SPEED EFFICIENT HARDWARE IMPLEMENTATION OF ADVANCED ENCRYPTION STANDARD (AES)

By

LOW CHIAU THIAN

A Dissertation submitted for partial fulfilment of the requirement for the degree of Master of Science (Microelectronic Engineering)

August 2017

ACKNOWLEDGEMENTS

First of all, I would to express my greatest gratitude to my supervisor, Dr. Bakhtiar Affendi Rosdi. His willingness of giving his time generously to supervise me with expert comments and guidance has been very much appreciated. If it wasn't his guidance and encouragement throughout this research period, I wouldn't had complete my thesis in time successfully.

Besides, I would like to thank my family and friends for spending their hectic moment to accompany me throughout the thesis. Their advices and opinions given have proven to be helpful. Without their assistance, this thesis will not have been as successful as it is.

Last but not least, I would like to express my greatest appreciation to Intel Penang Company for providing me the study sponsorship for this Microelectronic Engineering course. Without their financial support, it wouldn't be me to pursue my Master degree in School of Electrical and Electronic Engineering Universiti Sains Malaysia.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Pelaksanaan Perkakasan Untuk Piawai Penyulitan Lanjutan Dengan Kelajuan Yang Berkesan

ABSTRAK

Kriptografi memainkan peranan yang penting dalam keselamatan data terhadap serangan dari pihak ketiga. Dalam tesis ini, tumpuan adalah untuk melaksanakan algoritma kriptografi yang biasa digunakan seperti Piawai Penyulitan Lanjutan (AES) dan meningkatkan prestasi kelajuannya. Motivasi adalah untuk membuat proses penyulitan semakin pendek untuk meningkatkan keupayaan sistem dalam pemprosesan jumlah data yang besar. FPGA dipilih sebagai platform kerana ia tidak mempunyai overhed perisian dan sesuai untuk aplikasi masa nyata. Kebanyakan kajian mengoptimumkan sumber perkakasan untuk melaksanakan AES di FPGA dan cara-cara termasuklah pengiraan aktif dan gelung bergulir seni bina yang mengurangkan sistem kelajuan. Tesis ini adalah untuk membentangkan reka bentuk pemprosesan yang tinggi dalam 128-bit AES algoritma dengan menggunakan gelung membuka, seni bina saluran maklumat dan pendekatan LUT untuk bekerja secara selari dalam penyegerakan yang tepat untuk memenuhi keperluan aplikasi masa nyata. Reka bentuk sistem dikodkan daripada Verilog HDL di dalam ModelSim dan reka bentuk perkakasan dianalisa melalui Altera Cyclone II daripada Quartus II. Proses penyulitan boleh dicapaikan dalam daya pemprosesan maksimum 32 Gbits/s beroperasi dengan kekerapan 250 MHz. Selain itu, penyulitan AES 128-bit yang beroperasi dalam satu kitaran penuh hanya memerlukan 41 kitar jam untuk mendapatkan data yang disulitkan. Perbandingan dengan kerja-kerja berkaitan dilakukan dan akhirnya mencapai penghantaran lebih tinggi daripada kerja-kerja berkaitan dengan 3.47% dan 22% masing-masing. Dua objektif yang ditetapkan dalam tesis ini telah dicapai.

Speed Efficient Hardware Implementation Of Advanced Encryption Standard (AES)

ABSTRACT

Cryptography plays a vital role in data security against the attacks from the third party. In this thesis, the focus is to leverage existing, commonly used cryptography algorithm which is the Advanced Encryption Standard (AES) and improve its speed performance. The motivation is to make encryption process as short as possible to aid in increasing a system's ability to process large amount of data. FPGA is chosen as the platform due to it does not have software overhead and is meant to be customized for real time applications. Most of the researches are done on the area of optimizing hardware resources to implement AES on FPGA. The methods of optimization include on the fly computations and looping architecture, where all these of methods reduce the speed. This thesis presents a high throughput design of the 128-bit AES algorithm using loop unrolling, pipelined architecture and LUT approach which is able to work in parallel to allow accurate synchronization in order to fulfill the real time application needs. The system design is coded using Verilog HDL in ModelSim and the hardware design is analyzed through Altera Cyclone II in Quartus II. The maximum throughput of 32 Gbits/s operating at 250 MHz for the encryption process can be achieved. Also, one full cycle of a 128-bit AES encryption only needs 41 clock cycles in order to get the encrypted data. The comparison with the related works is done and eventually achieved higher throughput than the related works by 3.47% and 22% respectively. The two objectives set in this thesis are achieved.

CHAPTER 1

INTRODUCTION

1.1 Background

In this world of technology, privacy and secrecy are the vital aspects to be contemplated due to the explosive growth of internet. Cryptography is the security mechanism that protect and prevent the data from leaking to the public access. Cryptography is known as “secret-writing” in Greek origin word which is meant to make the data immune to attacks and keep them safe and secure. In the old days, cryptography had been used as a communication medium between two people for their top secret. Nowadays, the purpose of cryptography has been changed based on the high technology demand of the people into an algorithm (Joshi, 2015). Cryptography is the practice of secret writing for secure communication with the conversion of plain text form into cipher text form, where only desired entity can retrieves information. On the other way, the cipher text can be transformed into plain text only if the receiver has the correct cipher key (Talha et al., 2016).

