prevent such hacking cases, many manufacturers are using encryption techniques to secure their Internet of Things devices [12]-[16].

One of the issues related to the Internet of Things security is limited resources [17], [18]. Many Internet of Things devices cannot afford security, or even simple algorithms because they usually use low-performance central processing units or microprocessing units. Therefore, it is necessary to implement reliable security modules using a few resources. Besides, since many Internet of Things devices use manufacturer-specific security algorithms, many of them are incompatible with each other when connected to a network [17]. In other words, standardized encryption algorithms are required for interconnection.

In this paper, we implemented a cryptographic algorithm for the security of the Internet of Things as a hardware system. The used encryption algorithm is the Advanced Encryption Standard (AES) algorithm. The AES algorithm is a cryptographic algorithm used as a US standard [19]. The encryption block size is 128 bits, and the block size can be extended [20], [21]. In addition, unlike the Data Encryption Standard (DES) algorithm, which is a symmetric-key algorithm with the 56-bit key size for the encryption of digital data [22], the AES algorithm has a large block size and is almost impossible to hack using the brute force method [23]. The AES algorithm is considered as one of the suitable algorithms for Internet of Things applications [24]-[26]. Instead, the AES algorithm requires more processing than encryption algorithms such as DES, so it must be designed to operate quickly [27].

Previously, many AES algorithms were designed in software [28]-[30]. Security algorithms for the Internet of Things are applied to data transmitted and received in real-time. However, the encryption algorithm implemented in the software does not perform well in terms of speed because data are processed serially [31]. Therefore, we implemented the AES algorithm in hardware instead of software.

A cryptographic device implemented in hardware can be faster than a cryptographic device implemented in software because it can use parallel processing techniques. So, it can be used without any inconvenience and overloading the primary system when applied to the Internet of Things. Reference [32] implemented the AES algorithm in hardware. Since the encryption module was designed using dedicated hardware, it did not put a load on the system and was naturally faster than those implemented in software. However, in processing the encryption algorithm, not only was it processed serially but also did not consider the difference in processing time according to the complexity of the algorithm. So, the overall speed improvement was not considerable.

In consideration of these problems, we propose a fast AES algorithm hardware module that uses pipeline techniques as well as parallel techniques. In session II, we introduce the AES algorithm and describe the architecture of the proposed AES hardware. Session III shows simulation and implementation results. The next section draws the conclusion.

## II. MATERIALS AND METHOD

### A. *AES Algorithm*

The AES algorithm consists mainly of initial round, main round, and final round. Fig. 1 shows how each of the three rounds is organized and in what order the algorithms are performed. The initial round includes one AddRoundKey block. The main round consists of SubBytes, ShiftRows, Mixcolumns, and AddRoundKey blocks, which in turn perform their operations. This main round is repeated nine times in total. The final round consists of SubBytes, ShiftRows and AddRoundKey blocks.

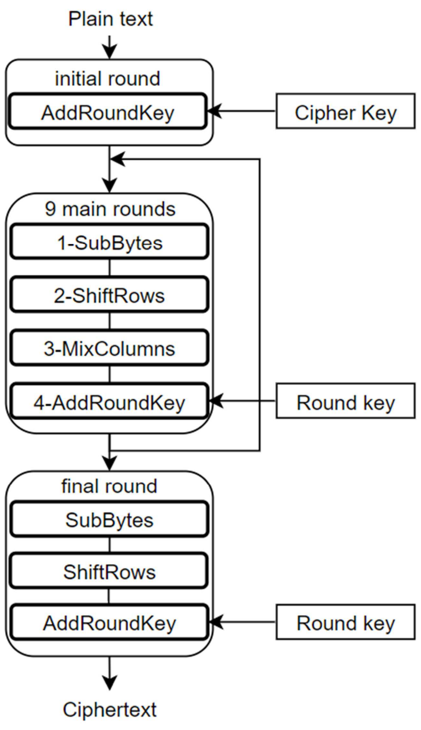
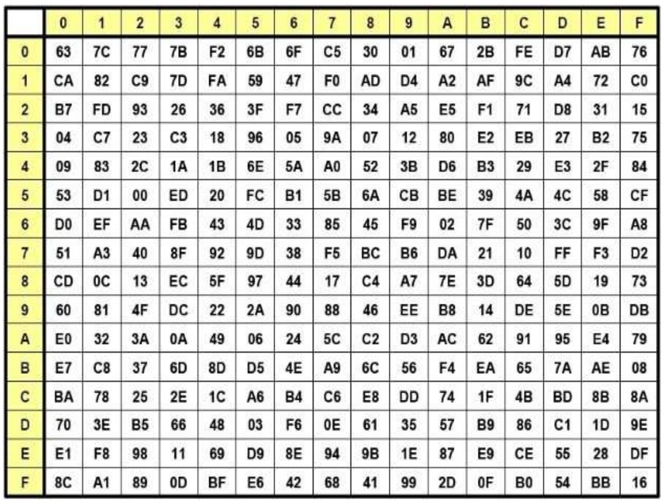


Fig. 1 Ciphering of AES Algorithm

The SubBytes block replaces input byte data in the hexadecimal format according to the S-box, as shown in Fig. 2. The entered hexadecimal front bits correspond to the column in the replacement table, and the back bits correspond to the row. For example, if a hexadecimal number ‘53’ is entered, a replacement number is ‘ED’.



