

2004

# A VHDL implementation of the Advanced Encryption Standard-Rijndael Algorithm

Rajender Manteena  
*University of South Florida*

Follow this and additional works at: <http://scholarcommons.usf.edu/etd>

 Part of the [American Studies Commons](#)

---

## Scholar Commons Citation

Manteena, Rajender, "A VHDL implementation of the Advanced Encryption Standard-Rijndael Algorithm" (2004). *Graduate Theses and Dissertations*.

<http://scholarcommons.usf.edu/etd/1149>

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact [scholarcommons@usf.edu](mailto:scholarcommons@usf.edu).

A VHDL Implementation of the Advanced Encryption Standard-Rijndael Algorithm

by

Rajender Manteena

A thesis submitted in partial fulfillment  
of the requirements for the degree of  
Master of Science in Electrical Engineering  
Department of Electrical Engineering  
College of Engineering  
University of South Florida

Major Professor: Wilfrido Moreno, Ph.D.  
James Leffew, Ph.D.  
Wei Qian, Ph.D.

Date of Approval:  
March 23, 2004

Keywords: decryption, state, cipher.

© Copyright 2004 , Rajender Manteena

## **ACKNOWLEDGEMENTS**

I would like to thank Dr. Moreno, my major professor, for his guidance. Dr. Moreno deserves my deepest gratitude for supporting me financially throughout my master degree program. I would like to thank Dr. Leffew who helped me to build this thesis. I also acknowledge with gratitude the contribution of journals, papers, organizations and books, which I have referred to in the pages of the references.

My parents and my wife play an important role in my life. To them I dedicate my thesis. Without their love, affection, motivation and support this thesis would not have been possible. Last but not the least I would like to thank my friends Narender, Nikhil, Bhasker who have always been with me throughout the work of this thesis.

## TABLE OF CONTENTS

LIST OF FIGURES	iii
ABSTRACT	v
CHAPTER 1 INTRODUCTION	1
1.1. Background of the Algorithm	1
1.2. Notation and Conventions	2
1.2.1. Inputs and Outputs	2
1.2.2. Bytes	3
1.2.3. Arrays of Bytes	4
1.2.4. The State	4
1.2.5. The State as an Array of Columns	6
1.3. Mathematical Background	6
1.3.1. Addition	6
1.3.2. Multiplication	7
1.3.3. Multiplication by $x$	9
1.3.4. Polynomials with Coefficients in $GF(2^8)$	10

CHAPTER 2 ENCRYPTION	13
2.1. Encryption Process	13
2.2. Bytes Substitution Transformation	14
2.3. Shift Rows Transformation	16
2.4. Mixing of Columns Transformation	17
2.5. Addition of Round Key Transformation	18
2.6. Key Schedule Generation	19
CHAPTER 3 DECRYPTION	21
3.1. Decryption Process	21
3.2. Inverse Bytes Substitution Transformation	22
3.3. Inverse Shift Rows Transformation	23
3.4. Inverse Mixing of Columns Transformation	24
CHAPTER 4 IMPLEMENTATION AND RESULTS	26
4.1. Encryption Implementation	26
4.2. Decryption Implementation	30
4.3. Hardware Implementation	32
4.4. Conclusions	33
REFERENCES	34
APPENDICES	35
Appendix A: Terms and Definitions	36
Appendix B: Cipher Example	38
Appendix C: Example Vectors	41
Appendix D: VHDL Design Code	56

## LIST OF FIGURES

Figure 1. Hexadecimal Representation of Bit Patterns	3
Figure 2. Indices for Bytes and Bits	4
Figure 3. State Array Input and Output	5
Figure 4. Encryption Process	13
Figure 5. Matrix Notation of S-box	15
Figure 6. Application of the S-box to Each Byte of the State	15
Figure 7. S-box Values for All 256 Combinations in Hexadecimal Format	16
Figure 8. Cyclic Shift of the Last Three Rows of the State	17
Figure 9. Mixing of Columns of the State	18
Figure 10. Exclusive-OR Operation of State and Cipher Key Words	19
Figure 11. Key-Block- Round Combinations	20
Figure 12. Decryption Process	21
Figure 13. Application of the Inverse S-box to Each Byte of the State	22
Figure 14. Inverse S-box Values for All 256 Combinations in Hexadecimal Format	23

Figure 15. Inverse Cyclic Shift of the Last Three Rows of the State	24
Figure 16. Inverse Mix Column Operation on State	25
Figure 17. Basic Characteristics of the ACEX1K Family Devices	27
Figure 18. Waveforms of 8-bit Byte Substitution	28
Figure 19. Waveforms of Shift Row Transformation	28
Figure 20. Waveforms of Mix Column Transformation	29
Figure 21. Waveforms of Key Schedule Generation	29
Figure 22. Waveforms of 8-bit Inverse Byte Substitution	30
Figure 23. Waveforms of Inverse Shift Row Transformation	31
Figure 24. Waveforms of Inverse Mix Column Transformation	31
Figure 25. Block Diagram of AES Hardware Implementation	32

**A VHDL IMPLEMENTATION OF THE ADVANCED ENCRYPTION  
STANDARD-RIJNDAEL ALGORITHM**

**Rajender Manteena**

**ABSTRACT**

The National Institute of Standards and Technology (NIST) has initiated a process to develop a Federal information Processing Standard (FIPS) for the Advanced Encryption Standard (AES), specifying an Advanced Encryption Algorithm to replace the Data Encryption standard (DES) the Expired in 1998. NIST has solicited candidate algorithms for inclusion in AES, resulting in fifteen official candidate algorithms of which Rijndael was chosen as the Advanced Encryption Standard.

The Advanced Encryption Standard can be programmed in software or built with pure hardware. However Field Programmable Gate Arrays (FPGAs) offer a quicker, more customizable solution. This research investigates the AES algorithm with regard to FPGA and the Very High Speed Integrated Circuit Hardware Description language (VHDL). Altera Max+plus II software is used for simulation and optimization of the synthesizable VHDL code. All the transformations of both Encryptions and Decryption are simulated using an iterative design approach in order to minimize the hardware consumption. Altera ACEX1K Family devices are utilized for hardware evaluation.



## CHAPTER 1

### INTRODUCTION

#### 1.1. Background of the Algorithm

The National Institute of Standards and Technology, (NIST), solicited proposals for the Advanced Encryption Standard, (AES). The AES is a Federal Information Processing Standard, (FIPS), which is a cryptographic algorithm that is used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt, (encipher), and decrypt, (decipher), information. Encryption converts data to an unintelligible form called cipher-text. Decryption of the cipher-text converts the data back into its original form, which is called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

Many algorithms were originally presented by researchers from twelve different nations. Fifteen, (15), algorithms were selected from the first set of submittals. After a study and selection process five, (5), were chosen as finalists. The five algorithms selected were MARS, RC6, RIJNDAEL, SERPENT and TWOFISH. The conclusion was that the five Competitors showed similar characteristics. On October 2<sup>nd</sup> 2000, NIST announced that the Rijndael Algorithm was the winner of the contest. The Rijndael Algorithm was chosen since it had the best overall scores in security, performance, efficiency, implementation ability and flexibility, [NIS00b]. The Rijndael algorithm was

developed by Joan Daemen of Proton World International and Vincent Fijmen of Katholieke University at Leuven.

The Rijndael algorithm is a symmetric block cipher that can process data blocks of 128 bits through the use of cipher keys with lengths of 128, 192, and 256 bits. The Rijndael algorithm was also designed to handle additional block sizes and key lengths. However, the additional features were not adopted in the AES. The hardware implementation of the Rijndael algorithm can provide either high performance or low cost for specific applications. At backbone communication channels or heavily loaded servers it is not possible to lose processing speed, which drops the efficiency of the overall system while running cryptography algorithms in software. On the other side, a low cost and small design can be used in smart card applications, which allows a wide range of equipment to operate securely. [6]

## **1.2. Notation and Conventions**

### **1.2.1. Inputs and Outputs**

The input and output for the AES algorithm consists of sequences of 128 bits. These sequences are referred to as blocks and the numbers of bits they contain are referred to as their length. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and Cipher Key lengths are not permitted by this standard. The bits within such sequences are numbered starting at zero and ending at one less than the sequence length, which is also termed the block length or key length. The number “ $i$ ” attached to a bit is known as its index and will be in one of the ranges  $0 \leq i < 128$ ,  $0 \leq i < 192$  or  $0 \leq i < 256$  depending on the block length or key length specified.

### 1.2.2. Bytes

The basic unit of processing in the AES algorithm is a byte, which is a sequence of eight bits treated as a single entity. The input, output and Cipher Key bit sequences described in Section 1.1 are processed as arrays of bytes that are formed by dividing these sequences into groups of eight contiguous bits to form arrays of bytes. For an input, output or Cipher Key denoted by  $a$ , the bytes in the resulting array are referenced using one of the two forms,  $a_n$  or  $a[n]$ , where  $n$  will be in a range that depends on the key length. For a key length of 128 bits,  $n$  lies in the range  $0 \leq n < 16$ . For a key length of 192 bits,  $n$  lies in the range  $0 \leq n < 24$ . For a key length of 256 bits,  $n$  lies in the range  $0 \leq n < 32$ .

All byte values in the AES algorithm are presented as the concatenation of the individual bit values, (0 or 1), between braces in the order  $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$ . These bytes are interpreted as finite field elements using a polynomial representation

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^7 b_i x^i \quad (1)$$

For example,  $\{01100011\}$  identifies the specific finite field element  $x^6 + x^5 + x + 1$ . It is also convenient to denote byte values using hexadecimal notation with each of two groups of four bits being denoted by a single hexadecimal character. The hexadecimal notation scheme is depicted in Figure.1.

