AN ABSTRACT OF THE THESIS OF

Wen-Chun Yang for the degree of Master of Science in

Electrical and Computer Engineering presented on May 19th, 2005.

Title: Hardware Realization for Advanced Encryption Standard Key Generation

Abstract approved:

Çetin Kaya Koç

Advanced Encryption Standard (AES) is one of the secret key algorithms used in Cryptography. It is applied in a variety of applications including smart cards, internet web servers, automated teller machines (ATMs), etc. Both hardware and software implementations are taken into consideration while addressing AES algorithms. In addition to reaching standard requirements, the hardware realization provides better security than the software realization while selecting the AES algorithm. An efficient architecture for the hardware implementation of Advanced Encryption Standard (AES) key expansion is presented.

Hardware Realization for Advanced Encryption Standard Key Generation

By

Wen-Chun Yang

A Thesis

submitted to

Oregon State University

In partial fulfillment of
The requirements for the
degree of

Master of Science

Completed May 19, 2005
Commencement June 2005

Master of Science thesis of Wen-Chun Yang presented on May 19, 2005

APPROVED:

Major Professor, representing Electrical and Computer Engineering

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

Wen-Chun Yang, Author

ACKNOWLEDGEENTS

The author expresses sincere appreciation to the following individuals. Dr. Çetin Kaya Koç for his mentorship and guidance. Onur Acıiçmez for his help in the designing phase of AES key generation and for his enthusiasm. Gerald Lai for his help in the testing phase of VHDL coding. Minho Kim for his help in the investigating phase of AES hardware implementation. Hong Zhu for her strong believe in me and her everlasting cheerfulness.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

For my dad, my mom, my sister, and my cousin

Kuang-Li Yang

Hsiao- Chin Chen

Irene W.F. Yang

Chih-Yao J. Wang

Hardware Realization for Advanced Encryption Standard Key Generation

1. INTRODUCTION

Advanced Encryption Standard (AES) is one of the secret key algorithms used in Cryptography. It is applied in a variety of applications including smart cards, internet web servers, automated teller machines (ATMs), etc. Both hardware and software implementations are taken into consideration while addressing AES algorithms. In addition to achieving standard requirements, hardware realization provides better security than the software realization while selecting the AES algorithm. An efficient architecture for hardware implementation of Advanced Encryption Standard (AES) key expansion is presented.

In this paper, we focus on how the key expansion affects the resource-constrained AES algorithm as applied using different key lengths. Three kinds of architectural approaches, Pipelined, Sub-pipelined, and Loop-unrolling, are considered while addressing the hardware optimizations. In addition, roundkey generation is discussed as part of the hardware realization. The goal of this study is to accomplish and to simulate the best combination of the architectural approach and the roundkey generation. Furthermore, an analysis of these architecture-key generation combinations is evaluated as part of the conclusion in this research.

2. ADVANCED ENCRYPTION STANDARD (AES)

2.1 Overview

Cryptography is one of the most important topics related to network security and transmission today. It plays an important role in technology and involves a great deal of background knowledge of computer science and mathematics. Advanced Encryption Standard (AES) was introduced by the National Institute of Standards and Technology (NIST) in January, 1977. This standard requires an algorithm that should be operated on blocks of 128 input data bits and use key sizes of 128, 192, or 256 bits. The idea for this standard was to replace the old public key algorithm, Data Encryption Standard (DES). To do this, Rijndael was chosen as the AES algorithm, which was designed by Joan Daemen and Vincent Rijmen [1, 2].

A symmetric block cipher with a 128-bit block size is the requirement for AES. Hence, both the receiver and sender use one single key to encrypt or to decrypt the information. However, an iterated block cipher is designed using the Rijndael algorithm, which uses 128, 160, 192, 224 or 256 bits for its block sizes and key lengths. Therefore, the extra allowance is not included in AES.

The internal AES operations are performed on a two dimensional array of bytes called State, which contains an 4×4 array that maps, using the divided input data. Each byte contains eight bits and is denoted by $S_{i,j} (0 \le i, j < 4)$, which is considered an element of GF (2^8). Furthermore, the State consists of four rows of bytes, where N_b bytes are used in each row. The encryption/decryption process is performed on the

State where the input and output data bytes are mapped.

The AES key generation is also mapped to four rows of bytes. However, the numbers of bytes used in each row is different and is denoted by N_k. In addition, the value of N_k is four, six, or eight when the key length K is 128, 192, or 256 bits respectively. A single iteration in the AES algorithm is called a round. Similarly, the total number of rounds, N_r, 10, 12, or 14 is dependent on the key length. The details are included in Figure 2.1.

Key Size (Bits)	128	192	256
Key Lengths (N_k words)	4	6	8
Block Size (N_b words)	4	4	4
Number of rounds (N_r)	10	12	14
Round key size (words)	4	4	4
Expanded key length (Serial)	44	52	60
Expanded key length (Parallel)	10	8	7

FIGURE 2.1. Information for different key lengths

2.2. AES Algorithm

Four basic steps called layers are performed on the incoming data while doing the encryption process. They are:

1. The ByteSub Transformation.
2. The ShiftRow Transformation.
3. The MixColumn Transformation.
4. AddRoundKey.

The above four operations together form a round that proceeds through the steps in chronological order. The ByteSub Transformation is used for resistance with differential and linear cryptanalysis attacks; the distribution of bits over multiple rounds is produced by this linear ShiftRow Transformation step, and the same strategy is applied to the MixColumn operation where its result is XORed with the roundkey at the AddRoundKey layer.

These four steps used in the encryption are inverted in a straightforward decryption structure. Consequently the operations used in the decryption process are:

1. The InvByteSub Transformation.
2. The InvShiftRow Transformation.
3. The InvMixColumn Transformation.
4. AddRoundKey

The major problem caused by the encryption/decryption structure is that the transformation differences used in both configurations increase resource dissipations while implementing the encryptor/decryptor hardware designs. Properties [2] indicate that making changes in the roundkey generation step will result in an equivalent structure for both encryption and decryption. The equivalent structure is shown implemented in Figure 2.2.

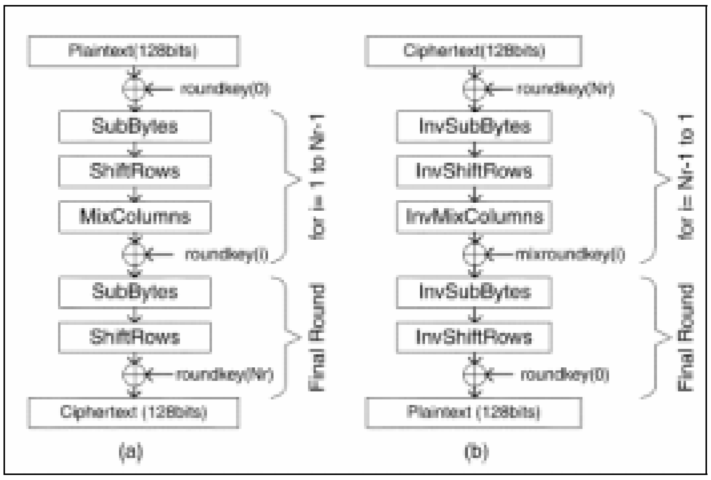


Figure 2.2. The equivalent AES structure

2.2.1 The ByteSub Transformation

The ByteSub Transformation is used for resistance with differential and linear cryptanalysis attacks. It is a simple look-up table, which uses a 16×16 matrix of byte values called S-Box. The S-Box includes a combination of all the 256 possible 8-bit values. The way it works is to map every single byte of the State into a new byte. It uses the leftmost four bits of the byte, as a row value and the rightmost four bits, as a column value. The row and column values are used as an index to check up the 8-bit output values of the S-Box (see Figure 2.3). In addition, the S-Box is used for encryption and an inverse S-Box is used in decryption.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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 2.3. S-Box

2.2.2 The ShiftRow Transformation

The distribution of bits over multiple rounds is produced in this linear ShiftRow Transformation step. In this operation, no changes are applied to the first row of State. A one-byte circular left shift operation is performed for the second row. Respectively, a two-byte and then a three-byte circular left shift operation are performed on the third and fourth row.

1A	1B	1C	1D
2A	2B	2C	2D
3A	3B	3C	3D
4A	4B	4C	4D

$\rightarrow$

1A	1B	1C	1D
2B	2C	2D	2A
3C	3D	3A	3B
4D	4A	4B	4C

FIGURE 2.4. ShiftRow Transformation

2.2.3 The MixColumn Transformation

A strategy similar to that used in the ShiftRow Transformation is applied in this operation. However, it is the most complicated step due to the arithmetic calculations. The elements of the data matrix, the four bytes in each column of State, are the elements of the Galois Field GF (2^8), which are used as the coefficient of the polynomials, and are multiplied with modulo (x^4+1) by the fixed polynomial a(x), given by:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

Overall, this step performs a matrix multiplication in GF (2^8) and is expressed as

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} \{02\}_H & \{03\}_H & \{01\}_H & \{01\}_H \\ \{01\}_H & \{02\}_H & \{03\}_H & \{01\}_H \\ \{01\}_H & \{01\}_H & \{02\}_H & \{03\}_H \\ \{03\}_H & \{01\}_H & \{01\}_H & \{02\}_H \end{bmatrix} \begin{bmatrix} S_{0,C} \\ S_{1,C} \\ S_{2,C} \\ S_{3,C} \end{bmatrix}, 0 \le C < 4$$

2.2.4 AddRoundKey

This operation is to XOR the output data from the MixColumn step with the roundkey generated from the key expansion design, which contains 128 bits and is mapped into a 4×4 matrix ($\text{Key}_{i,\, j}$). In addition, this layer outputs the result of the round. The $\oplus$ is used as a XOR function, but is an addition operation in GF (2^8).

$$\begin{pmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,}3 \end{pmatrix} \oplus \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k2,0 & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix} = \begin{pmatrix} e_{0,0} & e_{0,1} & e_{0,2} & e_{0,3} \\ e_{1,0} & e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,0} & e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,0} & e_{3,1} & e_{3,2} & e_{3,3} \end{pmatrix}$$

2.3. Key Expansion

Key generation is an important part of AES algorithm, which creates N_b (N_r+1) words. The method is to generate the initial set of N_k words first, and other roundkeys are produced by the key, K subsequently. An array of four-byte words is expected as the key expansion output, which is denoted by W_i as its parameter i ranges from zero to N_b (N_r+1). Each roundkey is generated as:

$$roundkey(i) = (w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3})$$

The key expansion algorithm is described in pseudo code in Figure 2.5 [2]. In this algorithm, SubWord uses the S-box from the ByteSub Transformation layer to get the internal elements to each of the four bytes in a word. The leftmost byte in each word of a constant word array, Rcon, contains a nonzero value. In addition, the RotWord operation performs a right-to-left rotation to each of the four bytes in a word.

```
        KeyExpansion (byte key (4*$N_k$), word w ($N_b$* ($N_r$+1)), $N_k$)
        Begin
        Word temp
        i = 0
        while (i < $N_k$)
        w (i) = word (key (4*i), key (4*i+1), key (4*i+2), key (4*i+3))
        i = i + 1
end while
        i = $N_k$
        while (i < $N_b$* ($N_r$+1))
temp = w (i-1)
if (i mod $N_k$=0)
temp = SubWord (RotWord (temp)) XOR Rcon (i / $N_k$)
else if ($N_k$ > 6 and i mod $N_k$=4)
temp = SubWord (temp)
end if
w (i) = w (i-$N_k$) XOR temp
                i = i+1
        end while
        end
```

FIGURE 2.5. Pseudo Code for Key Expansion

2.4. Architectural Optimization

A good architecture design is the key to success. As technology is improved along with new inventions, high processing speed, low power dissipation, and secure communication channels become the goals of modern Cryptography system designs. Software and hardware implementations are both taken into consideration to see if either will satisfy the needs of different applications.

