

# The Design of a High-Throughput Hardware Architecture for the AES-GCM Algorithm

Ming-Bo Lin<sup>1</sup>, Senior Member, IEEE, and Jen-Hua Chuang

**Abstract**—This paper proposes the hardware architecture capable of processing the AES-GCM algorithm in parallel and pipeline. To boost the hardware efficiency, two parallel AES processing units, each consisting of a mixed inner- and outer-round pipelining architecture, are employed in the encryption/decryption module. These two AES units share a key expansion unit to save the hardware resource. To optimize the architecture of the AES algorithm, the S-box function is directly realized with the built-in BRAM in the FPGA technique to decrease the number of LUTs used, and implemented with the composite field operation in the ASIC technique in order to save up to 25% logic gates. In the GHASH module design, two pipelining finite field  $GF(2^{128})$  multipliers with appropriate data scheduling are utilized. In addition, realizing the finite field multiplication with the Karatsuba algorithm can further save 50.7% of hardware resource. The resulting AES-GCM IP has been implemented and verified with a Xilinx Virtex 5 series device (XC5VLX220), and a tsmc 0.18- $\mu\text{m}$  cell library. In FPGA verification, the working frequency can achieve 223.314 MHz, yielding the maximum throughputs of 57.17 Gbps. The number of slices/BRAMs used is 5,944/109. The resulting ASIC can operate at 197.239 MHz in simulation and achieve a high throughput of 50.49 Gbps. The core area of the chip is 6,552  $\mu\text{m} \times 6,516 \mu\text{m}$ . The power consumption is 1,466.5 mW.

**Index Terms**—AES-GCM, AES, ASIC, authentication encryption, FPGA, GHASH, parallel, pipeline.

## I. INTRODUCTION

WITH the rapid development of network communication, the use of network to transfer messages has been become an indispensable tool for our daily life. To guarantee the confidentiality and intactness of the message and to authenticate the sender, the message encryption and authentication method has become a crucial issue in networking technique. To facilitate this, the AES-GCM algorithm [17] has been proposed and becomes one of such kinds of standards.

The AES-GCM algorithm is a symmetric message authentication and encryption algorithm. It was proposed in 2006 by the National Institute of Standard and Technology (NIST) as the SP 800-38D standard [16] for providing a method

of message authentication and encryption to ensure the confidentiality and integrity of message during transmission. Nowadays, the AES-GCM algorithm has been popular in many applications, such as MACsec, TLS 1.2, WiGig, SSH, IPsec [17], and Internet of Things (IoT) [20]. The AES algorithm [3] used by the AES-GCM algorithm is proposed by NIST as FIPS Publishing 197 in 2001 to replace the Data Encryption Standard (DES). The AES algorithm is a symmetric encryption/decryption technique, suitable for the encryption/decryption of mass data.

Since the design requirement is not the same for different applications, many different implementations of the AES-GCM algorithm are proposed and summarized as follows. The one proposed in [5] uses the tsmc 0.18- $\mu\text{m}$  process to design and implement the Media Access Control (MAC) layer security of the Ethernet Passive Optical Network (EPON), which includes an AES-GCM module. Nevertheless, it does not show the hardware resource used and the throughput. Another [21] also uses the 0.18- $\mu\text{m}$  process to realize the AES-GCM circuit. By way of balancing the critical paths associated with the AES circuit and the modular multiplier, its throughput can reach 34 Gbps. To achieve up to the order of 100 Gbps, the one proposed in [19] adopts four GHASH circuits operated in parallel. However, it cannot provide real-time operations because its AES-GCM circuit must be provided with the all-length message during the encryption and decryption. In addition, its throughput cannot be further promoted due to no pipelined design is applied. The design of a GHASH circuit using a single-pipeline  $GF(2^{128})$  multiplier with feedback is proposed in [2]. It can facilitate a throughput of 40 Gbps with a Xilinx Virtex 4 device.

In this paper, we will focus on the design of hardware architecture of the AES-GCM algorithm so as to speed up its operations. The philosophy will be centered around to explore the combination of advantages from pipelining and parallelism and the use of a fast multiplier to perform the required modular multiplication and to reduce hardware resource used so as to achieve a real-time, high throughput, low-cost circuit. The resulting architecture will be verified with both FPGA and ASIC techniques. To make good use of these two techniques, we will separately consider their individual features and adopt the most suitable way to optimize the design and implementation.

The rest of the paper is organized as follows. Section II introduces the AES-GCM algorithm. Section III discusses the design considerations of the proposed AES-GCM architecture. Section IV describes the proposed AES-GCM architecture.

Manuscript received 14 August 2023; revised 12 October 2023; accepted 10 November 2023. Date of publication 15 November 2023; date of current version 26 April 2024. (Corresponding author: Ming-Bo Lin.)

Ming-Bo Lin is with the Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology, Taipei City 106, Taiwan (e-mail: mblin@mail.ntust.edu.tw).

Jen-Hua Chuang is with the Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology, Taipei City 106, Taiwan, and also with Mediatek Inc., Hsinchu City 300, Taiwan (e-mail: M10902132@mail.ntust.edu.tw).

Digital Object Identifier 10.1109/TCE.2023.3332872

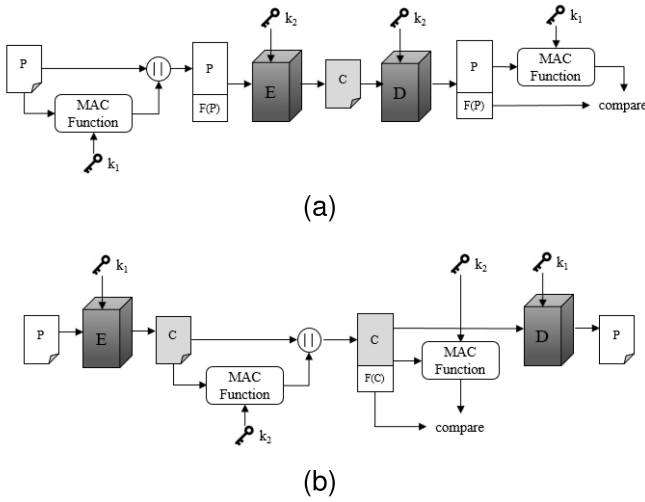


Fig. 1. The uses of MAC: (a) authentication tied to plaintext and (b) authentication tied to ciphertext.

Section V presents the experimental results. Section VI concludes this paper.

