IEIE Transactions on Smart Processing and Computing, vol. 6, no. 2, April 2017
https://doi.org/10.5573/IEIESPC.2017.6.2.133

Efficient FPGA Implementation of AES-CCM for IEEE 1609.2 Vehicle Communications Security

Chanbok Jeong[1] and Youngmin Kim[2]

[1] IVI Business Division, LG Electronics chanbok.jeong@lge.com
[2] Computer and Information Engineering, Kwangwoon University, Seoul, Korea youngmin@kw.ac.kr

***** Corresponding Author: Youngmin Kim

Received January 9, 2017; Accepted January 27, 2017; Published April 30, 2017

** Short Paper*

***Abstract*:** Vehicles have increasingly evolved and become intelligent with convergence of information and communications technologies (ICT). Vehicle communications (VC) has become one of the major necessities for intelligent vehicles. However, VC suffers from serious security problems that hinder its commercialization. Hence, the IEEE 1609 Wireless Access Vehicular Environment (WAVE) protocol defines a security service for VC. This service includes Advanced Encryption Standard–Counter with CBC-MAC (AES-CCM) for data encryption in VC. A high-speed AES-CCM crypto module is necessary, because VC requires a fast communication rate between vehicles. In this study, we propose and implement an efficient AES-CCM hardware architecture for high-speed VC. First, we propose a 32-bit substitution table (S_Box) to reduce the AES module latency. Second, we employ key box register files to save key expansion results. Third, we save the input and processed data to internal register files for secure encryption and to secure data from external attacks. Finally, we design a parallel architecture for both cipher block chaining message authentication code (CBC-MAC) and the counter module in AES-CCM to improve performance. For implementation of the field programmable gate array (FPGA) hardware, we use a Xilinx Virtex-5 FPGA chip. The entire operation of the AES-CCM module is validated by timing simulations in Xilinx ISE at a speed of 166.2 MHz.

***Keywords*:** AES-CCM, AES, V2X security, IEEE 1609.2, WAVE, Vehicle communications, FPGA

1. Introduction

Nowadays, many companies and researchers have been investigating autonomous vehicles, electric vehicles, and connected cars to develop the intelligent traffic system (ITS). Vehicle communications (VC) or networking technology in general is an essential element for the success of these systems. The IEEE developed and proposed the Wireless Access Vehicular Environment (WAVE) protocol [1, 2], which includes the IEEE 1609.2 standard for security services to provide reliable VC. A security service is a critical part of VC, because VC is strongly related to the safety of passengers and cars. For example, researchers recently demonstrated that they could control a vehicle (e.g., the engine, brakes, and instrument panel) through the controller area network (CAN) bus [3, 4]. If an attacker uses these kinds of techniques, and the VC network becomes vulnerable, there will be serious vehicle safety problems.

Many related works go into the specifics of the security problem in vehicle communications. For example, malicious cars can broadcast fake positions through VC [5-7] in a classic example of "anti-social" behavior. These malicious cars must be detected and isolated to ensure the integrity of position information in VC. In other words, the reliability of the position information for cars must be protected in VC, but this adds significant overhead to the system. VC has a unique characteristic that requires rigorous real-time constraints. Thus, security services in VC should also meet timing constraints.

Algredo-Badillo et al. [8] designed the Advanced Encryption Standard–Counter with Cipher Block Chaining-Message Authentication Code Protocol (AES-CCMP) and implemented it in field programmable gate

array (FPGA) hardware for IEEE 802.11i-2004. They carefully analyzed the Advanced Encryption Standard–Counter with CBC-MAC (AES-CCM) architecture to exploit the parallelization of some processes; this design is highly specialized in processing modules. They intended to design a fast and simple iterative AES-CCMP hardware architecture with low hardware requirements. The implemented Xilinx Virtex-4 FPGA consists of 1921 slices and 20 block memories and has 1.876 Gbps data throughput at 149 MHz. They also compared the hardware and software implementations in their paper. The software implementation had a higher frequency and throughput, but resulted in lower efficiency (e.g., data length per clock frequency) than the hardware implementation.