One of the requirements is that the proposed idea should be able to supply both software and hardware implementations while designing AES algorithm. However, many of the proposed ideas and publications show that software implementations are widely used for performance comparisons offered on various platforms [7].

Advantages of using software implementation are its efficiency, availability, and the portability causes access problems for the system. In addition, it is easy to use and to upgrade in order to keep the best preservation. Nevertheless, software implementation provides low physical security to the system, and its shortcoming is a limited key storage, which is another reason to question its reliability.

On the other hand, in the performance comparison between software and hardware implementation the priority is to evaluate which system provides higher security. Hardware's inflexibility eliminates possibility for the external changes to the system, and this results in a high quality physical security when compared with software implementations. Besides, most of the hardware implementations are tested either using the semi-custom Application Specific Integrated Circuits (ASICs) or the reconfigurable Field Programmable Gate Arrays (FPGAs) in order to understand the agility of AES algorithm.

Hardware implementation often results in secure application designs which meet high Cryptography standards. Depending on different needs for a system, how to choose the most appropriate architecture to accomplish all the requirements is the key to the achievement. Several architectures were taken into consideration, they are Pipelined architecture, Subpipelined architecture, and Loop-Unrolled architecture.

2.4.1 Design Method

Currently, two basic design methods often are used in hardware implementations. They are high level language based design and low level schematic based design. The language based design often uses synthesis tools in order to implement the hardware required by the applications, users, or the algorithms. In addition, language based design often can not provide the most optimized results for the speed and area implementation compared with the low level design since the synthesis tools are likely to use larger and slower devices. Nevertheless, the optimization performance is dependent on the chosen synthesis tools.

On the other hand, the schematic based design is used to provide better optimization compared with the high level design. However, it is now also replaced by a language based design named VHDL due to the growing architectural complexities in modern FPGA technology.

2.4.2 Operation Modes

The symmetric block ciphers are used in different operating modes, and these modes can be divided into two categories depending on the hardware implementation.

They are:

1. Feedback modes
2. NonFeedback Modes

The feedback modes include the Cipher Block Chaining mode (CBC), Cipher Feedback Mode (CFB), and Output Feedback Mode (OFB). The encryption is completed when the previous clock is finished processing, and the next block of data is about to be started. Currently, the encryption of data is performed mainly using the feedback mode to meet modern security standards.

On the other hand, Electronic Code Book mode (ECB) and counter mode are listed in the nonfeedback mode category. The nonfeedback modes are often used for encrypting the session keys for the key distribution process. During the encryption process, all blocks of data are encrypted in parallel.

2.4.3 Architecture Comparison

The architectural approach influences the performance and the speedup of the hardware implementation. Different architectures are applied to pursue the best performance depending on the requirements for each optimization approach. Pipelined, Subpipelined and Loop-unrolled are used in the non-feedback mode of the AES encryption/decryption structure to enhance performance by changing the hardware design at each round. Simple descriptions are listed as follows.

2.4.3.1 Pipelined

In order to increase the speedup at each round, multiple data blocks are processed concurrently for both the encryption and decryption structures. The pipelined method is one of the techniques often used for computer architecture. It takes advantage of parallelism, and in this case used the method to place registers between each round to form a pipelined stage so that several blocks of data are processed by the circuit at the same time [3]. Also, the concept of pipelined architecture is generally used to increase the amount of data processed by the digital circuit. During the process, the currently processing data is moved to the next stage which the next block of data takes its place. When the processed data meets the end of its depth, it is fed back to the beginning of the loop process. The speed of the pipelined structure is given by the number of bits processed by the design[8]

$$speed = 128/\#rounds \times reduced\_clock\_period$$

Note that the reduced_clock_period is the minimum period after the pipelined process. A completed pipelined architecture diagram is presented in Figure 2.6 [8].

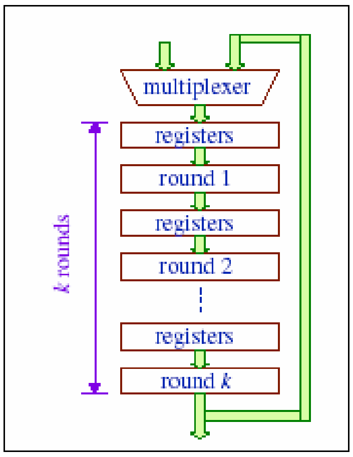


Figure 2.6. A completed pipelined architecture

2.4.3.2 Subpipelined

A large delay of clock cycles is created when the rounds of operation of the pipelined design are complicated. A subpipelined architecture would be the perfect solution for solving a situation like this. The theorem of the pipelined architecture is also applied to the subpipelined algorithm. Subpipelining not only places registers between each round, but also puts them inside each round unit while applying the idea of the loop-unrolled structure. Compared with the other two approaches, the subpipelined architecture accomplishes the maximum speedup [4]. In addition, the ByteSub and InvByteSub Transformation steps for encryption/decryption are usually implemented by the look-up tables [5-6].

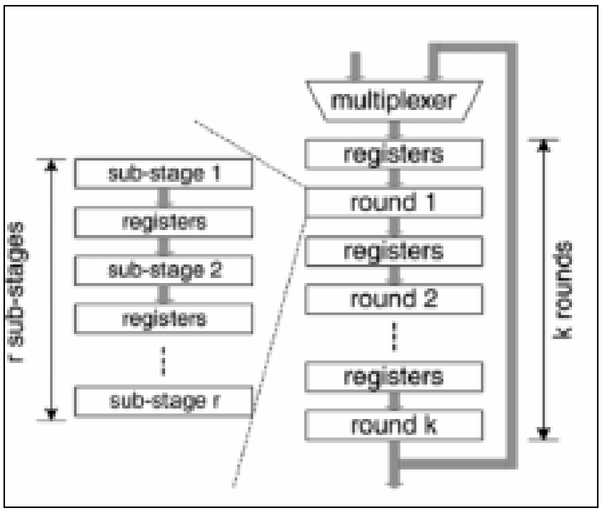


Figure 2.7. A completed subpipelined structure

2.4.3.3 Loop-Unrolled

This design is also called unfolded architecture. One data block is processed at a time in this approach; however, multiple rounds can be performed during each clock cycle. It can process one job at a time, but many tasks can be worked with each cycle. The idea is to implement the combinational logic gates of the design circuit into multiple numbers of rounds. This design is not suitable for the speed optimization in both feedback and nonfeedback operating modes due to the fact that the idea of this design results in a very slow clock signal for its system. However, the area used by this design is much smaller compared with other architecture designs and it often comes with an area penalty. A completed loop-unrolled architecture is presented in Figure 2.8.

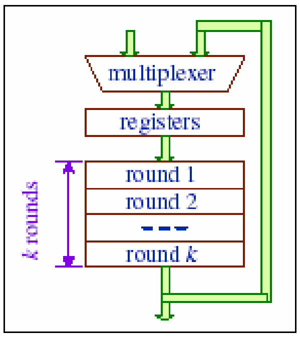


Figure 2.8. A completed loop-unrolled architecture

3 PROPOSED DESIGN AND IMPLEMENTATION DETAILS

In this section we give the details of our implementations and report the performance results in a non-feedback mode. First we present scalar implementations of our proposed AES key generation architecture and then use this structure as a base to analyze the effects of using a subpipelined AES key generation model.

3.1. Optimization Techniques

Here we explain the optimization techniques used in our implementations. The whole implementation is presented in Figure 3.1.

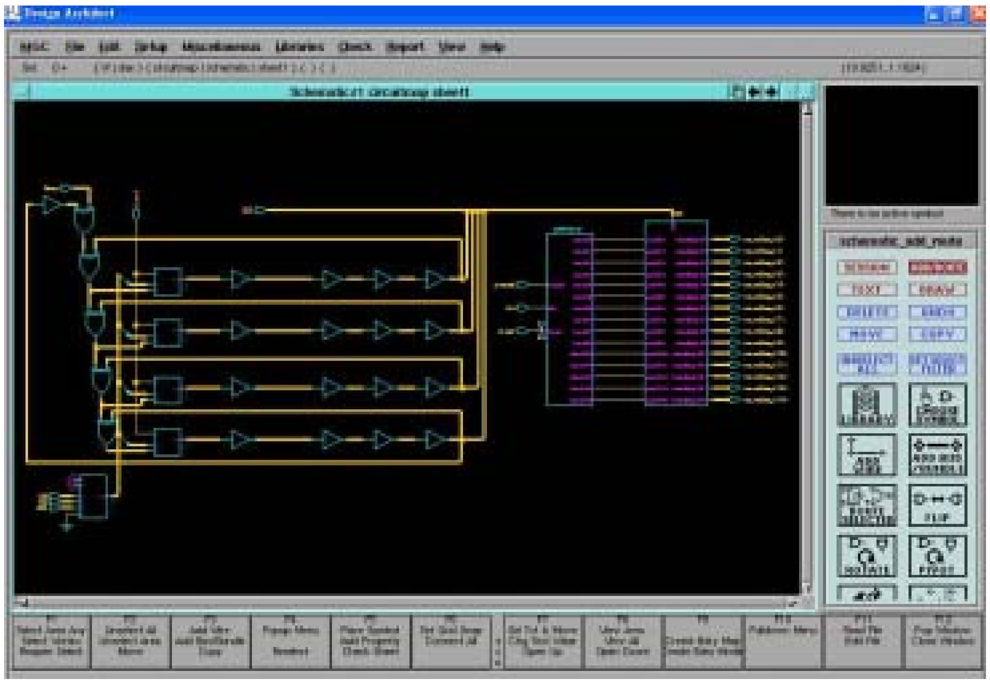


Figure 3.1. The whole implementation

3.1.1 Overall Design

The proposed structure is based on the concept of a subpipelined architecture. The design is divided into two different structures, the main architecture and the roundkey generation. In the architectural part of the design, the idea of a subpipelined architecture is used with different combinational logic gates, such as the SubWord data block, the RC constant array data block, the input selection data block, the multiplexers, and the registers. At the same time, the roundkey generation uses the on-the-fly technique, which is implemented by Johnson Counters, multiplexers, and registers. The design is able to generate all $N_r + 1$ roundkeys after $r \times N_r$ clock cycles. A “start” signal is used to determine the roundkey generation process. When the start signal is equal to “0”, the initial roundkey is loaded into to the registers to begin the whole process; the rest of the roundkeys are generated when the start is equivalent to “1”.