## II. AES-GCM ENCRYPTION/DECRYPTION ALGORITHM

The Galois/Counter Mode (GCM) is one of the operation modes of the AES algorithm. It belongs to the symmetric cryptology and facilitates the message encryption and authentication so as to ensure both the confidentiality and integrity [14], [15] of a message. Compared with the other operation modes of the AES algorithm, such as the Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB), the AES-GCM has better security strength, more robust against both the modification attack and the bit-flipping attack. In addition, the AES-GCM algorithm can be pipelined and parallelized in hardware implementation as well as can be worked with look-up tables and pipelining instructions in order to obtain good performance in software. For this reason, the AES-GCM algorithm has been broadly used in many standards, including IEEE 802.1 AE (MACsec) [8], where it is used to guarantee the security during the frame transmission on the media access control (MAC) layer.

Two general approaches for authentication encryption/decryption are authentication tied to plaintext and authentication tied to ciphertext, as illustrated in Figures 1(a) and (b), respectively. The method of authentication tied to plaintext is the one that the message authentication code (MAC) of the message is first generated through a MAC function and then the MAC along with the plaintext is encrypted and transmitted. At the receiver, the received message must be first decrypted and then the plaintext is authenticated to compare with the MAC. On the contrary, in the method of authentication tied to ciphertext, the message is first encrypted and then the MAC function is applied to the ciphertext to generate the MAC, which is then sent along with the ciphertext to the receiver. At the receiver, the decryption of the ciphertext and the authentication of the ciphertext can be proceeded in parallel. In the AES-GCM algorithm, the plaintext is encrypted by the AES-CTR mode and then the ciphertext is passed to the

GHASH function to yield the MAC. Thus, it is a kind of authentication tied to ciphertext.

The AES-GCM algorithm provides both message encryption/decryption and authentication at a time to ensure the confidentiality and integrity of a message during transmission. It utilizes the AES-CTR mode to encrypt/decrypt the concatenation of the plaintext and the additional authenticated data, and uses the GHASH hash function to yield the MAC [14], [16].

**AES-GCM encryption:** The inputs of the AES-GCM encryption algorithm [16] include the additional authenticated data ( $A$ ), plaintext ( $P$ ), initialization vector ( $IV$ ), and key ( $K$ ). The additional authenticated data ( $A$ ) has a length ranging from 0 to  $2^{64} - 1$  bytes. The plaintext length ranges from 0 to  $2^{39} - 256$  bytes. Padding 0 is carried out if the sum of the  $A$  and the plaintext ( $P$ ) is not a multiple of 128. The outputs contain the ciphertext ( $C$ ) and a tag (MAC). The ciphertext has the same length as the plaintext. The tag can have a length of 128, 112, 104 or 96 bits, determined by the actual requirement.

The AES-GCM encryption [16], [17] algorithm proceeds as the following steps:

- 1) Let  $H = \text{CIPH}_K(0^{128})$ . Here,  $\text{CIPH}_K$  means an encryption function (e.g., AES) with a key  $K$ .
- 2) If  $\text{len}(IV) = 96$ , then define  $J_0 = IV \parallel 0^{31} \parallel 1$ ; Here, “ $\parallel$ ” means the concatenation operator.  
If  $\text{len}(IV) \neq 96$ , let  $s = 128 \lceil \text{len}(IV)/128 \rceil - \text{len}(IV)$  and let  
 $J_0 = \text{GMASH}_H(IV \parallel 0^{s+64} \parallel [\text{len}(IV)]_{64})$ .
- 3) Let  $C = \text{GCTR}_K(\text{inc}_{32}(J_0), P)$ .
- 4) Let  $u = 128 \lceil \text{len}(C)/128 \rceil - \text{len}(C)$  and let  $v = 128 \lceil \text{len}(A)/128 \rceil - \text{len}(A)$ .
- 5) Define a block:  
 $S = \text{GHASH}_H(A \parallel 0^v \parallel C \parallel 0^u \parallel [\text{len}(A)]_{64} \parallel [\text{len}(C)]_{64})$ .
- 6) Let  $T = \text{MSB}_t(\text{GCTR}_K(J_0, S))$ , where  $t$  is the supported tag length.
- 7) Return  $(C, T)$ .

The GHASH algorithm carries out the message authentication. Its inputs include the hash subkey and a 128-multiple message to be authenticated; its output is the MAC. The GHASH algorithm proceeds as follows:

- 1) Let  $X = X_1 \parallel X_2 \parallel \dots \parallel X_{m-1} \parallel X_m$ , represents a 128m message segment.
- 2) Let  $Y_0 = 0^{128}$ .
- 3) For  $i = 1, \dots, m$ , let  $Y_i = (Y_{i-1} \oplus X_i) \cdot H$ .
- 4) Return  $Y_m$ .

The GCTR algorithm performs the message encryption. Its inputs include a plaintext, a key, and an initial counter block (ICB); its output is a ciphertext. The GCTR algorithm proceeds as follows:

- 1) If  $X$  is an empty string, then return the empty string as  $Y$ .
- 2) Let  $n = \lceil \text{len}(X)/128 \rceil$ .
- 3) Let  $X_1, X_2, \dots, X_{n-1}, X_n^*$ , denote the sequence of the plaintext, where  $1 \leq \text{len}(X_n^*) \leq 128$ .
- 4) Let  $\text{CB}_1 = \text{ICB}$ .
- 5) For  $i = 2$  to  $n$ , let  $\text{CB}_i = \text{inc}_{32}(\text{CB}_{i-1})$ .
- 6) For  $i = 1$  to  $n - 1$ , let  $Y_i = X_i \oplus \text{CIPH}_K(\text{CB}_i)$ .
- 7) Let  $Y_n^* = X_n^* \oplus \text{MSB}_{\text{len}(X_n^*)}(\text{CIPH}_K(\text{CB}_n))$ .

- 8) Let  $Y = Y_1 || Y_2 || \dots || Y_{n-1} || Y_n^*$ .
- 9) Return  $Y$ .

The  $\text{inc}_{32}(S)$  function increments the rightmost 32 bits of  $S$  by 1 mod  $2^{32}$ , and keeps the remaining bits unchanged.

**AES-GCM decryption:** The inputs of the AES-GCM decryption algorithm [16] include an additional authenticated data ( $A$ ), a ciphertext ( $C$ ), an initialization vector ( $IV$ ), a tag ( $T$ ), and a key ( $K$ ). The  $A$  has a length ranging from 0 to  $2^{64} - 1$  bytes. The ciphertext length ranges from 0 to  $2^{39} - 256$  bytes. Padding 0 is carried out if the sum of the  $A$  and  $C$  is not a multiple of 128. The outputs contain the plaintext ( $P$ ) and an indicator to identify whether the MAC is valid.

