International Journal of Engineering & Technology, 7 (3.24) (2018) 258-263

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Hardware Design of AES Core with High Throughput and Low Area

Alexander Owusu-Ansah Antwi, Kwangki Ryoo*

Department of Information and Communication Engineering, Hanbat National University, Daejeon 34158, Republic of Korea
Corresponding author E-mail: kkryoo@gmail.com

Abstract

Background/Objectives: The Advanced Encryption Standard is currently the most used algorithm for symmetric encryption. In this paper, we propose a hardware architecture of AES with an improved key generation unit.
Methods/Statistical analysis: We employ the use of a four-stage sub-pipelined architecture for encryption and decryption of all standard key sizes (128, 192 and 256 bits) of the Advanced Encryption Standard (AES). The implementation features an LUT-based S-Box as well as on-the-fly key generation. The RTL of the architecture was designed using Verilog HDL and simulated with ModelSim. The verified design was then synthesized in Synopsis Design Compiler with 180nm TSMC cell libraries.
Findings: Since the inception of AES, many implementations have been done in both software and hardware. For the purpose of robustness, the hardware implementation is much preferred. However, for area-constrained implementations, it is necessary for designers to present a very small area of the AES algorithm while keeping the AES structure and security unchanged. The proposed compact key generation unit contributed to the small area of 21.3K equivalent NAND2 gates. The S-Box was implemented as a ROM of size 9.152KB. In order to match the encryption/decryption, the on-the-fly also key generation was accordingly made to output round keys every four cycles. With this structure, there was a high average throughput yield of 11.51Gb/s, 9.75Gb/s and 8.46Gb/s for the 128-bit, 192-bit and 256-bit key lengths respectively, corresponding to a maximum frequency of 1GHz.
Improvements/Applications: In the future, we will investigate more techniques to reduce the area of the S-Box and Mix Column structures. We will implement the design on an SoC system for verification and testing.

***Keywords*:** On-the-fly key generation, ASIC, Encryption, Decryption, CMOS, Sub-pipelined architecture

1. Introduction

In recent times, information and the demand for it have increased due to technological advancements. From the start of the millennium up until now, there has been a steady rise in the demand and use of information due to improved banking, medical care, research, and many others. There are millions of smart mobile devices that facilitate in these areas of work and living. As part of making life easy for its users, these smart devices transmit sensitive information such as personal health diagnosis, daily routines, passwords and the like through the open internet. In fact, a smart home in today's world may be seen as an evolution in technology, but in reality, it poses a huge security risk, since the smart devices in the home transmit data between themselves as well as over the insecure internet. In effect, the whole house can be controlled remotely and thus compromising sensitive information. There have been a lot of breaches in the security of government and private businesses as well as individuals worldwide [1]. There is therefore, the need to regularly protect data both in storage and transmission from falling into the hands of these malicious hackers.

Two of the fields that are called into play when considering user data protection are cryptography and cryptanalysis. They involve the study of encryption, decryption, and techniques to counter activities by system intruders. Cryptography can further be classified into symmetric and asymmetric cryptography. In symmetric cryptographic systems, a single key is used for both encryption and decryption. In asymmetric cryptographic systems, however, a different key is used in both encryption and decryption. Examples of the symmetric systems include the Advanced Encryption Standard (AES) and the Data Encryption Standard (DES) while the asymmetric systems find examples in the Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) [2].

Since its inception in 2001, the AES algorithm has been very robust and is widely used in many systems such as smart cards, mobile phones, USB keys and banking systems [3]. The AES algorithm was designed to be implemented in both software and hardware. Although software-based encryption is very cost-effective, it is not as robust as hardware-based encryption. Unlike software which can be easily exploited even remotely, an intruder will need physical access to the hardware before he can do some serious damage to the system [4,5,6]. It is the reason why there have been various researches into hardware implementations of AES since its standardization. Researchers' main focus has been on optimization for area and throughput. Different design methods have been adopted in the research into these areas which are, iterative and unrolled architecture as well as sub-pipelined and fully pipelined architectures. There are two main hardware platforms which are considered when porting algorithms to hardware: Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). FPGAs have been a choice of implementation for many as compared to ASICs since the former are cheaper and they can be reprogrammed several times with several designs. ASICs do not have this

advantage – a design on an ASIC is not updatable. However, ASICs have the advantage of design flexibility and strong tamper resistance which is needed in security systems as well as area constrained applications. Also, there exist delays in FPGA circuitry due to its programmability which greatly affects the speed of designs. In ASIC, the designer has a 'blank sheet' to design on, where he does not have any limitations by means of area and of speed (frequency). As a result of this, a circuit implemented on an FPGA is naturally slower than on ASIC [5]. Due to these and many other bottlenecks in FPGAs, we find it proper to implement our AES design in ASIC.