There are two processes in cryptography algorithm which are encryption and decryption. Encryption is the process of encoding or converting data into form of code in order to protect users from being easily hacked by third parties. The data, which is the original plain text is converted into a cipher text, which is the unreadable text through encryption. On the other hand, decryption is a process which reverts the cipher text back to original plain text (Joshi, 2015). Both of encryption and decryption algorithms need either only one key or more than one key to complete the whole process. The two main

encryption algorithms include symmetric key algorithm and asymmetric key algorithm. Symmetric key algorithm uses only one key for the encryption and decryption process, thus it is called as private key algorithm. For asymmetric key algorithm, it uses two keys which one for the encryption process and another one for the decryption process. Therefore, asymmetric key algorithm is named as public key algorithm. The symmetric key algorithm (private key algorithm) is easy to implement and less complex when compared to asymmetric key algorithm (public key algorithm). Besides, symmetric key algorithm is fast when compared to asymmetric key algorithm as it uses only one key. Symmetric encryption uses fewer Computer Processer Unit (CPU) cycles than asymmetric encryption, hence symmetric algorithms are better than asymmetric algorithms in speed. Examples of symmetric algorithms are Data Encryption Standard (DES) and Advanced Encryption Standard (AES), where one of the example of asymmetric algorithm is Remote Secure Access (Priya et al., 2015). In this thesis, decryption will be not covered and will focus only on encryption.

Symmetric-key cryptography is known as the most instinctive of cryptography, which involves the utilization of one secret key only by the participants to perform encryption and decryption of data during the secure communication (Priya et al., 2015). Data Encryption Standard (DES) uses a key of 56 bits and converts a 64 bits of input block plain text into a 64 bits of output block cipher text during encryption. The key size of 56 bits is one of the most contentious aspects of DES algorithm as it is relatively small to cope with brute-force attack coming from modern computers (Deshpande et al., 2014). Triple DES, known as 3DES, which uses larger keys can generate better security. However, 3DES algorithm is relatively poor in software because DES was originally

practical in hardware implementation only (Oukili et al., 2016). Advanced Encryption Standard (AES) has better security than 3DES as it has symmetric block cipher with a 128 bits block size which hold up 3 different key lengths of 128 bits, 192 bits and 256 bits (Kaur et al., 2013). This allows AES algorithm to replace its forerunner DES algorithm as it supports better security, efficiency and adaptability with a larger key size. Key size is vital as it determines the security strength and power consumption (Deshpande et al., 2014). Hence, AES algorithm is chosen for the encryption purpose, as private key algorithm provides better in speed which is the main target to tackle on real time application in this thesis.

In this thesis, the target specification of a speed efficient encryption system need to be have high throughput to encrypt volume data in a shorter period in order to protect the real-time applications from being accessed by other people. Besides that, encryption has been widely used in many different real-time applications like cellular networks, ATM cards, online banking, wireless communications, electronic commerce and military communications as it provides information security in terms of confidentiality and integrity during communication. Therefore, there is a need to design an efficient and fast processing speed of 128-bit AES algorithm with pipelining hardware architecture to improve the data security of the real time application nowadays.

1.2 Problem Statements

AES encryption is known as an efficient platform for both software and hardware implementation (Ghewari et al., 2010). If AES algorithm is implemented at software level, the output will shows slower result when compared to hardware implementation that can execute the operation in parallel (Hongsongkiat et al., 2014). Moreover, hardware implementation of AES provides better performance and efficiency to the system in terms of security than software implementation. This is because software implementation of AES encryption often requires continuous updates in order to keep up with the hacking techniques (Kanguru, 2017).

Besides, software implementation of AES takes longer processing time than hardware. Hence, software implementation of AES unable to provide full proved security and creates time consuming system for financial industry and instance business. The only advantage of using software implementation is to save cost and chip area as no additional hardware is required and the program already resides in that of the instruction memory (Hongsongkiat et al., 2014).

Previously, there are many different proposed architectures on the size of the hardware implementation of AES algorithm. Mostly, people are concerned about how to reduce the hardware used to save cost, but this might in the same way decrease the speed of the application. In this thesis, high throughput is the main concern as real time application need fast processing speed to deliver data to the users which in the same time data need to be protected well by encryption process. There are few types of methods in order to achieve higher throughput by improving the encryption process which are

pipelining (Joshi, 2015), inner round pipelining (Thongkhome et al., 2011) and outer round pipelining (Rahimunnisa et al., 2013).

Regarding to (Tay et al., 2014), the area of the hardware used can be reduced by optimizing the AES S-Box in Sub Bytes, which is one of the most resources consuming in AES encryption. However, this type of architecture will decrease the processing speed of AES encryption because it takes more clock cycles to perform the substitution operations by reducing the S-Box numbers. Besides, reduction of pipeline registers is encouraged by (Alaoui, 2010) as claimed that pipeline registers are burden on hardware requirement. By the way, this is not suitable for high speed application as it will results in slower processing time.

1.3 Research Objectives

- To increase the processing speed of 128-bit AES algorithm by proposing a new hardware architecture.
- To evaluate the performance of the proposed architecture.

1.4 Research Scope

This thesis is mainly covered on the speed optimization of hardware implementation of AES encryption algorithm only by pipelined architecture. There will be no decryption algorithms as the Advanced Encryption Standard Algorithm Validation Suite (AESAVS) can be used to provide testing in order to determine the exactness of the AES algorithm implementation. The target is only on Register-Transfer Level (RTL) implementation, where the synthesize platform is FPGA. AES is also known as Rijndael in which the theory is robust enough, hence there will be no modifications on the mathematical concepts. On the other hand, AES' security measures such as side channel attack is not part of the research. Mode of operations in AES like counter mode or feedback is not covered in this research as well.

