IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 4, 2022, accepted February 21, 2022, date of publication March 2, 2022, date of current version March 23, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3156291*

# Lightweight and Low-Latency AES Accelerator Using Shared SRAM

**JAE SEONG LEE[1], PILJOO CHOI[2], (Member, IEEE), AND DONG KYUE KIM[3], (Member, IEEE)**

[1]Department of Electronics and Computer Engineering, Hanyang University, Seoul 04763, South Korea
[2]Department of Computer Engineering, Pukyong National University, Busan 48513, South Korea
[3]Department of Electronic Engineering, Hanyang University, Seoul 04763, South Korea

Corresponding author: Dong Kyue Kim (dqkim@hanyang.ac.kr)

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korea Government (MSIT) (RISC-V-based Secure CPU Architecture Design for Embedded System Malware Detection and Response) under Grant 2021-0-00724.

**ABSTRACT** In this study, we propose a lightweight and low-latency advanced encryption standard (AES) accelerator. Instead of being connected to the bus through its own slave wrapper, the proposed AES accelerator is located within the slave wrapper of the static random-access memory (SRAM) and is directly attached to the SRAM. Hence, the AES accelerator can directly access data in the SRAM and share SRAM space for storing expanded keys, resulting in no time for transferring input and output data, no resource usage for storing keys, and no power wastage for repeated key expansion. The proposed AES accelerator has a latency of 53 clock cycles per encryption/decryption process and has a gate count of 2912 when synthesized using 28 nm process technology. The latency is similar to that of another AES accelerator with the same 32-bit data path; however, the size of the proposed accelerator is 46.0% smaller. Furthermore, compared with other AES accelerators with 8-bit data path, the proposed AES accelerator has a 3.0–22.0 times smaller latency with a slightly larger area.

**INDEX TERMS** Coprocessors, cryptography, digital circuit, encryption.

## I. INTRODUCTION

Advanced encryption standard (AES) [1] is one of the most widely used block ciphers for data encryption, and its application ranges from high-performance to resource-constrained ones. The AES can be used by running AES software on a general-purpose processor of personal computers or microcontrollers. However, the AES software requires hundreds or thousands of clock cycles (CCs) to encrypt one block of data [2].

To increase the encryption speed, hardware AES accelerators can be used. For example, Satoh *et al.* [3] proposed an AES accelerator that can encrypt one block within 11–54 CCs. However, this method has the following two limitations. First, the CCs required for data transfer are not counted. The AES uses 128-, 192-, and 256-bit keys and 128-bit blocks. AES encryption with 128-bit key requires transfer of at least 12 words of input and output data, including the key, plaintext, and ciphertext, between the AES accelerator and static random-access memory (SRAM) in a 32-bit bus system. If an AES accelerator is attached to the bus system as a slave [4], data transfer from/to memory is controlled by a master, such as a processor and a direct memory access controller (DMAC). Consequently, data cannot be directly transferred between the AES accelerator and SRAM; the data can be only transferred via a master. Therefore, transferring 12 words requires at least 12 CCs, which are not negligible. Second, AES accelerators should repeat the same key expansion. When a 128-bit key is used, the AES expands the 128-bit input key into eleven 128-bit round keys for Round0–Round10. As the same key is often used for multiple blocks, the input key can be expanded once, and the expanded keys stored in the SRAM can be used repeatedly. This method is commonly used in software implementations, whereas most AES accelerators use on-the-fly key expansion, which expands the input key in every encryption/decryption process. This is because an AES accelerator requires its own SRAM or a large register to store the expanded key, thereby substantially increasing the cost of hardware implementation. However, repeating the same key extension wastes

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamad Afendee Mohamed.

power, which is undesirable, especially in power-constrained applications.

In this paper, we propose a method to directly attach an AES accelerator to the SRAM. Instead of transferring input and output data, the processor only passes the start address of the plaintext to the AES accelerator. Then, the AES accelerator reads and writes the data of the given address while performing key expansion, encryption, and decryption. The proposed AES accelerator provides two main advantages:

1) Fewer resources are required by storing the expanded keys in the SRAM instead of registers.
2) Less time and fewer resources are required for data transfer by directly accessing the SRAM.

First, a typical hardware AES accelerator has at least two 128-bit registers for the text and key. In contrast, the proposed accelerator exploits the SRAM to store both the input key and expanded keys. This SRAM sharing approach allows to remove the 128-bit register for storing the key. Second, data can be transferred directly between the SRAM and AES accelerator without using any bus. This approach can reduce the resources and time required for data transfer and allows the masters and bus to be used for other tasks.

The remainder of this paper is organized as follows. Background information is provided in Section II. The proposed methods are detailed in Section III. Implementation results are presented in Section IV. Finally, conclusions are drawn in Section V.

## II. PRELIMINARY

This section describes the AES algorithm and shows the structure of the microcontroller, the target device where the proposed AES accelerator can be used.

### A. AES ALGORITHM

The AES algorithm processes 16-byte intermediate values as a $(4 \times 4)$-byte array, called *state*. On the bytes, rows, columns, and the complete *state*, AES transformations including SubBytes (SBs), ShiftRows (SRs), MixColumns (MCs), and AddRoundKeys (ARKs) are performed, respectively.