The basic structure of AES consists of three modules; AddRoundKey, SubByte, ShiftRow and MixColumn and the key generation unit [7,8,9]. In the design of these modules, we can choose between using an on-the-fly key generation [10] or compute all the keys and store in memory to be retrieved when needed, at the end of the corresponding rounds [2]. There is also the option of implementing the S-Box as an LUT-based structure or as a composite field arithmetic combinational logic. When implemented this way, the critical path increases, thereby yielding a low frequency. In this paper, we propose a high throughput and small area AES with a compact on-the-fly key generation unit. The S-Box was implemented as a ROM unit.

2. Overview of the AES algorithm

The AES algorithm consists of three main parts – the Cipher, Inverse Cipher, and key generation unit. The Cipher module converts input text into gibberish by means of running four main sub-layers (AddRoundKey, SubByte, ShiftRow, and MixColumn) for a number of rounds. Inverse Cipher converts the unintelligible data back into the plaintext before encryption. It is made up of the inverse layers of the Cipher module with the exception of the AddRoundKey layer. Each of these layers is iterated upon for a number of rounds depending on the size of the encryption/decryption key [11,12,13]. Table 1 shows the different key sizes and the number of rounds required for each. The key generation unit takes as seed, an N-bit key (N is either 128, 192 or 256) and creates corresponding round keys for each round.

At the start of the encryption process, we XOR the input key with the input text. A state in AES is the 128-bit text that undergoes transformation to become the cipher. In order to have a good pictorial view of the AES state and operations, we represent it with a 4x4 matrix [10] as shown in figure 1. From the initial AddRoundKey module, the state proceeds to the SubByte stage and then to the ShiftRow to MixColumn. Finally, it enters the AddRoundKey stage where it is XORed with the round key. The process continues until the last stage where the MixColumn layer is not performed. In decrypting the cipher text, we do the opposite of what we did in the encryption. This means that the MixColumn layer is not performed in the first round. The ShiftRow and SubByte operations are swapped while the MixColumn and AddRoundKey stages are also swapped. The standard architecture of AES Cipher and Inverse Cipher are shown in figure 2. The following sub-sections give a summary of each of the sub-layers.

Table 1: AES rounds and sub-key lengths

	Number of Rounds	Sub-key Width	Number of Sub-keys	Total Number of Round Keys
AES-128	10	4 x 32	11	11
AES-192	12	6 x 32	8	13
AES-256	14	8 x 32	7	15

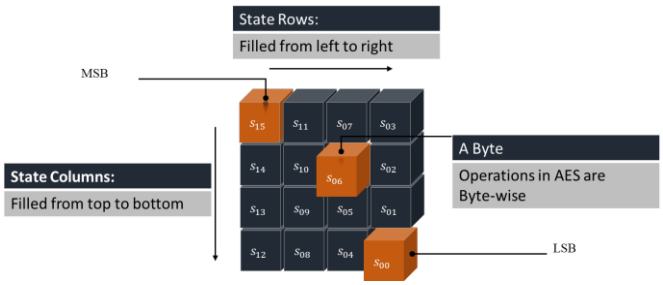

Figure 1: A state in AES.