The properties of Car-to-X communication (C2X) were investigated [9, 10]. C2X, in general, requires extremely low latency. In particular, at high speeds, when passive and active safety events occur, it requires millisecond latency. Thus, an additional security service is a serious overhead in C2X. Furthermore, cryptography plays a major role in securing VC because a C2X security service must provide an authentication and encryption method to protect short message data from many other vehicle objects. In the worst case scenario, the number of neighborhood cars can reach 200 with a beaconing rate of 1–10 Hz. To protect data, each vehicle needs to have between 400 and 4000 verifications or decryptions per second to exchange data. A parallel cryptography architecture in FPGA hardware and a hybrid system using the hardware and software co-design for a C2X security service was proposed.

Data throughput and minimum delay limits of the 802.11p VC protocol were analyzed [11]. The minimum delay in VC for a 27 Mbps data rate with 1000 bytes of payload data is 565.5 μs. Therefore, overall encryption processing must be completed no later than the minimum delay requirement.

In this paper, we design an efficient AES-CCM architecture by using Verilog Hardware Description Language (Verilog-HDL) and implement it in FPGA hardware. AES-CCM inherently has a significant influence on the workload in VC because of the repeated operations of the data encryption modules. To resolve this issue, we use a hardware implementation method for the AES-CCM module by proposing several design techniques in the FPGA to increase data processing throughput and reduce latency. For an efficient and effective hardware implementation, our main contributions can be summarized as follows.

- We propose a 32-bit parallel operations unit in the substitution box (S_Box), which is originally an eight-bit unit in the standard to improve the overall latency of the AES module.
- We employ key box registers to save key expansion results because key expansion operations are not always required for every AES iteration.
- We use internal register files bearing high usage of FPGA slices to store the input *Plaintext* and the output *Ciphertext* to protect the data from external attack.

Table 1. Key and Input Data Sizes of AES.

Input Key Size	4 Words / 128 Bits	6 Words / 192 Bits	8 Words / 128 Bits
Expanded Key Size	44 Words / 1408 Bits	52 Words / 1664 Bits	60 Words / 1920 Bits

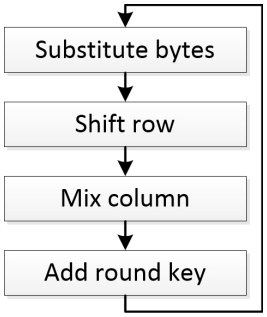

Fig. 1. General AES round structure in AES standard.

- The CBC-MAC module and the AES counter in AES-CCM are implemented in a parallel architecture to improve throughput.

The rest of this paper is organized as follows. Section 2 explains the AES encryption algorithm and standard. Section 3 explains the methodology in AES-CCM implementations. Section 4 presents the FPGA designs for the proposed AES and AES-CCM modules. Simulation results are discussed in Section 5, followed by conclusions in Section 6.

2. Advanced Encryption Standard

AES is the National Institute of Standards and Technology (NIST) encryption standard substituting for the Data Encryption Standard (DES) algorithm [12, 13]. DES has been known to have performance limitations and security issues when used for VC in an ITS [12, 13].

To develop AES, NIST set the following standard criteria: minimum system resource usage, open source algorithm, capable of hardware and software implementation, robust to any security attack, low complexity for encryption calculation, and can be implemented in any system environment [12, 13]. As a result, AES has variable key lengths of 128, 192, and 256 bits and a 128-bit wide input block. The inserted input key is expanded as shown in Table 1.