Let $s^i$ and $rk^i$ denote *state* and the round key at round $i$, respectively. $s^i$ $(0 \le i \le 10)$ is defined as

$$s^0 \leftarrow pt \oplus rk^0, \tag{1}$$

$$s^i \leftarrow MCs\left(SRs\left(SBs\left(s^{i-1}\right)\right)\right) \oplus rk^i \quad (1 \le i \le 9), \tag{2}$$

$$s^{10} \leftarrow SRs\left(SBs\left(s^9\right)\right) \oplus rk^{10}, \tag{3}$$

where $pt$ denotes the plaintext, and $s^{10}$ is the resulting ciphertext. SB, which is also called S-box, performs nonlinear substitutions on each byte. This nonlinear operation is typically described as a multiplicative inversion followed by an affine transformation using matrix multiplication and exclusive OR (XOR, $\oplus$) with a predefined vector. SR rotates each row $r_j$ by $j$ bytes to the left. MC mixes a column. Let $b_i$ and $b'_i$ be the $i$-th bytes in the input and output columns of MC, respectively. $b'_i (0 \le i < 4)$ is defined over GF($2^8$) as

$$b'_i \leftarrow (b_i * \{02\}) \oplus (b_{i+1 \bmod 4} * \{03\}) \oplus b_{i+2 \bmod 4} \oplus b_{i+3 \bmod 4}, \tag{4}$$

where $\{h_1 h_0\}$ represents a hexadecimal number and is equal to $h_1 \times 16 + h_0$, and $*$ represents a convolution using reduction polynomial $x^8 + x^4 + x^3 + x + 1$. ARK, that is, $\oplus rk^i$ $(0 \le i \le 10)$ in (1)–(3) adds a 128-bit round key to *state*. Since this addition is performed over GF($2^8$), it is equivalent to bitwise XOR.

For decryption, inverse transformations are performed, and the order of some transformations and round keys are changed as follows:

$$s^0 \leftarrow ct \oplus rk^{10}, \tag{5}$$

$$s^i \leftarrow MCs^{-1}\left(SRs^{-1}\left(SBs^{-1}\left(s^{i-1}\right)\right) \oplus rk^{10-i}\right) \quad (1 \le i \le 9), \tag{6}$$

$$s^{10} \leftarrow SRs^{-1}\left(SBs^{-1}\left(s^9\right)\right) \oplus rk^0, \tag{7}$$

where $ct$ denotes the ciphertext, and $s^{10}$ is the resulting plaintext. Inverse SB performs an inverse affine transformation followed by a multiplicative inversion. Inverse SR rotates each row $r_j$ by $j$ bytes to the right. The output bytes of inverse MC are computed as

$$b'_i \leftarrow (b_i * \{0e\}) \oplus (b_{i+1 \bmod 4} * \{0b\}) \oplus (b_{i+2 \bmod 4} * \{0d\}) \oplus (b_{i+3 \bmod 4} * \{09\}). \tag{8}$$

Compared with (2), ARK in (6), that is, $\oplus rk^{10-i}$ $(1 \le i \le 9)$ is performed before inverse MCs rather than after MCs.

The AES can be used in various block modes such as electronic code block (ECB), cipher block chaining (CBC), and counter (CTR) modes. For example, AES encryption and decryption are performed in the CBC mode as follows:

$$ct_i = Enc_k\left(pt_i\right) \oplus ct_{i-1}, \tag{9}$$

$$pt_i = Dec_k\left(ct_i \oplus ct_{i-1}\right), \tag{10}$$

where $pt_i, ct_i (i > 0)$ are the $i$-th blocks of plaintext and ciphertext, respectively, and $ct_0$ is an initial vector (IV). $Enc_k$ and $Dec_k$ represent AES encryption and decryption with the input key $k$, respectively. In the CTR mode, encryption and decryption are performed as follows:

$$ct_i = Enc_k([IV, CTR_i]) \oplus pt_i, \tag{11}$$

$$pt_i = Enc_k([IV, CTR_i]) \oplus ct_i, \tag{12}$$

where $[a, b]$ is concatenation of $a$ and $b$, and $CTR_k$ is a number that increases for each block. In this mode, AES decryption $Dec_K$ is not required, which can reduce the required resources.

### B. STRUCTURE OF MICROCONTROLLER

Fig. 1 shows the structure of a microcontroller [4]. The masters, such as an ARM Cortex-M3 processor and a DMAC; slaves, such as SRAM, read-only memory (ROM), and