3.1.2 The Subpipelined Architecture

The point of this research is to design an architecture used in three different key length inputs (128bit, 192 bit, and 256 bits.) In software design, it is easy to write a programming code for multiple resource constraints which produce no problems. However, hardware implementation is totally different than the software design. It can not be designed to use only the minimum number among data inputs. The plan has to be fitted for all possible data constraints. Thus, the input of the hardware device must be fit for maximum number of the key lengths, which is 256-bit long length of data in our case. Therefore, the data line is designed for 64-bit long length of data in our implementation.

First, the key data input is identified in the input selection data block and then is transformed to four 64-bit long multiplexers. As mentioned in the previous section, the start signal is used to determine the input loading as a selection line for the multiplexers. Then, in the architecture part, the 64-bit data is stored in the registers and is passed on to XOR with the result XORed from the SubWord and the RC constant array data block. On the other hand, the roundkey generation process uses a 64-bit long length of data that comes from the main architecture to generate and output the N_r roundkeys.

3.1.3 The SubWord Data Block

In the AES key expansion algorithm, the SubWord operation applies the ByteSub transformation to each of the four bytes in a word. Most of AES hardware implementations either build an S-Box into their designs or use a look-up table (LUT) of ByteSub transformation in order to perform this function. LUT is used to achieve this function in our design.

Eight ByteSub transformation blocks are used to implement the SubWord data block. The ByteSub transformation block is built into an 8×8 logic gate. The output is determined by the input data referred to in section 2.2.1. Our structure is designed to accommodate multiple resource constraints and a 64-bit data line is connected to the input of the SubWord data block. Each of the ByteSub transformation blocks takes 8-bit data from the 64-bit data line, and the output of each ByteSub transformation block is combined to form a 64-bit output data for the SubWord data block, which is

XORed with the output data of the RC constant array data block. The design is presented in Figure 3.2.

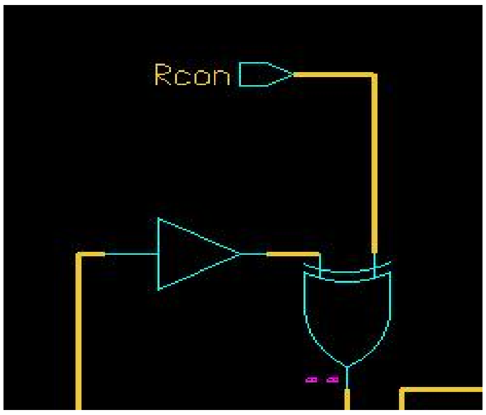


Figure 3.2. The Rcon design

3.1.4 The RC Constant Array Data Block

The RC constant array data block used in our design is 64-bit long and its value is calculated in the finite field Galois Field GF (2^8). Note that the model of GF (2^8) depends on a choice of the irreducible degree 8 polynomials, which is $X^8 + X^4 + X^3 + X + 1$ for the Rijndael design. The value of this array is designed as $Rcon[i] = [RC(i), \{00\}, \{00\}, \{00\}, \{00\}, \{00\}, \{00\}, \{00\}$. In order to calculate the $RC(i)$ value, a formula for calculating the Rcon (i) values is presented as follows.

$$RC(i+1) = \{00000010\} Rcon[i], (1 \le i < N_r - 1)$$

In our design, there is an internal four-bit selection line connected between the roundkey generation controller and the RC constant array data block. This selection line is used to decide which Rcon[i] value should be picked for that round. The selection line is presented in 16-bit long sections and each round is presented as “1”. For example, the first round is shown as “0000000000000001” for the selection line, and other rounds are also displayed in this way, respectively. The results of all the

Rcon[i] values are calculated in Figure 3.3.

Rcon[i]	Values
Rcon[1]	{01}
Rcon[2]	{02}
Rcon[3]	{04}
Rcon[4]	{08}
Rcon[5]	{10}
Rcon[6]	{20}
Rcon[7]	{40}
Rcon[8]	{80}
Rcon[9]	{1B}
Rcon[10]	{36}
Rcon[11]	{6C}
Rcon[12]	{D8}
Rcon[13]	{AB}

Figure 3.3. RC constant values

3.1.5 The Input Selection Data Block

A four-to-one multiplexer is used to perform the input selection operation. The selection line for this multiplexer is selected by the user. As mentioned previously, three different key lengths are inputted in our design. As considering the problems about three different input data lengths in the dataline, a multiplexer is used to perform a 256-bit output dataline, which provides the user the capability to choose the key length they want to use to perform the test. However, comparing the 256-bit data input with the 128-bit and 64-bit data input key length, differences appear as the process begins. Thus, we concatenate the smaller two key length differences with 128-bit and 192-bit inputs so that a 256-bit output is always performed in our design due to the hardware design.

3.1.6 Multiplexers

Four two-to-one multiplexers are used in our design for selecting the 64-bit dataline in order to generate N_r rounds. The input to this multiplexer came from the input selection data block and the results are XORed from the SubWord and RC constant array data block. The “start” signal selects which input data should be outputted to the registers. When the start signal equals to “0”, the multiplexer outputs the initial 64-bit data that came from the input selection data block. Otherwise, it passes on the 64-bit data from the results XORed from the SubWord and RC constant array data block. Note that the multiplexer uses every 64-bit from the overall 256-bit output started from its most significant bit (MSB) to its least significant bit (LSB). The design is presented in Figure 3.4.

3.1.7 Registers

The architecture design discussed in chapter two shows that the register is used between the multiplexer and the round. As the Subpipelined architecture is applied in our design, extra substages are divided from each round unit in order to accomplish the maximum speedup. We use registers with an equal delay to perform the substage operations. However, the number of substages used in our design is considered as another factor that might affect the speedup [9-10]. A comparison indicates that use r = 3 for the substage design achieves a higher speedup [11]. In our design, three extra registers are used in the round unit, and the data outputted from the last register of the round unit is collected as a 256-bit unit of data that is transferred to the roundkey generation. The design is presented in Figure 3.4

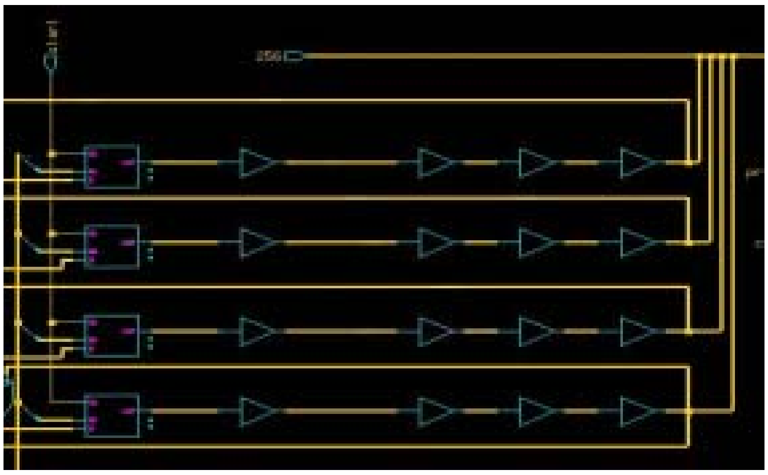


Figure 3.4. The round unit design

3.1.8 Roundkey Generation

Two ways of generating the roundkey have already been applied to the AES hardware implementation [9-10]. One is generating the roundkey and storing it in a memory register before it is needed for use, and the other is generating the roundkey on-the-fly. Both methods can be the best solution for generating a roundkey depending on application requirements.

A roundkey that is stored generated roundkey in memory is good for a design which does not require constant key changes. This method spends some extra time on reading the correct roundkey from the memory location, but this does not increase the delay in the decryption process. However, it does include a certain amount of time expense. On the other hand, generating on-the-fly is good for our design since constant roundkey changes are necessary in our design. In addition, it decreases the unnecessary time excuse, which reading the correct roundkey is good for our design.

In our design, the roundkey generation is separated into two parts, the roundkey controller and the control roundkey register shown in Figure 3.5.

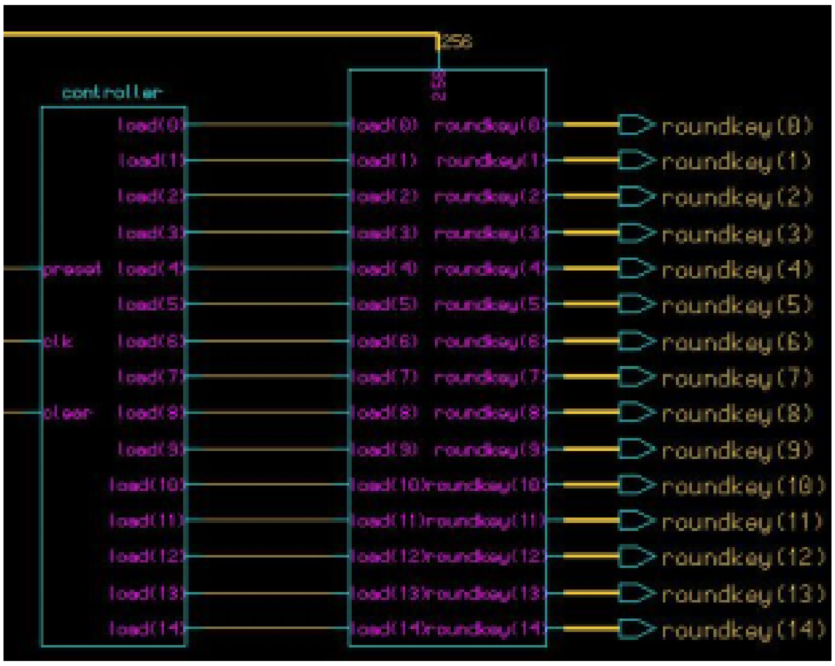


Figure 3.5. The roundkey generation

The roundkey controller is implemented by the Johnson Counter, which produces 16 different states. The counter starts from a state of "00000000" to a state of

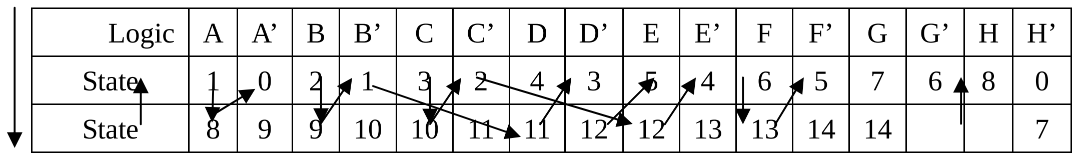
“10000000”. A “1” is repeated and performed the left-to-right shift until the whole state is shown as “11111111.” Then, a “0” is used to replace the “1” and is also performed in the left-to-right shift until the whole state is shown as “000000001.” The counter has originally designed to produce all $N_r + 1$ roundkeys after $r \times N_r$ clock cycles. In our case, it should be 15 roundkeys. However, 16 roundkeys are generated in our design due to the fact that some of the combinational logics of our design would pass on some undefined values “U” for the first few clock cycles while simulating in the Modelsim VHDL environment, which the roundkey generated in the first few clock cycles is not the original initial roundkey. However, the counter is still counting and for the above reason it is going to affect the output roundkey value. In order to avoid this factor, we decided to generate one more state in order to get the correct roundkey output. The counting sequence is presented in Figure 3.6.