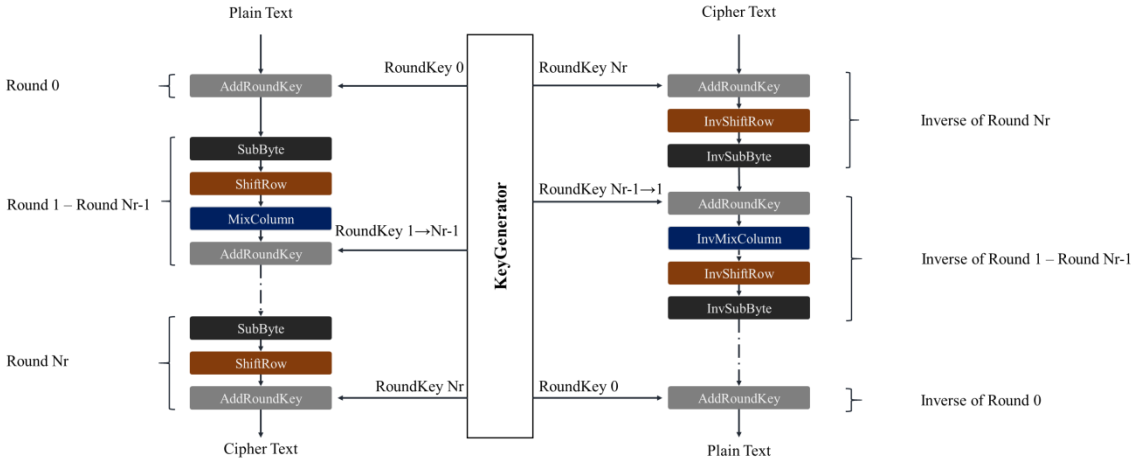

Figure 2: Standard AES architecture for encryption and decryption

2.1. SubByte/InvSubByte

The SubByte/InvSubByte layer is a non-linear transformation of the state. It can be implemented as an LUT-based unit, (S-Box for encryption, or Inv S-Box for decryption) or as a composite field arithmetic combinational logic unit. With the combinational logic unit, Galois fields (GF) are employed in computing the substitute of each byte in a state. The process is described in [3]. As a result of the composite logic structure yielding a long critical path, the S-Box and Inv S-Box have both been implemented with LUT-based logic in our design. There are sixteen identical S-Box/Inv S-Box modules in the Cipher/Inverse Cipher data path, and four identical S-Box modules in the key generation unit. In ASIC implementation, the S-Box is synthesized as a ROM unit and as BRAM in FPGAs.

2.2. ShiftRow/InvShiftRow

This operation is a free operation as it just involved byte rotation. Here, the rows of the State are rotated depending on the row number. The shifting is illustrated in figure 3.

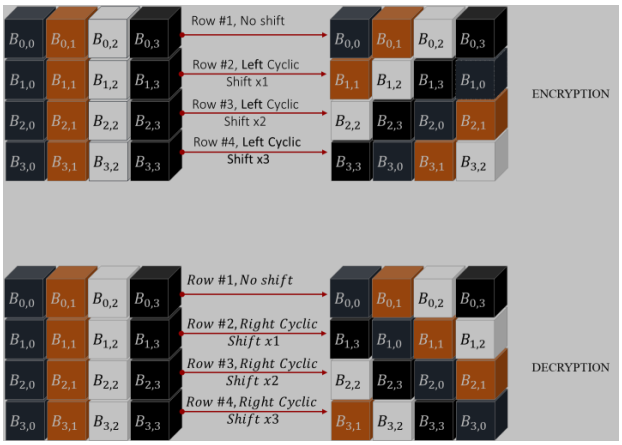

Figure 3. Structure of ShiftRow/InvShiftRow operations

It can be seen from figure 3 that rotating two bytes to the left is the same as rotating two bytes to the right in row 3 of both matrices. Also, from figure 3, in both encryption and decryption, row 1 is unchanged. Although the ShiftRow operation is of no cost itself, the output registers occupy some area in the design. With the knowledge of common operations, we can implement a shared ShiftRow module in order to save area by using lesser registers.

2.3. MixColumn/InvMixColumn

Each column of the State is multiplied by a fixed matrix. This matrix is unique to each of the encryption and decryption processes. The decryption step is more computationally intensive than that of the encryption. The standard MixColumn and InvMixColumn are computed in [7,8,9]. In hardware, multiplication by 02 or its multiple is nothing but a left-shift operation. By identifying and using this, we can avoid the use of multipliers, and hence reduce our area and critical path. The multiplication is thus shifting and XOR operations as depicted by equation (1) and table 2.

$$
\begin{aligned}
&Ai \cdot 01 = Ai \\
&if\ (MSB(Ai) = 1) \\
&\quad Ai \cdot 02 = (Ai \ll 1)\ \hat{}\ 8'b00011011 \\
&else \\
&\quad Ai \cdot 02 = Ai \ll 1 \\
&Ai \cdot 03 = (Ai \cdot 02)\ \hat{}\ Ai
\end{aligned}
\tag{1}
$$

Table 2: Multiplication in GF (2^8) for the MixColumn structure

$03 \cdot A_i$	$(A_i \cdot 02) + A_i$
$04 \cdot A_i$	$02 \cdot (A_i \cdot 02)$
$08 \cdot A_i$	$02 \cdot (A_i \cdot 04)$
$09 \cdot A_i$	$(A_i \cdot 08) + A_i$
$0B \cdot A_i$	$(A_i \cdot 09) + (A_i \cdot 02)$
$0D \cdot A_i$	$(A_i \cdot 09) + (A_i \cdot 04)$
$0E \cdot A_i$	$(A_i \cdot 08) + (A_i \cdot 04) + (A_i \cdot 02)$

We implemented our MixColumn module by taking advantage of parallel processing in hardware. This method contributes to the high frequency due to the reduced critical path.