Bit Pattern	Character
0000	0
0001	1
0010	2
0011	3

Bit Pattern	Character
0100	4
0101	5
0110	6
0111	7

Bit Pattern	Character
1000	8
1001	9
1010	a
1011	b

Bit Pattern	Character
1100	c
1101	d
1110	e
1111	f

Figure 1. Hexadecimal Representation of Bit Patterns [1]

Hence the element  $\{01100011\}$  can be represented as  $\{63\}$ , where the character denoting the four-bit group containing the higher numbered bits is again to the left. Some finite field operations involve one additional bit  $\{b_8\}$  to the left of an 8-bit byte. When the  $b_8$  bit is present, it appears as  $\{01\}$  immediately preceding the 8-bit byte. For example, a 9-bit sequence is presented as  $\{01\} \{1b\}$ .

### 1.2.3. Arrays of Bytes

Arrays of bytes are represented in the form  $a_0a_1a_2\cdots a_{15}$ . The bytes and the bit ordering within bytes are derived from the 128-bit input sequence,  $input_0input_1input_2\cdots input_{126}input_{127}$  as  $a_0 = \{input_0, input_1, \dots, input_7\}$ ,  $a_1 = \{input_8, input_9, \dots, input_{15}\}$  with the pattern continuing up to  $a_{15} = \{input_{120}, input_{121}, \dots, input_{127}\}$ . The pattern can be extended to longer sequences associated with 192 and 256 bit keys. In general,

$$a_n = \{input_{8n}, input_{8n+1}, \dots, input_{8n+7}\}.$$

An example of byte designation and numbering within bytes for a given input sequence is presented in Figure 2.

Input bit sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...
Byte number	0							1							2							...			
Bit numbers in byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

Figure 2. Indices for Bytes and Bits [1]

### 1.2.4. The State

Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the State. The State consists of four rows of bytes. Each row of a state contains  $N_b$  numbers of bytes, where  $N_b$  is the block length divided by 32. In the State array, which is denoted by the symbol  $\mathcal{S}$ , each individual byte has two indices. The first byte index is the row number  $\mathbf{r}$ , which lies in the range  $0 \leq \mathbf{r} \leq 3$  and the second byte

index is the column number  $c$ , which lies in the range  $0 \leq c \leq N_b - 1$ . Such indexing allows an individual byte of the State to be referred to as  $S_{r,c}$  or  $S_{[r,c]}$ . For the AES  $N_b = 4$ , which means that  $0 \leq c \leq 3$ . At the beginning of the Encryption and Decryption the input, which is the array of bytes symbolized by  $in_0 in_1 \dots in_{15}$  is copied into the State array. This activity is illustrated in Figure 3. The Encryption or Decryption operations are conducted on the State array. After manipulation of the state array has completed its final value is copied to the output, which is an array of bytes symbolized by  $out_0 out_1 \dots out_{15}$ .

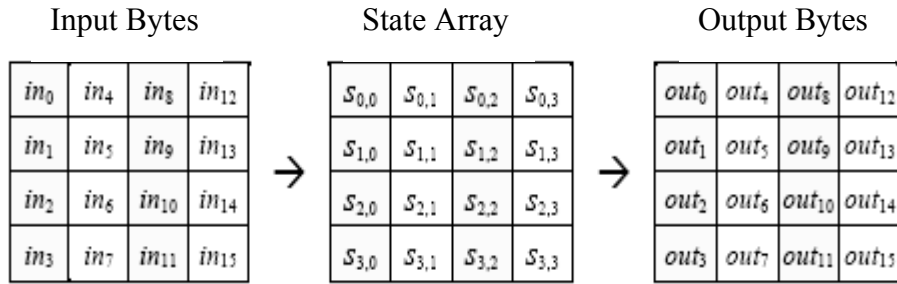


Figure 3. State Array Input and Output [1]

At the start of the Encryption or Decryption the input array is copied to the State array with

$$S_{[r,c]} = in[r + 4c]$$

where  $0 \leq r \leq 3$  and  $0 \leq c \leq N_b - 1$ . At the end of the Encryption and Decryption the State is copied to the output array with

$$out[r + 4c] = S_{[r,c]}$$

where  $0 \leq r \leq 3$  and  $0 \leq c \leq N_b - 1$ .

### 1.2.5. The State as an Array of Columns

The four bytes in each column of the State form 32-bit words, where the row number “r” provides an index for the four bytes within each word. Therefore, the state can be interpreted as a one-dimensional array of 32 bit words, which is symbolized by  $w_0...w_3$ . The column number  $c$  provides an index into this linear State array. Considering the State depicted in Figure3, the State can be considered as an array of four words where

$$w_0 = S_{0,0} S_{1,0} S_{2,0} S_{3,0},$$

$$w_1 = S_{0,1} S_{1,1} S_{2,1} S_{3,1},$$

$$w_2 = S_{0,2} S_{1,2} S_{2,2} S_{3,2}$$

and

$$w_3 = S_{0,3} S_{1,3} S_{2,3} S_{3,3}.$$

### 1.3. Mathematical Background

Every byte in the AES algorithm is interpreted as a finite field element using the notation introduced in Section.1.1.2. All Finite field elements can be added and multiplied. However, these operations differ from those used for numbers and their use requires investigation.

#### 1.3.1. Addition

The addition of two elements in a finite field is achieved by “adding” the coefficients for the corresponding powers in the polynomials for the two elements. The addition is performed through use of the XOR operation, which is denoted by the operator symbol  $\oplus$ . Such addition is performed modulo-2. In modulo-2 addition

$$1 \oplus 1 = 0,$$

$$1 \oplus 0 = 1,$$

$$0 \oplus 1 = 1$$

and

$$0 \oplus 0 = 0.$$

Consequently, subtraction of polynomials is identical to addition of polynomials. Alternatively, addition of finite field elements can be described as the modulo-2 addition of corresponding bits in the byte. For two bytes  $\{a_7a_6a_5a_4a_3a_2a_1a_0\}$  and  $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$ , the sum is  $\{c_7c_6c_5c_4c_3c_2c_1c_0\}$ , where each  $c_i = a_i \oplus b_i$  where  $i$  represents corresponding bits. For example, the following expressions are equivalent to one another.

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \text{ (Polynomial notation)}$$

$$\{01010111\} \oplus \{10000011\} = \{11010100\} \quad \text{(Binary notation)}$$

$$\{57\} \oplus \{83\} = \{d4\} \quad \text{(Hexadecimal notation)}$$

### 1.3.2. Multiplication

In the polynomial representation, multiplication in Galois Field  $GF(2^8)$  (denoted by  $\bullet$ ) corresponds with the multiplication of polynomials modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. For the AES algorithm, this irreducible polynomial is given by the equation (2).

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (2)$$

For example,  $\{57\} \bullet \{83\} = \{c1\}$  because

$$\begin{aligned}
 (x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) &= x^{13} + x^{11} + x^9 + x^8 + x^7 + \\
 &\quad x^7 + x^5 + x^3 + x^2 + x + \\
 &\quad x^6 + x^4 + x^2 + x + 1 \\
 &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \\
 x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 &\text{ Modulo } (x^8 + x^4 + x^3 + x + 1) \\
 &= x^7 + x^6 + 1.
 \end{aligned}$$

The modular reduction by  $m(x)$  ensures that the result will be a binary polynomial of degree less than 8, which can be represented by a byte. Unlike addition, there is no simple operation at the byte level that corresponds to this multiplication. The multiplication defined above is associative and the element  $\{01\}$  is the multiplicative identity. For any non-zero binary polynomial  $b(x)$  of degree less than 8, the multiplicative inverse of  $b(x)$ , denoted  $b^{-1}(x)$ , can be found. The inverse is found through use of the extended Euclidean algorithm to compute polynomials  $a(x)$  and  $c(x)$  such that

$$b(x)a(x) + m(x)c(x) = 1. \quad (3)$$

Hence,  $a(x) \bullet b(x) \text{ mod } m(x) = 1$ , which means

$$b^{-1}(x) = a(x) \text{ mod } m(x) \quad (4)$$

Moreover, for any  $a(x)$ ,  $b(x)$  and  $c(x)$  in the field, it holds that

$$a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x) \quad (5)$$

It follows that the set of 256 possible byte values, with XOR used as addition and multiplication defined as above, has the structure of the finite field  $\text{GF}(2^8)$ .



### 1.3.3. Multiplication by x

Multiplying the binary polynomial defined in equation (1) with the polynomial  $x$  results in

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x. \quad (6)$$

The result  $x \bullet b(x)$  is obtained by reducing the above result modulo  $m(x)$ . If  $b_7$  equals zero the result is already in reduced form. If  $b_7$  equals one the reduction is accomplished by subtracting the polynomial  $m(x)$ . It follows that multiplication by  $x$ , which is represented by  $\{00000010\}$  or  $\{02\}$ , can be implemented at the byte level as a left shift and a subsequent conditional bitwise XOR with  $\{1b\}$ . This operation on bytes is denoted by `xtime( )`. Multiplication by higher powers of  $x$  can be implemented by repeated application of `xtime( )`. Through the addition of intermediate results, multiplication by any constant can be implemented.