The AES-GCM decryption [16], [17] algorithm proceeds as the following steps:

- 1) If any of  $A$ ,  $C$ ,  $IV$ , or tag ( $T$ ) is invalid, then returns FAIL.
- 2) Let  $H = \text{CIPH}_K(0^{128})$ .
- 3) If  $\text{len}(IV) = 96$ , then define  $J_0 = IV || 0^{31} || 1$ ;  
If  $\text{len}(IV) \neq 96$ , let  $s = 128 \lceil \text{len}(IV)/128 \rceil - \text{len}(IV)$  and let  
 $J_0 = \text{GMASH}_H(IV || 0^{s+64} || [\text{len}(IV)]_{64})$ .
- 4) Let  $P = \text{GCTR}_K(\text{inc}_{32}(J_0), C)$ .
- 5) Let  $u = 128 \lceil \text{len}(C)/128 \rceil - \text{len}(C)$  and  
let  $v = 128 \lceil \text{len}(A)/128 \rceil - \text{len}(A)$ .
- 6) Define a block:  
 $S = \text{GHASH}_H(A || 0^v || C || 0^u || [\text{len}(A)]_{64} || [\text{len}(C)]_{64})$ .
- 7) Let  $T' = \text{MSB}_t(\text{GCTR}_K(J_0, S))$ , where  $t$  is the supported tag length.
- 8) If  $T = T'$ , then return the plaintext ( $P$ ); otherwise, return Fail.

### III. DESIGN CONSIDERATIONS OF THE PROPOSED HARDWARE ARCHITECTURE

To design a high-throughput AES-GCM hardware architecture, we must resolve the feedback and data dependent problems within both the GCTR and GHASH algorithms.

**AES round operations:** The most basic operation of the AES algorithm is the round operation [4], [17]. To complete one encryption or decryption, it needs  $N_r$  rounds and needs the same number of clock cycles if one round operation can be finished in one cycle.

To speed up the operation of the AES algorithm [7], many ways can be applied. Since the operations [3], including SubBytes, ShiftRows, MixColumns, and AddRoundKey, of each round of the AES algorithm are combinational logic in nature, there are many options can be applied to speed up the circuit. The most common ways include loop unrolling [13], outer-round pipelining, inner-round pipelining, and mixed inner- and outer-round pipelining.

The loop-unrolling method means to remove the feedback path and to use only one big combinational logic circuit to perform the operations of all  $N_r$  rounds, thereby leading to the use of maximum amount of hardware. The outer-round pipelining method also removes the feedback path but instead of using one big combinational logic circuit, a pipeline register is added in between two round circuits. The inner-round pipelining method pipelines the round combinational logic circuit without removing the feedback path. The mixed

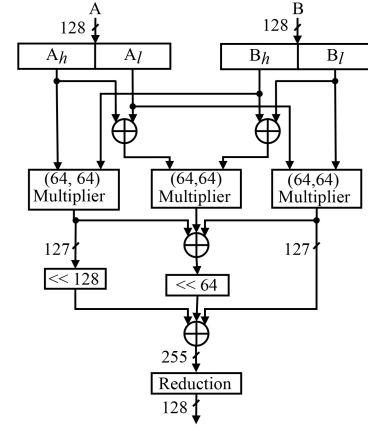


Fig. 2. The detailed structure of the  $\text{GF}(2^{128})$  multiplier.

inner- and outer-round pipelining combines the features of both the outer-round and the inner-round pipelining methods. Thus, it not only removes the feedback path by pipelining the round operations but also pipelining the operations within a round, thereby, leading to the most tiniest pipelining stage. In this paper, we will use this structure [4] due to its high throughput.

**The GHASH module:** The GHASH algorithm carries out the GHASH hash function and comprises a  $\text{GF}(2^{128})$  multiplier and a  $\text{GF}(2^{128})$  adder. To speed up the GHASH algorithm, a parallelized pipelining Karatsuba  $\text{GF}(2^{128})$  multiplier [11], [12] is used to achieve high hardware efficiency and data rate.

The Karatsuba multiplier partitions its two big-length operands into two roughly equal-size numbers and then carries out three multiplications together with a few additions and shift operations to complete the multiplication. For two  $n$ -bit numbers, the time complexity of this multiplier is about  $3n^{\log_2 3} \approx 3n^{1.585} = O(n^{1.585})$  rather than  $O(n^2)$  for the ordinary multiplier.

The Karatsuba multiplier proceeds as in the following steps:

```

function karatsuba(num1, num2) {
    if (num1 < 10 or num2 < 10)
        return num1 * num2; /* traditional multiplication */
    /* calculates the size of the numbers. */
    m = max(size_base_10(num1), size_base_10(num2));
    m2 = ⌊m/2⌋;
    /* split the digit sequences in the middle. */
    high1, low1 = split_at(num1, m2);
    high2, low2 = split_at(num2, m2);
    /* three recursive calls to numbers about half the size. */
    z0 = karatsuba(low1, low2);
    z1 = karatsuba(low1 + high1, low2 + high2);
    z2 = karatsuba(high1, high2);
    return (z2 * 102×m2) + ((z1 - z2 - z0) * 10m2 + z0);
} /* end of the algorithm */

```

The detailed structure of the  $\text{GF}(2^{128})$  multiplier is depicted in Figure 2. Due to the use of iterations in the Karatsuba algorithm, to understand how much hardware that the algorithm requires we implement it with several versions, from first-order to seventh-order, and then verify them with a Xilinx FPGA device and estimate their hardware cost in terms of LUTs. The results show that the fourth-order iteration will be

TABLE I  
DATAFLOW OF THE THREE-STAGE PIPELINED GHASH UNIT (EVEN-NUMBER BLOCKS)

Clock	$I_{0,0}$	$I_{0,1}$	$O_0$	$I_{1,0}$	$I_{1,1}$	$O_1$
1	$X_1$	$H^6$		$X_2$	$H^6$	
2	$X_3$	$H^6$		$X_4$	$H^6$	
3	$X_5$	$H^6$		$X_6$	$H^6$	
4	$X_1 \cdot H^6 + X_7$	$H^6$	$X_1 \cdot H^6$	$X_2 \cdot H^6 + X_8$	$H^5$	$X_2 \cdot H^6$
5	$X_3 \cdot H^6 + X_9$	$H^4$	$X_3 \cdot H^6$	$X_4 \cdot H^6 + X_{10}$	$H^3$	$X_4 \cdot H^6$
6	$X_5 \cdot H^6 + X_{11}$	$H^2$	$X_5 \cdot H^6$	$X_6 \cdot H^6 + X_{12}$	$H$	$X_6 \cdot H^6$
7			$(X_1 \cdot H^6 + X_7) \cdot H^6$			$(X_2 \cdot H^6 + X_8) \cdot H^5$
8			$(X_3 \cdot H^6 + X_9) \cdot H^4$			$(X_4 \cdot H^6 + X_{10}) \cdot H^3$
9			$(X_5 \cdot H^6 + X_{11}) \cdot H^2$			$(X_6 \cdot H^6 + X_{12}) \cdot H$

optimized. Thus, four Karatsuba multipliers with sizes of (16, 16), (32, 32), (64, 64), and (128, 128) will be used to create the desired (128, 128) GF(2<sup>128</sup>) multiplier. It needs 4,033 LUTs as compared 8,179 LUTs for the traditional multiplier and hence has 50.7% savings in hardware cost.