2.4. AddRoundKey

In this module, we add the 128-bit state to the round key. At the beginning of the encryption/decryption process, the state (plaintext/ciphertext) is first XORed with the input key. This layer is its own inverse since the XOR function is its own inverse.

3. Proposed Hardware Structure

In our proposed architecture, the AES algorithm was implemented as a four-stage sub-pipelined architecture. In each clock cycle, one layer of a round is computed. Unlike the row shifting and column mixing modules that were implemented as shared structures, the byte substitution modules were not. The full architecture is shown in figure 4. Choosing this style of implementation reduces the critical path of the S-Box. In table 3, the 4-stage pipeline structure is illustrated.

Table 3: Four-stage pipeline structure for encryption and decryption

ENCRYPTION						
	T0	T1	T2	T3	T4	T5
PlainText1	AddRoundKey	SubByte	ShiftRow	MixColumn	AddRoundKey	SubByte
PlainText2		AddRoundKey	SubByte	ShiftRow	MixColumn	AddRoundKey
PlainText3			AddRoundKey	SubByte	ShiftRow	MixColumn
PlainText4				AddRoundKey	SubByte	ShiftRow
DECRYPTION						
CipherText1	AddRoundKey	InvShiftRow	InvSubByte	AddRoundKey	InvMixColumn	InvShiftRow
CipherText2		AddRoundKey	InvShiftRow	InvSubByte	AddRoundKey	InvMixColumn
CipherText3			AddRoundKey	InvShiftRow	InvSubByte	AddRoundKey
CipherText4				AddRoundKey	InvShiftRow	InvSubByte

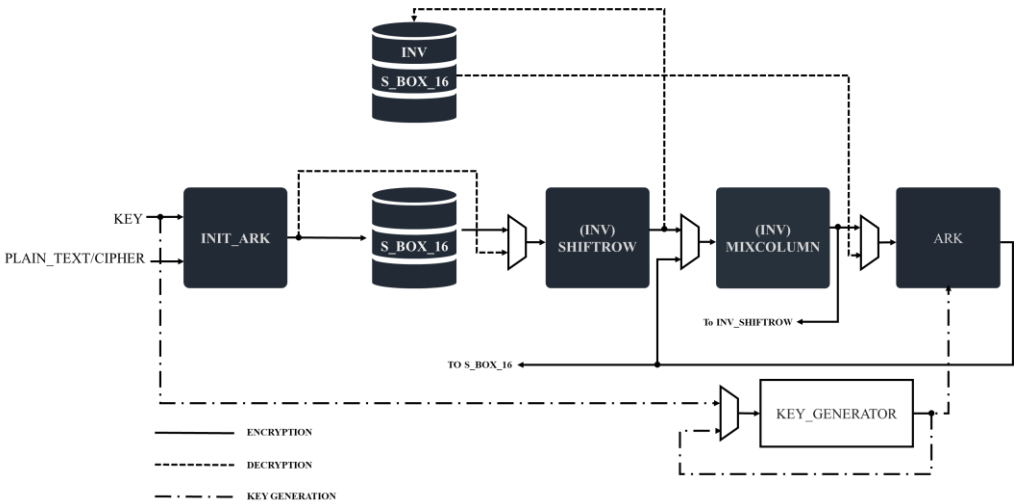

Figure 4: Proposed shared architecture for encryption and decryption

3.1. Key Generation

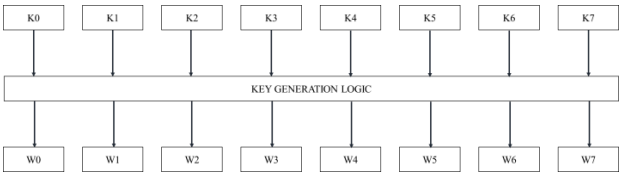

Figure 5: Overview of the key generation showing the word divisions

Figure 5 shows the overview of the standard key generation unit. It consists basically, of the XOR, RotWord and SubByte operations. The RotWord function rotates its input to the left by a byte. The key (or sub-key) is divided into 4, 6 or 8 words each of size 32 bits for the 128-bit, 192-bit, and 256-bit keys respectively. The calculation of the leftmost word for the 128-bit key is shown in equation (2).