Fig. 1 presents the general round structure of AES operation. The substitute bytes (SubByte) mode performs a byte-to-byte substitution of input data or previously encrypted data using a substitution table (S_Box). The shift row operation (ShiftRow) shifts the row-wise byte data according to the row index. The mix column process is for permutation of the column-wise bit data using a Galois field, GF(2^8). Mix column is similar to ShiftRow, but ShiftRow is a row-wise byte permutation, and mix column is a column-wise bit permutation. The Galois field is a finite field for an AES mix column operation. Add

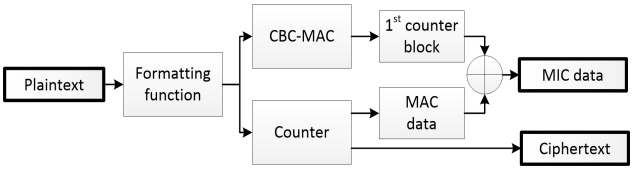

Fig. 2. AES-CCM structure.

round key operates bit-wise XOR for the mixed column results with the expanded key from the key expansion process. Not all the processes shown in Fig. 1 are applied to the initial and final rounds. The add round key process only operates in the initial round, and the ShiftRow, SubByte, and add round key processes are required in the final round [12, 13]. AES is the critical part of AES-CCM performance because it is an iterative process in AES-CCM. Therefore, careful analysis for an optimized AES implementation is extremely important.

3. AES-CCM

AES-CCM consists of the counter module and the cipher block chaining message authentication code (CBC-MAC) using AES as a NIST SP 800-38C encryption standard. The WAVE protocol uses this AES-CCM for data encryption. AES-CCM is a unique symmetric key encryption algorithm for the WAVE 1609.2 security standard; thus, all payload data must be encrypted by AES-CCM to be used under the WAVE protocol [2].

Fig. 2 shows the entire structure of the AES-CCM operation. As shown in the figure, first, the *Plaintext* is entered and formatted on the basis of the AES-CCM standard in the formatting function module to make a 128-bit data pattern. Then, it is processed by the CBC-MAC and the counter block in parallel. Finally, the encrypted *Ciphertext* is generated from the counter block, and the message integrity code (MIC) data are obtained from the XOR operations between the CBC-MAC results (MAC data) and the counter block (1st counter block) results. CBC-MAC validates the authenticity of data by using the cipher block chaining method, which chains all input data; the chaining data are truncated from MAC data for data authenticity.

The counter ensures the confidentiality of data by using the encryption process of the sequential counter blocks. The encrypted counter block is combined with nonce data and the counter index. The CBC-MAC and the counter use the same encryption key for AES-CCM data encryption [14].

AES-CCM is used in not only the IEEE 1609.2 WAVE security protocol but also in IEEE 802.11 and IEEE 802.15.4. Although other data encryption algorithms have a number of advantages, AES-CCM is already used in various data communications protocols by IEEE working groups and has already been proposed as an encryption standard by NIST. Therefore, it is accepted as the symmetric key cryptography algorithm of IEEE 1609.2 [2].

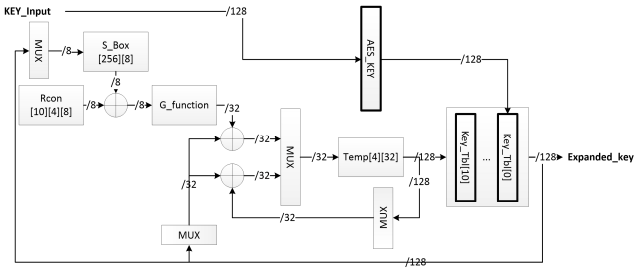

Fig. 3. Proposed key expansion structure.

4. FPGA Implementation

In this section, we introduce the hardware implementation of the proposed AES encryption, which is a critical and important module for AES-CCM, followed by the complete AES-CCM architecture design using the proposed AES module.

4.1 Hardware Design for AES

An optimized AES design is crucial to the performance of the entire AES-CCM, because CBC-MAC and the counter in AES-CCM are processed by the AES algorithm as explained in the previous sections.

AES-CCM does not require a separate AES decryption process for the data decryption–verification process. The generation–encryption process and the decryption–verification process in AES-CCM can be performed by using only the AES encryption process. Therefore, we focus on efficient design of the AES encryption structure. In addition, we use the 128-bit key length for the AES algorithm, because AES-CCM for IEEE 1609.2 only requires 128-bit key length operations.