For example,  $\{57\} \bullet \{13\} = \{fe\}$  because

$$\{57\} \bullet \{02\} = \text{xtime}(\{57\}) = \{ae\}$$

$$\{57\} \bullet \{04\} = \text{xtime}(\{ae\}) = \{47\}$$

$$\{57\} \bullet \{08\} = \text{xtime}(\{47\}) = \{8e\}$$

$$\{57\} \bullet \{10\} = \text{xtime}(\{8e\}) = \{07\},$$

Thus,

$$\begin{aligned} \{57\} \bullet \{13\} &= \{57\} \bullet (\{01\} \bullet \{02\} \bullet \{10\}) \\ &= \{57\} \bullet \{ae\} \bullet \{07\} \\ &= \{fe\}. \end{aligned}$$

### 1.3.4. Polynomials with Coefficients in GF (2<sup>8</sup>)

Four-term polynomials can be defined with coefficients that are finite field elements as the following equation (7)

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad (7)$$

which will be denoted as a word in the form [a0 , a1 , a2 , a3 ]. Note that the polynomials in this section behave somewhat differently than the polynomials used in the definition of finite field elements, even though both types of polynomials use the same indeterminate,  $x$ . The coefficients in this section are themselves finite field elements, i.e., bytes, instead of bits; also, the multiplication of four-term polynomials uses a different reduction polynomial, defined below. To illustrate the addition and multiplication operations, let

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \quad (8)$$

define a second four-term polynomial. Addition is performed by adding the finite field coefficients of like powers of  $x$ . This addition corresponds to an XOR operation between the corresponding bytes in each of the words – in other words, the XOR of the complete word values. Thus, using the equations of (7) and (8),

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0) \quad (9)$$

Multiplication is achieved in two steps. In the first step, the polynomial product

$c(x) = a(x) \bullet b(x)$  is algebraically expanded, and like powers are collected to give

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \quad (10)$$

where

$$\begin{aligned}
c_0 &= a_0 \bullet b_0 \\
c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \\
c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \\
c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \\
c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\
c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\
c_6 &= a_3 \bullet b_3
\end{aligned}$$

The result,  $c(x)$ , does not represent a four-byte word. Therefore, the second step of the multiplication is to reduce  $c(x)$  modulo a polynomial of degree 4; the result can be reduced to a polynomial of degree less than 4. For the AES algorithm, this is accomplished with the polynomial  $x^4 + 1$ , so that

$$x^i \bmod(x^4 + 1) = x^{i \bmod 4}. \quad (11)$$

The modular product of  $a(x)$  and  $b(x)$ , denoted by  $a(x) \bullet b(x)$ , is given by the four-term polynomial  $d(x)$ , defined as follows

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \quad (12)$$

with

$$\begin{aligned}
d_0 &= (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\
d_1 &= (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\
d_2 &= (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\
d_3 &= (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)
\end{aligned}$$

When  $a(x)$  is a fixed polynomial, the operation defined in equation (12) can be written in matrix form as the following equation (13).

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (13)$$

Because  $x^4 + 1$  is not an irreducible polynomial over  $\text{GF}(2^8)$ , multiplication by a fixed four-term polynomial is not necessarily invertible. However, the AES algorithm specifies a fixed four-term polynomial that does have an inverse is given by

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (14)$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \quad (15)$$

Another polynomial used in the AES algorithm has  $a_0 = a_1 = a_2 = \{00\}$  and  $a_3 = \{01\}$ , which is the polynomial  $x^3$ . Inspection of equation (13) above will show that its effect is to form the output word by rotating bytes in the input word. This means that  $[b_0, b_1, b_2, b_3]$  is transformed into  $[b_1, b_2, b_3, b_0]$ .

## CHAPTER 2

### ENCRYPTION

#### 2.1. Encryption Process

The Encryption process of Advanced Encryption Standard algorithm is presented below, in figure 4.

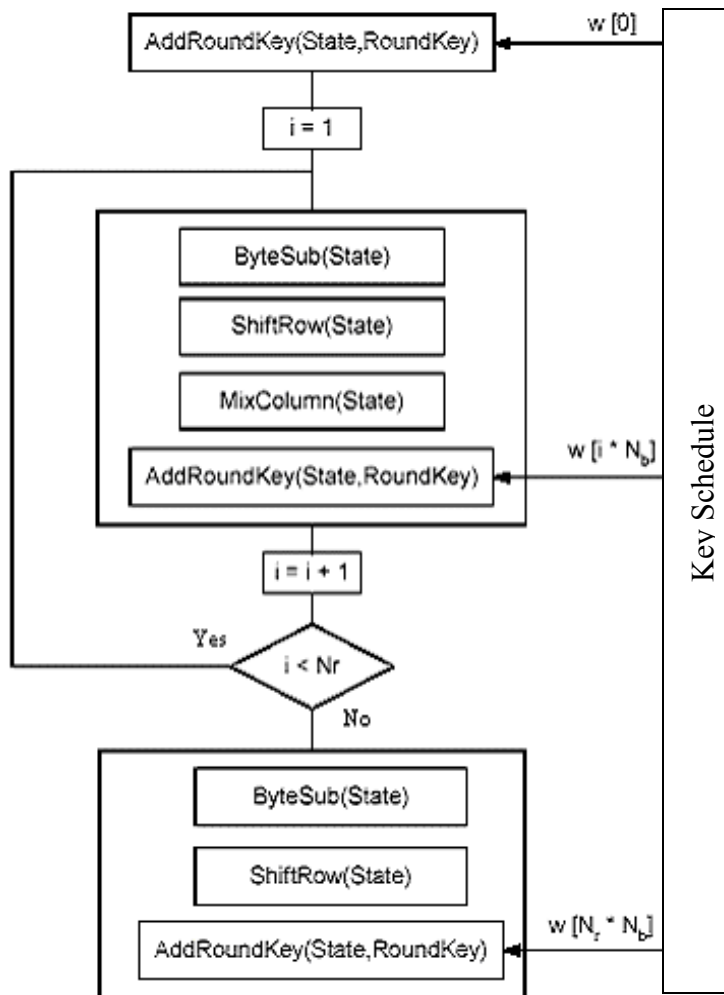


Figure 4. Encryption Process [6]

This block diagram is generic for AES specifications. It consists of a number of different transformations applied consecutively over the data block bits, in a fixed number of iterations, called rounds. The number of rounds depends on the length of the key used for the encryption process.

## 2.2. Bytes Substitution Transformation

The bytes substitution transformation Bytesub (state) is a non-linear substitution of bytes that operates independently on each byte of the State using a substitution table(S-box) presented in figure7. This S-box which is invertible, is constructed by composing two transformations

1. Take the multiplicative inverse in the finite field  $GF(2^8)$ , described in Section 1.3.2. The element  $\{00\}$  is mapped to itself.
2. Apply the following affine transformation (over  $GF(2)$ )

$$b'_i = b_i \oplus b_{(i+4)\text{mod}8} \oplus b_{(i+5)\text{mod}8} \oplus b_{(i+6)\text{mod}8} \oplus b_{(i+7)\text{mod}8} \oplus c_i \quad (16)$$

for  $0 \leq i \leq 7$ , where  $b_i$  is the  $i^{\text{th}}$  bit of the byte, and  $c_i$  is the  $i^{\text{th}}$  bit of a byte  $c$  with the value  $\{63\}$  or  $\{01100011\}$ . Here and elsewhere, a prime on a variable (e.g.,  $b'$ ) indicates that the variable is to be updated with the value on the right. In matrix form, the affine transformation element of the S-box can be expressed as

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Figure 5. Matrix Notation of S-box [1]

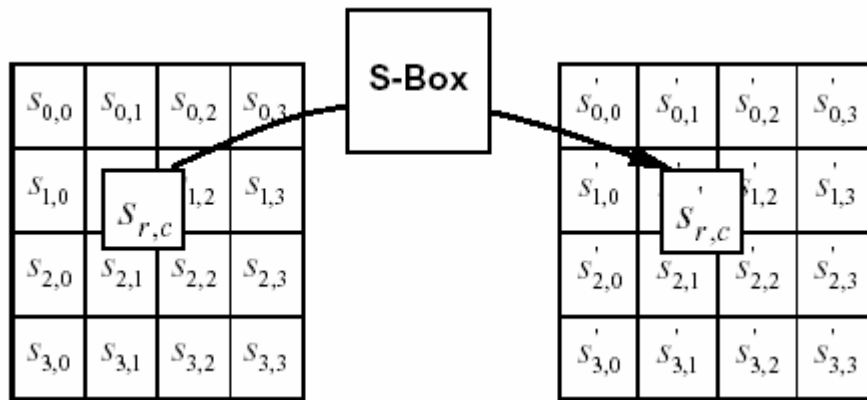


Figure 6. Application of S-box to the Each Byte of the State [1]

The S-box used in the Sub Bytes transformation is presented in hexadecimal form in figure 7. For example, if  $S_{1,1} = \{53\}$ , then the substitution value would be determined by the intersection of the row with index '5' and the column with index '3' in figure 7. This would result in  $S'_{1,1}$  having a value of  $\{ed\}$ .

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 7. S-box Values for All 256 Combinations in Hexadecimal Format [1]

### 2.3. Shift Rows Transformation

In the Shift Rows transformation  $ShiftRows()$ , the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row,  $r = 0$ , is not shifted. Specifically, the  $ShiftRows()$  transformation proceeds as follows

$$s'_{r,c} = s_{r,(c+shift(r,Nb)) \bmod Nb} \quad \text{for } 0 < r < 4 \text{ and } 0 \leq c \leq Nb,$$

Where the shift value  $shift(r, Nb)$  depends on the row number,  $r$ , as follows ( $Nb = 4$ )

$$Shift(1,4) = 1; Shift(2,4) = 2; Shift(3,4) = 3.$$

This has the effect of moving bytes to “lower” positions in the row (i.e., lower values of  $c$  in a given row), while the “lowest” bytes wrap around into the “top” of the row (i.e., higher values of  $c$  in a given row). Figure 7 illustrates the  $ShiftRows()$  transformation.