$$W[i] = W[4(i-1)] + g([W[4i-1]) \quad (2)$$

W represents a 32-bit word of the sub-key, where g is a special function that consists of S-Box, RotWord, and AddRoundConstant. The remaining words of the sub-key are calculated by equation (3).

$$W[4i + \mathrm{j}] = W[4i + j - 1] + W[4(i-1) + j] \quad (3)$$

Where $i = 1,2 \ldots,10$ and $\mathrm{j} = 1,2,3$. The calculation for the 192-bit and 256-bit key modes is similar to that of the 128-bit keys. However, the limit of the integers i and j vary for each of the key sizes. The number of iterations in the key generation unit is different for each key size; there are 10, 8 and 7 iterations for 128-bit, 192-bit, and 256-bit key modes respectively. These iterations do not correspond to the number of sub-keys.

The proposed architecture of the key generation unit showing the main computation circuit can be seen in figure 6. The unit utilizes only three 32-bit XORs in the calculation of the sub-words. OUT1 and OUT2 are computed each cycle and stored in a buffer, BUFF. After four cycles while operating in the 128-bit key mode, the key generation unit will output one round key, and, two round keys when operating in the 256-bit key mode. In the 192-bit key mode, the key generation unit will compute more than one round key in four cycles. There will be a remainder of 64 bits which will have to be carried into the next round. Consequently, we need to compute the sub-key for a new 6-word input key (192 bits). Meaning, for the 192-bit mode, two input keys will give rise to three round keys. The details are shown in table 4. For AES-256, another function, $h(x)$, exists aside $g(x)$ which is basically an S-Box of the 32-bit word, W_4. In hardware, we can implement resource sharing if a number of resources are used by different modules at different cycles or conditions. Since the $g(x)$ and $h(x)$ functions [8] have S-Box in common and are computed in different cycles, we implemented them as a shared module. The shared hardware structure, SRRC is shown in figure 8. MD selects the output between $h(x)$ and that from $g(x)$. Figure 7 shows the hardware structure of our proposed compact key generation unit. The full key generation of all the key modes is well illustrated in [7,8,9].

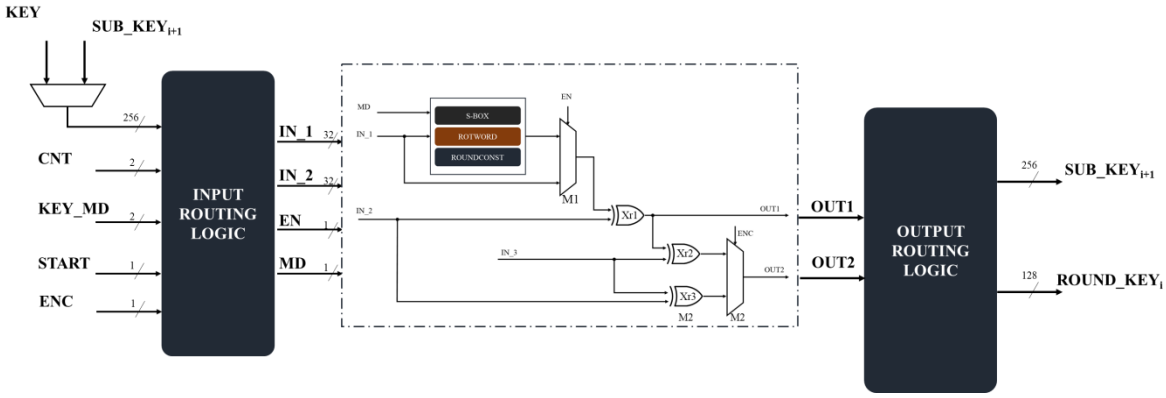

Figure 6: Proposed key generation unit architecture

3.1.1. Input Routing Logic

For every clock cycle, the combination, {CNT, KEY_MD, START} selects the three input words, IN_1, IN_2, and IN_3 to the Key generation unit based on equations (2) and (3). EN selects between IN_2 and the output of the SRRC module. For AES-128 and AES-192, EN is high for the first cycle and low in the next cycles for each sub-key computation of the left-most word. For AES-256, it is high in the first and fifth cycles because we need to compute $g(W_3)$ as well as $h(W_4)$.