1.5 Thesis Outline

The overall AES algorithm of encryption process will be further explained which includes the few main operations in Chapter 2. In Chapter 3, design to be implemented are proposed after reviewing related work in Chapter 2. The implementation of each method are explained in detailed for each operation. The results and discussion of the proposed architecture are presented in Chapter 4. Besides, the performance in terms of throughput is compared with the related works in a table. For Chapter 5, the summary about the achievement of objectives has been concluded and future works are also recommended to improve the overall design in this thesis.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, the overall process flow of AES algorithm during encryption process will be explained. Next, each of the sections in AES encryption algorithm will be explained in more details in each of the subchapters. Besides, hardware implementation of each of the transformations in AES encryption are presented by comparison with the related papers in terms of performance and the most important is concerned about the processing speed of the applications.

2.2 AES Algorithm

The AES algorithm is known as Rijndael algorithm, which consists of a series of mathematical operations and the basic unit of the process is called a byte (8 bits). This algorithm is widely used in Windows Vista's fault analysis software for security wise in configuration file. Besides, the AES algorithm is also used for transactions security in ATM machines. The AES algorithm can be implemented in many different of platform languages such as C, Matlab, Java but for reconfiguration purpose for hardware implementation and synthesized by RTL, Verilog HDL is the chosen one (Deshpande et al., 2014).

The AES algorithm is a symmetric key block cipher that encrypts and decrypts a 128-bit data block by using the same key. The input data block is fixed to 128 bits while it requires key size of 128-bit, 192-bit or 256-bit during encryption (Oukili et al., 2016).

The AES algorithm is iterative, which means every iteration is named as a Round. The number of rounds is basically dependent on the key size used which is shown in Table 2.1. During AES encryption, the 128-bit input data (plain text) is converted to 128-bit output data (cipher text) using an input 128 bit cipher key in order to lock the information from being hacked by third party (Saini et al., 2014).

Table 2.1: AES Key Size with Key length, Block Size and Number of rounds (Deshpande et al., 2014)

AES Key Size	Key Length (words)	Block Size (words)	Number of rounds
128-bit	4	4	10
192-bit	6	4	12
256-bit	8	4	14

A 128-bit of input plain text is arranged in a 4-by-4 matrix array or called as state, which is shown in Figure 2.1 (Talha et al., 2016). Each of the block in the state equals to 1 byte of data. Therefore, they are total 16 bytes of data in a state.

S_{15}	S_{11}	S_{7}	S_{3}
S_{14}	S_{10}	S_{6}	S_{2}
S_{13}	S_{9}	S_{5}	S_{1}
S_{12}	S_{8}	S_{4}	S_{0}

Figure 2.1: A 128-bit of input data is arranged in 4x4 state of array (Talha et al., 2016)

The 128-bit AES encryption process can be achieved by going through few steps of algorithms (Rao et al., 2015) which is formed mainly from 4 sections. The details of the four main sections will be further explained in the following subchapters:

1. Sub-Bytes
2. Shift Rows
3. Mix Columns
4. Add Round Key

As shown in Figure 2.2, the 128-bit AES encryption process can be divided into 3 parts:

1. Add Round Key: XOR-ing the 128-bit plain text (raw data) and 128-bit initial key (cipher key).
2. Repeated Round: (Sub Bytes -> Shift Rows -> Mix Columns -> Add Round Key) for 9 times
3. Final Round: (Sub Bytes -> Shift Rows -> Add Round Key), without Mix Columns