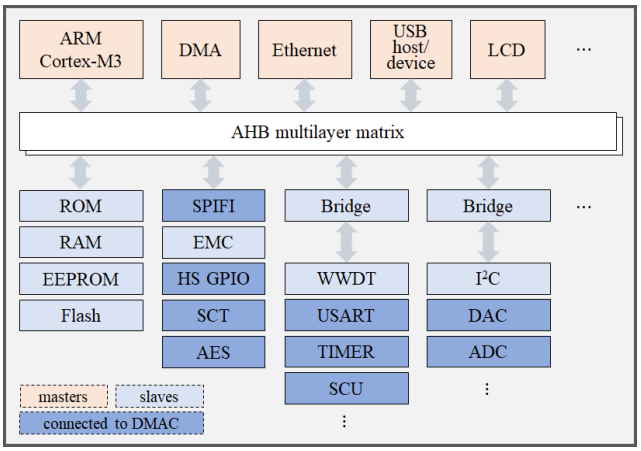



**FIGURE 1.** Structure of microcontroller [4].

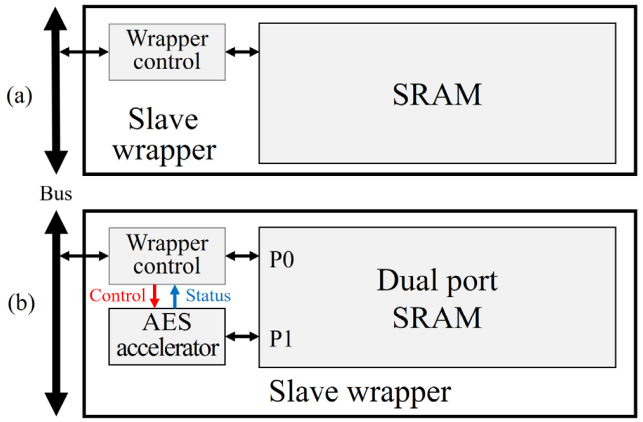



**FIGURE 2.** (a) Conventional and (b) proposed bus architectures.

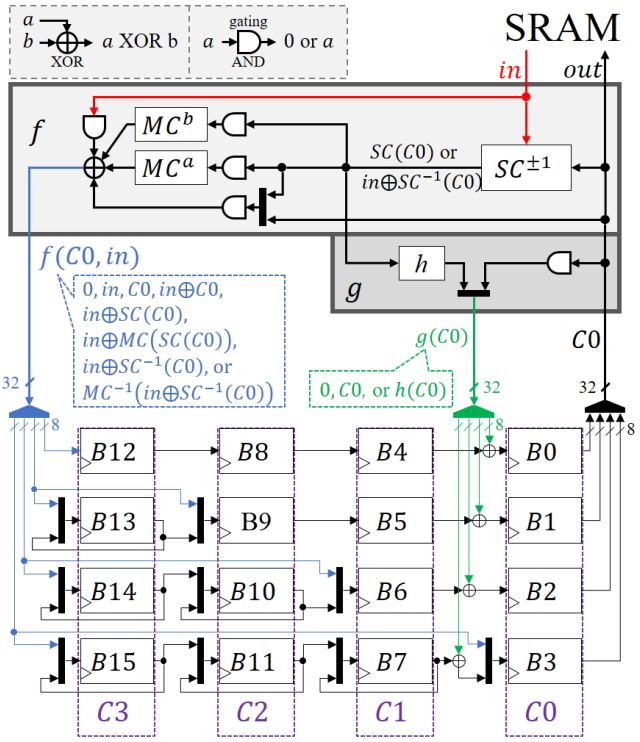



**FIGURE 3.** Architecture of the proposed AES accelerator.

## III. PROPOSED METHODS

In this section, we detail the proposed bus architecture, the proposed AES accelerator architecture, and its operation.

### A. PROPOSED BUS ARCHITECTURE

Slave wrappers are required to connect slave modules including the SRAM to the bus, as shown in Fig. 2(a). In the proposed bus architecture, the AES accelerator is not connected to the bus using its dedicated wrapper, but is located within the slave wrapper of the SRAM, as shown in Fig. 2(b). As the AES accelerator is directly connected to the SRAM, it can access the space shared with the processor in the SRAM without using a bus. The processor only needs to pass the start address of the plaintext instead of moving the entire data, such as round keys, plaintext, and ciphertext, to the AES accelerator.

### B. PROPOSED AES ACCELERATOR ARCHITECTURE

The architecture of the proposed AES accelerator is shown in Fig. 3. The AES accelerator has sixteen 8-bit registers, $B0, B1, \ldots, B15$, which store each byte of *state*, arranged in four columns, $C0, \ldots, C3$. As shown in Fig. 3, the value in each column is shifted to the right, and the AES transformations on the rightmost column, $C0$ are computed using $g(C0)$ and $f(C0, in)$, which are defined in Table 1. $i$ and $j$ denote the round number and word number, respectively, and $in$ denotes one word of plaintext, ciphertext, and round keys that is read from the SRAM. Using $i$ and $j$, the AES accelerator accesses a word in the SRAM as follows:

$$in \leftarrow Mem\,[startAddr + 16i + 4j]\,,$$

$$Mem\,[startAddr + 16i + 4j] \leftarrow C0,$$

where *startAddr* is the start address of the plaintext.

Function $g(C0)$ computes the expanded key values. During key expansion, $g(C0)$ produces a nonzero output, such as $h(C0)$ or $C0$, as shown in Table 1, where $h(C0) = RotWord(SC(C0)) \oplus Rcon$. SubColumn (SC) performs SBs on four bytes within one column, RotWord performs one-byte left rotation, and *Rcon* is a predefined vector for each round.

Function $f(C0, in)$ computes combinations of transformations on $C0$ during encryption and decryption. We implemented SC based on S-box optimization [9], [10]. In Fig. 3, $SC^{\pm 1}$ computes $SC(C0)$ in encryption and inverse SC with ARK, that is, $in \oplus SC^{-1}(C0)$ in decryption. $MC^a$ and $MC^b$