3.1.2. Proposed Key Generation Unit

Unlike the standard AES which utilizes 7 XORs, the proposed Key generation unit circuit has only three for the same operation. The key generation unit also consists of the SRRC module which has an S-Box, RotWord and an 8-bit RoundConstant addition unit. The 32-bit input into the SRRC module is first passed through the S-Box and then rotated a byte to the left. The most significant byte of the byte rotated word is XORed with a round constant. The round constants correspond to the sub-key numbers (not round keys) for each key mode. A sub-key of a particular key mode has the same width as the input key for that mode (128 bits, 192 bits or 256 bits). A round key, on the other hand, is exactly 128 bits in length and is what is added to the state in the AddRoundKey module. Figure 8 shows the shared module SRRC.M1 selects between the output of SRRC and IN_1. M2 selects between the output of Xr2 and Xr3. For encryption, the output of Xr2 is selected, whereas, for decryption, that of Xr3 is selected. OUT1 and OUT2 are pushed into a 256-bit buffer, BUFF. The number of bits of the sub-key has to match the 128-bit round key. For the 128-bit key mode, the size of a round key is the same as one sub-key and can be calculated in two cycles and stored in BUFF. For the 256-bit key mode, the computation of one sub-key generates two round keys. The 192-bit key mode generates a result that is the size of a round key (128 bits) plus a remainder of 64 bits. The remaining 64 bits are stored in a temporal register, TMP_REG. The next sub-key will also generate a round key and a half. Hence, grouping these two we have three round keys matched with two sub-keys. The details are given in sub-section 3.1.4.

3.1.3. S-Box, Rotword, and Roundconstant (SRRC)

This module is used in determining the leftmost word, W_0 of the sub-keys. For the 256-bit key mode, it is used again in determining the fifth sub-word, W_4. This is well illustrated in figure 7.

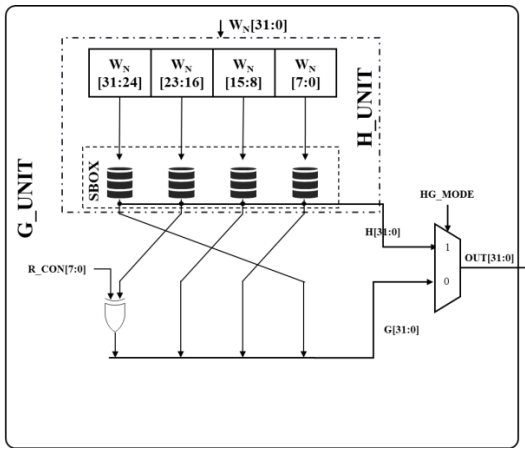

Figure 7: Structure of proposed SRRC module

3.1.4. Output Routing Logic

At the end of four cycles, the output routing has to decide which bit-select of BUFF will be passed as the key. We have only shown the output logic for Key generation unit during encryption since there is little variation in the decryption process. From figure 8, the output logic of the Key generation unit is shown. The output logic's main function is to select the correct 128 bits as the round key and also to feedback the generated sub-key into the input routing logic as the next key input. For the AES-128 key mode, the output RND_KEY is always the top 128 bits of BUFF. For the 256-bit key mode, it was earlier stated that one sub-key generates two round keys. For this reason, when FETCH_CNT is zero, the top 128 bits are selected as round key and a wait state is asserted that sets the START signal to the Key generation unit to zero. When FETCH_CNT is one, the bottom 128 bits are selected as the round key and the START signal is asserted. For the 192-bit key mode operation, FETCH_CNT has three possible values (0,1,2). Table 4 shows the selection of round keys as well as the assertion of START for the 192-bit key mode.