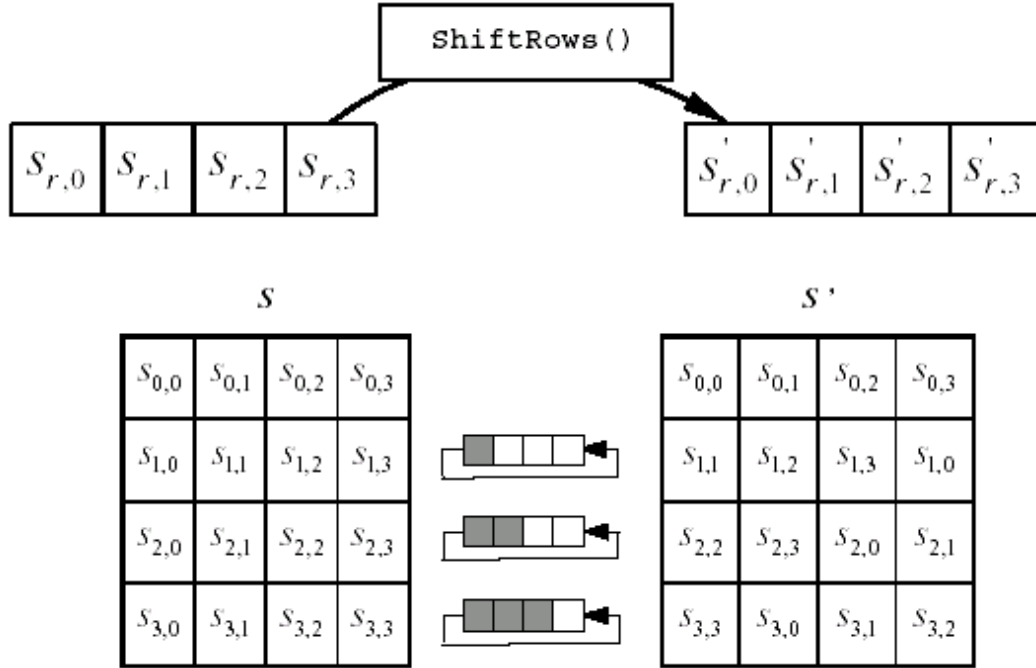


Figure 8. Cyclic Shift of the Last Three Rows of the State [1]

#### 2.4. Mixing of Columns Transformation

This transformation is based on Galois Field multiplication. Each byte of a column is replaced with another value that is a function of all four bytes in the given column. The MixColumns( ) transformation operates on the State column-by-column, treating each column as a four-term polynomial as described in Section.1.3.4. The columns are considered as polynomials over  $GF(2^8)$  and multiplied modulo  $x^4 + 1$  with a fixed polynomial  $a(x)$ , given by the following equation.

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

As described in Section. 1.3.4, this can be written as a matrix multiplication. Let

$$S'(x) = a(x) \otimes S(x)$$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad \text{for } 0 \leq c < \text{Nb.}$$

As a result of this multiplication, the four bytes in a column are replaced by the following

$$\begin{aligned} S'_{0,c} &= (\{02\} \bullet S_{0,c}) \oplus (\{03\} \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S'_{1,c} &= S_{0,c} \oplus (\{02\} \bullet S_{1,c}) \oplus (\{03\} \bullet S_{2,c}) \oplus S_{3,c} \\ S'_{2,c} &= S_{0,c} \oplus S_{1,c} \oplus (\{02\} \bullet S_{2,c}) \oplus (\{03\} \bullet S_{3,c}) \\ S'_{3,c} &= (\{03\} \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \bullet S_{3,c}) \end{aligned}$$

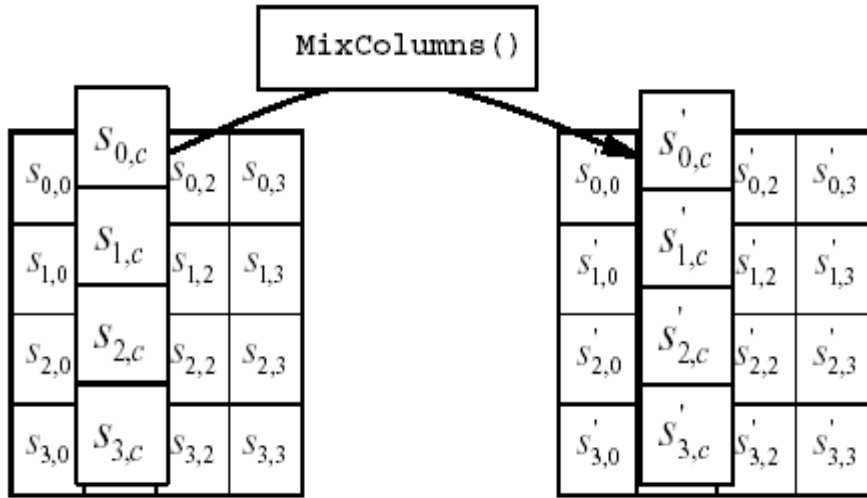


Figure 9. Mixing of Columns of the State [1]

## 2.5. Addition of Round Key Transformation

In the Addition of Round Key transformation `AddRoundKey()`, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of Nb words from the key schedule generation (described in following section 2.6). Those Nb words are each added into the columns of the State, such that

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus [W_{round * Nb}] \quad \text{for } 0 \leq c < Nb,$$

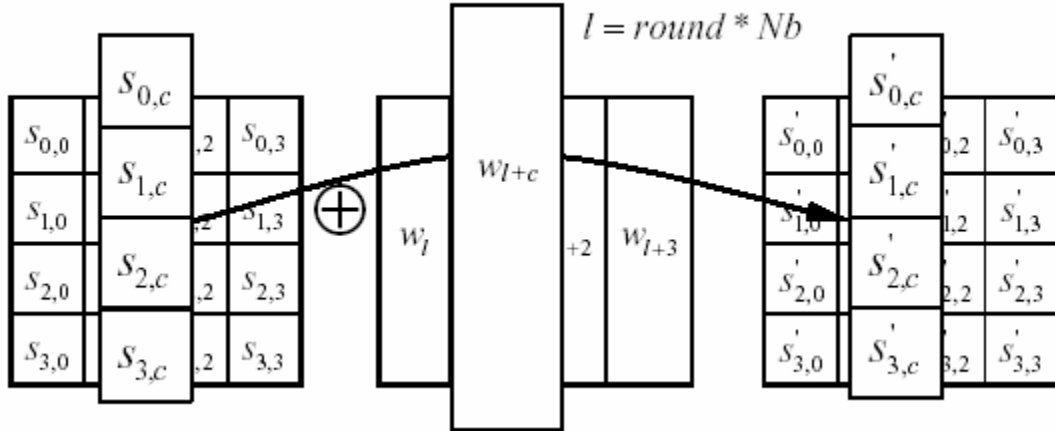


Figure 10. Exclusive-OR Operation of State and Cipher Key Words [1]

where  $[w_i]$  are the key generation words described in chapter 3, and round is a value in the range in the Encryption, the initial Round Key addition occurs when round = 0, prior to the first application of the round function. The application of the AddRoundKey ( ) transformation to the Nr rounds of the encryption occurs when  $1 \leq \text{round} \leq Nr$ . The action of this transformation is illustrated in figure10, where  $l = \text{round} * Nb$ . The byte address within words of the key schedule was described in Section1.2.1.

## 2.6. Key Schedule Generation

Each round key is a 4-word (128-bit) array generated as a product of the previous round key, a constant that changes each round, and a series of S-Box (figure6) lookups for each 32-bit word of the key. The first round key is the same as the original user input. Each byte ( $w_0 - w_3$ ) of initial key is XOR'd with a constant that depends on the current round, and the result of the S-Box lookup for  $w_i$ , to form the next round key. The number of rounds required for three different key lengths is presented in figure11.

	Key Length (Nk Words)	Block Size (Nb Words)	Number of Rounds (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Figure 11. Key-Block- Round Combinations [1]

The Key schedule Expansion generates a total of  $Nb(Nr + 1)$  words: the algorithm requires an initial set of  $Nb$  words, and each of the  $Nr$  rounds requires  $Nb$  words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted  $[w_i]$ , with  $i$  in the range  $0 \leq i < Nb(Nr + 1)$ .

## CHAPTER 3 DECRYPTION

### 3.1. Decryption Process

The Decryption process of Advanced Encryption Standard algorithm is presented below, in figure12.