**TABLE 1. Definitions of $g(C0)$ and $f(C0,in)$.**

| | | $i$ | $j$ | $f(C0,in)$ | $g(C0)$ |
|---|---|---|---|---|---|
| Key exp. | Data-in | 1 | 3, 0–2 | $in$ | 0 |
| | Exp. and data-out | 2–11 | 0 | $C0$ | $h(C0)$ |
| | | | 1–3 | | $C0$ |
| Enc. | Data-in | 0 | 0–3 | $in$ | 0 |
| | Round0 | 1 | 0–3 | $in \oplus C0$ | 0 |
| | Round1–9 | 1–10 | 0–3 | $in \oplus MC(SC(C0))$ | 0 |
| | Round10 | 11 | 0–3 | $in \oplus SC(C0)$ | 0 |
| | Data-out | 12 | 0–3 | 0 | 0 |
| Dec. | Data-in | 12 | 3–0 | $in$ | 0 |
| | Round0 | 11 | 3–0 | $in \oplus C0$ | 0 |
| | Round1–9 | 10–2 | 3–0 | $MC^{-1}(in \oplus SC^{-1}(C0))$ | 0 |
| | Round10 | 1 | 3–0 | $in \oplus SC^{-1}(C0)$ | 0 |
| | Data-out | 0 | 3–0 | 0 | 0 |

are used for MC and inverse MC. By modifying (4) and (8), the required resources for MC and inverse MC can be reduced [11]. The output byte of MC, $b'_i (0 \le i < 4)$ can be rewritten as

$$b'_i \leftarrow b_i \oplus b_{0123} \oplus ((b_i \oplus b_{i+1 \bmod 4}) * \{02\}), \quad (13)$$

where $b_{0123} = b_0 \oplus b_1 \oplus b_2 \oplus b_3$. Similarly, the output byte of inverse MC, $b'_i (0 \le i < 4)$ can be rewritten as

$$b'_i \leftarrow b_i \oplus b_{0123} \oplus ((b_i \oplus b_{i+1 \bmod 4}) * \{02\}) \oplus ((b_i \oplus b_{i+2 \bmod 4}) * \{04\}) \oplus (b_{0123} * \{08\}). \quad (14)$$

Excluding the XOR with $b_i$, $MC^a$ and $MC^b$ calculates the common part of (13)–(14) (highlighted in red) and the remainer of (14) (highlighted in blue) for the four bytes within the input column, respectively. Let $C$ be the output of $SC^{\pm 1}$. Using $MC^a$ and $MC^b$, we can define MC and inverse MC on $C$, that is, $MC(C)$ and $MC^{-1}(C)$, respectively, as follows:

$$MC(C) = C \oplus MC^a(C), \quad (15)$$

$$MC^{-1}(C) = C \oplus MC^a(C) \oplus MC^b(C). \quad (16)$$

By controlling the multiplexers and AND gates, $f(C0, in)$ produces various values, as detailed in Table 1.

In Fig. 3, the data path from $f(C0, in)$ to the registers is complex owing to SR, which is the only row-wise transformation. In the proposed architecture, SRs for the $i$-th round are performed in the preceding round. That is, during encryption, instead of (1)–(3), *state* is computed as

$$s^0 \leftarrow SRs(pt \oplus rk^0), \quad (17)$$

$$s^i \leftarrow SRs\left(MCs\left(SBs\left(s^{i-1}\right)\right) \oplus rk^i\right) \quad (1 \le i \le 9) \quad (18)$$

$$s^{10} \leftarrow SBs(s^9) \oplus rk^{10}. \quad (19)$$

Similarly, (5)–(7) are respectively replaced by

$$s^0 \leftarrow SRs(ct \oplus rk^{10}), \quad (20)$$

$$s^i \leftarrow SRs\left(MCs^{-1}\left(SBs^{-1}\left(s^{i-1}\right) \oplus rk^{10-i}\right)\right) \quad (1 \le i \le 9), \quad (21)$$

$$s^{10} \leftarrow SBs^{-1}(s^9) \oplus rk^0. \quad (22)$$

Compared with (5)–(7), the expressions in (20)–(22) use SRs instead of inverse SRs. Although the rotation directions of SR and its inverse are the opposite, words of ciphertext and round keys are read from the SRAM in the reversed order when decryption is performed in the proposed AES accelerator. This is shown in Table 1, where $i$ and $j$ increase during key expansion and encryption, but decrease during decryption. As a result, inverse SR for decryption is not required, and SR is used for both encryption and decryption.

### C. PROPOSED AES ACCELERATOR OPERATION

The proposed AES accelerator uses 208 bytes of the SRAM, which can be declared as a single array as follows:

```
unsigned char text[208];
```

After the address of text[0] is registered in the AES accelerator, the accelerator uses the array space as follows:

- text[0]–text[15] for plaintext,
- text[16]–text[191] for round keys,
- text[192]–text[207] for ciphertext.

During key expansion, only text[16]–text[191] are used, and key expansion proceeds as follows:

1) KeyEx1: processor stores input key in text[16]–text[31]
2) KeyEx2: processor commands AES accelerator to start key expansion
3) KeyEx3: AES accelerator reads the input key, $rk_j^{i-1}$ $(i = 1, j = 3, 0, 1, 2)$ from text[28]-text[31] and text[16]-text[27]
4) KeyEx4: AES accelerator calculates round keys, $rk_j^{i-1}$ $(2 \le i \le 11, 0 \le j \le 3)$ and stores them in text [32]–text[191]
5) KeyEx5: AES accelerator sets DONE flag and clears the registers

The details of KeyEx3–KeyEx5 are shown in Fig. 4. As detailed in Table 1, $f(C0, in)$ and $g(C0)$ return different values depending on $i$ and $j$. When $i = 1$ (KeyEx3), words of the input key read from the SRAM are filled in the registers using $f(C0, in) = in$ and $g(C0) = 0$. For $2 \le i \le 11$ (KeyEx4), the values in registers $C3, C2, C1$, and $C0$ are rotated using $f(C0, in) = C0$, and a word of expanded key is generated using $C1 \oplus g(C0)$, where $g(C0)$ is either $h(C0)$ or $C0$. When $i = 12$ (KeyEx5), the registers are cleared using $f(C0, in) = 0$.