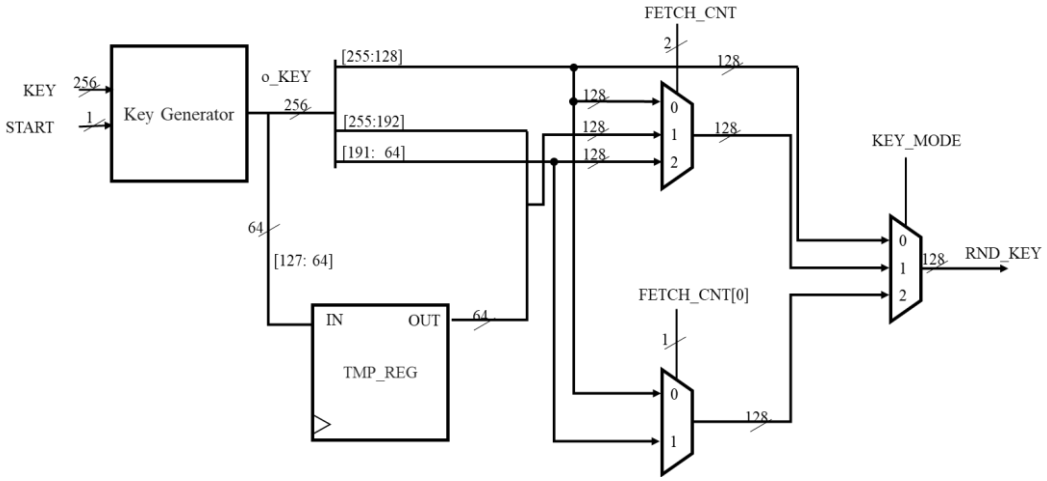

Figure 8: Output routing logic for key generation

Table 4: Round key selection for 192-bit key

ENCRYPTION			
FETCH_CNT	START	RNDKEY	TMP_KEY
0	1	$BUFF_1$[255:128]	$BUFF_1$[127:64]
1	0	{$BUFF_2$[255:192], TMP_REG}	-
2	1	$BUFF_2$[191: 64]	-
DECRYPTION			
0	1	$BUFF_1$[191: 64]	$BUFF_1$[255:192]
1	0	{$BUFF_2$[127:64], TMP_REG}	-
2	1	$BUFF_2$[255:128]	-

3.2. Proposed Mixcolumn Common Unit

Figure 9 shows our proposed structure for the MixColumn layer. We created a common unit in which each matrix multiplication is done for both encryption and decryption. The matrix computations involve left shift and XOR operations as described in table 2 above. There are four common units, one for each byte of the 32-bit input vector, A. After the multiplication is done, the next step is to route and XOR the corresponding outputs from the common unit. The results are then assigned to the 32-bit output vector B.

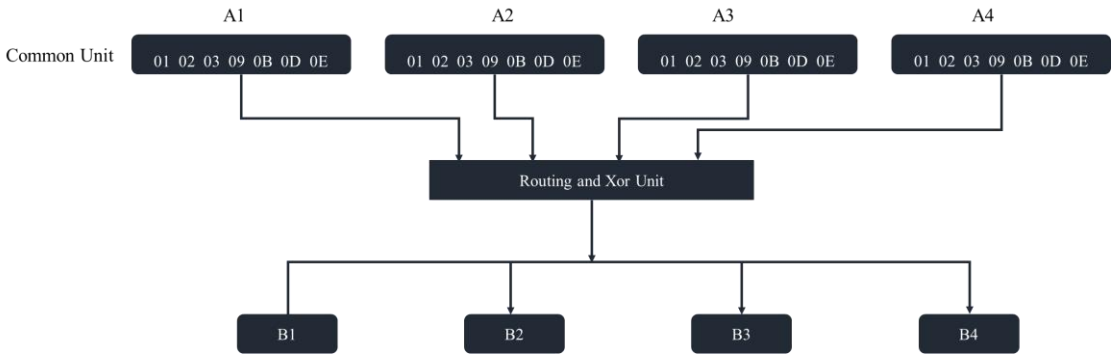

Figure 9: Overview of proposed mixcolumn/invmixcolumn structure

4. Implementation Results

Table 5 shows the comparison of our module with various 180nm implementations of AES. In all the implementations, efforts were made to improve on throughput and area by means of one or more of the various hardware implementation pathways of AES to choose from. Usually, a pipelined hardware has a larger area as

compared to a non-pipelined one. By using a rolled architecture, [2] gained an average throughput of 1.6Gbps. [3] adopted an on-the-fly key generation scheme and composite S-Box structure that caused a reduction in the area, but cost them in terms of the critical path. Proposed_1 was synthesized for a maximum frequency of 300MHz in order to compare to that of [10] since their architecture was close to ours (three stage pipeline) and to show that ours outperforms in terms of area and throughput. Our design showed dominance in terms of area and throughput. The remainder of the results is shown in table 5.