To promote the throughput of the 4th-order Karatsuba multiplier, three pipeline registers are added after the (16, 16) multiplier, the (64, 64) multiplier, and the (128, 128) multiplier output, thereby, creating a three-stage pipelining structure.

The GHASH algorithm comprises a sequence of GF(2<sup>128</sup>) multiplication-addition operations, where each operation depends on its previous result. As a result, it seems hard to parallelize and pipeline the data processing, thereby, limiting the throughput. Fortunately, a GHASH unit based on a single-pipeline GF(2<sup>128</sup>) multiplier is proposed in [2]. In this paper, on the basis of this multiplier, we extend the GHASH unit into two parallelized pipelining GF(2<sup>128</sup>) multipliers in order to obtain twice throughputs.

In the new architecture, we need to consider the number of input blocks of the GHASH unit is even or odd. By expanding the GHASH hash function expression, we can obtain that the result of the hash function is indeed composed of several 128-bit segment message times some integer powers of the key  $H$ . that is,

$$\begin{aligned}
 Y_m &= ((((((X_1 \cdot H \oplus X_2) \cdot H \oplus X_3) \cdot H) \oplus X_4) \cdot H \oplus \dots) \cdot H \oplus X_m) \cdot H \\
 &= X_1 \cdot H^m \oplus X_2 \cdot H^{m-1} \oplus X_3 \cdot H^{m-2} \oplus X_4 \cdot H^{m-3} \\
 &\quad \oplus \dots \oplus X_{m-1} \cdot H^2 \oplus X_m \cdot H
 \end{aligned}$$

To fit our design with two parallelized pipelining GF(2<sup>128</sup>) multipliers, we need a new equivalent expression. For this, let

$$\begin{aligned}
 R_1 &= ((X_1 \cdot H^6 \oplus X_7) \cdot H^6 \oplus \dots \oplus X_{m-5}) \cdot H^6 \\
 R_2 &= ((X_2 \cdot H^6 \oplus X_8) \cdot H^6 \oplus \dots \oplus X_{m-4}) \cdot H^5 \\
 R_3 &= ((X_3 \cdot H^6 \oplus X_9) \cdot H^6 \oplus \dots \oplus X_{m-3}) \cdot H^4 \\
 R_4 &= ((X_4 \cdot H^6 \oplus X_{10}) \cdot H^6 \oplus \dots \oplus X_{m-2}) \cdot H^3 \\
 R_5 &= ((X_5 \cdot H^6 \oplus X_{11}) \cdot H^6 \oplus \dots \oplus X_{m-1}) \cdot H^2 \\
 R_6 &= ((X_6 \cdot H^6 \oplus X_{12}) \cdot H^6 \oplus \dots \oplus X_m) \cdot H
 \end{aligned}$$

Then,  $Y_m$  can be represented as in the following expression:

$$\begin{aligned}
 Y_m &= X_1 \cdot H^m \oplus X_2 \cdot H^{m-1} \oplus X_3 \cdot H^{m-2} \oplus X_4 \cdot H^{m-3} \\
 &\quad \oplus \dots \oplus X_{m-1} \cdot H^2 \oplus X_m \cdot H \\
 &= R_1 \oplus R_2 \oplus R_3 \oplus R_4 \oplus R_5 \oplus R_6
 \end{aligned}$$

The new GHASH unit based on this expression is depicted in Figure 6, where except the last five input data, all the other input data need to be added with the output from the multiplier and then times  $H^6$ . The last fifth data will be times  $H^5$ , the last fourth data will be times  $H^4$ , and so on until all data are completely input. Then, in the next three cycles the two multipliers output their registers (i.e.,  $R_1$  to  $R_6$ ) and add them together to become the final GHASH output.

To further illustrate the operation of the GHASH unit, consider an input data to the GHASH unit that can be partitioned into 12 blocks (i.e., even-number blocks),  $X_1$  to  $X_{12}$ . Then the detailed computations of the GHASH unit are shown in Table I. As indicated in the table, the outputs after the seventh cycle are the values of  $R_1$  to  $R_6$ . If we add these together, we may obtain the desired result. That is,

$$\begin{aligned}
 &((X_1 \cdot H^6 + X_7) \cdot H^6 + (X_2 \cdot H^6 + X_8) \cdot H^5 \\
 &\quad + (X_3 \cdot H^6 + X_9) \cdot H^4 + (X_4 \cdot H^6 \\
 &\quad + X_{10}) \cdot H^3 + (X_5 \cdot H^6 + X_{11}) \cdot H^2 \\
 &\quad + (X_6 \cdot H^6 + X_{12}) \cdot H \\
 &= X_1 \cdot H^{12} + X_2 \cdot H^{11} + X_3 \cdot H^{10} + X_4 \cdot H^9 \\
 &\quad + X_5 \cdot H^8 + X_6 \cdot H^7 + X_7 \cdot H^6 + X_8 \cdot H^5 \\
 &\quad + X_9 \cdot H^4 + X_{10} \cdot H^3 + X_{11} \cdot H^2 + X_{12} \cdot H
 \end{aligned}$$

An illustration of the operation of the GHASH unit on the odd-number input data blocks is shown in Table II, where we assume that the input data to the GHASH unit can be partitioned into 11 blocks,  $X_1$  to  $X_{11}$ . Then the detailed computations of the GHASH unit are shown in the table. As indicated, the last six outputs are not symmetric, and hence need 4 cycles to add them together. That is,

$$\begin{aligned}
 &X_6 \cdot H^6 + (X_1 \cdot H^6 + X_7) \cdot H^5 + (X_2 \cdot H^6 + X_8) \cdot H^4 \\
 &\quad + (X_3 \cdot H^6 + X_9) \cdot H^3 + (X_4 \cdot H^6 + X_{10}) \cdot H^2 \\
 &\quad + (X_5 \cdot H^6 + X_{11}) \cdot H
 \end{aligned}$$