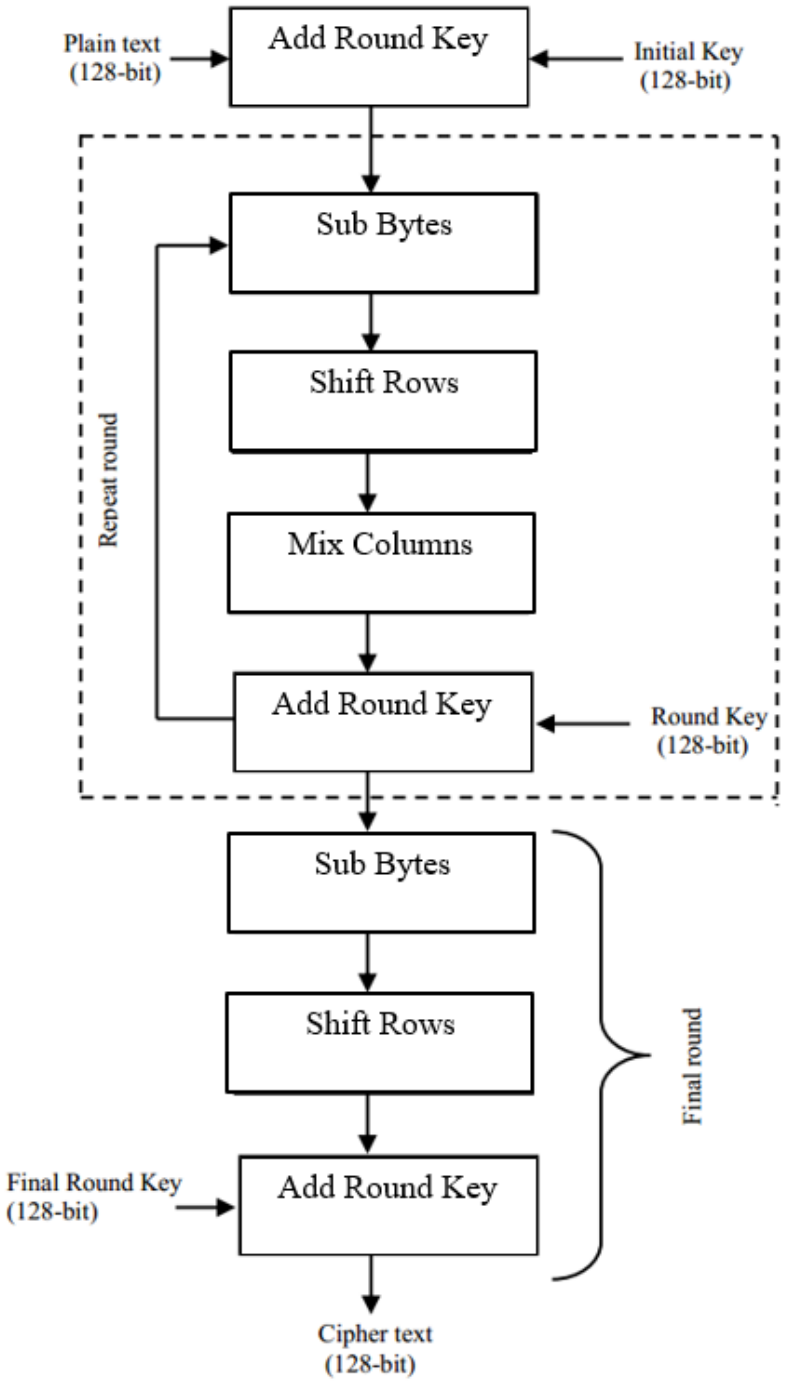

Figure 2.2: 128-bit AES encryption algorithm (Rao et al., 2015)

2.2.1 Sub Bytes

Sub Bytes is able to operate independently in each byte of the State, where each byte is referred and substituted regarding to the corresponding byte available in the Substitution box (S-Box). S-Box reveals the property of confusion and is known as one of the basic substances of symmetric key algorithm that performs substitution. This confusion property is used to obscure the finding of the key from the cipher text. In brief, S-Box takes one byte (8 bits) input and does transformation in order to deliver one byte (8 bits) output (Dao et al., 2015)

For the S-Box transformation, it consists of two operations which are the Galois field (GF) inversion and affine transformation over GF (2^8). Firstly, for the GF inversion, where the field computations have been taken over the binary field GF (2^8) with that of the constant irreducible polynomial, $(x) = x^8 + x^4 + x^3 + x + 1$. An element of GF ($2^8$) will be each of the 8-bit input and computes the inverse of the 8-bit input. Next, for the affine transformation over GF (2^8), the input is represented as an 8-bit vector and it will be multiplied by a fixed matrix M with 8x8 bits shown in Figure 2.3. Lastly, it be will added to a fixed vector C with 8-bit as shown in Figure 2.4. All of these properties provide non-linearity over cryptanalysis (Oukili et al., 2016).

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 2.3: The matrix M for S-Box transformation (Oukili et al., 2016)

$$C = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure 2.4: The vector C for S-Box transformation (Oukili et al., 2016)

Finally, An S-Box table with a 16×16 matrix of bytes is generated as shown in Table 2.2. A byte (8 bits) will be categorized as the leftmost 4 bits MSB and the rightmost 4 bits LSB, where the 4 bits MSB will be used as row value (group X) and the 4 bits LSB will be used as column value (group Y). All of these values of row and column are served as the indexes into the Table 2.2 to select a special value of 8-bit output as shown in Figure 2.5. Take one value for example, if the hexadecimal value is “53” references number of row 5 and column 3 referring to S-Box table, whereas their intersection value is “ED”. Therefore, the hexadecimal value “53” is mapped onto the value “ED” as seen in Table 2.2 (Nadjia et al., 2015).

Table 2.2: AES Encryption S-Box table (Deshpande et al., 2014)

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

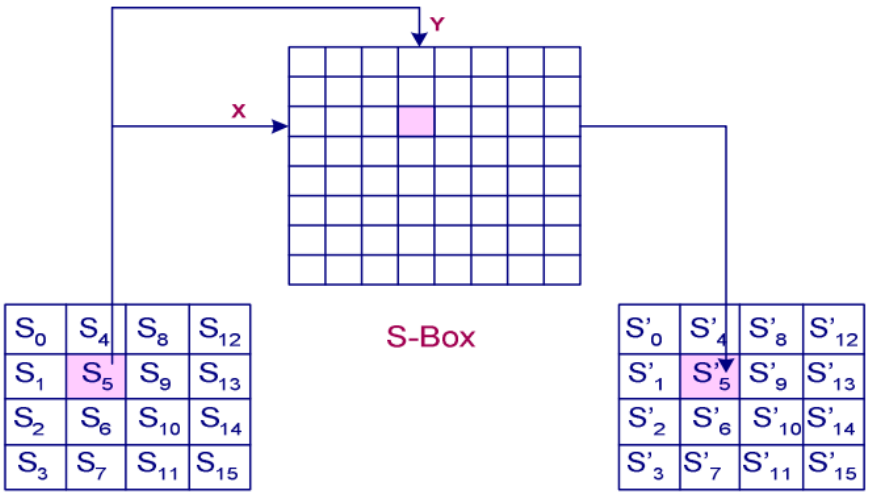

Figure 2.5: S-Box Transformation (Nadjia et al., 2015)

There are two common Sub Bytes hardware implemented which are Look Up Table (LUT) and combinational logic calculation. The LUT approach is implemented in Sub Bytes regarding to design of (Nadjia et al., 2015). Combinational logic calculation method is not implemented in this thesis as this on-the-fly method takes longer time to calculate as it needs to go through 2 main calculations which are affine transformation and multiplicative inverse in GF(2^8). LUT is chosen as the method of substitution process in AES algorithm as it is faster and easier to implement for better processing speed circuit design.