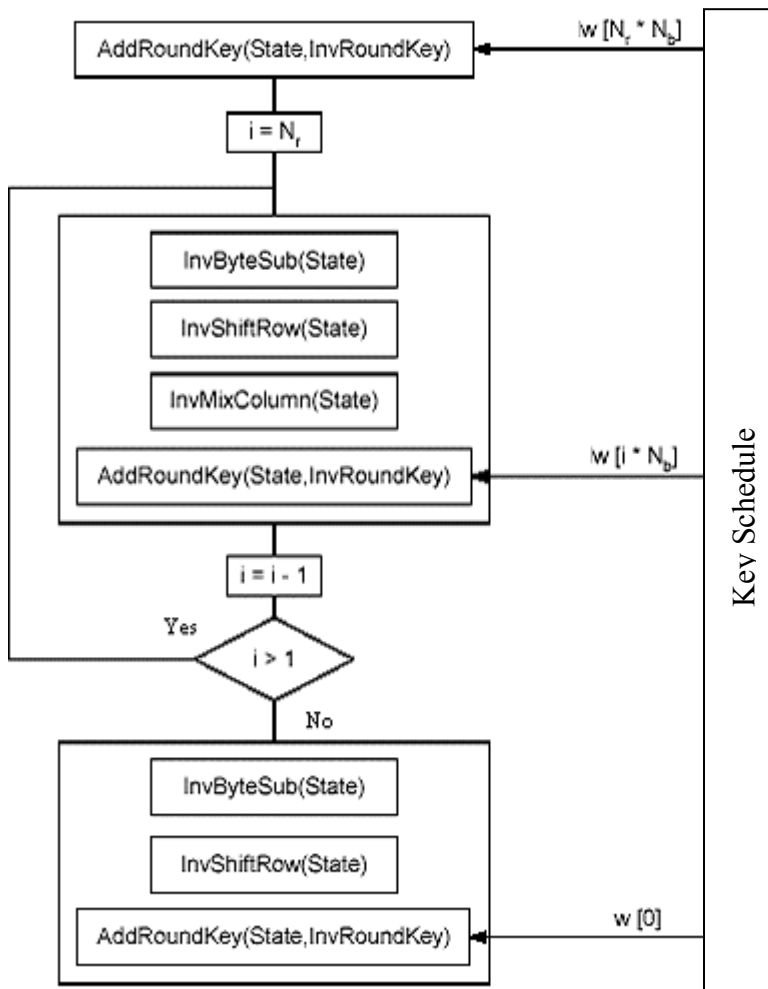


Figure 12. Decryption Process [6]

This process is direct inverse of the Encryption process (chapter2). All the transformations applied in Encryption process are inversely applied to this process. Hence the last round values of both the data and key are first round inputs for the Decryption process and follows in decreasing order.

### 3.2. Inverse Bytes Substitution Transformation

Inverse Byte Substitution Transformation  $\text{InvSubBytes}()$  is the inverse of the byte substitution transformation, in which the inverse S-Box (figure14) is applied to each byte of the State. This is obtained by applying the inverse of the affine transformation to the equation (16) followed by taking the multiplicative inverse in  $\text{GF}(2^8)$ .

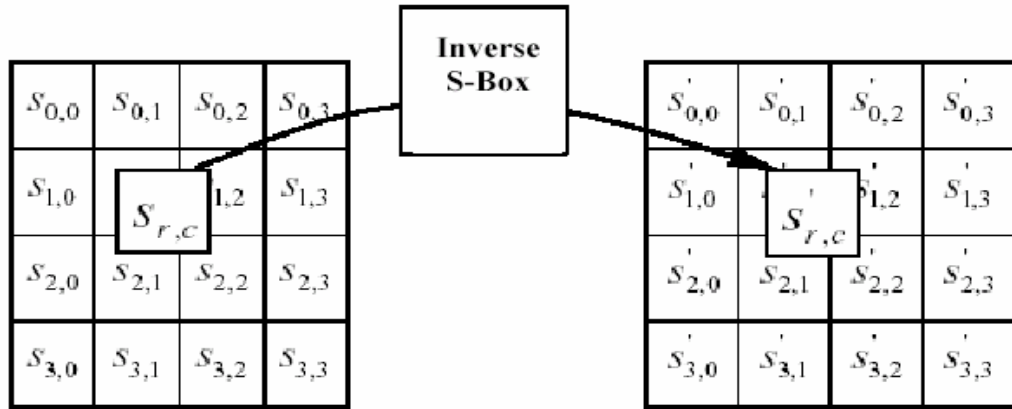


Figure 13. Application of the Inverse S-box to Each Byte of the State [1]

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 14. Inverse S-box Values for All 256 Combinations in Hexadecimal Format

### 3.3. Inverse Shift Rows Transformation

Inverse Shift Rows Transformation  $InvShiftRows()$  is the inverse of the  $ShiftRows()$  transformation presented in Chater2. The bytes in the last three rows of the State are cyclically shifted over different numbers of bytes. The first row,  $r = 0$ , is not shifted. The bottom three rows are cyclically shifted by  $Nb-shift(r, Nb)$  bytes, where the shift value  $shift(r, Nb)$  depends on the row number, and is explained in Section.2.3. Specifically, the  $InvShiftRows()$  transformation proceeds as follows

$$S'_{r,(c+shift(r,Nb))\bmod Nb} = S_{r,c} \quad \text{for } 0 \leq r < 4 \text{ and } 0 \leq c < Nb$$

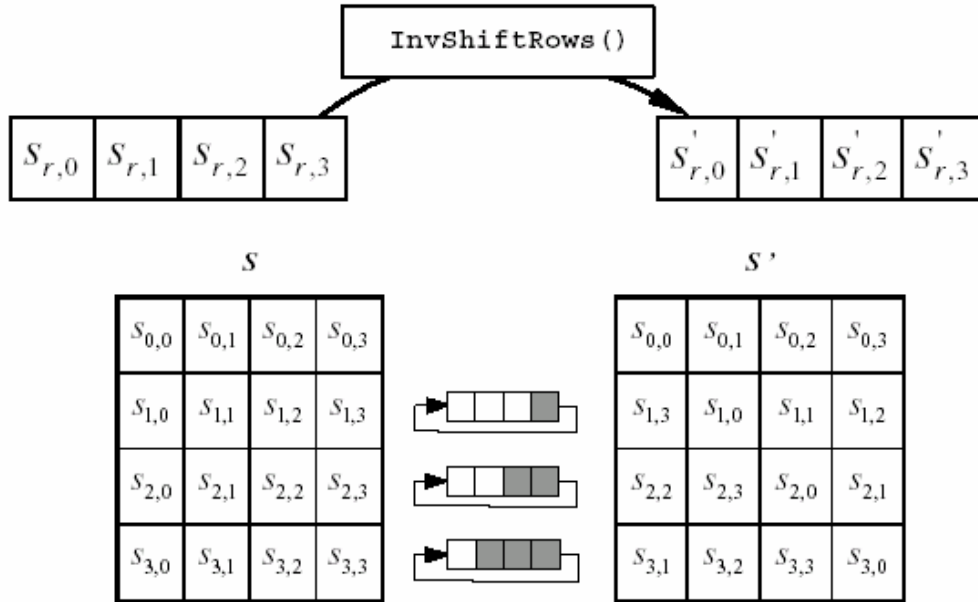


Figure 15. Inverse Cyclic Shift of the Last Three Rows of the State [1]

### 3.4. Inverse Mixing of Columns Transformation

Inverse Mixing of Columns Transformation  $\text{InvMixColumns}()$  is the inverse of the  $\text{MixColumns}()$  transformation presented in chapter 2.  $\text{InvMixColumns}()$  operates on the State column-by-column, treating each column as a four term polynomial as described in Section 1.3.4. The columns are considered as polynomials over  $\text{GF}(2^8)$  and multiplied modulo  $x^4 + 1$  with a fixed polynomial  $a^{-1}(x)$ , given by

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

As described in Section 1.3.4, this can be written as a matrix multiplication. Let

$$S'(x) = a^{-1}(x) \otimes S(x)$$



$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad \text{for } 0 \leq c < \text{Nb.}$$

As a result of this multiplication, the four bytes in a column are replaced by the following equations.

$$\begin{aligned} S'_{0,c} &= (\{0e\} \bullet S_{0,c}) \oplus (\{0b\} \bullet S_{1,c}) \oplus (\{0d\} \bullet S_{2,c}) \oplus (\{09\} \bullet S_{3,c}) \\ S'_{1,c} &= (\{09\} \bullet S_{0,c}) \oplus (\{0e\} \bullet S_{1,c}) \oplus (\{0b\} \bullet S_{2,c}) \oplus (\{0d\} \bullet S_{3,c}) \\ S'_{2,c} &= (\{0d\} \bullet S_{0,c}) \oplus (\{09\} \bullet S_{1,c}) \oplus (\{0e\} \bullet S_{2,c}) \oplus (\{0b\} \bullet S_{3,c}) \\ S'_{3,c} &= (\{0b\} \bullet S_{0,c}) \oplus (\{0d\} \bullet S_{1,c}) \oplus (\{09\} \bullet S_{2,c}) \oplus (\{0e\} \bullet S_{3,c}) \end{aligned}$$

Hence, State can be represented as

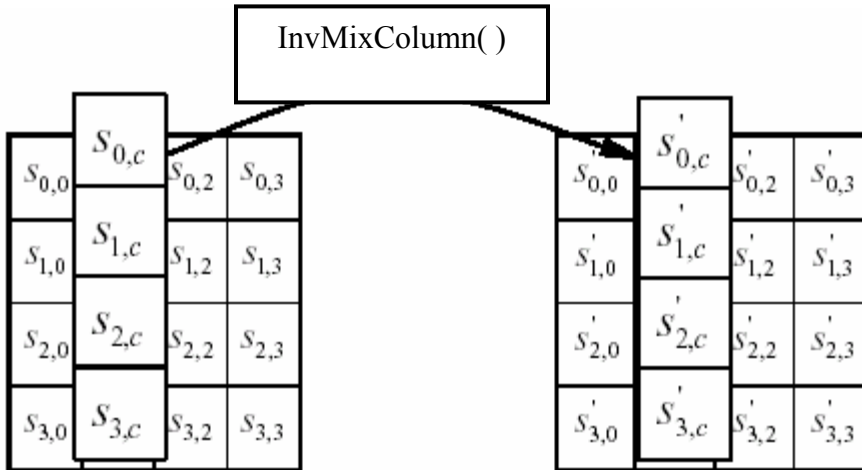


Figure 16. Inverse Mix Column Operation on State [1]

## **CHAPTER 4**

### **IMPLEMENTATION AND RESULTS**

#### **4.1. Encryption Implementation**

VHDL is used as the hardware description language because of the flexibility to exchange among environments. The code is pure VHDL that could easily be implemented on other devices, without changing the design. The software used for this work is Altera Max+plus II 10.2. This is used for writing, debugging and optimizing efforts, and also for fitting, simulating and checking the performance results using the simulation tools available on MaxPlus II design software.