Figure 3.6 shows that the controller is implemented by eight D type Flip-Flops (FFs) with the required logic gates by presenting a heuristic algorithm to determine the combinational logic.

Then, the output branch can be distinguished by removing the redundant states from both sets respectively. The output of this logic starts from State 8 -> 1 -> 10 -> 3 -> 12 -> 5 -> 14 -> 7 -> 0 -> 9 -> 2 -> 11 -> 4 -> 13 -> 6. Each of the output states (0 to 15) is shown as “1” in the final output data. For example, output state“0000000000000001” is shown for state 1, respectively. The output state diagram and the circuit implementation are presented in Figure 3.7 and Figure 3.8.

State	A	B	C	D	E	F	G	H	Required Logic
0	0	0	0	0	0	0	0	0	A’H’
1	1	0	0	0	0	0	0	0	AB’
2	1	1	0	0	0	0	0	0	BC’
3	1	1	1	0	0	0	0	0	CD’
4	1	1	1	1	0	0	0	0	DE’
5	1	1	1	1	1	0	0	0	EF’
6	1	1	1	1	1	1	0	0	FG’
7	1	1	1	1	1	1	1	0	GH’
8	1	1	1	1	1	1	1	1	AH
9	0	1	1	1	1	1	1	1	A’B
10	0	0	1	1	1	1	1	1	B’C
11	0	0	0	1	1	1	1	1	C’D
12	0	0	0	0	1	1	1	1	D’E
13	0	0	0	0	0	1	1	1	E’F
14	0	0	0	0	0	0	1	1	F’G
15	0	0	0	0	0	0	0	1	G’H

Figure 3.6. The Johnson Counter sequence



Logic	A	A’	B	B’	C	C’	D	D’	E	E’	F	F’	G	G’	H	H’
State	1	0	2	1	3	2	4	3	5	4	6	5	7	6	8	0
State	8	9	9	10	10	11	11	12	12	13	13	14	14			7

Figure 3.7. The Johnson counter state output diagram

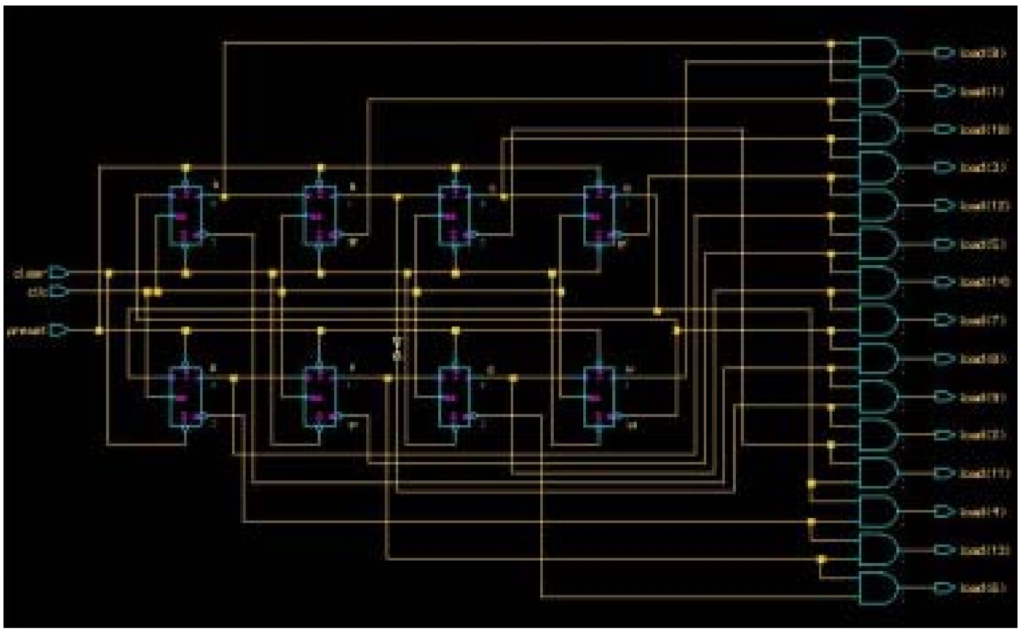


Figure 3.8. The roundkey controller circuit implementation

Once the output state of the roundkey controller is generated, it is passed on to the control register. There are two inputs used in the control register, and they are the output state from the roundkey controller, and the output of the 256-bit data passed from the main architecture. As a result, each output of the 256-bit is matched with the roundkey controller’s output state in order to perform the final 16 roundkeys used for AES encryption/decryption process. Note that, there are 16 final roundkey outputs, and the roundkey (0) should not be used in the AES encryption/decryption design.

3.2 The Experiment Results

Our design is sketched in the Design Architect (DA) software provided by Mentor Graphics. After getting the overall implementation shown in DA (see Figure 3.1), all the functions are accomplished in a Very High Speed Integrated Circuit

(VHSIC) Hardware Description Language (VHDL) environment. Eight VHDL coding files are used in order to implement our design and they are listed in the Appendix section. Every function is tested in the Modelsim, and the testing process is presented in the Figure 3.9 - 12.

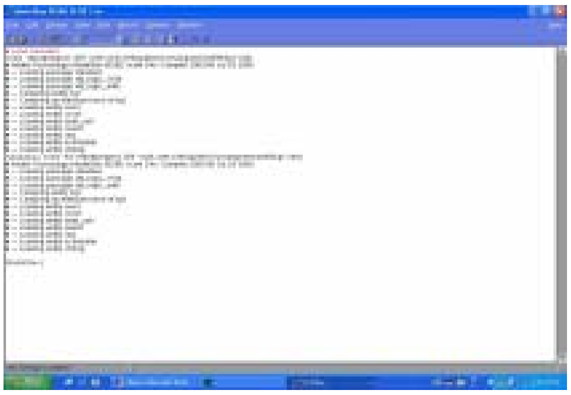


Figure 3.9. The simulating process in Modelsim

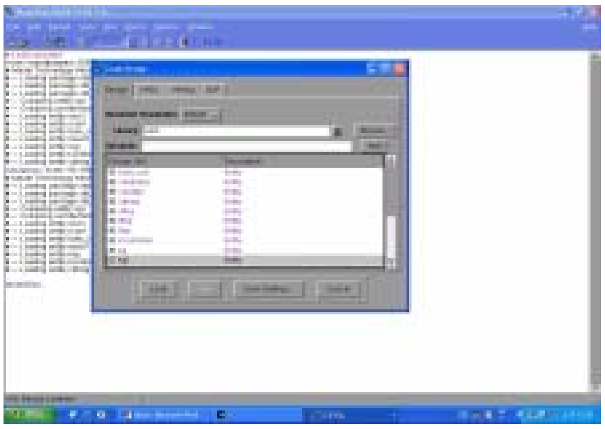


Figure 3.10. The load process in Modelsim

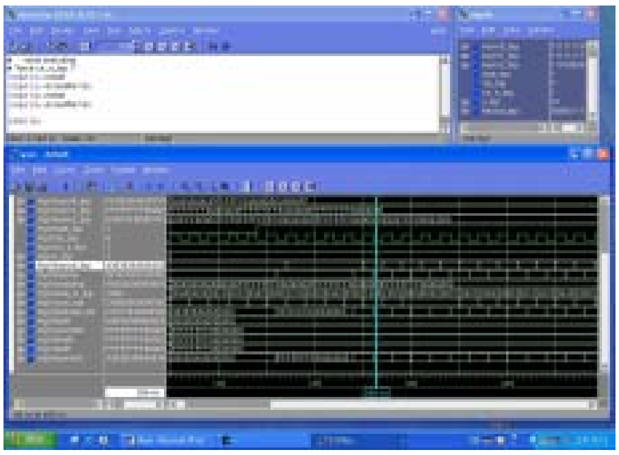


Figure 3.11. The simulation process in Modelsim

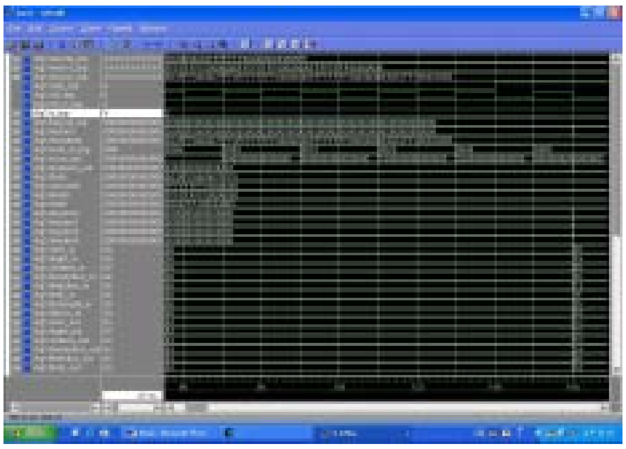


Figure 3.12. The simulated waveform shown in Modelsim

After getting the results from the Modelsim simulating tool, the simulated waveforms show all the final roundkey outputs operated on different key lengths as we planed. In our design, every cycle takes 20 ns to perform, which means it takes 120 ns in order to generate one round. As mentioned in the previous section, the first roundkey is not to be used. In that case, the actually roundkey is designed to be perform after 4 clock cycles in order get the first output roundkey. Then, the roundkeys are generated congruously. The critical path of the key expansion is included in five registers, five XOR gates, and one 2-1 multiplexer (the SubWord data block is not included here.)

Next, our design is to be tested on the synthesis tool named "LeonardoSpectrum" provided by Mentor Graphics. This software is a high level design tool used for synthesizing CPLD, FPGA, and ASIC. It provides a variety of functions that can be used for different technologies. In our design, we use "Level 3" to synthesize our hardware implementation. Note that Level 3 supports the algorithms targeted on ASIC, FPGA and CPLD technologies.