In this non-linear Sub Bytes Module, it contains 16 S-boxes which can be seen in Figure 2.6. The 128-bit input data (State) is divided into 16 bytes, where each byte of the State is the input to the S-Box. Each input byte of the State will be mapped to the S-Boxes to substitute the corresponding value with the LUT. All the output from S-boxes are concatenated to compose the output of the Sub Bytes Module, where the S-Boxes are stored in RAM (Random Access Memory).

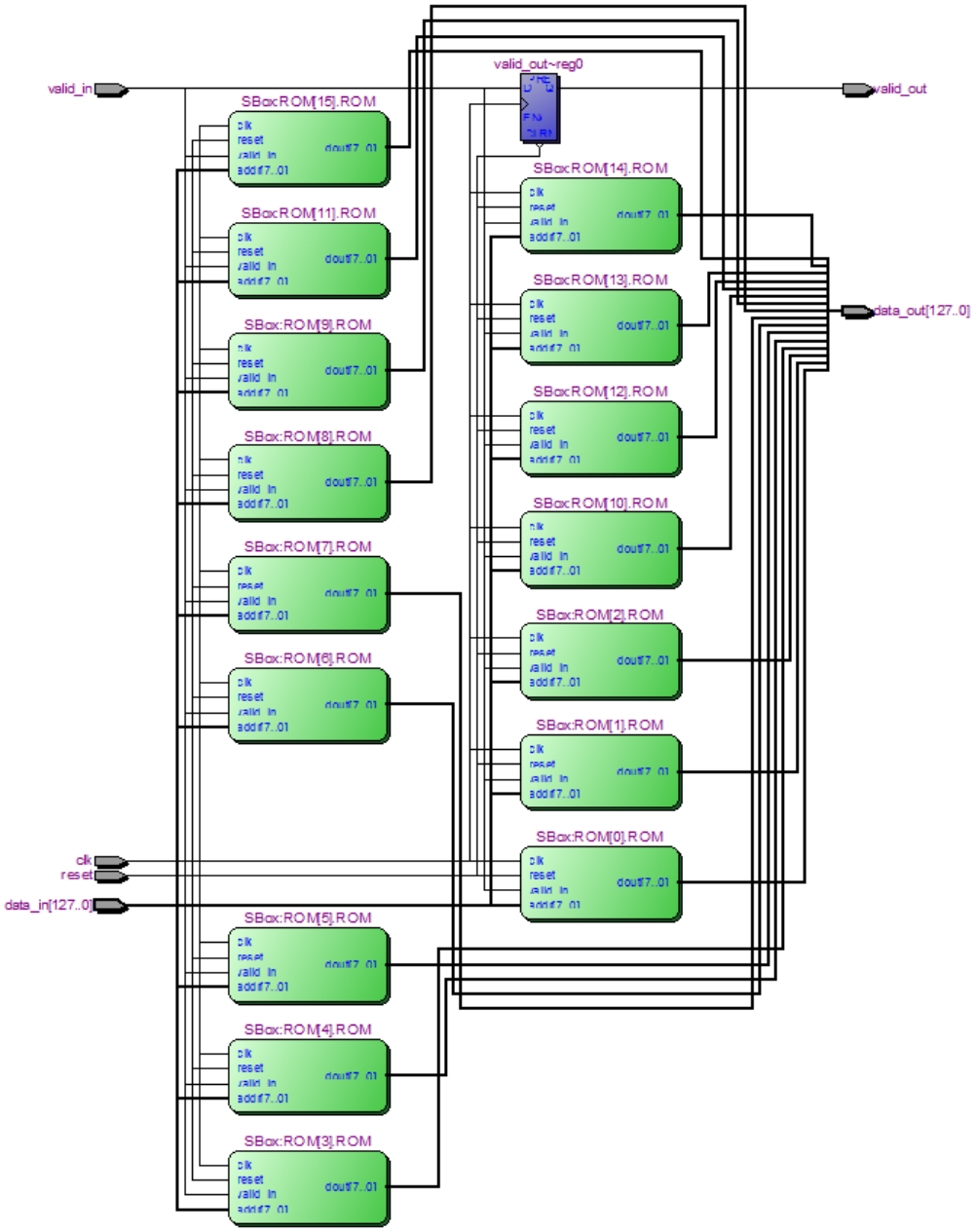

Figure 2.6: Circuit design of Sub Bytes Module

2.2.2 Shift Rows

Shift Rows transformation rotates row-wisely the bytes of the State for this last 3 rows respectively except for the first row. In this transformation, the last 3 rows of the State are left shifted cyclically by different offsets (Deshpande et al., 2014). The Shift Rows transformation is shown in Figure 2.7. There is a register triggered by clock signal to act as the medium to perform row shifting to the State (Maraghy et al., 2013).

1. Row 0: Not shifted.
2. Row 1: One byte shifted to the left.
3. Row 2: Two bytes shifted to the left.
4. Row 3: Three bytes shifted to the left.

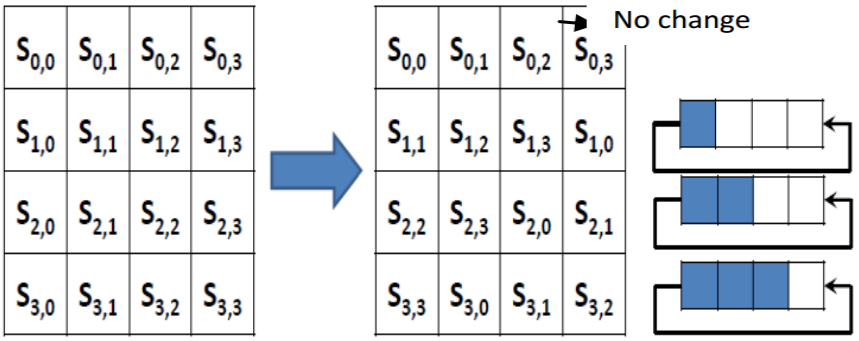

Figure 2.7: Shift Row Operation (Deshpande et al., 2014)

The Shift Rows Module is working with the operation of the shifting of connection (Gao et al., 2008). The shifting operation can be applied by rerouting the wire of each byte of the State from the source to the destination which can be seen in Figure 2.8. This rerouting method does not requires clock cycle to run the Shift Rows operation. This proposed method requires no additional hardware overheads and it is considered as an added advantage to the design.