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 7C | 77 | 7B | F2 | 6B | 6F | C5 | 30 | 01 | 67 | 2B | FE | D7 | AB | 76 |
| 1 | CA | 82 | C9 | 7D | FA | 59 | 47 | F0 | AD | D4 | A2 | AF | 9C | A4 | 72 | C0 |
| 2 | B7 | FD | 93 | 26 | 36 | 3F | F7 | CC | 34 | A5 | E5 | F1 | 71 | D8 | 31 | 15 |
| 3 | 04 | C7 | 23 | C3 | 18 | 96 | 05 | 9A | 07 | 12 | 80 | E2 | EB | 27 | B2 | 75 |
| 4 | 09 | 83 | 2C | 1A | 1B | 6E | 5A | A0 | 52 | 3B | D6 | B3 | 29 | E3 | 2F | 84 |
| 5 | 53 | D1 | 00 | ED | 20 | FC | B1 | 5B | 6A | CB | BE | 39 | 4A | 4C | 58 | CF |
| 6 | D0 | EF | AA | FB | 43 | 4D | 33 | 85 | 45 | F9 | 02 | 7F | 50 | 3C | 9F | A8 |
| 7 | 51 | A3 | 40 | 8F | 92 | 9D | 38 | F5 | BC | B6 | DA | 21 | 10 | FF | F3 | D2 |
| 8 | CD | 0C | 13 | EC | 5F | 97 | 44 | 17 | C4 | A7 | 7E | 3D | 64 | 5D | 19 | 73 |
| 9 | 60 | 81 | 4F | DC | 22 | 2A | 90 | 88 | 46 | EE | B8 | 14 | DE | 5E | 0B | DB |
| A | E0 | 32 | 3A | 0A | 49 | 06 | 24 | 5C | C2 | D3 | AC | 62 | 91 | 95 | E4 | 79 |
| B | E7 | C8 | 37 | 6D | 8D | D5 | 4E | A9 | 6C | 56 | F4 | EA | 65 | 7A | AE | 08 |
| C | BA | 78 | 25 | 2E | 1C | A6 | B4 | C6 | E8 | DD | 74 | 1F | 4B | BD | 8B | 8A |
| D | 70 | 3E | B5 | 66 | 48 | 03 | F6 | 0E | 61 | 35 | 57 | B9 | 86 | C1 | 1D | 9E |
| E | E1 | F8 | 98 | 11 | 69 | D9 | 8E | 94 | 9B | 1E | 87 | E9 | CE | 55 | 28 | DF |
| F | 8C | A1 | 89 | 0D | BF | E6 | 42 | 68 | 41 | 99 | 2D | 0F | B0 | 54 | BB | 16 |

Fig. 2 S-box: replacement table for the SubBytes block

To perform the ShiftRows, first, configure 4x4 array in 16-byte unit. And then, ShiftRows shifts each row of 4x4 data to the right, where the Nth row is shifted by N-1.

As shown in Fig. 3, MixColumns uses an array formula using Galois Field, which multiplies the left S matrix with a constant matrix and performs an exclusive OR operation of each term to find the value of the right S' matrix. As a result, the 4-byte data $\{s_0, s_1, s_2, s_3\}$ are replaced by the following equation (1)-(4).

$$s'_{0,c} = (\{02\}\cdot s_{0,c}) \oplus (\{02\}\cdot s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \quad (1)$$
$$s'_{1,c} = (\{02\}\cdot s_{1,c}) \oplus (\{02\}\cdot s_{2,c}) \oplus s_{3,c} \oplus s_{0,c} \quad (2)$$
$$s'_{2,c} = (\{02\}\cdot s_{2,c}) \oplus (\{02\}\cdot s_{3,c}) \oplus s_{0,c} \oplus s_{1,c} \quad (3)$$
$$s'_{3,c} = (\{02\}\cdot s_{3,c}) \oplus (\{02\}\cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \quad (4)$$