All the results are based on simulations from the Max+plus II and Quartus tools, using Timing Analyzer and Waveform Generator. All the individual transformation of both encryption and decryption are simulated using FPGA ACEX1K family and EP1K100 devices. The characteristics of the devices are presented in figure 17. An iterative method of design is implemented to minimize the hardware utilization and the fitting is done by the Altera's Quartus fitter Technology.

General Characteristics of the ACEX Family				
Device	EP1K10	EP1K30	EP1K50	EP1K100
Typical Gates	10,000	30,000	50,000	100,000
Maximum System Gates	56,000	119,000	199,000	257,000
Logic Elements	576	1,728	2,880	4,992
Embedded Array Blocks (EABs)	3	6	10	12
Maximum RAM Bits	12,288	24,576	40,960	49,152
Speed Grades	-1, -2, -3	-1, -2, -3	-1, -2, -3	-1, -2, -3
Package (mm)	Maximum User I/O Pins			
100-Pin TQFP	66			
144-Pin TQFP	92	102	102	
208-Pin PQFP	120	147	147	147
256-Pin (BGA)	136	171	186	186
484-Pin (BGA)			249	333

Figure 17. Basic Characteristics of the ACEX1K Family Devices [4]

In order to allow a full parallel process of the state, it is necessary to implement all the transformations over 128 bits. The most expensive one is the Byte substitution, because it is a table lookup operation, implemented as ROM. Each 8 bits requires a 2048 bit ROM. To process 128 bits it is necessary 32768 bits. The Key Expansion uses a Byte substitution operation over 32 bits also, so another 8192 bits should be allocated.

The following figure 18 shows the waveforms generated by the 8-bit byte substitution transformation. The inputs are clock of 100ns time period, Active High reset, and 8-bit state as a standard logic vector, whose output is 8-bit S-box lookup substitution. This design utilizes 32% of the area of EP1K100FC484-1, around 1631 logic elements are consumed to implement only 8-bit S-box lookup table. Hence, approximately 20,000 logic elements are necessary to implement the complete 128-bit byte substitution transformation. It can be done by the APEX20K family devices.

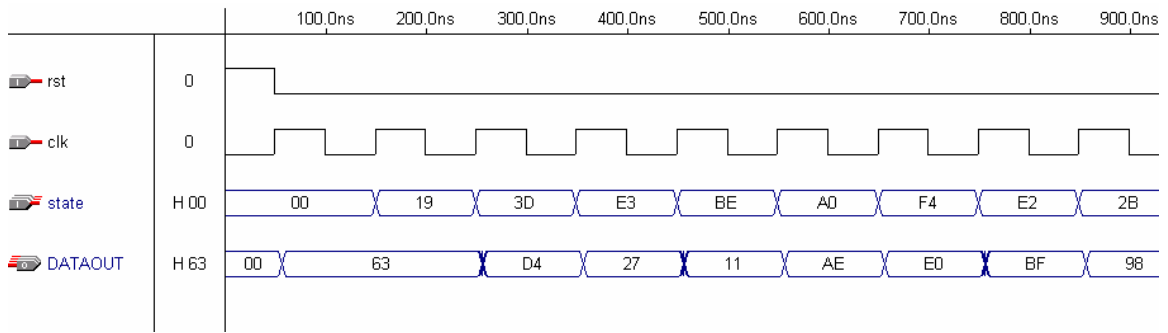


Figure 18. Waveforms of 8-bit Byte Substitution

The following figure 19 represents the waveforms generated by the 8-bit byte substitution transformation. The inputs are clock of 100ns time period, Active High reset, and 128-bit state as a standard logic vector, whose output is shifted as explained in the section 2.3. Design utilizes 2% of the area of EP1K100FC484-1, around 128 logic elements are consumed.

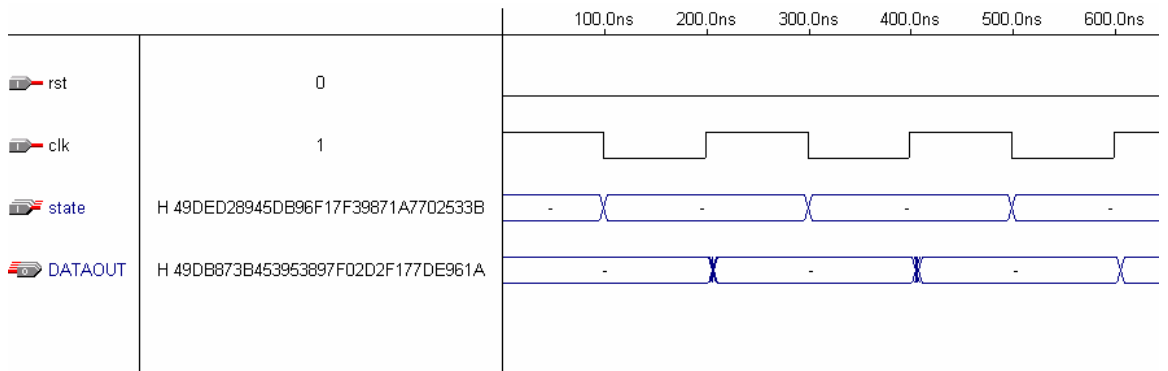


Figure 19. Waveforms of Shift Row Transformation

The following figure 20 represents the waveforms generated by the 12 8-bit Mix Columns transformation. The inputs are clock of 100ns time period, Active High reset, and 128-bit state as a standard logic vector, whose output is shifted as explained in the section 2.4. Design utilizes 5% of the area of EP1K100FC484-1, around 156 logic elements are consumed.

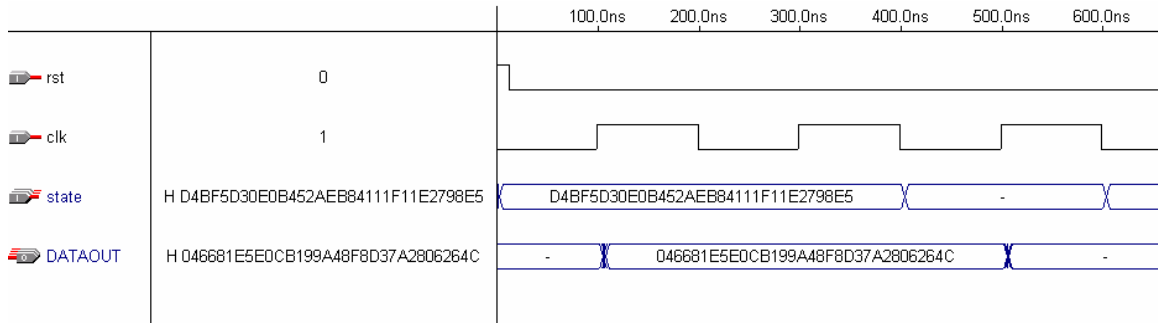


Figure 20. Waveforms of Mix Column Transformation

The following figure 21 represents the waveforms generated by the 128-bit Key Schedule Generation. The inputs are clock of 100ns time period, Active High reset, round, and 128-bit state as a standard logic vector, whose output is the 128-bit key for round one is generated. Design utilizes 74% of the area of EP1K100FC484-1, around 3700 logic elements are consumed.

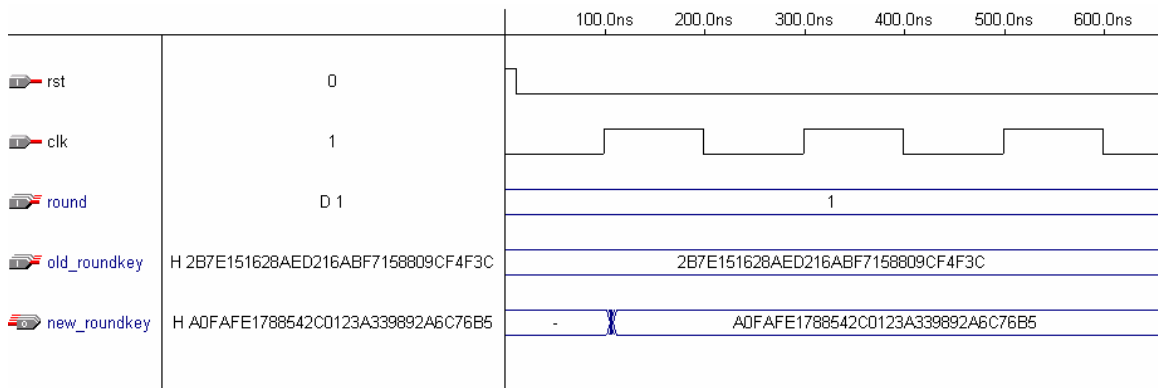


Figure 21. Waveforms of Key Schedule Generation

## 4.2. Decryption Implementation

The decryption implementation results are similar to the encryption implementation. The key schedule generation module is modified in the reverse order. In which last round key is treated as the first round and decreasing order follows.