Fig. 3 shows the proposed key expansion architecture for AES. The key value (KEY_Input) is entered into the encrypted key tables (Key_Tbl) through registers (AES_KEY), and the expanded key is generated on the basis of the index from the controller through the key expansion operations. As mentioned, it is a 128-bit key length architecture, and the expanded key data are saved into the specific key table registers, Key_Tbl. The reason for saving the expanded key data is that the expanded key data remain unchanged until the input key data are modified. We can remove additional clocks for the key expansion processes by excluding the initial key expansion process using the proposed key-saving method.

The key expansion process should be performed with a 32-bit unit. However, the first column of each round is processed with only an eight-bit unit because of the SubByte operation. Therefore, the processing of the first column requires four clock cycles. To resolve this problem and improve performance, four–S_Box parallelism in the hardware architecture with an eight-bit unit that requires only one clock cycle can be considered. However, this approach is not efficient, because it consumes significant hardware resources in order to save the time consumed by only 30 clock cycles.

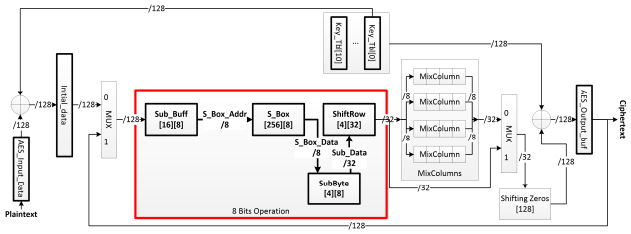

Fig. 4. Standard AES structure and 8 bits operation.

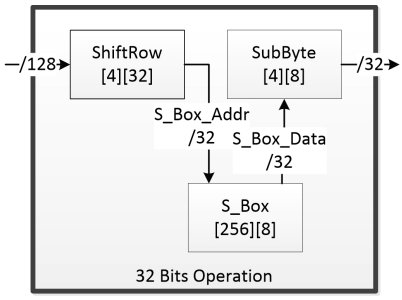

Fig. 5. Proposed simplified 32-bit operation structure for AES.

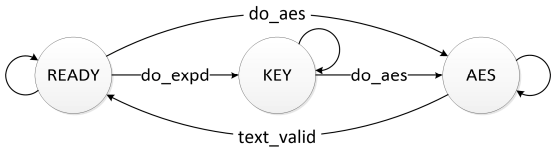

Fig. 6. Finite state machine for AES operation.

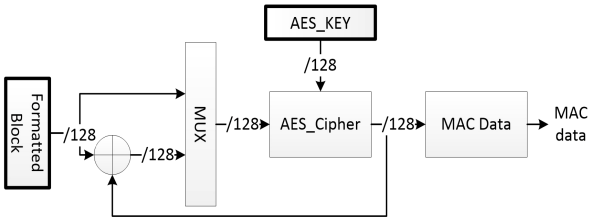

Fig. 7. Proposed CBC-MAC architecture used in AES-CCM.

Fig. 4 shows the standard architecture of the AES encryption process. The formatted data, AES_Input_Data, are entered and the temporal AES data are saved into buffers through AES operations for the next round. The initial round performs a simple bit-wise XOR operation with 0^{th} expanded key data in the key tables (Key_Tbl) and AES_Input_Data. The final valid data are ready after 11 rounds.

The eight-bit operation modules of SubByte and ShiftRow in Fig. 4 (red box) are replaced with the proposed 32-bit structures shown in Fig. 5. The implemented AES encryption operates in the following order with the proposed 32-bit architecture: ShiftRow, SubByte, mix column, and add round key. As shown in Figs. 4 and 5, the processing order is not the same as that for the AES encryption standard shown in Fig. 1. The reason for the order change is to re-use these modules in the decryption process. In fact, our AES encryption design considers the decryption process, as well. In other words, the AES encryption module can be used in the AES decryption process without any design modification. Therefore, to use it in the decryption process, an inversion of the sequence of the encryption round is required. Hence, our AES design has the reversed SubByte and ShiftRow operations in the encryption process. However, the order between ShiftRow and SubByte does not affect the encryption result. This order inversion is called an *equivalent inverse cipher* [12, 13].