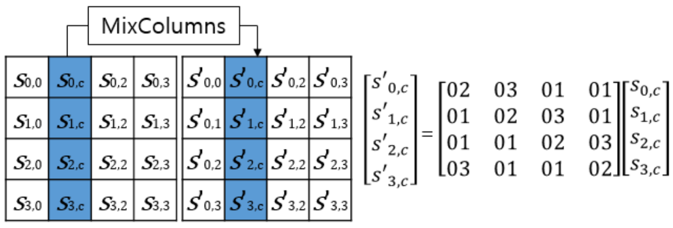


Fig. 3 Mixcolumns operation

AddRoundKey is the process of performing a bitwise exclusive OR operation of Roundkey and the output of MixColumns. The input of AddRoundKey, Round key, is generated through the KeyExpansion process, as shown in Fig.4. It takes the CipherKey as input and generates the RoundKeys by repeating the main round 43 times. The main round for KeyExpansion consists of RotWord, SubWord, XORwithRcon, and XORwithW[i-4] processes. RotWord moves 8 bits from the MSB to LSB of a 32-bit word, and the remaining bits are shifted to the MSB as many as 8 bits empty. SubWord functions the same as SubBytes described above. XORwithRcon is the process of performing a bitwise exclusive OR operation of Rcon, as shown in Fig. 5, and the output of SubWord. And then, the output of XORwithW[i-4], w[i], is the result of a bitwise exclusive OR operation on the previous value w[i-1] and the four previous value w[i-4].

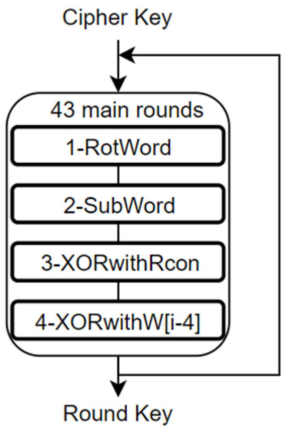


Fig. 4 KeyExpansion Process

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 01 | 02 | 04 | 08 | 10 | 20 | 40 | 80 | 1b | 36 |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

Fig. 5 Rcon

The encryption process ends when the plain text goes through the first round, the main round with nine repetitions and the final round, using the CipherKey modified through the KeyExpansion process.

## *B. AES Hardware Design*

First, the algorithm was designed in C language and verified using MS Visual Studio before it was implemented in hardware. Three rounds were implemented as separate functions. Analysis of functions designed in C language showed that the Cipherkey generation process, or KeyExpansion, was expected to be more complex than the encryption process. Moreover, MixColumns was expected to be more complex than ShiftRows and SubBytes. Based on these results, we decided on the architecture of an AES hardware module.

For the hardware-designed AES module in [32], after the 128-bit Cipher Key Expansion was completed over 47 clocks, the encryption process was performed on plain data using the CipherKey and the generated RoundKeys over 13 clocks. Thus, a total of 60 clocks were required. As expected, KeyExpansion has more computational capacity than the encryption process. Thus, the clock rate of the AES module in [32] was determined by the clock rate of KeyExpansion, which takes the longest processing time.

To increase the clock speed while reducing the total number of clocks required, we have designed the AES hardware module by applying parallel processing techniques and pipeline techniques that reflect the computational complexity and processing order.

In the AES algorithm, the input plain text passes through each process in the order of SubBytes, ShiftRows, and MixColumns. Next, an exclusive OR is performed on it with Roundkey. If the plain text encryption process is run; after all, RoundKeys are generated, like [32], the overall processing time will be extended. Thus, as shown in Fig 6, we proceeded with KeyExpansion and plain text encryption at the same time.

KeyExpansion consists of a total of 43 rounds, and the encryption process consists of a total of 11 rounds. The encryption processes data in 16-byte units, while KeyExpansion processes data in 4-byte units. So the encryption only needs to operate one round while KeyExpansion operates in four rounds. Therefore, the one main round of the encryption was broken down into the four-stage pipeline, as shown in Fig. 6.

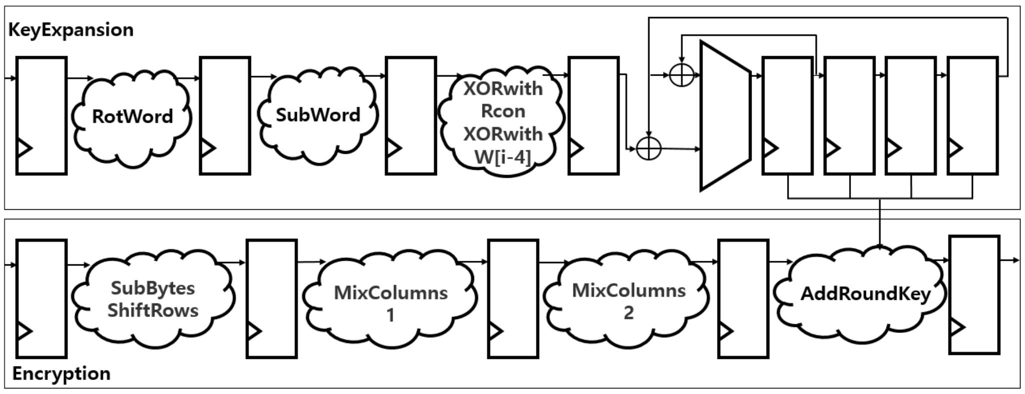


Fig. 6 AES Block Diagram

The main round of the encryption consists of SubBytes, ShiftRows, MixColumns, and AddRoundKey. As expected, MixColumns is more complex than SubBytes and ShiftRows. Since it is good that all the stages take the same amount of time to process their work, SubBytes and ShiftRows were grouped into one stage, and MixColumns was divided into two stages. This pipelining significantly increased the clock rate of the encryption. To increase the KeyExpanion clock rate to a level similar to the encryption, KeyExpansion was also divided, as shown in Fig. 6. As a result, the proposed hardware took a total of 43 clocks to encrypt the plain text.