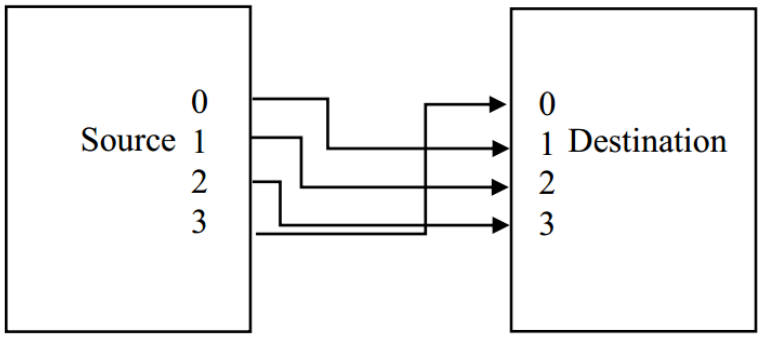

Figure 2.8: Rerouting of each byte of the State in Shift Rows Module

2.2.3 Mix Columns

In Mix Columns transformation, each column of the State bytes in Figure 2.9 is multiplied by a fixed matrix as shown in Figure 2.10. The two column vectors represent the State bytes' i-th column and i-th column of the transformed Mix Columns State matrix in Figure 2.10, for i = 0, 1, 2 and 3 respectively (Balamurugan et al., 2014).

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Figure 2.9: State bytes used for Mix Columns (Balamurugan et al., 2014)

$$\begin{bmatrix} S'_{0,i} \\ S'_{1,i} \\ S'_{2,i} \\ S'_{3,i} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{o,i} \\ S_{1,i} \\ S_{2,i} \\ S_{3,i} \end{bmatrix}$$

Figure 2.10: Fixed matrix used for encryption process (Balamurugan et al., 2014)

In the Mix Columns operation, the state bytes will be treated as the polynomials of Galois Field algebra, GF (2^8). This operation can be entitled as a matrix multiplication which can be seen in Figure 2.11, where S represents the initial state and S' represent the final state after this operation. (Prathyusha et al., 2013)

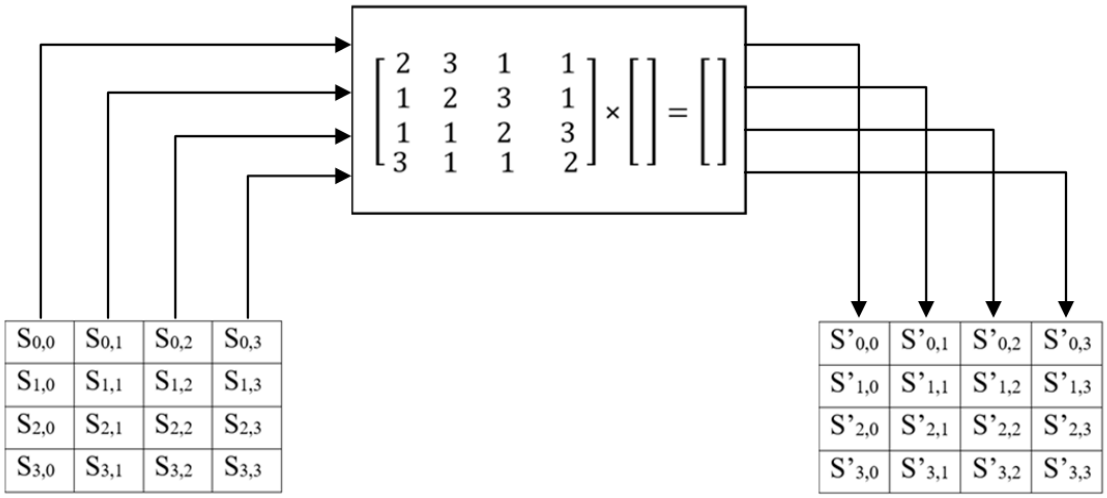

Figure 2.11: Mix Columns multiplication process (Prathyusha et al., 2013)

The Mix Columns Module is designed with Galois multiplication and 4 inputs XOR operation by referring to (Narasimhulu et al., 2014). But unlike combinational implementation of Galois field multiplication, this proposed design which uses Galois multiplication with ROM based implementation as this method is significantly faster than the combinational implementation in order to avoid combinational delays. For an 8-bit (one byte) of input data, there will be 256 multiplication conditions, where all these conditions will be readily stored in the (256 x 8) ROM. The Mix Columns Module uses two ROMs for Galois multiplication of "2" and "3" in order to perform 4 Input XOR operation. This design offers higher speed in the hardware implementation.

2.2.4 Add Round Key

Add Round Key Transformation is the process of a 128-bit state (Input Bytes) XOR with a 128-bit round key, which the 128-bit round key is generated from the cipher key through the Key Expansion module as seen as Figure 2.12. The output from this transformation will creates the Output Bytes in a form of 128-bit (Prathyusha et al., 2013).