TABLE II  
DATAFLOW OF THE THREE-STAGE PIPELINED GHASH UNIT (ODD-NUMBER BLOCKS)

Clock	$I_{0,0}$	$I_{0,1}$	$O_0$	$I_{1,0}$	$I_{1,1}$	$O_1$
1	$X_1$	$H^6$		$X_2$	$H^6$	
2	$X_3$	$H^6$		$X_4$	$H^6$	
3	$X_5$	$H^6$		$X_6$	$H^6$	
4	$X_1 \cdot H^6 + X_7$	$H^5$	$X_1 \cdot H^6$	$X_2 \cdot H^6 + X_8$	$H^4$	$X_2 \cdot H^6$
5	$X_3 \cdot H^6 + X_9$	$H^3$	$X_3 \cdot H^6$	$X_4 \cdot H^6 + X_{10}$	$H^2$	$X_4 \cdot H^6$
6	$X_5 \cdot H^6 + X_{11}$	$H$	$X_5 \cdot H^6$			$X_6 \cdot H^6$
7			$(X_1 \cdot H^6 + X_7) \cdot H^5$			$(X_2 \cdot H^6 + X_8) \cdot H^4$
8			$(X_3 \cdot H^6 + X_9) \cdot H^3$			$(X_4 \cdot H^6 + X_{10}) \cdot H^2$
9			$(X_5 \cdot H^6 + X_{11}) \cdot H$			

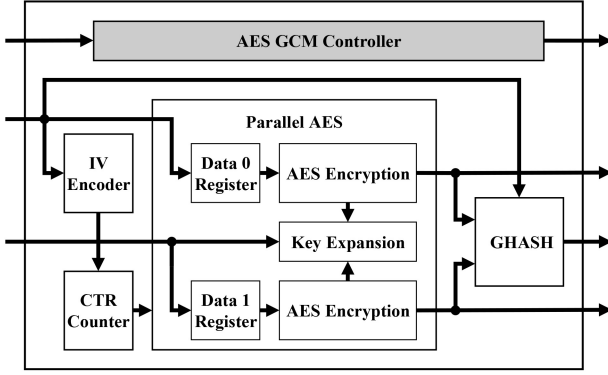


Fig. 3. The internal blocks of the proposed AES-GCM architecture.

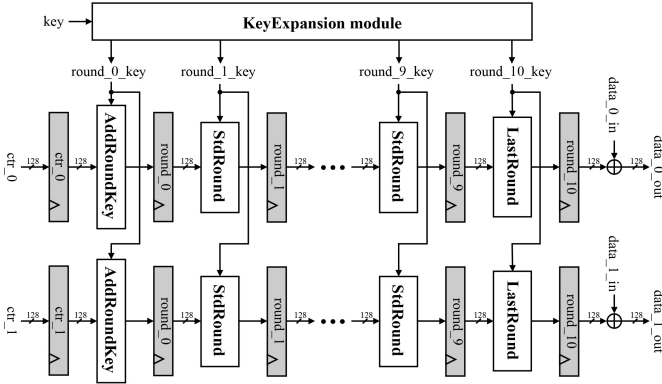


Fig. 4. The detailed structure of the AES module.

$$\begin{aligned}
 &= X_1 \cdot H^{11} + X_2 \cdot H^{10} + X_3 \cdot H^9 + X_4 \cdot H^8 + X_5 \cdot H^7 \\
 &\quad + X_6 \cdot H^6 + X_7 \cdot H^5 + X_8 \cdot H^4 + X_9 \cdot H^3 \\
 &\quad + X_{10} \cdot H^2 + X_{11} \cdot H.
 \end{aligned}$$

#### IV. THE PROPOSED HARDWARE ARCHITECTURE

The proposed AES-GCM architecture is shown in Figure 3. It consists of an IV encoder, a CTR counter, a parallel (dual) AES module, a GHASH module, and a controller.

*The parallel AES module:* The parallel AES module contains two AES encryption units and one key-expansion module, as depicted in Figure 4. These two AES units share the key-expansion module. They can accept two individual 128-bit data inputs at the same time and then proceed the encryption with the AES-CTR mode.

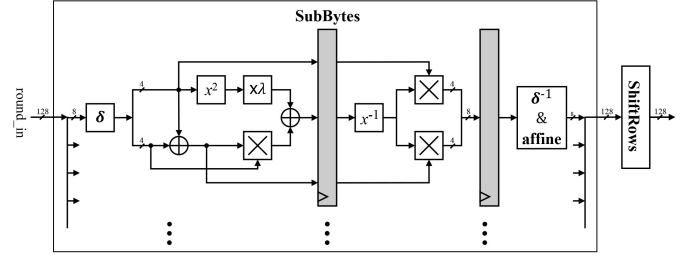


Fig. 5. The implementation of the S-box in ASIC design.

To match with the hardware features of FPGA devices and the ASIC library, different design considerations are applied to design the AES module. To obtain a better throughput under the same hardware cost or chip area, we use the fully expanded mixed inner- and outer-round pipelining architecture. Generally, the S-box of the AES algorithm may be realized by a look-up table to store the S-box table or a combinational logic circuit to compute the desired value on the fly. To efficiently use the best hardware feature of the FPGA device and of the ASIC library, we separately design and implement the S-box.

In FPGA design, the hardware resource can be significantly saved with block random access memory (BRAM) facilitated by the FPGA device rather than using logic circuits. To speed up the operation of the S-box with BRAMs, a pipeline register is added at the output of the SubBytes stage to complete the inner-round pipelining.

In ASIC design, the S-box function is computed with the GF(2<sup>8</sup>) operation [10], [22] to save the hardware resource. To speed up the operation, two pipeline registers are used to partition the circuit into two pipeline stages, as depicted in Figure 5.

*The GHASH module:* The GHASH module is shown in Figure 6. To speed up the operation, a two-stage pipelining architecture is constructed. In computing the powers of key  $H$ , the two GF(2<sup>128</sup>) multipliers in the GHASH module are used to compute the  $H^2$  to  $H^6$  in parallel.