As shown in Fig. 7, the decryption process consists of an initial round, main round, and final round. The decryption reverses all processes of the encryption. Since there were no

Mixcolums in the encryption final round, there were no Inv_Mixcolumns in the initial decryption round. Inv_AddRoundKey, Inv_MixColumns, Inv_ShiftRows, and SubBytes of the decryption main round reverse AddRoundKye, MixColumns, ShiftRows, and SubBytes, respectively. The decryption main round also reverses the whole process of the leading encryption round and is repeated nine times as the first encryption round. Moreover, the final round includes only one Inv_AddRoundKey, like the initial round of the encryption.

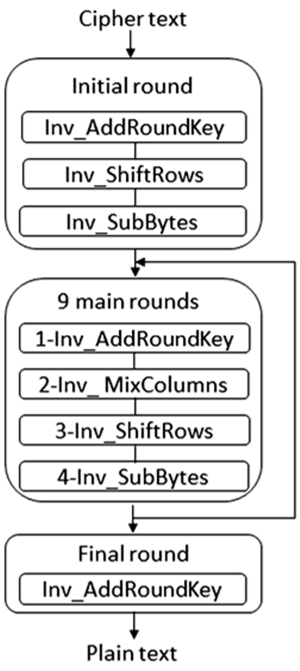


Fig. 7 Decryption Algorithm

## C. *Test System Design*

A simple door lock system has been designed to show the adaptability to Internet of Things applications of the proposed AES hardware module. The door lock system was designed using Arduino with an 8-bit Atmega328 and GPIOs. Fig. 8 shows the block diagram of the door lock system. This system has a keypad and an SPI master block for communication with FPGA providing an SPI slave block, storage and the AES algorithm module designed in this paper.

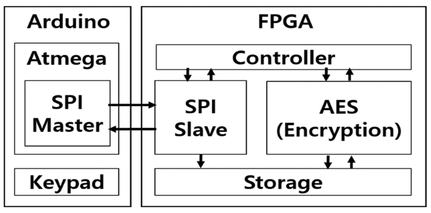


Fig. 8 Door lock System Block Diagram

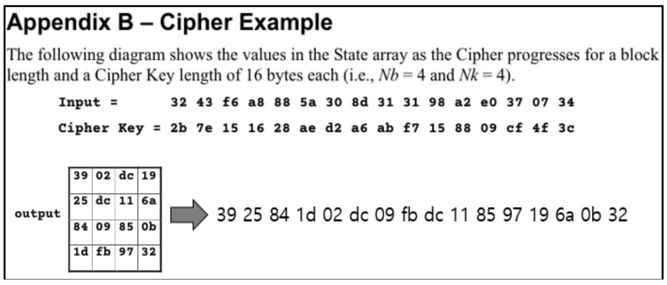
When in password setting mode, the Arduino system passes the password entered via the keypad to the FPGA via SPI. The AES module encrypts the number, stores an encrypted password in the storage, and deletes the raw password. When a number is entered in the door lock system to verify the password, the SPI master sends the number to SPI slave that stores it. The AES module encrypts the number and compares the stored encrypted password with the newly encrypted password. The AES module sends a matched/unmatched signal to the Arduino system using SPI.

We also designed the encryption module as well. When a user enters a number, the stored encrypted password can be decrypted and compared to the entered number. However, in order to prevent the raw password from being disclosed, the stored encrypted password is not decrypted. Instead, the entered number is encrypted to compare the result with the stored encrypted password.

## III. Result and Discussion

### A. *Function Simulation Results*

We designed an AES module using the Verilog-HDL language. And then, it was simulated in ModelSim. It was verified by comparing simulation results with the example data from the standard, as shown in Fig. 9 [7].



**Appendix B – Cipher Example**

The following diagram shows the values in the State array as the Cipher progresses for a block length and a Cipher Key length of 16 bytes each (i.e., $Nb = 4$ and $Nk = 4$).

```
Input =      32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
Cipher Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
```

output

| | | | |
|---|---|---|---|
| 39 | 02 | dc | 19 |
| 25 | dc | 11 | 6a |
| 84 | 09 | 85 | 0b |
| 1d | fb | 97 | 32 |

39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32

Fig. 9 Example Data from the Standard

Fig. 10 and Fig. 11 show the simulation results of the encryption module. Signals for encryption are clock, reset, data_in, restart, data_out, data_in_valid, and data_out_valid from above. The data_in in Fig. 10 is 128-bit 0x3243f6a8885a308d313198a2 e0370734 and data_in_valid is 1, which means values of data_in is ready to perform encryption. The data_out in Fig. 11 is 128-bit 0x3925841d02dc09fbdc118597196a0b32 and data_ out_valid is 1, which means values of data_out is valid encryption results. The encryption results are the same as the example values, so the encryption process has been verified to operate normally.