The implementation of Add Round Key Module is the simplest steps in 128-bit AES algorithm, which only requires XOR gates to make the transformation as shown in Figure 2.13. Although this method might use a lot of hardware resources as each bit of the input data will XOR with each bit of the round key. However, this is the fastest way which fulfill the faster processing speed in this thesis. In this case, 128-bit AES requires 128 XOR gates to get the transformation done.

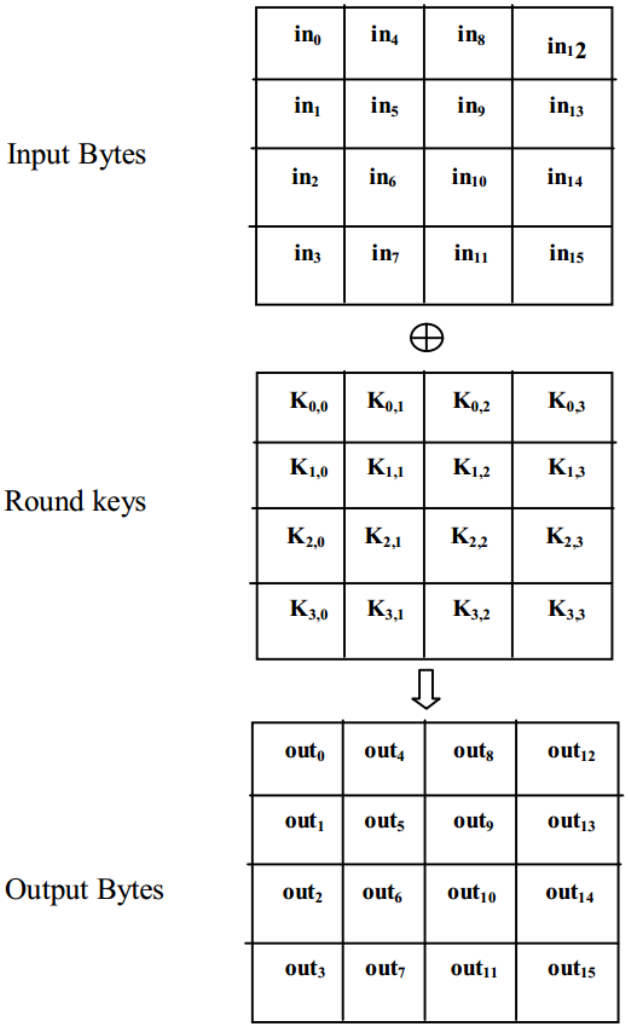

Figure 2.12: Add Round Key Process in AES algorithm

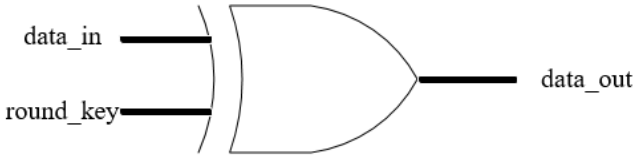

Figure 2.13: Circuit design of Add Round Key Module

2.2.5 Key Expansion

The Key Expansion in AES algorithm is mainly used to generate the 128-bit round key from the cipher key for each round. In this thesis, 128-bit AES algorithm is used where 10 rounds of transformation will be performed, therefore there will be 10 round keys being generated (Berent, 2017).

There are 3 important functions in this algorithm which is:

1. Rot Word
2. Sub Word
3. RCON

Here is the brief explanation on the 3 functions in Key Expansion algorithm:

2.2.5.a Rot Word

Rot Word is quite similar to the Shift Row Transformation, where it has circular shifts on 4 bytes. The simple process is shown in Table 2.3.

Table 2.3: Before and after Rot Word process

Before Rot Word (4 bytes)	After Rot Word (4 bytes)
3,4,5,6	4,5,6,3

2.2.5.b Sub Word

Sub word utilizes the S-box table in Table 2.2 for substitution which is similar to Sub Bytes Transformation where each of the 4 bytes is in the argument.

2.2.5.c RCON

RCON function provides exponentiation of 2 in Rijndael finite field to the round keys. The RCON table is shown in Table 2.4, where "i" is the round number and the equation for RCON[i] is 2^{i-1}. Taking Round 1 as example where i=1, then RCON [1]= $2^{1-1} = 2^0 = 1$.

Table 2.4: RCON table with value at respective round

Round, i	1	2	3	4	5	6	7	8	9	10
RCON[i]	01	02	04	08	10	20	40	80	1B	36
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00

2.2.5.d Key expansion flow with 128-bit of cipher key to generate 128-bit of Round Key