The device used in our synthesis testing is a FPGA chip used the Spartan 3 technology designed from Xilinx. The testing clock cycle used in our synthesis is 100 MHz ,and the results are showed in the Appendix section. The synthesis results shows a 9.27 ns for the critical data path require time for our hardware implementation. A 0.27 ns is used for the slack time. Ten critical paths are used for testing in Leonardo Spectrum. The synthesizing circuits are presented in Figure 3.13 - 15. A speedup of 1.028 is calculated by our hardware implementation. Note that a design for a pure 256-bit key input or a design for multiple key inputs is not often published in the AES research. Most of the hardware implementations for AES roundkey generation are targeted to the 128-bit key input.

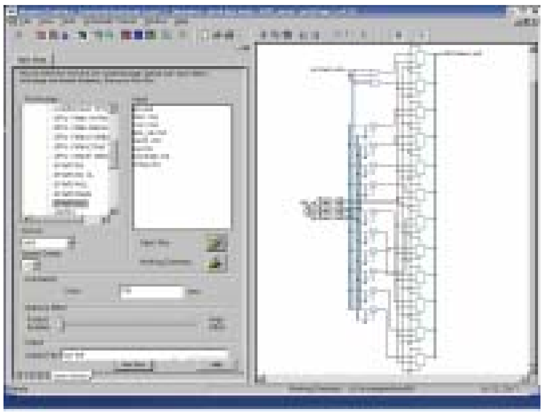


Figure 3.13. The synthesis result performed on a FPGA chip

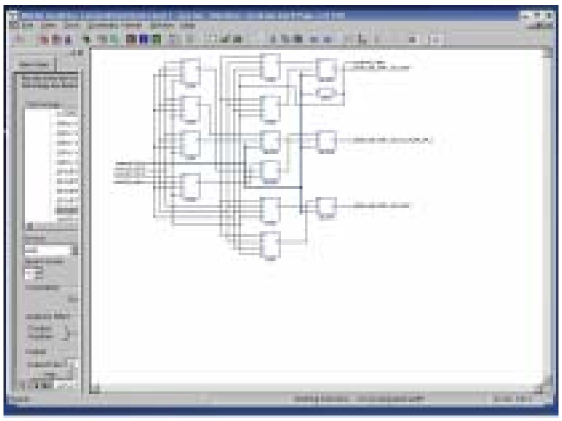


Figure 3.13. The synthesis result performed on the RTL level

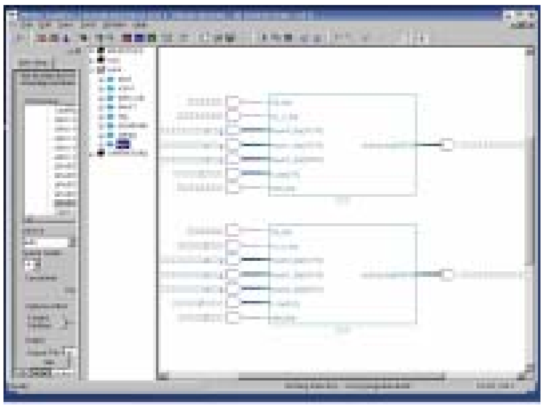


Figure 3.15. The final implementation block diagram

4. CONCLUSION

A completed subpipelined architecture of the AES key generation is presented in this thesis. We introduced and demonstrated that it is possible to obtain a subpipelined implementation with on-the-fly roundkey generation in AES algorithm. In addition, we derived a number of agendas applicable to implement the AES hardware design for multiple resource constraints. However, the results of our hardware implementation can not clearly demonstrate the best solution for designing for multiple resource constraints since not many publications or hardware implementations have experimented with the same criteria. The final conclusion of this study is that variable key length implementations using finite filed arithmetic can result a good performance and obtain a speedup of 1.028 in our study.

BIBLIOGRAPHY

[1] J. Daemen, and V. Rijmen, “AES Proposal: Rijndael,” version 2, 1999, http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf.

[2] National Institute of Standards and Technology (NIST), “Advanced Encryption Standard (AES),” FIPS Publication 197, November 26, 2001, http://csrc.nist.gov/encryption/aesindex.html.

[3] Computer Architecture: a Quantitative Approach, 3rd Edition. Hennessy, J.L., Patterson, D.A. Morgan Kaufmann, 2003, pp. A-2-4.

[4] X. Zhang, and K.K. Parhi. “Implementation Approaches for the Advanced Encryption Standard Algorithm,” *IEEE Circuits and Systems Magazine*, vol. 2, pp. 24-46, September 2002.

[5] H. Kuo, and I. Verbauwhede, “Architectural Optimization for a 1.82 Gbits/sec VLSI Implementation of the AES Rijndael Algorithm,” *Proc. CHES 2001*, Paris, France, May 2001, pp. 51-64.

[6] M. McLoone and J. B. McCanny, “Rijndael FPGA Implementation Utilizing look-up Tables,” *IEEE Workshop on Signal Processing Systems*, September 2001, pp. 349-360.

[7] National Institute of Standards and Technology (NIST), 2nd Advanced Encryption Standard (AES) Conference, Rome, Italy, March 1999.

[8] K.Gaj and P. Chodowiec, “Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware,” *Proc. 3rd AES Conf (AES3)*, http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/aes3papers.html.

[9] A. J. Elbert, W. Yip, B. Chetwynd, C. Parr, “An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists,” *Proc. 3rd AES Conf (AES3)*, http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/aes3papers.html.

[10] X. Zhang, and K.K. Parhi, “High-Speed VLSI Architectures for the AES Algorithm,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 12, pp. 957-967, September 2004.

APPENDICES

Performance Metrics of Our Code

Byte_Sub.vhd

```
-------------------------------------------
-- byte_sub.vhd
-- Wen-Chun Yang
-------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-- entity declaration
entity byte_sub is
port
(
        data_in: in std_logic_vector(7 downto 0);
        clk: in std_logic;
        data_out: out std_logic_vector( 7 downto 0)
);
end byte_sub;

-- architecture body
architecture byte_sub_architecture of byte_sub is

begin
with data_in(7 downto 0) select

data_out(7 downto 0) <=
                        -- first row
                        "01100011"    when "00000000",
                        "01111100"    when "00000001",
                        "01110111"    when "00000010",
                        "01111011"    when "00000011",
                        "11110010"    when "00000100",
                        "01101011"    when "00000101",
                        "01101111"    when "00000110",
                        "11000101"    when "00000111",
                        "00110000"    when "00001000",
                        "00000001" when "00001001",
                        "01100111" when "00001010",
                        "00101011" when "00001011",
                        "11111110" when "00001100",
                        "11010111" when "00001101",
                        "10101011" when "00001110",
                        "01110110" when "00001111",
                        -- second row
                        "11001010" when "00010000",
                        "10000010" when "00010001",
                        "11001001" when "00010010",
                        "01111101" when "00010011",
                        "11111010" when "00010100",
                        "01011001" when "00010101",
                        "01000111" when "00010110",
                        "11110000" when "00010111",
                        "10101101" when "00011000",
                        "11010100" when "00011001",
                        "10100010" when "00011010",
                        "10101111" when "00011011",
                        "10011100" when "00011100",
                        "10100100" when "00011101",
                        "01110010" when "00011110",
                        "11000000" when "00011111",
                        -- third row
                        "10110111" when "00100000",
                        "11111101" when "00100001",
                        "10010011" when "00100010",
                        "00100110" when "00100011",
                        "00110110" when "00100100",
                        "00111111"    when "00100101",
                        "11110111" when "00100110",
                        "11001100" when "00100111",
                        "00110100" when "00101000",
                        "10100101" when "00101001",
                        "11100101" when "00101010",
                        "11110001"   when "00101011",
                        "01110001" when "00101100",
                        "11011000"   when "00101101",
                        "00110001" when "00101110",
                        "00010101" when "00101111",
                        -- fourth row
                        "00000100" when "00110000",
                        "11000111" when "00110001",
                        "00100011" when "00110010",
                        "11000011" when "00110011",
                        "00011000" when "00110100",
                        "10010010"   when "00110101",
                        "00000101" when "00110110",
                        "10011010" when "00110111",
                        "00000111" when "00111000",
                        "00010010" when "00111001",
                        "10000000" when "00111010",
                        "11100010" when "00111011",
                        "11101011" when "00111100",
                        "00100111" when "00111101",
                        "10110010" when "00111110",
                        "01110101" when "00111111",
                        -- fifth row
                        "00001001" when "01000000",
                        "10000011" when "01000001",
                        "00101100" when "01000010",
                        "00011010" when "01000011",
                        "00011011" when "01000100",
                        "01101110" when "01000101",
                        "01011010" when "01000110",
                        "10100000" when "01000111",
                        "01010010" when "01001000",
                        "00111011" when "01001001",
                        "11010110" when "01001010",
                        "10110011" when "01001011",
                        "00101001" when "01001100",
                        "11100011" when "01001101",
                        "00101111" when "01001110",
                        "10000100" when "01001111",
                        -- sixth row
                        "01010011" when "01010000",
                        "11010001" when "01010001",
                        "00000000" when "01010010",
                        "11101101" when "01010011",
                        "00100000" when "01010100",
                        "11111100" when "01010101",
                        "10110001" when "01010110",
                        "01011011" when "01010111",
                        "01101010" when "01011000",
                        "11001011" when "01011001",
                        "10111110" when "01011010",
                        "00111001" when "01011011",
                        "01001010" when "01011100",
                        "01001100" when "01011101",
                        "01011000" when "01011110",
                        "11001111" when "01011111",
                        -- seventh row
                        "11010000" when "01100000",
                        "11101111" when "01100001",
                        "10101010" when "01100010",
                        "11111011" when "01100011",
                        "01000011" when "01100100",
                        "01001101" when "01100101",
                        "00110011" when "01100110",
                        "10000101" when "01100111",
                        "01000101" when "01101000",
                        "11111001" when "01101001",
                        "00000010" when "01101010",
                        "01111111" when "01101011",
                        "01010000" when "01101100",
                        "00111100" when "01101101",
                        "10011111" when "01101110",
                        "10101000" when "01101111",
                        -- eighth row
                        "01010001" when "01110000",
                        "10100011" when "01110001",
                        "01000000" when "01110010",
                        "10001111" when "01110011",
                        "10010010" when "01110100",
                        "10011101" when "01110101",
                        "00111000" when "01110110",
                        "11110101" when "01110111",
                        "10111100" when "01111000",
                        "10110110" when "01111001",
                        "11011010" when "01111010",
                        "00100001" when "01111011",
                        "00010000" when "01111100",
                        "11111111" when "01111101",
                        "11110011" when "01111110",
                        "11010010" when "01111111",
                        -- ninth row
                        "11001101" when "10000000",
                        "00001100" when "10000001",
                        "00010011" when "10000010",
                        "11101100" when "10000011",
                        "01011111" when "10000100",
                        "10010111" when "10000101",
                        "01000100" when "10000110",
                        "00010111" when "10000111",
                        "11000100"    when "10001000",
                        "10100111" when "10001001",
                        "01111110" when "10001010",
                        "00111101" when "10001011",
                        "01100100" when "10001100",
                        "01011101" when "10001101",
                        "00011001" when "10001110",
                        "01110011" when "10001111",
                        -- tenth row
                        "01100000" when "10010000",
                        "10000001" when "10010001",
                        "01001111" when "10010010",
                        "11011100" when "10010011",
                        "00100010" when "10010100",
                        "00101010" when "10010101",
                        "10010000" when "10010110",
                        "10001000" when "10010111",
                        "01000110" when "10011000",
                        "11101110" when "10011001",
                        "10111000" when "10011010",
                        "00010100" when "10011011",
                        "11011110" when "10011100",
                        "01011101" when "10011101",
                        "00001011" when "10011110",
                        "11011011" when "10011111",
                        -- eleventh row
                        "11100000" when "10100000",
                        "00110010" when "10100001",
                        "00111010" when "10100010",
                        "00001010" when "10100011",
                        "01001001" when "10100100",
                        "00000110" when "10100101",
                        "00100100" when "10100110",
                        "01011100" when "10100111",
                        "11000010" when "10101000",
                        "11010011" when "10101001",
                        "10101100" when "10101010",
                        "01100010" when "10101011",
                        "10010001" when "10101100",
                        "10010101" when "10101101",
                        "11100100" when "10101110",
                        "01111001" when "10101111",
                        -- twelveth row
                        "11100111" when "10110000",
                        "11001000" when "10110001",
                        "00110111" when "10110010",
                        "01101101" when "10110011",
                        "10001101" when "10110100",
                        "11010101" when "10110101",
                        "01001110" when "10110110",
                        "10101001" when "10110111",
                        "01101100" when "10111000",
                        "01010110" when "10111001",
                        "11110100" when "10111010",
                        "11101010" when "10111011",
                        "01100101" when "10111100",
                        "01111010" when "10111101",
                        "10101110" when "10111110",
                        "00001000" when "10111111",
                        -- thirteenth row
                        "10111010" when "11000000",
                        "01111000" when "11000001",
                        "00100101" when "11000010",
                        "01011110" when "11000011",
                        "00011100" when "11000100",
                        "10100110" when "11000101",
                        "10110100" when "11000110",
                        "11000110" when "11000111",
                        "11101000" when "11001000",
                        "11011101" when "11001001",
                        "01110100" when "11001010",
                        "00011111" when "11001011",
                        "01001011" when "11001100",
                        "10111101" when "11001101",
                        "10001011" when "11001110",
                        "10001010" when "11001111",
                        -- fourteenth row
                        "01110000" when "11010000",
                        "00111110" when "11010001",
                        "10110101" when "11010010",
                        "01100110" when "11010011",
                        "01001000" when "11010100",
                        "00000011" when "11010101",
                        "11110110" when "11010110",
                        "00001110" when "11010111",
                        "01100001" when "11011000",
                        "00110101" when "11011001",
                        "01010111" when "11011010",
                        "10111001"    when "11011011",
                        "10000110" when "11011100",
                        "11000001" when "11011101",
                        "00011101" when "11011110",
                        "10011110" when "11011111",
                        -- fifteen row
                        "11100001" when "11100000",
                        "11111000" when "11100001",
                        "10011000" when "11100010",
                        "00010001" when "11100011",
                        "01101001" when "11100100",
                        "11011001" when "11100101",
                        "10001110" when "11100110",
                        "10010100" when "11100111",
                        "10011011" when "11101000",
                        "00011110" when "11101001",
                        "10000111" when "11101010",
                        "11101001" when "11101011",
                        "11001110" when "11101100",
                        "01010101" when "11101101",
                        "00101000" when "11101110",
                        "11011111" when "11101111",
                        -- sixteen row
                        "10001100" when "11110000",
                        "10100001" when "11110001",
                        "10001001" when "11110010",
                        "00001101" when "11110011",
                        "10111111" when "11110100",
                        "11100110" when "11110101",
                        "01000010" when "11110110",
                        "01101000" when "11110111",
                        "01000001" when "11111000",
                        "10011001" when "11111001",
                        "00101101" when "11111010",
                        "00001111" when "11111011",
                        "10111101" when "11111100",
                        "01010100" when "11111101",
                        "10111011" when "11111110",
                        "00010110" when "11111111",

                                "XXXXXXXX" when others;
end byte_sub_architecture;
```