Using the input and expanded keys for ARKs, encryption proceeds as follows:

1) Enc1: processor stores plaintext in text[0]–text[15]
2) Enc2: processor commands AES accelerator to start encryption
3) Enc3: AES accelerator reads $pt_j (0 \le j \le 3)$ from text[0]–text[15], where $pt_j$ is the $j$-th word of plaintext
4) Enc4: AES accelerator reads words of round keys, that is, $r_j^{i-1} (1 \le i \le 11, 0 \le j \le 3)$ in text[16]–text[191] and performs encryption transformations
5) Enc5: AES accelerator stores the results, $ct_j (0 \le j \le 3)$ in text[192]–text[207] and clears the registers, where $ct_j$ is the $j$-th word of ciphertext
6) Enc6: AES accelerator sets DONE flag

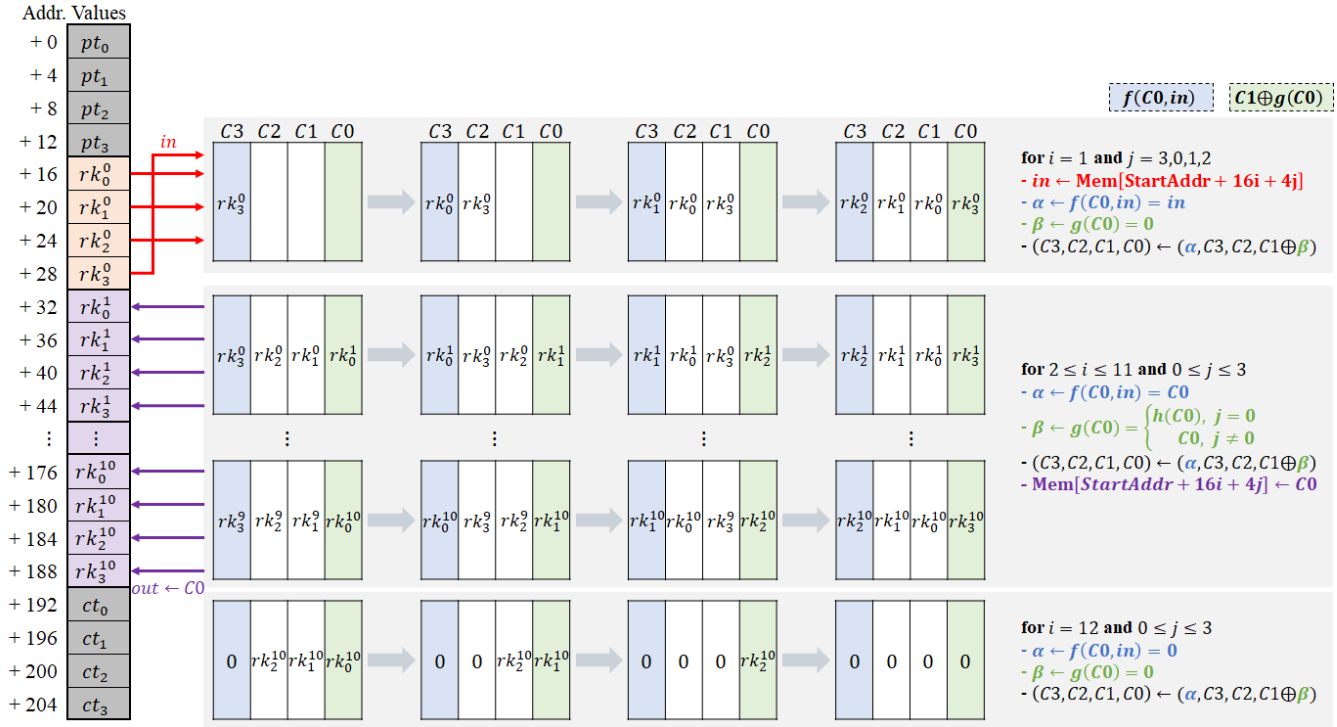



**FIGURE 4.** Key expansion in proposed AES accelerator.

The details of Enc3–Enc5 are shown in Fig. 5, where $g(C0)$ always returns zero. When $i = 0$ (Enc3), words of the plaintext read from the SRAM are filled in the registers using $f(C0, in) = in$, which is similar to KeyEx3. Enc4 is divided into Round0 ($i = 1$) performing ARK and SR; Round1–Round9 ($2 \leq i \leq 10$) performing SC, MC, ARK, and SR; and Round10 ($i = 11$) performing SC and ARK with different return values of $f(C0, in)$. In Enc5, the values in the registers (i.e., words of ciphertext) are stored in the SRAM, and the registers are cleared using $f(C0, in) = 0$.

Decryption proceeds analogously as follows:

1) Dec1: processor stores ciphertext in text[192]–text[207]
2) Dec2: processor commands AES accelerator to start decryption
3) Dec3: AES accelerator reads $ct_j (3 \geq j \geq 0)$
4) Dec4: AES accelerator reads words of round keys, that is, $rk_j^{i-1} (11 \geq i \geq 1, 3 \geq j \geq 0)$ in text[188]–text[191], text[184]–text[187], ..., text[16]–text[19], and it performs decryption transformations
5) Dec5: AES accelerator stores the results, $pt_j (3 \geq j \geq 0)$ in text[12]–text[15], text[8]–text[11], ..., text[0]–text[3], and it clears the registers
6) Dec6: AES accelerator sets DONE flag

Compared with encryption, the order in which words are read and written is reversed during decryption, and inverse transformations, except for SR, are performed.