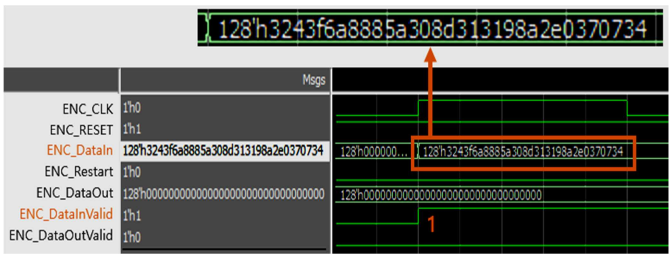

128'h3243f6a8885a308d313198a2e0370734
Msgs
ENC_CLK 1'h0
ENC_RESET 1'h1
ENC_DataIn 128'h3243f6a8885a308d313198a2e0370734
ENC_Restart 1'h0
ENC_DataOut 128'h00000000000000000000000000000000
ENC_DataInValid 1'h1
ENC_DataOutValid 1'h0
128'h000000...
128'h3243f6a8885a308d313198a2e0370734
128'h00000000000000000000000000000000
1

Fig. 10 Encryption Simulation – Input Data

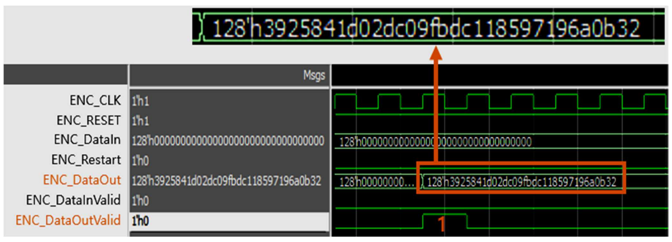

128'h3925841d02dc09fbdc118597196a0b32
Msgs
ENC_CLK 1'h1
ENC_RESET 1'h1
ENC_DataIn 128'h00000000000000000000000000000000
ENC_Restart 1'h0
ENC_DataOut 128'h3925841d02dc09fbdc118597196a0b32
ENC_DataInValid 1'h0
ENC_DataOutValid 1'h0
128'h00000000000000000000000000000000
128'h00000000...
128'h3925841d02dc09fbdc118597196a0b32
1

Fig. 11. Encryption Simulation – Output Data

Fig. 12 and Fig. 13 show simulation results of the decryption module. Signals for decryption are clock, reset, data_in, restart, data_out, data_out_valid, and data_in_valid from above. The data_in in Fig. 12 is 128-bit 0x3925841d02dc09fbdc118597196a0b32 and data_in_valid is 1, which means values of data_in is ready to perform decryption. The data_out in Fig. 13 is 128-bit 0x3243f6a8885a308d313198a2e0370734 and data_out_valid is 1, which means values of data_out is valid decryption results. The decryption results are the same as the example values. Therefore, the decryption process has been verified to operate normally.

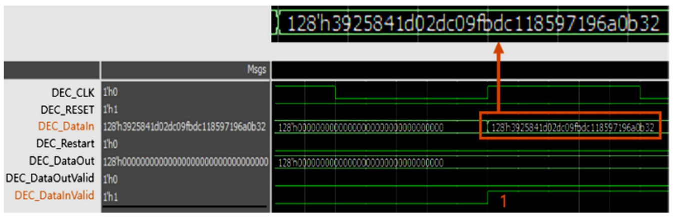


Fig 12. Decryption Simulation – Input Data

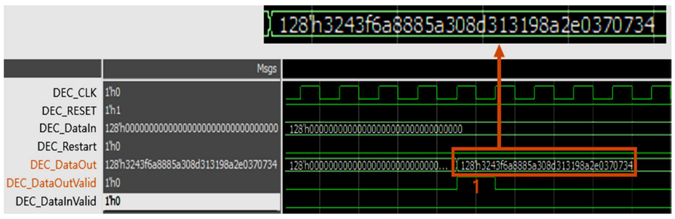


Fig 13. Decryption Simulation – Output Data

Fig. 14 shows the simulation results of the test door lock system. Signals for the door lock system are spi_clk, spi_slavle_sel, spi_master_out_slave_in (SPI_MOSI), spi_master_in_slave_out (SPI_MISO), system_clk, ram_data [0]~[7], ram_write_enable, ram_address, enc_start and cmp_start from above. When in password setting mode, the spi master sends a setting command and an 8-byte password (SPI_CLK, SPI_SLAVE_SEL, and SPI_MOSI are active). The spi slave received and stored the password (RAM_WEN is active). In this case, the length of a password is 8 bytes. When enc_start is 1, the AES module encrypts the password using system_clk that is faster than spi_clk.