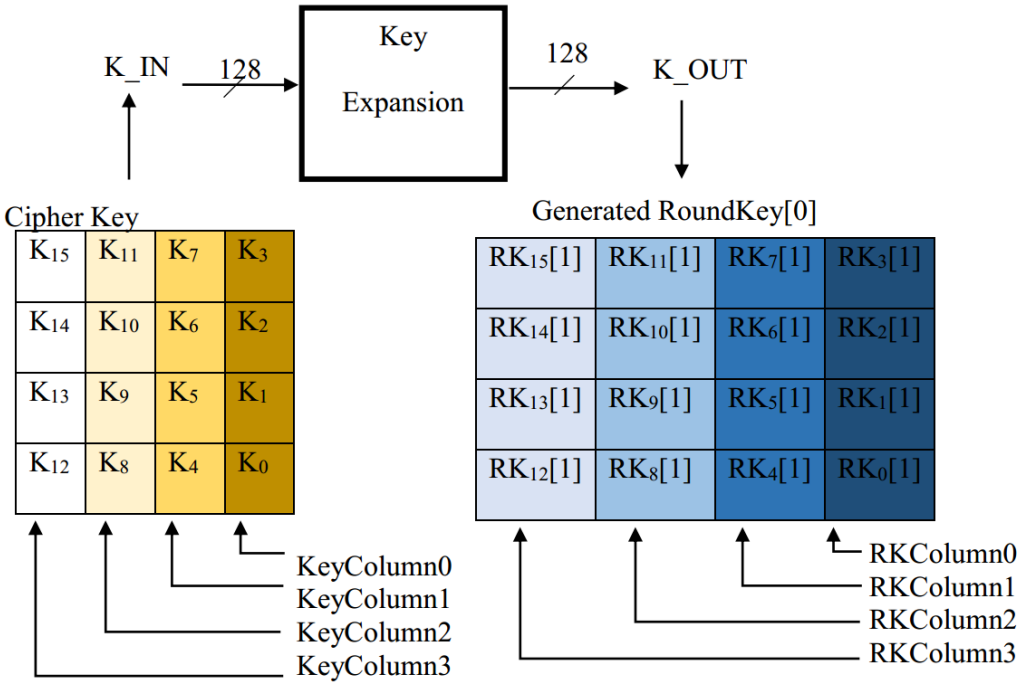

Figure 2.14: Key expansion flow which uses 128-bit of cipher key to generate 128-bit of RoundKey[0]

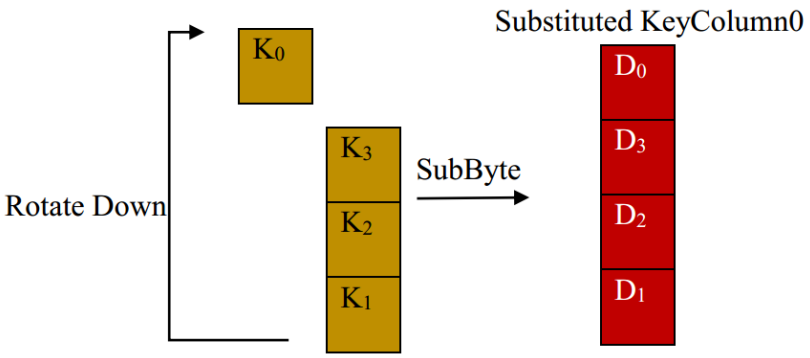

Figure 2.15: KeyColumn0 is rotated downwards and substituted with S-box value

The full flow of Key Expansion to generate a 128-bit of cipher key is shown in Figure 2.14. As shown in Figure 2.15, the Rot Word function takes places where the rightmost column of cipher key which is KeyColumn0 is rotated one byte downwards. Next, it will goes to Sub Word function. Each byte of the KeyColumn0 will be substituted with a new value through the Sub-Bytes operation with the aid of S-Box table shown in Table 2.2.

RCON, the substituted rightmost KeyColumn0 is XOR-ed with the leftmost unsubstituted KeyColumn3 and RCON[i]. For RCON[i], the value of "i" is a constant which will be changed according to the number of rounds as shown in Table 2.4. Hence, the leftmost column of Generated RoundKey[0], which the RKColumn3 is generated as shown in Figure 2.16.

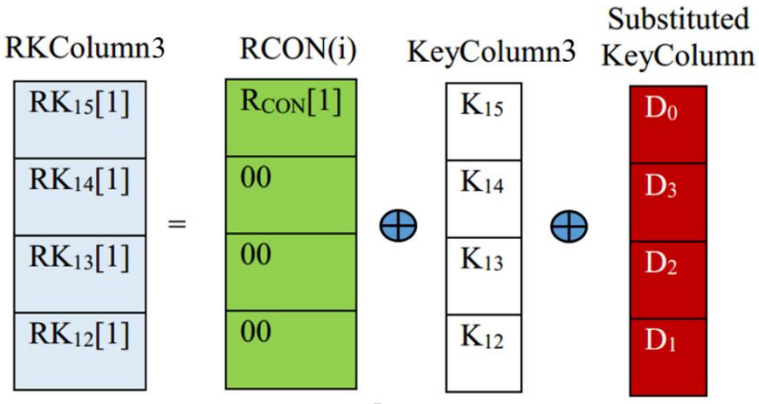

Figure 2.16: Calculation to obtain RKColumn3

Next, the process continues to move on when the newly generated RKColumn3 is XOR-ed with KeyColumn1 to obtain RKColumn2 as shown in Figure 2.17. The repeated process will be executed with the generated RKColumn2 and KeyColumn to generate RKColumn1 and RKColumn0 as shown in Figure 2.18 and Figure 2.19 respectively. Finally, the generated RoundKey[1] can be used in Add Round Key Transformation during encryption.

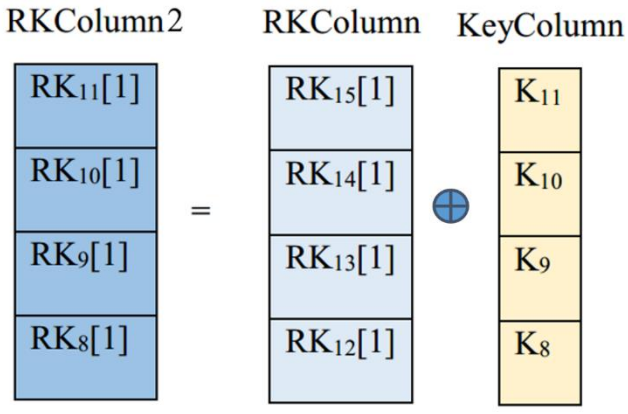

Figure 2.17: Calculation to obtain RKColumn2