In this paper, because the 32-bit S_Box can perform an eight-bit SubByte function, we use a single S_Box for the SubByte operation of both the eight-bit substitution in key expansion and the 32-bit substitution in AES. The reason for the 32-bit implementation is to remove the delay penalty of SubByte in the eight-bit operation. In the eight-bit SubByte operation, 223 clock cycles are consumed to encrypt 128-bit plaintext. However, only 73 clock cycles are required to encrypt the same plaintext with the 32-bit SubByte. That means the 32-bit implementation is three times faster than the eight-bit implementation. Therefore, the 32-bit data path is a suitable structure for high-performance AES hardware implementation. The path in the red box of Fig. 4 is the critical path of the standard AES structure. As explained above, this path is replaced by the 32-bit data path in the proposed architecture, as shown in Fig. 5. To process the SubByte operation, 16-byte registers are required, and they yield additional 16 clock delays, because the SubByte operation is a byte-wise operation and ShiftRow is a row-wise operation. Therefore, the ShiftRow operation waits until the SubByte operation is completed. However, the delay penalty in the standard architecture can be reduced by changing the order of SubByte, ShiftRow, and the 32-bit SubByte operation, which follows the order of ShiftRow in the proposed architecture. In this paper, the S_Box in Figs. 3 and 5 is the same module, because one S_Box can be shared by AES encryption and the key expansion.

The finite state machine (FSM) of the AES operation is shown in Fig. 6. During the **READY** state, if the **do_expd** control signal becomes true, then the state is changed to the **KEY** state, and the key expansion operation is performed. If the **do_expd** signal is false and the **do_aes** control signal is true, then the key expansion is skipped, and AES is performed directly. Whenever the key expansion is finished, the **do_aes** signal will be set to true, and the execution of AES will begin. When **AES** is completed, the **text_valid** signal is set to true, and the state will return to **READY** for the next encryption. The key expansion requires an eight-bit substitution operation. In addition, key expansion and **AES** spend nine clock cycles and seven clock cycles, respectively, for each round. Therefore, **KEY** and **AES** must be separated in our design.

4.2 Hardware Design for AES-CCM

As explained in the previous sections, for the AES-CCM implementation, both the CBC-MAC module and the counter module are required.

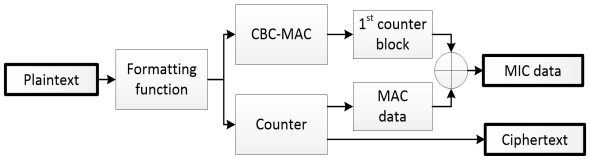

Fig. 8. Proposed counter architecture used in the AES-CCM.

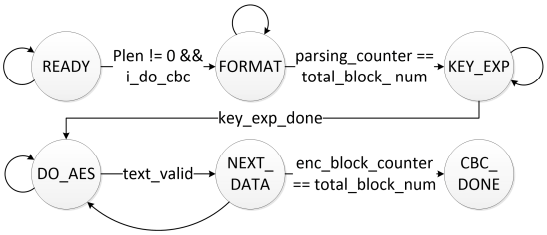

Fig. 9. Finite state machine for the proposed AES-CCM.

Fig. 7 shows the structure of the CBC-MAC module. CBC-MAC performs the AES encryption (i.e., AES_Cipher) using chaining block data, which is the result of an XOR operation with the encrypted data and the current formatted block data. However, the first chaining block does not require an XOR operation, because it does not contain encrypted data; it only has the current formatted data. Therefore, AES is performed directly for the first chaining data. The CBC-MAC process is the most critical function block in AES-CCM, because it requires as many iterations as nonce data, associated data, and payload data.

Fig. 8 presents the architecture of the counter module. Different from the CBC-MAC module, the counter module performs AES encryption with 128-bit counter block data (counter block). The counter block is made up of nonce data and counter values. An XOR operation is performed between the encrypted counter block from AES_Cipher and the formatted payload data (formatted block) to obtain the final encrypted data, *Ciphertext*. The first counter block is XOR-ed with the CBC-MAC result (MAC data) to generate the MIC data.