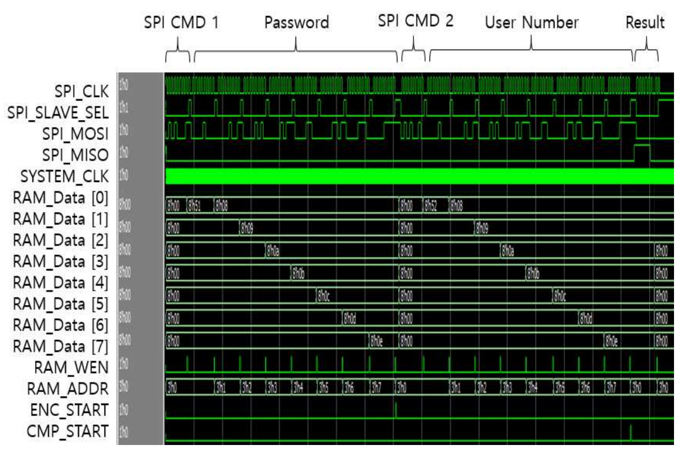
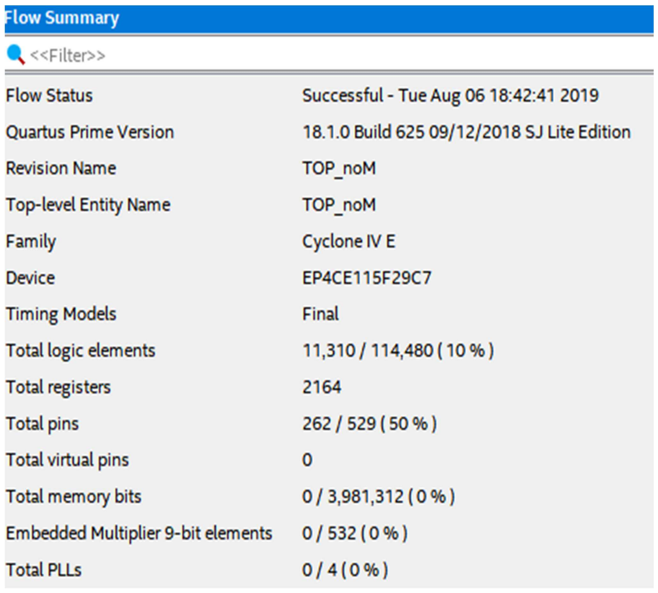


Fig 14. DoorLock Hardware Part Simulation

In verification mode, the spi master sends a compare command and an 8-byte number. After the number is received, it is encrypted according to cmp_start. Compare result is sent to the system via the SPI (SPI_MISO is active). We checked the overall operation of the door lock system.

## B. *Implementation Results*

We implemented the AES hardware using Quartus Prime and an Altera CycloneIV EP4CE115F29C7. As shown in Fig. 15, 11,310 logic elements and 2,164 registers were used.

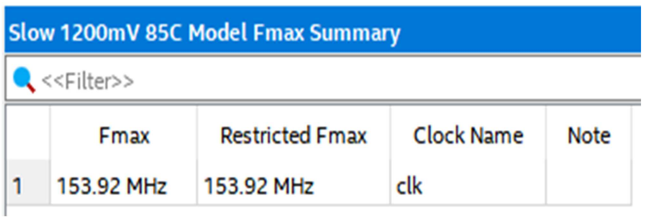

Flow Summary

<<Filter>>

| | |
|---|---|
| Flow Status | Successful - Tue Aug 06 18:42:41 2019 |
| Quartus Prime Version | 18.1.0 Build 625 09/12/2018 SJ Lite Edition |
| Revision Name | TOP_noM |
| Top-level Entity Name | TOP_noM |
| Family | Cyclone IV E |
| Device | EP4CE115F29C7 |
| Timing Models | Final |
| Total logic elements | 11,310 / 114,480 ( 10 % ) |
| Total registers | 2164 |
| Total pins | 262 / 529 ( 50 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 3,981,312 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 532 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

Fig 15. Implementation Summary

Fig. 16 shows the maximum frequency that can be implemented. The maximum frequency is about 154 MHz.


Slow 1200mV 85C Model Fmax Summary

<<Filter>>

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 153.92 MHz | 153.92 MHz | clk | |

Fig.16. Maximum Frequency Summary

Table 1 compares the implementation results of the previous AES hardware with the results of the implementations of the proposed AES hardware. The previous design requires 60 clocks to encrypt 128-byte plain text, and its maximum frequency is 45 Mhz. The proposed design requires 43 clocks, and its maximum frequency is 153.92 MHz. In other words, the data processing speed of the previous design was 3.75 cycles/bytes, but the data processing speed of the proposed design was 2.68 cycles/byte, which is about 140 % improvement. And the latency decreased by about a sixth from 1,758 ns to 280 ns. Not only did the performance improve, but the number of logic elements and registers also decreased to 83.6% and 92.8%, respectively.