Before computing the GHASH hash function, the key  $H$  is entered the GHASH module through  $h\_in$  and stored in the  $H$ ,  $A$ , and  $B$  registers in preparation to compute  $H^2$ . After three cycles, the first output of the GF(2<sup>128</sup>) multiplier is  $H^2$ . By feeding  $H^2$  into registers  $A$ ,  $C$ ,  $D$ , and  $H^2$  and feeding the

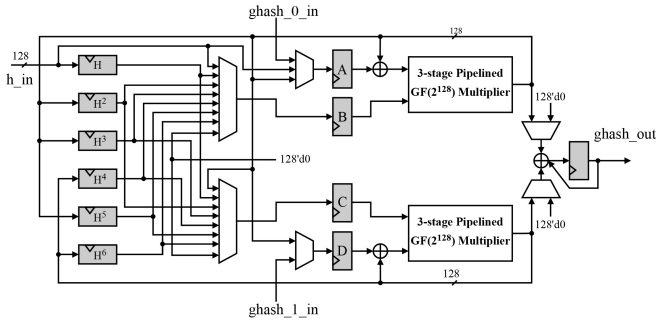


Fig. 6. The parallelized pipelining GHASH module.

value in  $H$  into register  $B$ , the output of the first  $GF(2^{128})$  multiplier is  $H^3$  and of the second multiplier is  $H^4$ .

By feeding  $H^3$  into registers  $A$ ,  $C$ ,  $D$ , and  $H^3$  and feeding the value in  $H^2$  into register  $B$ , the output of the first  $GF(2^{128})$  multiplier is  $H^5$  and of the second multiplier is  $H^6$ . From the above discussion, except for  $H^2$ , the other values of  $H$ , i.e.,  $H^3$  to  $H^6$  can be computed in parallel. After  $H^2$  to  $H^6$  are computed, the GHASH module can receive two 128-bit messages from  $ghash\_0\_in$  and  $ghash\_1\_in$  and output the computed result through  $ghash\_out$ .

*The AES-GCM datapath:* The complete datapath of the AES-GCM architecture is depicted in Figure 7. It consists of an IV module composed of an IV register and two adders, two pipelined AES-CTR modules, one GHASH module, and a number of data input and output registers. Before proceeding message authentication/encryption, register  $J\_0$  is cleared so as to prepare for creating the hash key  $H$ . The key and IV are input into the AES and IV modules through  $data\_0\_in$  and  $data\_1\_in$ , respectively. At this time, the value in  $J\_0$  (cleared) and the encoded IV are input to the AES module as the first data input. The encoded IV plus 1 is entered the  $J\_0$  register and the encoded IV plus 2 is entered the  $J\_1$  register, which in turn become the second input of the AES module. After this, both registers  $J\_0$  and  $J\_1$  are added by 2 repeatedly and serve as the input of the AES module until the first encrypted message appears at the output of the AES module. The two outputs from the AES module are the hash key  $H$  and  $AES_k(J_0)$ .  $H$  is input into the GHASH module and stored in the  $H$  register whereas  $AES_k(J_0)$  is stored in register  $AES_k(J_0)$ . Now, the controller stops the AES computation and the GHASH module starts the computation of the various powers of  $H$ . After these powers of  $H$  are computed, the datapath can accept the external message and perform the ASE-CTR encryption and compute the GHASH hash function.

Since the AES-GCM architecture may accept two sets of messages at the same time, it is possible to occur that both the additional authenticated data and the plaintext are input simultaneously. To resolve this, we add the XOR (indicated by  $\oplus$ ) operation for the first set of data input and the AES second round operation as well as for the second sets of data input and the AES first round operation through the use of multiplexers to coordinate their operations.

Because the AES-GCM module does not need to know the message length in advance, we require two registers to

TABLE III  
AES-GCM TEST PATTERNS

Variable	Value (hexadecimal)
Key	feffe992_8665731c_6d6a8f94_67308308
Initial vector	cafebabe_facedbad_decaf888
Additional authenticated data	feedface_deadbeef_feedface_deadbeef abaddad2
Plaintext	d9313225_f88406e5_a55909c5_aff5269a 86a7a953_1534f7da_2e4c303d_8a318a72 1c3c0c95_95680953_2fc0e24_49a6b525 b16aedf5_aa0de657_ba637b39
Ciphertext	42831ec2_21777424_4b7221b7_84d0d49c e3aa212f_2c02a4e0_35c17e23_29aca12e 21d514b2_5466931c_7d8f6a5a_ac84aa05 1ba30b39_6a0aac97_3d58e091
MAC	5bc94fbc_3221a5db_94fae95a_e7121a47

memorize the length of the additional authenticated data and the plaintext. In addition, for the design of the GHASH module, we need to know three cycles in advance before the end of the data; thereby, it must delay three cycles before data entering the GHASH module after the data entered the AES-GCM module. To solve this difficulty, we add three registers in both of these data paths and use the last two registers as the ciphertext output registers. Before entering the GHASH module, the data passes through two mask modules, which determine whether append 0s or not according to the effective bit signal of the external data.

The difference between the authentication decryption and the authentication encryption circuits lies on the tag register, which stores the MAC to be verified, and a comparator to compare and verify the validity of the incoming message.

## V. EXPERIMENTAL RESULTS

To verify the proposed AES-GCM architecture that can satisfy the NIST standard, we verified the resulting system with the NIST test pattern, as listed in Table III.

Since both the pipeline and parallelism are used in the proposed AES-GCM architecture, two 128-bit messages can be processed at the same time. Hence, the throughput of the AES-GCM architecture can be defined as follows

$$\text{Throughput} = \frac{128 \times n}{T} \times f \quad (1)$$

where  $n$  is the number of 128-bit data that can be processed at a time; in this paper, it is fixed to 2.  $T$  is the number of clock cycles needed for processing one 128-bit message; it is 1.  $f$  is the operating frequency,  $f = 223.314$  MHz. Based on this, the maximum throughput is 57.17 Gbps in the implementation with Virtex 5 (XC5VLX220) of the proposed architecture. The comparison with the others is summarized in Table IV.

From the table, we can know that the use of block RAM (BRAMs) provided by the FPGA device may save a lot of slices used. Due to the use of pipeline and parallelism, the proposed architecture has a much high throughput and gains a much better ratio of throughput to hardware resource (i.e., throughput/slices plus BRAMs) than existing designs.