The following figure 22 represents the waveforms generated by the 8-bit byte substitution transformation. The inputs are clock of 100ns time period, Active High reset, and 8-bit state as a standard logic vector, whose output is 8-bit Inverse S-box lookup substitution. This design utilizes 50% of the area of EP1K30TC144-1, around 877 logic elements are consumed to implement only 8-bit S-box lookup table

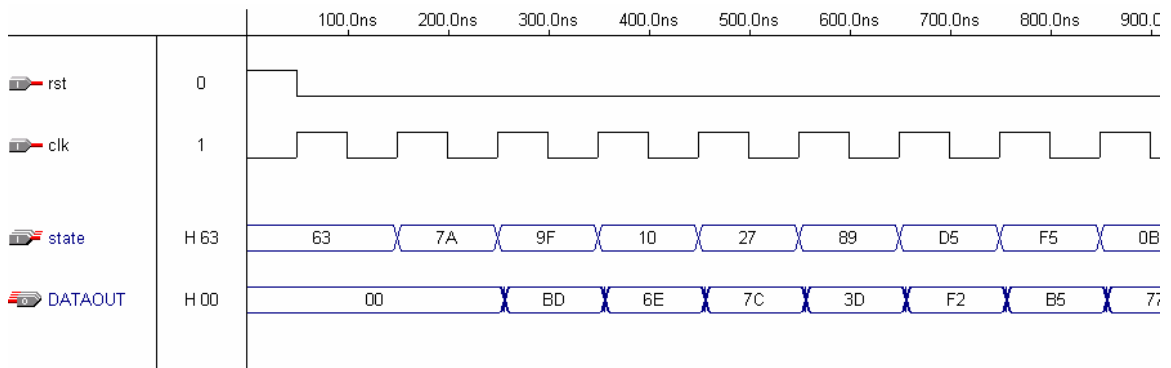


Figure 22. Waveforms of 8-bit Inverse Byte Substitution

The following figure 23 represents the waveforms generated by the 8-bit Inverse byte substitution transformation. The inputs are clock of 100ns time period, Active High reset, and 8-bit state as a standard logic vector whose output is shifted as explained in the section 3.3. Design utilizes 2% of the area of EP1K100FC484-1, around 128 logic elements are consumed.

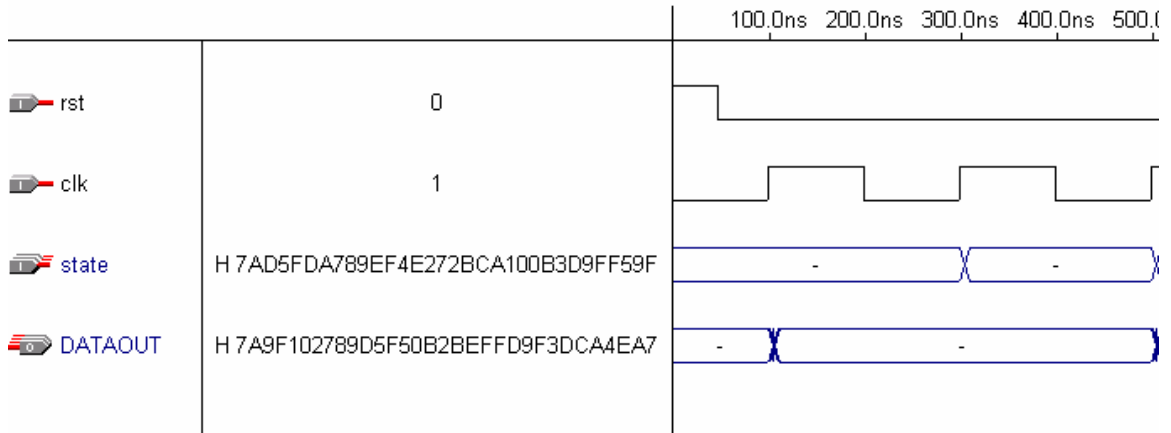


Figure 23. Waveforms of Inverse Shift Row Transformation

The following figure 24 represents the waveforms generated by the 8-bit byte substitution transformation. The inputs are clock of 100ns time period, Active High reset, and 8-bit state as a standard logic vector, whose output is shifted as explained in the section 3.4. Design utilizes 12% of the area of EP1K100FC484-1, around 624 logic elements are consumed.

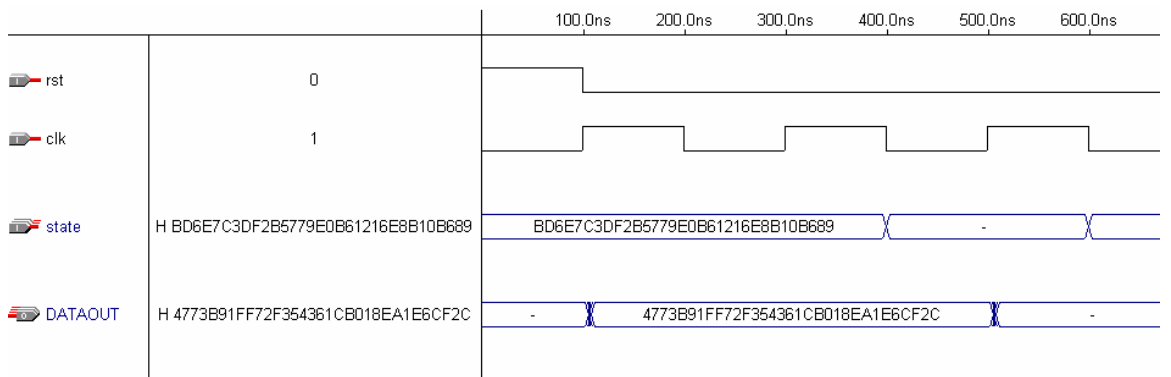


Figure 24. Waveforms of Inverse Mix Column Transformation

### 4.3. Hardware Implementation

The following figure 25 represents complete hardware implementation of the both encryption and decryption with key generation modules.

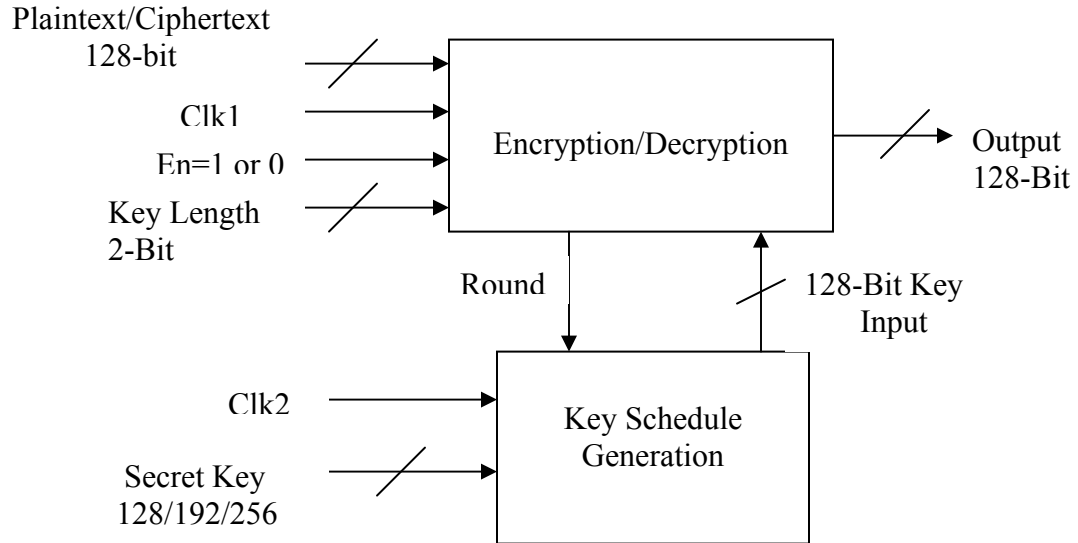


Figure 25. Block Diagram of AES Hardware Implementation

Key Schedule Generation block can generate the required keys for the process with secret key and Clk2 as inputs; these generated keys are stored in internal ROM and read by Encryption/Decryption block for each round to obtain a distinct 128-bit key with Round counter, where Encryption/Decryption module takes 128-bit plaintext or ciphertext as input with respective to the Clk1 (If En=1 or 0 process is encryption or decryption respectively). In order to distinguish the number of rounds, a 2-bit Key Length input is given to this module where 00, 01, 10 represents 10(128-bit key), 12(192-bit key), 14(256-bit key) rounds respectively, generates the final output of 128-bit cipher or plaintext.



#### **4.4. Conclusions**

Optimized and Synthesizable VHDL code is developed for the implementation of both encryption and decryption process. Each program is tested with some of the sample vectors provided by NIST and output results are perfect with minimal delay. Therefore, AES can indeed be implemented with reasonable efficiency on an FPGA, with the encryption and decryption taking an average of 320 and 340 ns respectively (for every 128 bits). The time varies from chip to chip and the calculated delay time can only be regarded as approximate. Adding data pipelines and some parallel combinational logic in the key scheduler and round calculator can further optimize this design.