TABLE I
COMPARISON OF IMPLEMENTATION RESULTS

| | cycles/byte | Max. Frequency | Latency | Total logic elements | Total registers | Implementation Device |
|---|---|---|---|---|---|---|
| Previous design in [13] | 3.75 | 45 MHz | 1,758 ns | 13,517 | 2330 | CycloneII EP2C70F896C8 |
| Proposed design | 2.68 | 153.92 MHz | 280 ns | 11,310 | 2164 | CycloneIV EP4CE115F29C7 |

The door lock system was implemented and verified, as shown in Fig. 17. A keypad was connected to the ATmega 328 through GPIOs of the Arduino. The SPI master was implemented in the ATmega 328 by software, and SPI input/output signals are allocated to GPIOs of the Arduino.

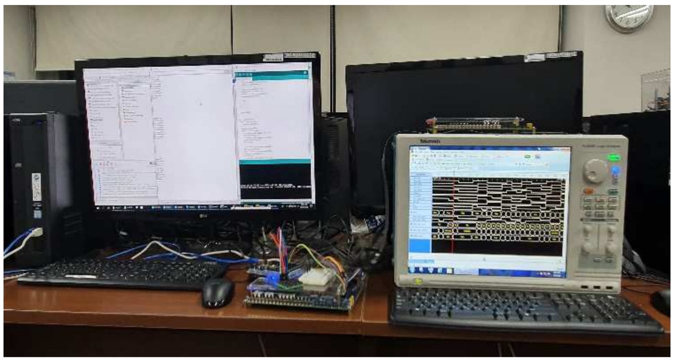


Fig 17. System Emulation Environments

The SPI slave was implemented in the FPGA, and SPI input/output signals were assigned to the GPIOs of the FPGA board and connected to the spi master pins of Arduino. By connecting the PC with the Arduino as a serial interface, the status of the door lock system can be checked on the PC. In addition, a logic analyzer was also used to check signals of modules implemented in the FPGA, including SPI signals. The door lock system has been verified for normal operation by repeating the password setting mode and verification mode.

## IV. CONCLUSIONS

In this paper, we implemented a fast AES hardware security module for Internet of Things applications. After designing and analyzing the AES algorithm in C language, we reflected the result and designed it in hardware using Verilog-HDL. After functional verification in ModelSim, it was implemented using Altera CycloneIV EP4CE115F29C7.

In order to improve the processing speed of the AES hardware module, it was designed by applying parallel processing techniques and pipeline techniques to reflect the computational complexity and processing order of the algorithm. The proposed AES hardware module has a processing speed of 2.68 cycles/byte. The total logic elements, the total registers, and the clock rate are 11310l, 2164, and 154 Mhz, respectively, on the EP4CE115F29C7. Thus, while reducing the size to about 90 % compared to the previous paper, the processing speed is 1.4 times faster and the latency is reduced to a sixth.

In addition, we confirmed the applicability to the Internet of Things by implementing the door lock system, including the designed AES module and an Arduino. The designed AES module is implemented as separate hardware and is small in size and fast, so it can be used to protect the information in various embedded systems, including the Internet of Things.

## REFERENCES

[1] M. Chui, M. Löffler, and R. Roberts. "The internet of things," *McKinsey Quarterly*, March 2010.

[2] (2017) IEEE Courses. Computing. [Online]. What Is the Internet of Things: An Introduction. Available: https://ieeexplore.ieee.org/courses/ details/EDP486

[3] (2017) IEEE Courses. [Online]. Computing. Exploring IoT Industry Applications: The Evolution of Internet of Things for Healthcare. Available: https://ieeexplore.ieee.org/courses/details/EDP483

[4] D. Yang, and H. Ren, “The research on the technology of Internet of Things and embedded system,” in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)* pp. 395-398. Nov. 1999.

[5] M. Hossain, and S.M. RiazulIslam, F. Ali, K.S. Kwak, and R. Hasan, “An Internet of Things-based health prescription assistant and its security system design,” *Future Generation Computer Systems*, vol. 20, pp. 422-439, May. 2018.

[6] C. Z. E. Li, and Z. W. Deng, “The Embedded Modules Solution of Household Internet of Things System and The Future Development,” *Procedia Computer Science*, vol. 166, pp350-356, 2020.

[7] M. Ahmad, and K. K. Salahb, “IoT security: Review, blockchain solutions, and open challenges,” *Future Generation Computer Systems*, vol. 82, pp395-411, May. 2018.

[8] N. Moustafa, B. Turnbull, and K. K. R. Choo, “An Ensemble Intrusion Detection Technique Based on Proposed Statistical Flow Features for Protecting Network Traffic of Internet of Things,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp 4815-4830, June. 2019.

[9] E. Bertino, and N. Islam, “Botnets and Internet of Things Security,” *Computer*, vol. 50, no.2, pp 76-79, Feb. 2017.