In the ASIC verification, we synthesize the resulting architecture with the tsmc 0.18- $\mu\text{m}$  process, the resulting chip

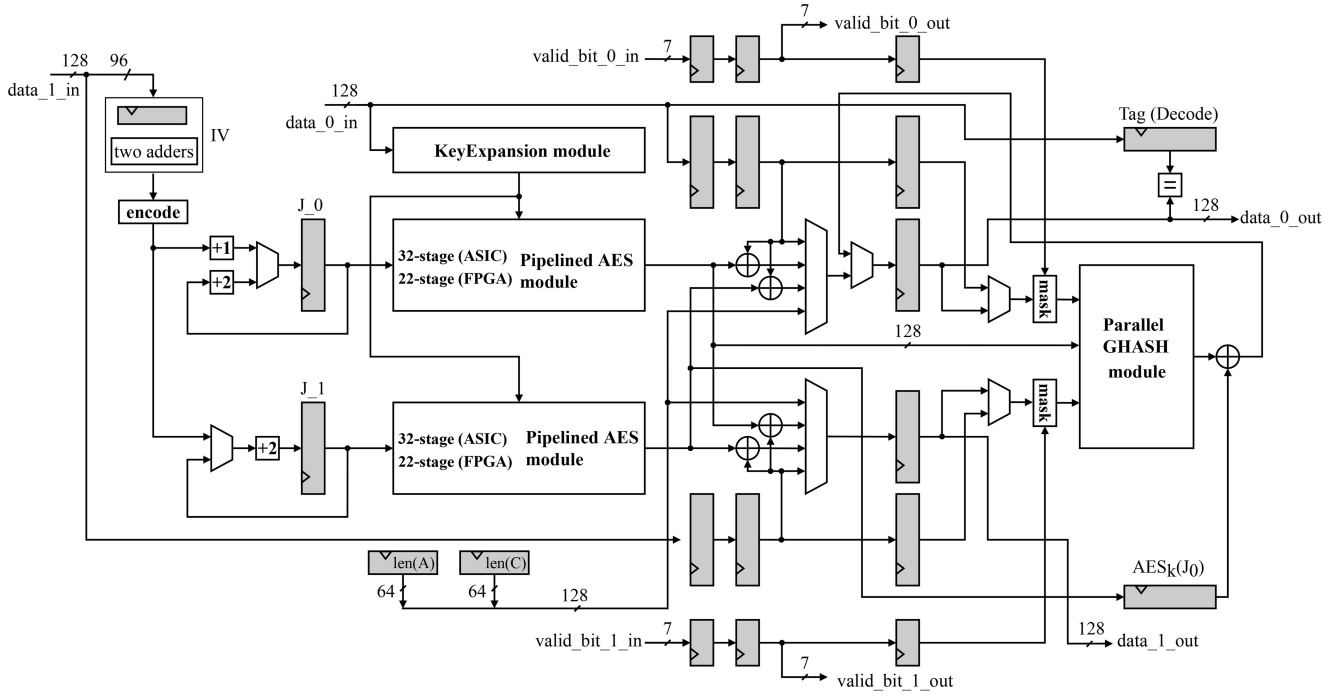


Fig. 7. The complete datapath of the AES-CGM architecture.

TABLE IV  
PERFORMANCE COMPARISON OF VARIOUS FPGA IMPLEMENTATIONS

Design	FPGA Type	S-box	Slices	BRAMs	Freq. (MHz)	(Gbps)
[1]	Virtex 5					
	XC5VLX220	Logic Mapping	12,152	0	200	102.40
[6]	Virtex 5	Logic Mapping	14,799	0	233	119.30
	XC5VLX220	Complex field	18,505	0	233	119.30
[23]	Virtex 5	Logic Mapping	5,961	0	296	37.89
	XC5VLX85	Complex field	8,077	0	305	39.04
		BRAM	4,115	59	287	36.74
This work	Virtex 5					
	XC5VLX220	BRAM	5,944	109	223.314	57.17

TABLE V  
ASIC SPECIFICATION

Process	tsmc 0.18 $\mu\text{m}$
Chip area	6928.3 $\mu\text{m} \times 6892 \mu\text{m}$
Core area (without IO PAD)	6552.3 $\mu\text{m} \times 6516 \mu\text{m}$
Equivalent gates	348.986 k
Operating frequency	197.239 MHz
Power dissipation of core	860.648 mW
Power dissipation of I/O pads	605.883 mW
Operating temperature	0° to 125°
Operating voltage	1.8 V $\pm$ 10%

TABLE VI  
PERFORMANCE COMPARISON OF ASICs

Design	Process ( $\mu\text{m}$ )	Frequency (MHz)	Throughput (Gbps)	Gate Count (kgates)	Hardware Efficiency (kpbs/Gate)
[21]	0.18	271	34.69	498.658	69.57
[9]	0.09	824	10.5	49.633	211.55
	0.13	500	6.4	40.335	158.67
[18]	0.13	333.3	42.67	297.542	143.41
[19]	0.13	200	102.4	600.44	170.54
This work	0.18	197.239	50.49	348.986	144.68

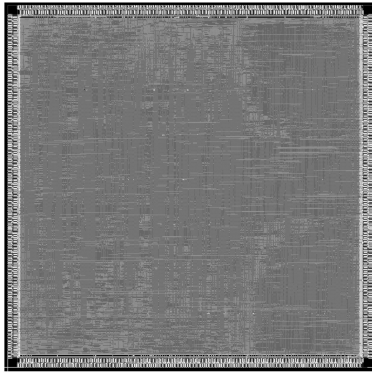


Fig. 8. The chip layout.

layout is depicted in Figure 8. The detailed specifications are listed in Table V.

Table VI compares the performance and hardware efficiency with others. Here, the hardware efficiency is defined as follows

$$\text{Hardware efficiency} = \frac{\text{Throughput}}{\text{Gate count}} \quad (2)$$

The maximum throughput of the proposed architecture is 50.49 Gbps. As compared with [21], the hardware efficiency of the proposed architecture is  $2.08\times$  that of [21] under the same process technology. In addition, the hardware efficiency of the proposed architecture can keep up with that of [18], which uses 0.13- $\mu\text{m}$  process, a more advanced one than that used in this work.

## VI. CONCLUSION

In this paper, we proposed a high hardware-efficiency AES-GCM architecture. To make good use of the hardware features

inherent in FPGA and ASIC techniques, the different method is employed to design and implement the S-box of the AES algorithm. In FPGA design, the built-in BRAMs are used to reduce the amount of LUTs used whereas in ASIC design the finite field computation is used to save up to 25% logic gates. For the  $GF(2^{128})$  multiplier of the GHASH algorithm, the Karatsuba fast multiplication algorithm is utilized to reduce the amount of hardware resource. In FPGA design, a savings can be up to 50.7% LUTs. In addition, pipeline registers are added in the critical path of the  $GF(2^{128})$  multiplier to speed up the data processing. Furthermore, in the AES-GCM datapath, two parallel AES modules with a hybrid inner- and outer-round pipelining structure along with a parallel and pipeline GHASH computation unit are used to promote the system throughput.