Fig. 9 shows the finite state machine of AES-CCM. During the **READY** state, if the payload length (**Plen**) is not equal to 0, and the **i_do_cbc** control signal is true, then the state transits to **FORMAT** and performs the formatting function. When the number of generated formatting block counters (**parsing_counter**) equals the number of total block counters (**total_block_num**), then the state transits to the **KEY_EXP** state. Furthermore, the **KEY_EXP** state expands the input key based on the key length, and the state moves to the **DO_AES** state when the key expansion is completed. The **DO_AES** state encrypts the formatted block data using the AES algorithm and transits to the **NEXT_DATA** state when the text becomes valid (**text_valid**). The **DO_AES** state can transit to the **CBC_DONE** state whenever the amount of encrypted data (**enc_block_counter**) and the total number of blocks (**total_block_num**) are the same. Otherwise, the **NEXT_DATA** state returns to the **DO_AES** state and performs AES encryption until the **CBC_DONE** condition is satisfied.

Table 3. Register File Usage.

Register File Name	Width (bit)	Depth
Input_data	128	16.7
Input_register	8	256
Parser_memory	128	256
Key_box	128	22
Ctr_memory	128	256

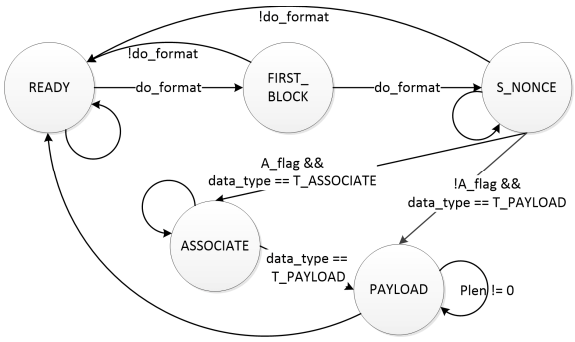

Fig. 10. Finite state machine for the formatting function.

Fig. 10 shows the finite state machine of the formatting function used in AES-CCM. This operation is part of the **FORMAT** state in the AES-CCM finite state machine, as shown in Fig. 9. During the **READY** state, when the **do_format** control signal becomes true, the state transits to the **FIRST_BLOCK** and generates a flag octet of the first block of the formatting and moves to the **S_NONCE** state. Otherwise, the state returns to the **READY** state. In the **S_NONCE** state, the first block is generated completely by using the flag octet, nonce data, and payload length. When the **S_NONCE** state is executed, if the associated data flag (**A_flag**) control signal is true and the **data_type** is **T_ASSOCIATE**, then the state transits to the **ASSOCIATE** state. The **ASSOCIATE** state performs the formatting function using the associated data, and the state changes to the **PAYLOAD** state if the **data_type** is **T_PAYLOAD**. The **PAYLOAD** state performs the payload data formatting.

If **A_flag** is false when the **S_NONCE** state is executed, the **PAYLOAD** state starts directly and generates the formatted payload data. In the **PAYLOAD** state, the system generates the formatted data until the payload data length (**Plen**) equals 0. When **Plen** equals 0, the state returns to the **READY** state and waits until the next data become available. Furthermore, the **FIRST_BLOCK** state and the **S_NONCE** state return to the **READY** state directly when the **do_format** control signal becomes false.

5. Result

In this study, we used a Xilinx Virtex-5 FPGA chip and an ISE 14.5 synthesis tool [15, 16] to implement the

Table 2. FPGA implementation result and comparison with previous works.

Device	FPGA Slice	Clock Frequency (MHz)
Spartan-3 [17]	523	63.7
Virtex-2 [18]	3474	80.3
Virtex-2 [8]	1609	117.88
Virtex-4-LX [8]	1921	149
Virtex-5-LX [19]	1809	213
Virtex-5-SX [this work]	11,913	166.20

proposed AES-CCM system. The FPGA hardware implementation is summarized and compared with other designs in Table 2. As shown in the table, the number of slice registers is 11,913 with a 166.2 MHz system clock. The number of look-up tables (LUT) and flip flops (F/F) used in the synthesized design is 24,062 and 31,461, respectively.