Ctrlreg.vhd

```
----------------------------------
-- ctrlreg.vhd
-- Wen-Chun Yang
----------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity ctrlreg is
PORT
```

```
        (
                keyin: in std_logic_vector (255 downto 0);
                load_in : in std_logic_vector (15 downto 0);
                clk, clr_n: in std_logic;
                keyout: out std_logic_vector(255 downto 0)
        );
end ctrlreg;

architecture bech of ctrlreg is
signal y_ps: std_logic_vector(255 downto 0);

begin
        Process(clk,clr_n)
        BEGIN
                        if (clr_n = '0') then
                                y_ps <= (others=>'0');
                        elsif(clk'event and clk='1') then
                                y_ps <=keyin;
                        end if;
        end Process;

        process(load_in, keyin)
        begin
               keyout <= (others =>'0');
                case load_in is
                        when "0000000000000001" =>
                                keyout <= keyin;
                        when "0000000000000010" =>
                                keyout <= keyin;
                        when "0000000000000100" =>
                                keyout <= keyin;
                        when "0000000000001000" =>
                                keyout <= keyin;
                        when "0000000000010000" =>
                                keyout <= keyin;
                        when "0000000000100000" =>
                                keyout <= keyin;
                        when "0000000001000000" =>
                                keyout <= keyin;
                        when "0000000010000000" =>
                                keyout <= keyin;
                        when "0000000100000000" =>
                                keyout <= keyin;
                        when "0000001000000000" =>
                                keyout <= keyin;
                        when "0000010000000000" =>
                                keyout <= keyin;
                        when "0000100000000000" =>
                                keyout <= keyin;
                        when "0001000000000000" =>
                                keyout <= keyin;
                        when "0010000000000000" =>
                                keyout <= keyin;
                        when "0100000000000000" =>
                                keyout <= keyin;
                        when "1000000000000000" =>
                                keyout <= keyin;
                        when others =>
                                keyout <= (others=>'0');
                end case;
        end process;

        keyout <= y_ps;                         -- concurrent statement

end bech;
```

Kcontroller.vhd

```

-----------------------------------------
```

```
-- kcontroller.vhd
-- Wen-Chun Yang
-----------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity kcontroller is
port(
            clk, clr_n: in std_logic;
            Q_vector: buffer std_logic_vector(7 downto 0);
            load: out std_logic_vector(15 downto 0)
      );
end kcontroller;

architecture bech of kcontroller is
signal temp : std_logic;

begin

     --temp <= not Q_vector(7);

      process(clk, clr_n, Q_vector, temp)
            begin
            if (clr_n ='0') then
                  Q_vector<=(others=>'0');
            elsif (clk='1' and clk'event) then
                  for I in 6 downto 0
                  loop
                        Q_vector(I+1)<= Q_vector(I);
                  end loop;
                  Q_vector(0) <= not Q_vector(7);
            end if;
      end process;
```

```
process(Q_vector)
begin
    load <= (others =>'0');
     case Q_vector is
          when "00000000" =>
               load(0)<= (not Q_vector(0) and not Q_vector(7));
          when "10000000"=>
               load(1)<= (Q_vector(0) and not Q_vector(1));
          when "11000000"=>
               load(2)<= (Q_vector(1) and not Q_vector(2));
          when "11100000"=>
               load(3)<= (Q_vector(2) and not Q_vector(3));
          when "11110000"=>
               load(4)<= (Q_vector(3) and not Q_vector(4));
          when "11111000"=>
               load(5)<= (Q_vector(4) and not Q_vector(5));
          when "11111100"=>
               load(6)<= (Q_vector(5) and not Q_vector(6));
          when "11111110"=>
               load(7)<= (Q_vector(6) and not Q_vector(7));
          when "11111111"=>
               load(8)<= (Q_vector(0) and Q_vector(7));
          when "01111111"=>
               load(9)<= (not Q_vector(0) and Q_vector(1));
          when "00111111"=>
               load(10)<= (not Q_vector(1) and Q_vector(2));
          when "00011111"=>
               load(11)<= (not Q_vector(2) and Q_vector(3));
          when "00001111"=>
               load(12)<= (not Q_vector(3) and Q_vector(4));
          when "00000111"=>
               load(13)<= (not Q_vector(4) and Q_vector(5));
          when "00000011"=>
```

```
                    load(14)<= (not Q_vector(5) and Q_vector(6));
                when "00000001"=>
                    load(15)<= (not Q_vector(6) and Q_vector(7));
                when others =>
                    load <= (others=>'0');
            end case;
        end process;
end bech;
```

Mux1.vhd

```
----------------------------------------
-- mux1.vhd
-- Wen-Chun Yang
----------------------------------------



library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;



entity mux1 is
port(
        I3: in std_logic_vector(127 downto 0);
        I2: in std_logic_vector(255 downto 0);
        I1: in std_logic_vector(191 downto 0);
        I0: in std_logic_vector(127 downto 0);
        S: in std_logic_vector(1 downto 0);
        clk, clr_n : in std_logic;
        O: out std_logic_vector(255 downto 0)
);
end mux1;

architecture behavior of mux1 is
```

```
signal t1 : std_logic_vector(127 downto 0) := (OTHERS => '0');
signal t2 : std_logic_vector(63 downto 0) := (OTHERS => '0');

begin
	Process(clk,clr_n, S)
	BEGIN
			if (clr_n = '0') then
				O <= (others=>'0');
			elsif(clk'event and clk='1') then
				if (S = "00") then
					O <= t1 & I0;
				elsif (S="01") then
					O <= t2 & I1;
				elsif (S="10") then
					O <= I2;
				else
					O <= t1 & I3;
				end if;
			end if;
	end process;
end behavior;
```

Mux21.vhd

```
----------------------------------------
-- mux21.vhd
-- Wen-Chun Yang
----------------------------------------



library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

```
entity mux21 is
port(
            I1,I0: in std_logic_vector(63 downto 0);
            s,clk,clr_n: in std_logic;
            o: out std_logic_vector(63 downto 0)
      );
end mux21;

Architecture bech of mux21 is
begin
      Process(clk,clr_n)
      BEGIN
                  if (clr_n = '0') then
                        o <= (others=>'0');
                  elsif(clk'event and clk='1') then
                        if (s = '0') then
                              o <= I0;
                        else
                              o <= I1;
                        end if;
                  end if;
      end Process;
end bech;
```

Reg.vhd

```
--------------------------------
-- 64-bit register
-- Wen-Chun Yang
--------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;


entity reg is
```

```
port(
            clk, clr_n    : in std_logic;
            input: in std_logic_vector( 63 downto 0);
            output: out std_logic_vector(63 downto 0)
      );
end reg;

architecture bech of reg is
signal y_ps: std_logic_vector(63 downto 0);