The resulting AES-GCM architecture has been implemented and verified with a Xilinx Virtex 5 series device (XC5VLX220) and a tsmc 0.18- $\mu\text{m}$  cell library. In the FPGA verification, the operating frequency can achieve 223.314 MHz, yielding the maximum throughputs of 57.17 Gbps. The number of slices used is 5,944, and the number of BRAMs is 109. In ASIC verification, the resulting chip can operate at 197.239 MHz in simulation and achieve a high throughput of 50.49 Gbps. The core area of the chip is  $6,552.3 \mu\text{m} \times 6,516 \mu\text{m}$ , equivalent to 348.986 kgates. The core power consumption is 860.648 mW, and the I/O Pad power consumption is 605.883 mW.

## REFERENCES

- [1] K. M. Abdellatif, R. Chotin-Avot, and H. Mehrez, "Improved method for parallel AES-GCM cores using FPGAs," in *Proc. Int. Conf. Reconfig. Comput. FPGAs (ReConFig)*, Dec. 2013, pp. 1–4.
- [2] T. Chen, W. Huo, and Z. Liu, "Design and efficient FPGA implementation of Ghash core for AES-GCM," in *Proc. Int. Conf. Comput. Intell. Softw. Eng. (CISE)*, Wuhan, China, Dec. 2010, pp. 1–4.
- [3] *Advanced Encryption Standard*, document FIPS 197, NIST, Gaithersburg, MD, USA, Nov. 2001.
- [4] K. Gaj and P. Chodowiec, "Comparison of the hardware performance of the AES candidates using reconfigurable hardware," in *Proc. 3rd Adv. Encrypt. Stand. (AES3) Candidate Conf.*, New York, NY, USA, Apr. 2000, pp. 1–15.
- [5] K.-S. Han, K.-O. Kim, T. W. Yoo, and Y. Kwon, "The design and implementation of MAC security in EPON," in *Proc. IEEE Int. Conf. Adv. Commun. Technol. (ICACT)*, Gangwon, South Korea, Feb. 2006, pp. 20–22.
- [6] L. Henzen and W. Fichtner, "FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications," in *Proc. Eur. Conf. Solid-State Circuits (ESSCIRC)*, Sevilla, Spain, Sep. 2010, pp. 202–205.
- [7] Y. Huang, Y. Lin, K. Hung, and K. Lin, "Efficient implementation of AES IP," in *Proc. IEEE Asia-Pac. Conf. Circuits Syst.*, Singapore, Dec. 2006, pp. 1418–1421.
- [8] *IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security*, IEEE Standard 802.1AE-2006, Aug. 2006.
- [9] IP Cores. "GCM1/GCM2 802.1ae (MACSec) GCM/AES Cores." 2006. [Online]. Available: <http://www.ipcores.com/IEEE802.1AE-AES-GCM-Core.htm>
- [10] A. Joshi, P. K. Dakhole, and A. Thatere, "Implementation of S-box for advanced encryption standard," in *Proc. IEEE Int. Conf. Eng. Technol. (ICETECH)*, Coimbatore, India, Mar. 2015, pp. 1–5.
- [11] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *Proc. USSR Acad. Sci.*, vol. 14, no. 145, pp. 293–294, Dec. 1962.
- [12] A. A. Karatsuba, "The complexity of computations (PDF)," *Proc. Steklov Inst. Math.*, vol. 211, pp. 169–183, Jan. 1995.
- [13] M. B. Lin, *Digital System Designs and Practices: Using Verilog HDL and FPGAs*. Singapore: Wiley, 2008.
- [14] D. A. McGrew and J. Viega. "The galois/counter mode of operation (GCM)." May 2005. [Online]. Available: <http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-revised-spec.pdf>
- [15] D. A. McGrew and J. Viega. "The security and performance of the galois/counter mode (GCM) of operation (full version)." 2008. [Online]. Available: <http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-ad.pdf>
- [16] *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) for Confidentiality and Authentication*, NIST SP 800-38D, NIST, Gaithersburg, MD, USA, Apr. 2006. [Online]. Available: [http://csrc.nist.gov/publications/drafts/Draft-NIST\\_SP800-38D\\_Public\\_Comment.pdf](http://csrc.nist.gov/publications/drafts/Draft-NIST_SP800-38D_Public_Comment.pdf)
- [17] W. Stallings, *Cryptography and Network Security Principles and Practice*, 8th ed. London, U.K.: Pearson, 2023.
- [18] A. Satoh, "High-speed hardware architectures for authenticated encryption mode GCM," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Kos, Greece, May 2006, pp. 4831–4834.
- [19] A. Satoh, "High-speed parallel hardware architecture for Galois counter mode," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2007, pp. 1863–1866.
- [20] Y. Sovyn, V. Khoma, and M. Podpora, "Comparison of three CPU-core families for IoT applications in terms of security and performance of AES-GCM," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 339–348, Jan. 2020.
- [21] B. Yang, S. Mishra, and R. Karri. "High speed architecture for galois/counter mode of operation (GCM)." Jun. 2005. [Online]. Available: <http://eprint.iacr.org/2005/146.pdf>
- [22] X. Zhang and K. K. Parhi, "High-speed VLSI architectures for the AES algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 9, pp. 957–967, Sep. 2004.
- [23] G. Zhou and H. Michalik, "Improving throughput of AES-GCM with pipelined Karatsuba multipliers on FPGAs," in *Reconfigurable Computing: Architectures, Tools and Applications (Lecture Notes in Computer Science 5453)*. Heidelberg, Germany: Springer, 2009.



**Ming-Bo Lin** (Senior Member, IEEE) received the B.Sc. degree in electronic engineering from the National Taiwan Institute of Technology (currently is the National Taiwan University of Science and Technology), Taipei City, the M.Sc. degree in electrical engineering from National Taiwan University, Taipei City, and the Ph.D. degree in electrical engineering from the University of Maryland at College Park, College Park. He has been a Professor with the Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology since 2001. His research interests include VLSI systems design, mixed-signal integrated circuit designs, parallel architectures and algorithms, and embedded computer systems.



**Jen-Hua Chuang** received the B.Sc. degree in computer science and engineering from Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung City, and the M.Sc. degree in electronic engineering from the Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology, Taipei City, in 2022. He has been working with Mediatek Inc. as a Digital IC Design Engineer since 2022. His research interests and expertise are on the field of digital ASIC design and implementation.