Furthermore, the register file usage and size are summarized in Table 3. As shown in Tables 2 and 3, the proposed design uses more register slices than previous works. The reason for the higher usage of FPGA resource and register files is that both the input data and the processed data are saved in the register files in our design. Therefore, the system occupies a number of registers and LUTs. Even with such a high usage of resources, we can achieve fast operating speed by using the proposed parallel architectures and pipelining approaches in the long paths. In addition to the higher speed, such an FPGA implementation method (in which the processed data are saved inside the system) is safer from the perspective of protecting the encrypted data from external attacks than by saving data outside the system.

As mentioned, the entire AES-CCM process must be finished in 565.5 μs to send VC messages without congestion. Our proposed AES-CCM hardware results in a delay of 132.4 ns for 1-byte payload data (i.e., it requires only 133 μs to encrypt 1000 bytes of payload data). Therefore, the operation of our AES-CCM implementation can satisfy the delay constraint of VC, and it has very low encryption overhead.

Fig. 11 shows the register-transfer level (RTL) synthesis result of the proposed AES-CCM implementation in FPGA. The two AES modules are placed parallel to CBC-MAC and the counter module, and the FSM for AES-CCM is placed inside the CCM controller to send control signals for the ASE-CCM operations. The remaining blocks are interface modules for the input and output data.

6. Conclusion

In this study, we have implemented an AES-CCM FPGA hardware encryption module that satisfies the IEEE 1609.2 VC security service. VC inherently has a considerable communications load, because a vehicle communicates with many vehicle objects simultaneously.

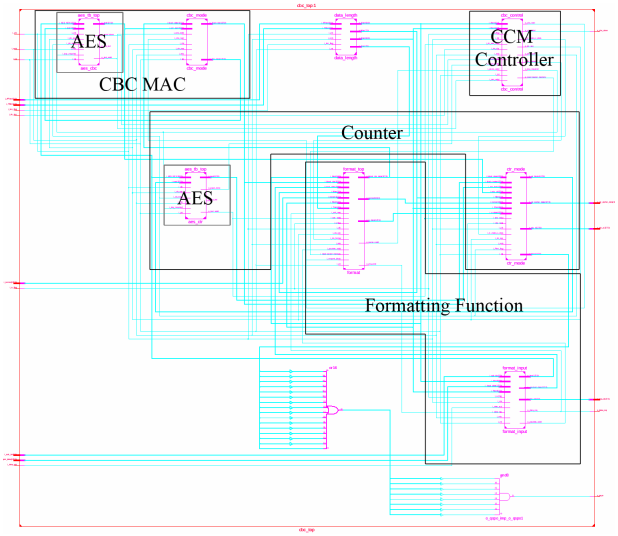

Fig. 11. RTL synthesis result of the proposed AES-CCM.

Therefore, in addition to a reliable security service, an efficient design of the AES-CCM module, which is an essential security component in VC, is important. In this paper, several design methodologies are proposed to improve the efficiency of the AES modules and AES-CCM. These designs are implemented using a Xilinx Virtex-5 chip; the functional operations are confirmed via post-map timing simulations; and the real hardware signals are verified using Chip-scope [16]. As a result, the implemented hardware uses 11,913 register slices and operates with a 166.2 MHz system clock. We use internal register files to save the input data and the processed data to protect them from possible external attacks.

Acknowledgement

This present research has been conducted by the Research Grant of Kwangwoon University in 2016.

Reference

[1] R. A. Uzcátegui and G. Acosta-Marum, “WAVE: A Tutorial,” IEEE Communications Magazine, May 2009, pp. 126-133.