## IV. EXPERIMENTATION AND IMPLEMENTATION RESULTS

This section presents the execution time and implementation area of the proposed AES accelerator and a comparison with the results of other studies.

### A. EXECUTION TIME AND IMPLEMENTATION AREA OF THE PROPOSED AES ACCELERATOR

In some block modes such as the CTR mode, decryption is not required. By removing the logic circuits for decryption, a smaller and faster accelerator can be obtained. Thus, we designed two versions of AES accelerators: AES-ED that supports key expansion, encryption, and decryption; and AES-E that supports only key expansion and encryption.

The measured CCs are listed in Table 2. Key expansion, which requires 46 CCs, is much less frequently performed than encryption and decryption because the same key is often used over a certain period. Therefore, excluding the CCs for key expansion, the proposed AES accelerator requires only 53 CCs for encryption or decryption of one block.

**TABLE 2.** Execution time and areas of the proposed AES accelerator.

| | Execution time (CCs) | | | Areas (GEs @ MHz) | |
|---|---|---|---|---|---|
| | Key exp. | Enc. | Dec. | @ 333 MHz | @ Max freq. |
| AES-ED | 46 | 53 | 53 | 2912 | 4481 @ 667 |
| AES-E | 46 | 53 | – | 2442 | 3399 @ 769 |

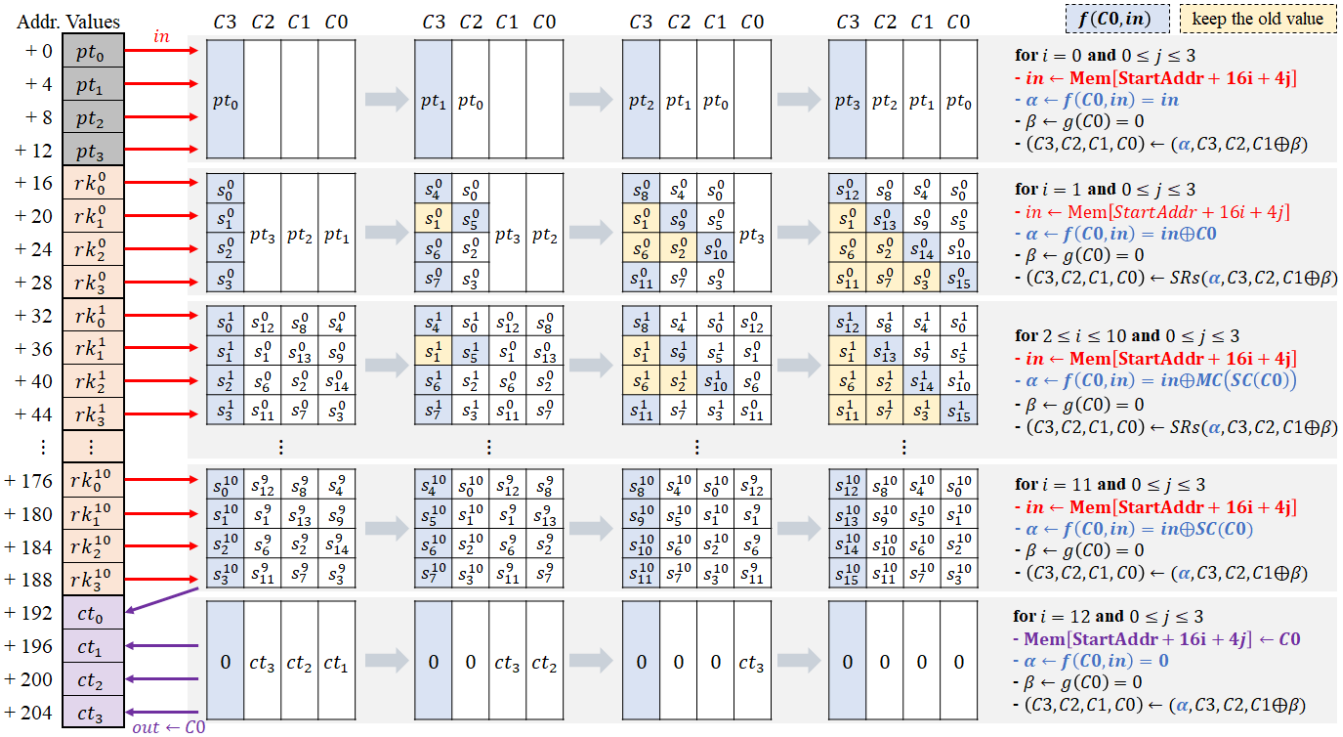



**FIGURE 5.** Encryption in proposed AES accelerator.

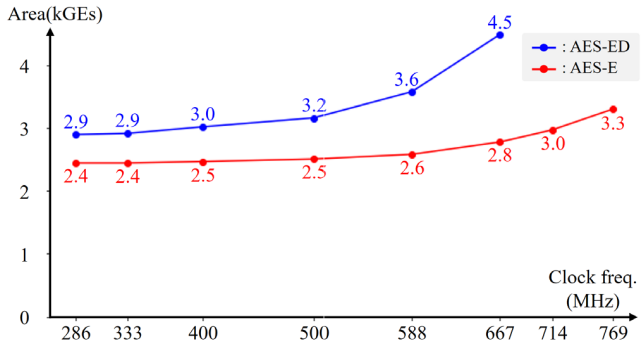



**FIGURE 6.** Implementation area of proposed AES accelerators.

We synthesized AES-ED and AES-E using 28-nm complementary metal-oxide semiconductor (CMOS) process technology, and the results are shown in Table 2 and Fig. 6. At a clock frequency of 333 MHz, AES-ED and AES-E require 2912 and 2442 gate equivalents (GEs), respectively. AES-ED is synthesizable at a maximum clock frequency of 667 MHz with 4481 GEs, and AES-E is synthesizable at a maximum clock frequency of 769 MHz with 3399 GEs.