[10] D.S. Kim. (2017) Special Economy. [Online]. Available: http://www. speconomy.com/news/articleView.html?idxno=86731

[11] CISOMAG. (2019) CISO MAG. [Online]. Available: cisomag.com /hackers-take-over-smart-home/

[12] P. Marius, C. TOMA, C. Boja, and Z. Alin, “Privacy and Security in Connected Vehicles Ecosystems,” *Informatica Economica*, vol. 21, no.4, pp. 29-40, 2017.

[13] C. Wootton, “Samsung ARTIK Reference: The Definitive Developers Guide,” Apress, 2016.

[14] H. Naeem, F. Ullah, M. R. Naeem, S. Khalid, D. Vasan, S. Jabbar, and S. Saeed, “Malware detection in industrial internet of things based on hybrid image visualization and deep learning model,” *Ad Hoc Networks*, vol. 105, pp. 102154, Aug. 2020.

[15] Y. H. Liu, S. Zhang, “Information security and storage of Internet of Things based on block chains,” *Future Generation Computer Systems*, vol 106, pp 296-303, May 2020.

[16] G. D. L. T. Parra, P. Rad, K. K. R. Choo, and N. Beebe, “Detecting Internet of Things attacks using distributed deep learning,” Journal of Network and Computer Applications, vol. 163, pp 102662, Aug. 2020.

[17] M. Rouse. (2018) IoT security(internet of things security), the IoT Agenda website. [Online]. Available: https://internetofthingsagen da.techtarget.com/definition/IoT-security-Internet-of-Things-security.

[18] M. Hossain, M. Fotouhi, and R. Hasan, “Towards an analysis of security issues, challenges, and open problems in the internet of things,” *in IEEE 11th World Congress on Services*, 2015.

[19] S. K. Rao, and D. Mahto, and D. A. Khan, “A Survey on Advanced Encryption Standard,” *International Journal of Science and Research (IJSR)*, vol. 6, no. 1, Jan. 2017.

[20] J. Daemen and V. Rijmen, “AES proposal: Rijndael,” 1999.

[21] M. F. Mushtaq, S. Jamel, A. H. Disina, Z. A. Pindar, N. S. A. Shakir, and M. M. Deris, “A Survey on the Cryptographic Encryption Algorithms,” *International Journal of Advanced Computer Science and Applications(IJACSA)*, vol. 8, no. 11, pp 333-344, 2017.

[22] (2020) The Wikipedia website. [Online]. Available: https://en.wikiped ia.org/wiki/Data_Encryption_Standard

[23] N. ALEISA, “A Comparison of the 3DES and AES Encryption Standards,” *International Journal of Security and Its Applications*, vol.8, no. 7, pp. 241-246, 2015.

[24] M. Bahnasawi, A., K. Ibrahim, A. Mohamed, M. Khalifa, A. Moustafa, K.Abelmonim, Y. ismail, and H. Mostafa, “ASIC-Oriented Comparative Review of Hardware Security Algorithms for the Internet of Things Applications,” in *IEEE International Conference on Microelectronics (ICM 2016)*, pp. 285-288, 2016.

[25] R. Nivedhaa, and J.Jean Justus, “A Secure Erasure Cloud Storage System Using Advanced Encryption Standard Algorithm and Proxy Re-Encryption,” in *2018 International Conference on Communication and Signal Processing (ICCSP)*, 2018.

[26] M. E. Hameed, M. M. Ibrahim, and N. A. Manap, “Review on Improvement of Advanced Encryption Standard (AES) Algorithm based on Time Execution, Differential Cryptanalysis and Level of Security,” *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 10, no. 1., pp. 139-145, 2018.

[27] S. P. Singh, and R. Maini. "Comparison of data encryption algorithms." *International Journal of Computer Science and Communication,* vol.2, no. 1, pp. 125-127, 2011.

[28] D. Budiyanto, and P. A. W Putro, “Comparison of Implementation Tiny Encryption Algorithm (TEA) and Advanced Encryption Standard (AES) Algorithm on Android Based Open Source Cryptomator Library,” in *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2018.

[29] D. Nurmalasari, E. Mulyana, and M. Irfan, “Security Implementation of the Internet of Things Using the Advanced Encryption Standard (AES) Algorithm,” in *2019 IEEE 5th International Conference on Wireless and Telematics (ICWT)*, 2019.

[30] R. Riyaldhi, Rojali, and A. Kurniawan, “Improvement of Advanced Encryption Standard Algorithm With Shift Row and S.Box Modification Mapping in Mix Column,” *Procedia Computer Science*, vol. 116, pp. 401-407, 2017.

[31] D. J. Bernstein and P. Schwabe, “New AES software speed records,” in International Conference on Cryptology in India, Berlin, Heidelberg: Springer, 2008.

[32] H. K. Park and K. Lee, “A Design of an AES-based Security Chip for IoT Applications using Verilog HDL,” *The Transactions of the Korean Institute of Electrical Engineers*, no. 1, pp. 9-14, 2018.