[2] IEEE Std. 1609.2-2013. (April, 2013.) IEEE Standard for Wireless Access in Vehicular Environments Security Services for Applications and Management Messages. Accessed Aug. 2014. Article (CrossRef Link)

[3] T. Zhang and L. Delgrossi, Vehicle Safety Communications: Protocols, Security, and Privacy, Hoboken, New Jersey: John Wiley Sons, Inc., 2012.

[4] P. Papadimitratos, “Secure Vehicular Communication Systems: Design and Architecture,” IEEE Communication Magazine, vol. 46, no. 11, 2008, pp. 100-109.

[5] G. Yan, S. Olariu, and M. C. Weigle, “Providing VANET Security Through Active Position Detec-

tion," Computer Communications, vol. 31, no. 12, July 2008, pp. 2883-2897.
[6] N. Wang and Y. Huang, "A Novel Secure Communication Scheme in Vehicular Ad Hoc Networks," Computer Communications, vol. 31, no. 12, July 208, pp. 2827-2837.
[7] M. Faezipour, M. Nourani, A. Saeed, and S. Addepalli, "Progress and Challenges in Intelligent Vehicle Area Networks," Communications of the ACM, vol. 55, no. 2, Feb. 2012, pp. 90-1000.
[8] I. Algredo-Badillo et al, "Efficient Hardware Architecture for the AES-CCM Protocol of the IEEE 802.11i Standard", Computers & Electrical Engineering, 2010, pp. 565-577.
[9] T. Schütze, "Automotive Security: Cryptography for Car2X Communication," tech. rep., Rodhe & Schwarz, Germany, Mar. 2011, pp. 1-16.
[10] E. Schoch and F. Kargl, "On the Efficiency of Secure Beaconing in VANETs," The 3rd ACM Conference on Wireless Network Security, Mar. 2010, pp. 111-116.
[11] Y. Wang et al., "Throughput and Delay Limits of 802.11p and its Influence on Highway Capacity," Procedia - Social and Behavioral Sciences, vol. 96, Nov. 2013, pp. 2096-2104.
[12] W. Stallings, Cryptography and Network Security Principles and Practices, 4th ed. Prentice Hall, 2005.
[13] NIST FIPS-197. Advanced Encryption Standard. Accessed Aug. 2014. Article (CrossRef Link)
[14] NIST SP 800-38C. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. Accessed Aug. 2014. Article (CrossRef Link)
[15] Xilinx. XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices. Accessed Aug. 2014. Article (CrossRef Link)
[16] Xilinx. ChipScope Pro Software and Cores User Guide. Accessed Aug. 2014. Article (CrossRef Link)
[17] A. Aziz, A. Samiah, and N. Ikram, "A Secure Framework for Robust Secure Wireless Network (RSN) using AES-CCMP," The fourth International Bhurban Conference on Applied Sciences and Technology, 2005.
[18] N. Smyth, M. McLoone, and J. V. McCanny, "WLAN Security Processor," IEEE Trans. on Circ. and Syst.-I, 2006, pp. 1506-1520.
[19] H. Rha and H. Choi, "Efficient Pipelined Multistream AES CCMP Architecture for Wireless LAN," International Conference on Information Science and Applications (ICISA), May 2012, pp. 1-5.

Chanbok Jeong received a BS from the Computer Engineering at Gyeongsang National University in Feb. 2013, and received an MS in electrical and computer engineering from the Ulsan National Institute of Science and Technology (UNIST), Ulsan, Korea, in 2015. He is now with the LG Electronics.

Youngmin Kim received a BS in electrical engineering from Yonsei University, Seoul, Korea, in 1999, and an MS and a PhD in electrical engineering from the University of Michigan, Ann Arbor, in 2003 and 2007, respectively. He held a senior engineer position at Qualcomm in San Diego, CA. He is currently an Associate Professor at Kwangwoon University, Seoul, South Korea. Prior to joining Kwangwoon University, he was with the school of electrical engineering and computer engineering at the Ulsan National Institute of Science and Technology (UNIST), Ulsan, South Korea. His research interests include variability-aware design methodologies, design for manufacturability, design and technology co-optimization methodologies, and low-power and 3D IC designs.