### *B. COMPARISON WITH RESULTS OF OTHER STUDIES*

Small AES accelerators can be divided into two types according to their data paths of 32 and 8 bits as listed in Table 3. Satoh *et al.* [3] developed five versions of AES accelerators using logic optimization. The smallest version with four S-boxes and a 32-bit data path can encrypt a block within 54 CCs. For decryption, 10 more CCs are required to generate the initial decryption key. Moreover, its area is 5398 GEs at 131.24 MHz.

For smaller areas, other accelerators have one or two S-boxes with 8-bit data paths [12]–[17]. The AES accelerator proposed by Feldhofer *et al.* [12] had 3400 GEs, but the encryption required 1032 CCs. Although this number of CCs included the input/output operations, it was substantially slower than the accelerator proposed by Satoh *et al.* [3]. Despite its larger area, the AES accelerator proposed by Mathew *et al.* [13] provided a very higher throughput. Banik *et al.* [14] further reduced the number of CCs and area. They proposed two versions of AES accelerators. The first version had a latency of 226 CCs with 2605 GEs, and the second version was smaller, but required more CCs. To reduce the area, some accelerators do not support decryption [15]–[17], requiring only 1.5–2.6 kGEs.

The proposed AES accelerators have 32-bit data paths, which is similar to that proposed by Satoh *et al.* [3]. However, AES-ED is 46.0% smaller with almost the same number of CCs. Compared with the AES accelerators with 8-bit data paths [12]–[17], AES-ED is 11.8–30.8% larger than the smallest accelerator that supports both encryption and decryption [14], and AES-E is 67.6% larger than the accelerator that supports only encryption [17]. However, the proposed accelerators are much faster. Table 3 shows that the proposed AES accelerators have the highest throughput. Although we used a very small process technology, the required CCs were 3.0–22.0 times fewer than those of the accelerators

**TABLE 3. Performance comparison of AES accelerators.**

| | | Type[a] | Size of data path (bits) | Process technology (nm) | No. CCs | | Post-synthesis area (GEs) | Clock frequency (MHz) | Encryption throughput (Mbps) |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Enc. | Dec. | | | |
| Asiacrypt'01 [3] | | ED | 32 | 110 | 54 | 64 | 5398 | 131.24 | 311 |
| IET IS'05 [12] | | ED | 8 | 350 | 1032 | 1165 | 3400 | 80 | 9.9 |
| JSSC'15 [13] | | ED | 8 | 22 | 336 | 216 | 4037[b] | 1133 | 432 |
| JCE'19 [14] | | ED | 8 | 90 | 226 | | 2605 | 165.6 | 93.8 |
| | | | | | 246 | 326 | 2227 | 169.9 | 88.4 |
| Eurocrypt'11 [15] | | E | 8 | 180 | 226 | – | 2400 | 0.1 | 0.061 |
| EL'17 [16] | | E | 8 | 65 | 160<br>210 | – | 2600[b]<br>2400[b] | 130.9<br>127.2 | 104.7<br>77.5 |
| ISCAS'19 [17] | | E | 8 | 65 | 242 | – | 1457[b] | 200 | 105.8 |
| This work | AES-ED<br>AES-E | ED<br>E | 32 | 28 | 53 | | 2912<br>2442 | 333 | 805 |

[a]ED supports both encryption and decryption, and E supports only encryption
[b]Post-layout results

in [12]–[17]. In particular, the CCs required for data transfer are not included in Table 3 for most acclerators. As the proposed AES accelerators can directly access the SRAM, the proposed accelerators are more advantageous in terms of the encryption throughput including the data transmission time.

## V. FUTURE WORK

The proposed data interface method can be applied to other block ciphers and public-key cryptographies. In particular, it can be used to design a lightweight version of public-key cryptography coprocessors, which are resource-intensive for processing large numbers and performing complex computations. For instance, the proposed method can be applied in the following cases:

1) RSA and elliptic curve cryptography (ECC): RSA and ECC are widely used public key cryptosystems. Their main operation is modular multiplication of large numbers, such as 256-, 512-, and 1024-bit values. A lightweight coprocessor for RSA and ECC can be implemented by software/hardware co-design with a hardware multiplier. The hardware only multiplies large numbers, while the software controls the hardware multiplier and combines the results to compute the RSA and ECC operations. Although frequent data transfer to/from the hardware multiplier is time-consuming, most of the time required for data transfer can be reduced by using the proposed method.
2) Ideal lattice-based cryptography: Ideal lattice-based cryptography is popularly used in post-quantum cryptography and homomorphic encryption. The main operation is multiplication of polynomials, requiring a large memory capacity but simple computations. Choi *et al.* [18] demonstrated the implementation of ring Lizard, which is an ideal lattice-based cryptosystem and a candidate in round 2 of the Post-Quantum Cryptography Standardization project conducted by the National Institute of Standards and Technology. The coprocessor requires only a few adders and small registers for computation but a large memory capacity for storing large polynomials. By applying the proposed method, the coprocessor can be implemented without dedicated memory, thus significantly reducing the required resources. Similarly, the proposed method can be applicable to other ideal lattice-based cryptosystems, such as NTRU [19].