BEGIN


      Process(clk,clr_n)
      BEGIN
                  if (clr_n = '0') then
                        y_ps <= (others=>'0');
                  elsif(clk'event and clk='1') then
                        y_ps <=input;
                  end if;
      end Process;

      output <= y_ps;                         -- concurrent statement
end bech;
```

The synthesis result from LeonardoSpectrum

```
*******************************************************
Cell: kg1     View: bech     Library: work
*******************************************************
Total accumulated area :
 Number of BUFGP :                              1
 Number of Dffs or Latches :             1800
 Number of Function Generators :         2091
 Number of GND :                                1
 Number of IBUF :                             580
```

```
  Number of MUXF5 :                        644
  Number of MUXF6 :                        256
  Number of MUXF7 :                        128
  Number of MUXF8 :                         64
  Number of OBUF :                         256
  Number of gates :                      2010
  Number of accumulated instances :    5821
  Number of ports :                        837
  Number of nets :                       5570
  Number of instances :                  3219
  Number of references to this view :      0

  Cell       Library   References      Total Area
BUFGP         xis3       1 x        1         1 BUFGP
FDC           xis3       8 x        1         8 Dffs or Latches
IBUF          xis3     580 x        1       580 IBUF
LUT1          xis3      66 x        1        66 Function Generators
LUT2          xis3      78 x        1        78 Function Generators
LUT3          xis3      70 x        1        70 Function Generators
LUT4          xis3    1175 x        1      1175 Function Generators
MUXF5         xis3     515 x        1       515 MUXF5
MUXF6         xis3     256 x        1       256 MUXF6
MUXF7         xis3     128 x        1       128 MUXF7
MUXF8         xis3      64 x        1        64 MUXF8
OBUF          xis3     256 x        1       256 OBUF
ctrlreg     work       1 x       24        24 gates
                                    1         1 GND
                                   28        28 Function Generators
                                    1         1 MUXF5
                                  256       256 Dffs or Latches
mux1          work       1 x      385       385 gates
                                  394       394 Function Generators
                                  128       128 MUXF5
                                  256       256 Dffs or Latches
mux21         work       4 x       65       260 gates
```

```
                                        66      264 Function Generators
                                        64      256 Dffs or Latches
reg          work     16 x        1      16 gates
                                         1       16 Function Generators
                                        64     1024 Dffs or Latches

*****************************************************
Cell: mux1     View: behavior     Library: work
*****************************************************
Total accumulated area :
  Number of Dffs or Latches :              256
  Number of Function Generators :          394
  Number of MUXF5 :                        128
  Number of gates :                        385
  Number of accumulated instances :        778
  Number of ports :                        836
  Number of nets :                        1358
  Number of instances :                    778
  Number of references to this view :        1

  Cell      Library   References     Total Area
FDC         xis3     256 x        1     256 Dffs or Latches
LUT1        xis3      10 x        1      10 Function Generators
LUT2        xis3     128 x        1     128 Function Generators
LUT3        xis3     192 x        1     192 Function Generators
LUT4        xis3      64 x        1      64 Function Generators
MUXF5       xis3     128 x        1     128 MUXF5

*****************************************************
Cell: mux21     View: bech     Library: work
*****************************************************
Total accumulated area :
  Number of Dffs or Latches :               64
  Number of Function Generators :           66
  Number of gates :                         65
  Number of accumulated instances :        130
  Number of ports :                            195
  Number of nets :                              261
  Number of instances :                       130
  Number of references to this view :       4

  Cell       Library   References        Total Area
FDC         xis3       64 x         1        64 Dffs or Latches
LUT1       xis3         2 x         1          2 Function Generators
LUT3       xis3       64 x         1        64 Function Generators

*******************************************************
Cell: reg       View: bech       Library: work
*******************************************************
Total accumulated area :
  Number of Dffs or Latches :                  64
  Number of Function Generators :              1
  Number of gates :                                  1
  Number of accumulated instances :          65
  Number of ports :                              130
  Number of nets :                                131
  Number of instances :                          65
  Number of references to this view :       16
  Cell       Library   References        Total Area
FDC         xis3       64 x         1        64 Dffs or Latches
LUT1       xis3         1 x         1          1 Function Generators

*******************************************************
Cell: ctrlreg       View: bech       Library: work
*******************************************************
Total accumulated area :
  Number of Dffs or Latches :                256
  Number of Function Generators :            28
  Number of GND :                                     1
  Number of MUXF5 :                                  1
  Number of gates :                                   24
  Number of accumulated instances :         286
  Number of ports :                                  530
  Number of nets :                                   560
  Number of instances :                            286
  Number of references to this view :         1

  Cell       Library    References         Total Area
FDCE         xis3     256 x          1       256 Dffs or Latches
GND           xis3        1 x          1          1 GND
LUT1         xis3         3 x          1          3 Function Generators
LUT2         xis3         6 x          1          6 Function Generators
LUT3         xis3         1 x          1          1 Function Generators
LUT4         xis3       18 x          1        18 Function Generators
MUXF5        xis3         1 x          1          1 MUXF5

WARNING:
No resource information is available for the part specified. Please check the specified part name 'auto'.

Using default wire table: STD
                                      Clock Frequency Report
        Clock                            : Frequency
          ------------------------------------
        clk_top                        : 102.8 MHz
                                      Slack Table at End Points

End points                                          Slack          Arrival                     Required
                                                                        rise          fall          rise          fall


ctrlreg_unit/reg_keyout(0)/CE    :      0.27           9.10      9.10           9.37      9.37
ctrlreg_unit/reg_keyout(128)/CE :      0.27           9.10      9.10           9.37      9.37
ctrlreg_unit/reg_keyout(129)/CE :      0.27           9.10      9.10           9.37      9.37
ctrlreg_unit/reg_keyout(171)/CE :      0.27           9.10      9.10           9.37      9.37
ctrlreg_unit/reg_keyout(170)/CE :      0.27           9.10      9.10           9.37      9.37
ctrlreg_unit/reg_keyout(84)/CE :      0.27           9.10      9.10           9.37      9.37
```

```
ctrlreg_unit/reg_keyout(83)/CE :      0.27          9.10      9.10          9.37      9.37
ctrlreg_unit/reg_keyout(172)/CE :      0.27          9.10      9.10          9.37      9.37
ctrlreg_unit/reg_keyout(81)/CE :      0.27          9.10      9.10          9.37      9.37
ctrlreg_unit/reg_keyout(80)/CE :      0.27          9.10      9.10          9.37      9.37


                              Critical Path Report
Critical path #1, (path slack =     0.3):
NAME                                    GATE                    ARRIVAL                     LOAD
-----------------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                                                  0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(1)/Q      FDC             0.00    0.62 up                  10.00
kcontroller_unit_modgen_eq_472_nx20/O    LUT4            0.85    1.47 up                  4.50
nx6181/O                                   LUT4             0.85    2.31 up                  2.40
load_in(1)/O                                LUT2             0.85    3.16 up                  2.70
ctrlreg_unit/modgen_eq_503_nx36/O          LUT4             0.85    4.00 up                  2.40
ctrlreg_unit/ix2690/O                       MUXF5            0.86    4.87 up                  1.30
ctrlreg_unit/nx2681/O                      LUT4             0.85    5.71 up                  1.30
ctrlreg_unit/nx2680/O                      LUT4             0.85    6.56 up                  1.30
ctrlreg_unit/nx2677/O                       LUT4             0.85    7.40 up                  1.30
ctrlreg_unit/nx2684/O                      LUT4             0.85    8.25 up                  2.70
ctrlreg_unit/sel_keyin_repl1/O              LUT4             0.85    9.10 up                  10.00
ctrlreg_unit/reg_keyout(129)/CE            FDCE             0.00    9.10 up                  0.00
data arrival time                                                                          9.10
data required time    (default specified - setup time)                            9.37
-----------------------------------------------------------------------------------------------
data required time                                                                         9.37
data arrival time                                                                          9.10
                                                                                        ----------
slack                                                                                          0.27
-----------------------------------------------------------------------------------------------
Critical path #2, (path slack =     0.3):
NAME                          GATE                    ARRIVAL                     LOAD
-----------------------------------------------------------------------------------------------
clock information not specified
```

```
delay thru clock network                                                           0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(2)/Q      FDC          0.00    0.62 up           10.00
kcontroller_unit_modgen_eq_472_nx20/O       LUT4         0.85    1.47 up            4.50
nx6181/O                                    LUT4         0.85    2.31 up            2.40
load_in(1)/O                                LUT2         0.85    3.16 up            2.70
ctrlreg_unit/modgen_eq_503_nx36/O           LUT4         0.85    4.00 up            2.40
ctrlreg_unit/ix2690/O                       MUXF5        0.86    4.87 up            1.30
ctrlreg_unit/nx2681/O                       LUT4         0.85    5.71 up            1.30
ctrlreg_unit/nx2680/O                       LUT4         0.85    6.56 up            1.30
ctrlreg_unit/nx2677/O                       LUT4         0.85    7.40 up            1.30
ctrlreg_unit/nx2684/O                       LUT4         0.85    8.25 up            2.70
ctrlreg_unit/sel_keyin_repl1/O              LUT4         0.85    9.10 up           10.00
ctrlreg_unit/reg_keyout(129)/CE             FDCE         0.00    9.10 up            0.00
data arrival time                                                                   9.10
data required time    (default specified - setup time)                      9.37
--------------------------------------------------------------------------------------------------
data required time                                                                  9.37
data arrival time                                                                   9.10
                                                                                 ----------
slack                                                                                0.27
--------------------------------------------------------------------------------------------------
Critical path #3, (path slack =    0.3):
NAME                        GATE                  ARRIVAL                    LOAD
--------------------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                                           0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(3)/Q      FDC          0.00    0.62 up           10.00
kcontroller_unit_modgen_eq_472_nx20/O       LUT4         0.85    1.47 up            4.50
nx6181/O                                    LUT4         0.85    2.31 up            2.40
load_in(1)/O                                LUT2         0.85    3.16 up            2.70
ctrlreg_unit/modgen_eq_503_nx36/O           LUT4         0.85    4.00 up            2.40
ctrlreg_unit/ix2690/O                       MUXF5        0.86    4.87 up            1.30
ctrlreg_unit/nx2681/O                       LUT4         0.85    5.71 up            1.30
ctrlreg_unit/nx2680/O                       LUT4         0.85    6.56 up            1.30
ctrlreg_unit/nx2677/O                       LUT4         0.85    7.40 up            1.30
```