Table 5: Results and comparisons for 180nm CMOS process

		[2]	[3]	[6]	[10]	Proposed_1	Proposed_2
Frequency (MHz)		125	100	300	300	300	1000
Throughput (Gbps)	128	1.6	1.16	10.656	3.84	3.84	11.52
	192		0.99		3.199	3.2	9.68
	256		0.85		2.743	2.648	8.124
Gate Count (KGates)		58.445	19.5	-	39.98	20.51	21.63
Memory(S-Box) [Bytes]		-	-	-	-	9.152K	

5. Conclusion

In this paper, we proposed a four-stage sub-pipelined architecture of AES with a compact on-the-fly key generation. There are two ways of implementing the SubByte currently: as a composite logic or a look-up table. The composite field arithmetic implementation of the S-Box increases the critical path and hence reduces the throughput unless we use sub-pipelining, which then increases the area due to additional registers. We chose to design the S-Box as an LUT-based structure which allows for one-time access and reduces critical path. We implemented a compact version of the key generation that caused a reduction in area. The adopted four-stage sub-pipeline structure causes an increase in the throughput. The compact Key generation unit was proposed, which utilizes three 32-bit XORs compared to the traditional AES which utilizes 7 XORs. A common unit for the MixColumn was also proposed to help reduce the critical path by means of parallel computation. The proposed AES was designed with Verilog HDL and synthesized with Synopsys Design Compiler using a 180nm CMOS cell library. The total operating frequency peaked at 1GHz, giving rise to an average throughput of 11.51Gbps, 9.75Gbps, and 8.46Gbps for AES-128, AES-192, and AES-256 respectively. The core area was 21.63 equivalent NAND2 gates and 9.152KB of memory

Acknowledgment

This research was supported by the MSI (Ministry of Science, ICT and Future Planning), Korea, under the Global IT Talent support program (IITP-2017-0-01681) supervised by the IITP (Institute for Information and Communication Technology Promotion).

References

[1] CSO online. The 17 biggest data breaches of the 21st century. https://www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html. Revised January 2. Accessed September 2, 2018.

[2] Shastry PVS, Kulkarni A & Sutaone MS (2012), ASIC implementation of AES. *Proceedings of the 2012 Annual IEEE India Conference (INDICON)* 1255-1259.

[3] Cao Q & Li S (2009), A high-throughput cost-effective ASIC implementation of the AES Algorithm. *Proceedings of the 2009 IEEE 8th International Conference on ASIC* 805-808.

[4] Tales from the Crypt: Hardware vs Software [Internet]. Infosecurity group. 2015. [updated 2015 June 23; cited 2018 Aug 31] Available from: https://www.infosecurity-magazine.com/magazine-features/tales-crypt-hardware-software

[5] Gaj K & Chodowiec P (2009) Cryptography Engineering: FPGA and ASIC Implementations of AES. Boston: Springer US;

[6] Saravanan P, Devi RN, Swathi G & Kalpana P (2011), A High-Throughput ASIC implementation of Configurable Advanced Encryption Standard (AES) Processor. *International Journal of Computer Applications (IJCA)* 3, 1-6.

[7] Advanced Encryption Standard. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard. Revised August 25. Accessed September 2, 2018.

[8] Rijmen V & Daemen, J (2002) The design of Rijndael: AES-The Advance Encryption Standard. Berlin: Springer-Verlag.

[9] Paar C & Pelzl J (2010) Understanding Cryptography. Berlin Heidelberg: Springer-Verlag.

[10] Li H (2006), Efficient and flexible architecture for AES. *IEE Proceedings - Circuits, Devices and Systems* 153, 533-538.

[11] Dao VL, Nguyen AT, Hoang VP& Tran TA (2015), An ASIC implementation of low area AES encryption core for wireless networks. *Proceedings of the 2015 International Conference on Communications, Management and Telecommunications (ComManTel)* 99-102.

[12] López RL, García ML & Navarro EC (2018), Hardware Architecture Implemented on FPGA for Protecting Cryptographic Keys against Side-Channel Attacks. *IEEE Transactions on Dependable and Secure Computing* 15, 898-905.

[13] Kalaiselvi K & Mangalam H (2015) Power efficient and high-performance VLSI architecture for AES algorithm. *Journal of Electrical Systems and Information Technology* 2, 178-183.