## VI. CONCLUSION

We proposed an AES accelerator with a novel data interface for accessing the SRAM. Instead of being attached to the bus with its own slave wrapper, the proposed AES accelerator was located within the wrapper of the SRAM and shared some space of the SRAM with the processor. This allowed the AES accelerator to directly access the SRAM and to use its space for storing the expanded key without requiring additional registers. As a result, we reduced the required resources for storing the key and the power consumed during key expansion, which is unnecessarily repeated for every block of encryption/decryption. The proposed AES accelerator can be used in resource- and power-constrained applications such as Internet-of-Things (IoT).

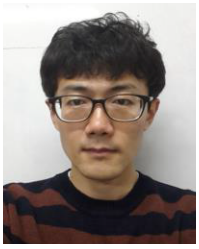
## ACKNOWLEDGMENT

*(Jae Seong Lee and Piljoo Choi contributed equally to this work.)* The chip fabrication and EDA tool were supported by the IC Design Education Center (IDEC), South Korea.

## REFERENCES

[1] *Advanced Encryption Standard (AES)*, document FIPS 197, National Institute of Standards and Technology, 2001.
[2] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, "Fast software AES encryption," in *Proc. FSE*, Seoul, South Korea, 2010, pp. 75–93.
[3] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Proc. ASIACRYPT*, Gold Coast, QLD, Australia, Dec. 2001, pp. 239–254.
[4] LPC18Sxx. *NXP*. Accessed: Feb. 4, 2022. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/LPC18S5X_S3X.pdf
[5] *Low-Power Features of SAM L Series Devices*. Accessed: Feb. 4, 2022. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Low-Power-Features-SAML-00002709A.pdf

[6] AES-IP-39. *Rambus*. Accessed: Feb. 4, 2022. [Online]. Available: https://www.rambus.com/security/crypto-accelerator-hardware-cores/basic-crypto-blocks/aes-ip-39/
[7] STM32F4 Series. *STM*. Accessed: Feb. 4, 2022. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f413-423.html
[8] MSP430FRxx. *Texas Instrument*. Accessed: Feb. 4, 2022. [Online]. Available: http://www.ti.com/lit/slau367
[9] D. Canright, "A very compact S-box for AES," in *Proc. CHES*, Edinburgh, U.K., Aug. 2005, pp. 441–445.
[10] *sbox.verilog*. Accessed: Feb. 4, 2022. [Online]. Available: https://github.com/coruus/canright-aes-sboxes/blob/master/verilog/sbox.verilog
[11] E. G. Ahmed, E. Shaaban, and M. Hashem, "Lightweight mix columns implementation for AES," in *Proc. ICAIC*, 2009, pp. 253–258.
[12] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES implementation on a grain of sand," in *Proc. IEE Inf. Secur.*, Oct. 2005, vol. 152, no. 1, pp. 13–20.
[13] S. Mathew, S. Satpathy, V. Suresh, M. Anders, H. Kaul, A. Agarwal, S. Hsu, G. Chen, and R. K. Krishnamurthy, "340 mv–1.1 v, 289 gbps/w, 2090-gate nanoaes hardware accelerator with area-optimized encrypt/decrypt gf (2 4) 2 polynomials in 22 nm tri-gate CMOS," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1048–1058, Apr. 2015.
[14] S. Banik, A. Bogdanov, and F. Regazzoni, "Compact circuits for combined AES encryption/decryption," *J. Cryptograph. Eng.*, vol. 9, no. 1, pp. 69–83, Apr. 2019.
[15] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of AES," in *Proc. Eurocrypt*, Tallinn, Estonia, May 2011, pp. 69–88.
[16] V. Hoang, V. Dao, and C. Pham, "Design of ultra low power AES encryption cores with silicon demonstration in SOTB CMOS process," *Electron. Lett.*, vol. 53, no. 23, pp. 1512–1514, Nov. 2017.
[17] A. Shreedhar, K.-S. Chong, N. K. Z. Lwin, N. A. Kyaw, L. Nalangilli, W. Shu, J. S. Chang, and B.-H. Gwee, "Low gate-count ultra-small area nano advanced encryption standard (AES) design," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.
[18] P. Choi, J.-H. Kim, and D. K. Kim, "Fast and power-analysis resistant ring lizard crypto-processor based on the sparse ternary property," *IEEE Access*, vol. 7, pp. 98684–98693, 2019.
[19] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory—ANTS* (Lecture Notes in Computer Science), vol. 1423. Berlin, Germany: Springer, 1998, pp. 267–288.



**JAE SEONG LEE** received the B.S. and M.S. degrees in electronic engineering from Hanyang University, Seoul, South Korea, in 2007 and 2010, respectively, where he is currently pursuing the Ph.D. degree in electronics and computer engineering. He has been working with the SoC Development Team, ICTK Holdings Company Ltd. His research interests include the areas of security SoCs, PUF, and crypto-coprocessors.

**PILJOO CHOI** (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronics and computer engineering from Hanyang University, Seoul, South Korea, in 2010, 2012, and 2017, respectively. From 2017 to 2019, he was a Software Education Professor with the Software Education Committee, Hanyang University. Since 2020, he has been an Assistant Professor with the Department of Computer Engineering, Pukyong National University, South Korea.

**DONG KYUE KIM** (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, in 1992, 1994, and 1999, respectively. From 1999 to 2005, he was an Assistant Professor with the Division of Computer Science and Engineering, Pusan National University. Since 2006, he has been a Professor with the Department of Electronic Engineering, Hanyang University.