```
ctrlreg_unit/nx2684/O                      LUT4        0.85   8.25 up              2.70
ctrlreg_unit/sel_keyin_repl1/O             LUT4        0.85   9.10 up              10.00
ctrlreg_unit/reg_keyout(129)/CE            FDCE        0.00   9.10 up              0.00
data arrival time                                                                9.10
data required time    (default specified - setup time)                  9.37
---------------------------------------------------------------------------------------------------
data required time                                                               9.37
data arrival time                                                                9.10
                                                                               ----------
slack                                                                              0.27
---------------------------------------------------------------------------------------------------
Critical path #4, (path slack =    0.3):
NAME                   GATE                  ARRIVAL                  LOAD
---------------------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                                         0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(0)/Q     FDC         0.00   0.62 up              10.00
kcontroller_unit_modgen_eq_472_nx20/O      LUT4        0.85   1.47 up              4.50
nx6181/O                                   LUT4        0.85   2.31 up              2.40
load_in(1)/O                               LUT2        0.85   3.16 up              2.70
ctrlreg_unit/modgen_eq_503_nx36/O          LUT4        0.85   4.00 up              2.40
ctrlreg_unit/ix2690/O                      MUXF5       0.86   4.87 up              1.30
ctrlreg_unit/nx2681/O                      LUT4        0.85   5.71 up              1.30
ctrlreg_unit/nx2680/O                      LUT4        0.85   6.56 up              1.30
ctrlreg_unit/nx2677/O                      LUT4        0.85   7.40 up              1.30
ctrlreg_unit/nx2684/O                      LUT4        0.85   8.25 up              2.70
ctrlreg_unit/sel_keyin_repl1/O             LUT4        0.85   9.10 up              10.00
ctrlreg_unit/reg_keyout(128)/CE            FDCE        0.00   9.10 up              0.00
data arrival time                                                                9.10
data required time    (default specified - setup time)                  9.37
---------------------------------------------------------------------------------------------------
data required time                                                               9.37
data arrival time                                                                9.10
                                                                               ----------
slack                                                                              0.27
```

```
-----------------------------------------------------------------------------------------------------
Critical path #5, (path slack =    0.3):
NAME                          GATE                        ARRIVAL                      LOAD
-----------------------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                                                    0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(1)/Q         FDC              0.00    0.62 up                    10.00
kcontroller_unit_modgen_eq_472_nx20/O          LUT4             0.85    1.47 up                     4.50
nx6181/O                                       LUT4             0.85    2.31 up                     2.40
load_in(1)/O                                   LUT2             0.85    3.16 up                     2.70
ctrlreg_unit/modgen_eq_503_nx36/O              LUT4             0.85    4.00 up                     2.40
ctrlreg_unit/ix2690/O                          MUXF5            0.86    4.87 up                     1.30
ctrlreg_unit/nx2681/O                          LUT4             0.85    5.71 up                     1.30
ctrlreg_unit/nx2680/O                          LUT4             0.85    6.56 up                     1.30
ctrlreg_unit/nx2677/O                          LUT4             0.85    7.40 up                     1.30
ctrlreg_unit/nx2684/O                          LUT4             0.85    8.25 up                     2.70
ctrlreg_unit/sel_keyin_repl1/O                 LUT4             0.85    9.10 up                    10.00
ctrlreg_unit/reg_keyout(128)/CE                FDCE             0.00    9.10 up                     0.00
data arrival time                                                                              9.10
data required time    (default specified - setup time)                                9.37
-----------------------------------------------------------------------------------------------------
data required time                                                                             9.37
data arrival time                                                                              9.10
                                                                                            ----------
slack                                                                                              0.27
-----------------------------------------------------------------------------------------------------
Critical path #6, (path slack =    0.3):
NAME                          GATE                        ARRIVAL                      LOAD
-----------------------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                                                    0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(2)/Q         FDC              0.00    0.62 up                    10.00
kcontroller_unit_modgen_eq_472_nx20/O          LUT4             0.85    1.47 up                     4.50
nx6181/O                                       LUT4             0.85    2.31 up                     2.40
load_in(1)/O                                   LUT2             0.85    3.16 up                     2.70
```

```
ctrlreg_unit/modgen_eq_503_nx36/O           LUT4        0.85   4.00 up              2.40
ctrlreg_unit/ix2690/O                       MUXF5       0.86   4.87 up              1.30
ctrlreg_unit/nx2681/O                        LUT4       0.85   5.71 up              1.30
ctrlreg_unit/nx2680/O                       LUT4        0.85   6.56 up              1.30
ctrlreg_unit/nx2677/O                       LUT4        0.85   7.40 up              1.30
ctrlreg_unit/nx2684/O                       LUT4        0.85   8.25 up              2.70
ctrlreg_unit/sel_keyin_repl1/O               LUT4       0.85   9.10 up              10.00
ctrlreg_unit/reg_keyout(128)/CE              FDCE       0.00   9.10 up              0.00
data arrival time                                                              9.10
data required time    (default specified - setup time)                  9.37
-----------------------------------------------------------------------------------------------
data required time                                                             9.37
data arrival time                                                              9.10
                                                                             ----------
slack                                                                             0.27
-----------------------------------------------------------------------------------------------
Critical path #7, (path slack =    0.3):
NAME                          GATE                 ARRIVAL                   LOAD
-----------------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                                       0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(3)/Q      FDC         0.00   0.62 up              10.00
kcontroller_unit_modgen_eq_472_nx20/O    LUT4        0.85   1.47 up              4.50
nx6181/O                                  LUT4        0.85   2.31 up              2.40
load_in(1)/O                             LUT2        0.85   3.16 up              2.70
ctrlreg_unit/modgen_eq_503_nx36/O        LUT4        0.85   4.00 up              2.40
ctrlreg_unit/ix2690/O                     MUXF5       0.86   4.87 up              1.30
ctrlreg_unit/nx2681/O                     LUT4        0.85   5.71 up              1.30
ctrlreg_unit/nx2680/O                    LUT4        0.85   6.56 up              1.30
ctrlreg_unit/nx2677/O                     LUT4        0.85   7.40 up              1.30
ctrlreg_unit/nx2684/O                      LUT4        0.85   8.25 up              2.70
ctrlreg_unit/sel_keyin_repl1/O            LUT4        0.85   9.10 up              10.00
ctrlreg_unit/reg_keyout(128)/CE           FDCE        0.00   9.10 up              0.00
data arrival time                                                              9.10
data required time    (default specified - setup time)                  9.37
```

```
--------------------------------------------------------------------------------------------------
data required time                                                                        9.37
data arrival time                                                                         9.10
                                                                                          ----------
slack                                                                                        0.27
--------------------------------------------------------------------------------------------------
Critical path #8, (path slack =    0.3):
NAME                    GATE                     ARRIVAL                     LOAD
--------------------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                                                  0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(0)/Q        FDC              0.00    0.62 up              10.00
kcontroller_unit_modgen_eq_472_nx20/O         LUT4             0.85    1.47 up               4.50
nx6181/O                                      LUT4             0.85    2.31 up               2.40
load_in(1)/O                                  LUT2             0.85    3.16 up               2.70
ctrlreg_unit/modgen_eq_503_nx36/O             LUT4             0.85    4.00 up               2.40
ctrlreg_unit/ix2690/O                         MUXF5            0.86    4.87 up               1.30
ctrlreg_unit/nx2681/O                         LUT4             0.85    5.71 up               1.30
ctrlreg_unit/nx2680/O                         LUT4             0.85    6.56 up               1.30
ctrlreg_unit/nx2677/O                         LUT4             0.85    7.40 up               1.30
ctrlreg_unit/nx2684/O                         LUT4             0.85    8.25 up               2.70
ctrlreg_unit/sel_keyin/O                      LUT4             0.85    9.10 up              10.00
ctrlreg_unit/reg_keyout(0)/CE                 FDCE             0.00    9.10 up               0.00
data arrival time                                                                         9.10
data required time    (default specified - setup time)                             9.37
--------------------------------------------------------------------------------------------------
data required time                                                                        9.37
data arrival time                                                                           9.10
                                                                                          ----------
slack                                                                                        0.27
--------------------------------------------------------------------------------------------------
Critical path #9, (path slack =    0.3):
NAME                    GATE                     ARRIVAL                     LOAD
--------------------------------------------------------------------------------------------------
clock information not specified
```

```
delay thru clock network                                                      0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(1)/Q        FDC          0.00    0.62 up          10.00
kcontroller_unit_modgen_eq_472_nx20/O         LUT4         0.85    1.47 up           4.50
nx6181/O                                      LUT4         0.85    2.31 up           2.40
load_in(1)/O                                  LUT2         0.85    3.16 up           2.70
ctrlreg_unit/modgen_eq_503_nx36/O             LUT4         0.85    4.00 up           2.40
ctrlreg_unit/ix2690/O                         MUXF5        0.86    4.87 up           1.30
ctrlreg_unit/nx2681/O                         LUT4         0.85    5.71 up           1.30
ctrlreg_unit/nx2680/O                         LUT4         0.85    6.56 up           1.30
ctrlreg_unit/nx2677/O                         LUT4         0.85    7.40 up           1.30
ctrlreg_unit/nx2684/O                         LUT4         0.85    8.25 up           2.70
ctrlreg_unit/sel_keyin/O                      LUT4         0.85    9.10 up          10.00
ctrlreg_unit/reg_keyout(0)/CE                 FDCE         0.00    9.10 up           0.00
data arrival time                                                                9.10
data required time    (default specified - setup time)                    9.37
------------------------------------------------------------------------------------------------
data required time                                                               9.37
data arrival time                                                                9.10
                                                                               ----------
slack                                                                            0.27
------------------------------------------------------------------------------------------------
Critical path #10, (path slack =    0.3):
NAME                    GATE                   ARRIVAL                  LOAD
------------------------------------------------------------------------------------------------
clock information not specified
delay thru clock network                                                      0.00 (ideal)
kcontroller_unit_reg_Q_vector_reg(2)/Q            FDC          0.00    0.62 up          10.00
kcontroller_unit_modgen_eq_472_nx20/O             LUT4         0.85    1.47 up           4.50
nx6181/O                                          LUT4         0.85    2.31 up           2.40
load_in(1)/O                                      LUT2         0.85    3.16 up           2.70
ctrlreg_unit/modgen_eq_503_nx36/O                 LUT4         0.85    4.00 up           2.40
ctrlreg_unit/ix2690/O                             MUXF5        0.86    4.87 up           1.30
ctrlreg_unit/nx2681/O                             LUT4         0.85    5.71 up           1.30
ctrlreg_unit/nx2680/O                             LUT4         0.85    6.56 up           1.30
ctrlreg_unit/nx2677/O                             LUT4         0.85    7.40 up           1.30
```

```
ctrlreg_unit/nx2684/O                          LUT4        0.85   8.25 up          2.70
ctrlreg_unit/sel_keyin/O                       LUT4        0.85   9.10 up         10.00
ctrlreg_unit/reg_keyout(0)/CE                  FDCE        0.00   9.10 up          0.00
data arrival time                                                             9.10
data required time    (default specified - setup time)              9.37
-----------------------------------------------------------------------------------------------------
data required time                                                           9.37
data arrival time                                                            9.10
                                                                             ----------
slack                                                                        0.27
-----------------------------------------------------------------------------------------------------
```