## REFERENCES

- [1] FIPS 197, “Advanced Encryption Standard (*AES*)”, November 26, 2001  
<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [2] J. Daemen and V. Rijmen, “*AES Proposal: Rijndael*”, AES Algorithm Submission, September 3, 1999  
<http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaedocV2.zip>
- [3] ALTERA. Max+plus II VHDL. San Jose. Altera, 1996
- [4] ALTERA “ACEX1K Embedded Programmable Logic Family Data Sheet”, pdf files,  
<http://www.altera.com/literature/ds/acex.pdf> (May 2003)
- [5] ALTERA High-Speed Rijndael Encryption/Decryption Processors,  
[http://www.altera.com/literature/wp/wp\\_hcores\\_rijnfast.pdf](http://www.altera.com/literature/wp/wp_hcores_rijnfast.pdf)
- [6] Marcelo B. de Barcelos Design Case, “Optimized performance and area implementation of Advanced Encryption Standard in Altera Devices, by,  
<http://www.inf.ufrgs.br/~panato/artigos/designcon02.pdf>
- [7] “FPGA Simulations of Round 2 Advanced Encryption Standards”  
<http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/presentations/elbirt.pdf>.
- [8] [http://en.wikipedia.org/wiki/Extended\\_Euclidean\\_algorithm](http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm)
- [9] Tilborg, Henk C. A. van. “Fundamentals of Cryptology: A Professional Reference and Interactive Tutorial”, New York Kluwer Academic Publishers, 2002
- [10] Peter J. Ashenden, “The Designer's Guide to VHDL”, 2<sup>nd</sup> Edition, San Francisco, CA, Morgan Kaufmann, 2002

## **APPENDICES**

## Appendix A: Terms and Definitions

The following definitions are used throughout this standard

Terms	Definitions
AES	Advanced Encryption Standard.
Affine Transformation	A transformation consisting of multiplication by a matrix followed by the addition of a vector.
Array	An enumerated collection of identical entities.
Bit	A binary digit having a value of 0 or 1. Block Sequence of binary bits that comprise the input, output, State, and Round Key. The length of a sequence is the number of bits it contains. Blocks are also interpreted as arrays of bytes.
Byte	A group of eight bits that is treated either as a single entity or as an array of 8 individual bits.
Encryption or ciphertext	Cipher Series of transformations that converts plaintext to using the Cipher Key
Cipher Key	Secret, cryptographic key that is used by the Key Expansion routine to generate a set of Round Keys; can be pictured as a rectangular array of bytes, having four rows and $Nk$ columns.

## Appendix A (continued)

Ciphertext	Data output from the Encryption or input to the decryption.
Decryption or Inverse Cipher	Series of transformations that converts ciphertext to plaintext using the Cipher Key.
Key Schedule	Routine used to generate a series of Round Keys from the Cipher Key.
Plaintext	Data input to the Cipher or output from the Inverse Cipher.
Rijndael	Cryptographic algorithm specified in this Advanced Encryption Standard (AES). Round Key Round keys are values derived from the Cipher Key using the Key Expansion routine; they are applied to the State in the Cipher and Inverse Cipher.
State	Intermediate Cipher result that can be pictured as a rectangular array of bytes, having four rows and Nb columns.
S-Box	A Non-linear substitution table used in several byte substitution transformations and in the Key Schedule routine to perform a one-for-one substitution of a byte value.
Word	A group of 32 bits that is treated either as a single entity or as an array of 4 bytes

## Appendix B: Cipher Example

The following diagram shows the values in the State array as the Encryption progresses for a block length and a Key length of 16 bytes each (i.e.,  $Nb = 4$  and  $Nk = 4$ ).

Input = 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

Cipher Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

Round Number	Start of Round	After SubBytes	After ShiftRows	After MixColumns	Round Key Value																																																																																
input	<table border="1"> <tr><td>32</td><td>88</td><td>31</td><td>e0</td></tr> <tr><td>43</td><td>5a</td><td>31</td><td>37</td></tr> <tr><td>f6</td><td>30</td><td>98</td><td>07</td></tr> <tr><td>a8</td><td>8d</td><td>a2</td><td>34</td></tr> </table>	32	88	31	e0	43	5a	31	37	f6	30	98	07	a8	8d	a2	34	<table border="1"> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>																	<table border="1"> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>																	<table border="1"> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>																	<table border="1"> <tr><td>2b</td><td>28</td><td>ab</td><td>09</td></tr> <tr><td>7e</td><td>ae</td><td>f7</td><td>cf</td></tr> <tr><td>15</td><td>d2</td><td>15</td><td>4f</td></tr> <tr><td>16</td><td>a6</td><td>88</td><td>3c</td></tr> </table>	2b	28	ab	09	7e	ae	f7	cf	15	d2	15	4f	16	a6	88	3c
32	88	31	e0																																																																																		
43	5a	31	37																																																																																		
f6	30	98	07																																																																																		
a8	8d	a2	34																																																																																		
2b	28	ab	09																																																																																		
7e	ae	f7	cf																																																																																		
15	d2	15	4f																																																																																		
16	a6	88	3c																																																																																		
1	<table border="1"> <tr><td>19</td><td>a0</td><td>9a</td><td>e9</td></tr> <tr><td>3d</td><td>f4</td><td>c6</td><td>f8</td></tr> <tr><td>e3</td><td>e2</td><td>8d</td><td>48</td></tr> <tr><td>be</td><td>2b</td><td>2a</td><td>08</td></tr> </table>	19	a0	9a	e9	3d	f4	c6	f8	e3	e2	8d	48	be	2b	2a	08	<table border="1"> <tr><td>d4</td><td>e0</td><td>b8</td><td>1e</td></tr> <tr><td>27</td><td>bf</td><td>b4</td><td>41</td></tr> <tr><td>11</td><td>98</td><td>5d</td><td>52</td></tr> <tr><td>ae</td><td>f1</td><td>e5</td><td>30</td></tr> </table>	d4	e0	b8	1e	27	bf	b4	41	11	98	5d	52	ae	f1	e5	30	<table border="1"> <tr><td>d4</td><td>e0</td><td>b8</td><td>1e</td></tr> <tr><td>bf</td><td>b4</td><td>41</td><td>27</td></tr> <tr><td>5d</td><td>52</td><td>11</td><td>98</td></tr> <tr><td>30</td><td>ae</td><td>f1</td><td>e5</td></tr> </table>	d4	e0	b8	1e	bf	b4	41	27	5d	52	11	98	30	ae	f1	e5	<table border="1"> <tr><td>04</td><td>e0</td><td>48</td><td>28</td></tr> <tr><td>66</td><td>cb</td><td>f8</td><td>06</td></tr> <tr><td>81</td><td>19</td><td>d3</td><td>26</td></tr> <tr><td>e5</td><td>9a</td><td>7a</td><td>4c</td></tr> </table>	04	e0	48	28	66	cb	f8	06	81	19	d3	26	e5	9a	7a	4c	<table border="1"> <tr><td>a0</td><td>88</td><td>23</td><td>2a</td></tr> <tr><td>fa</td><td>54</td><td>a3</td><td>6c</td></tr> <tr><td>fe</td><td>2c</td><td>39</td><td>76</td></tr> <tr><td>17</td><td>b1</td><td>39</td><td>05</td></tr> </table>	a0	88	23	2a	fa	54	a3	6c	fe	2c	39	76	17	b1	39	05
19	a0	9a	e9																																																																																		
3d	f4	c6	f8																																																																																		
e3	e2	8d	48																																																																																		
be	2b	2a	08																																																																																		
d4	e0	b8	1e																																																																																		
27	bf	b4	41																																																																																		
11	98	5d	52																																																																																		
ae	f1	e5	30																																																																																		
d4	e0	b8	1e																																																																																		
bf	b4	41	27																																																																																		
5d	52	11	98																																																																																		
30	ae	f1	e5																																																																																		
04	e0	48	28																																																																																		
66	cb	f8	06																																																																																		
81	19	d3	26																																																																																		
e5	9a	7a	4c																																																																																		
a0	88	23	2a																																																																																		
fa	54	a3	6c																																																																																		
fe	2c	39	76																																																																																		
17	b1	39	05																																																																																		
2	<table border="1"> <tr><td>a4</td><td>68</td><td>6b</td><td>02</td></tr> <tr><td>9c</td><td>9f</td><td>5b</td><td>6a</td></tr> <tr><td>7f</td><td>35</td><td>ea</td><td>50</td></tr> <tr><td>f2</td><td>2b</td><td>43</td><td>49</td></tr> </table>	a4	68	6b	02	9c	9f	5b	6a	7f	35	ea	50	f2	2b	43	49	<table border="1"> <tr><td>49</td><td>45</td><td>7f</td><td>77</td></tr> <tr><td>de</td><td>db</td><td>39</td><td>02</td></tr> <tr><td>d2</td><td>96</td><td>87</td><td>53</td></tr> <tr><td>89</td><td>f1</td><td>1a</td><td>3b</td></tr> </table>	49	45	7f	77	de	db	39	02	d2	96	87	53	89	f1	1a	3b	<table border="1"> <tr><td>49</td><td>45</td><td>7f</td><td>77</td></tr> <tr><td>db</td><td>39</td><td>02</td><td>de</td></tr> <tr><td>87</td><td>53</td><td>d2</td><td>96</td></tr> <tr><td>3b</td><td>89</td><td>f1</td><td>1a</td></tr> </table>	49	45	7f	77	db	39	02	de	87	53	d2	96	3b	89	f1	1a	<table border="1"> <tr><td>58</td><td>1b</td><td>db</td><td>1b</td></tr> <tr><td>4d</td><td>4b</td><td>e7</td><td>6b</td></tr> <tr><td>ca</td><td>5a</td><td>ca</td><td>b0</td></tr> <tr><td>f1</td><td>ac</td><td>a8</td><td>e5</td></tr> </table>	58	1b	db	1b	4d	4b	e7	6b	ca	5a	ca	b0	f1	ac	a8	e5	<table border="1"> <tr><td>f2</td><td>7a</td><td>59</td><td>73</td></tr> <tr><td>c2</td><td>96</td><td>35</td><td>59</td></tr> <tr><td>95</td><td>b9</td><td>80</td><td>f6</td></tr> <tr><td>f2</td><td>43</td><td>7a</td><td>7f</td></tr> </table>	f2	7a	59	73	c2	96	35	59	95	b9	80	f6	f2	43	7a	7f
a4	68	6b	02																																																																																		
9c	9f	5b	6a																																																																																		
7f	35	ea	50																																																																																		
f2	2b	43	49																																																																																		
49	45	7f	77																																																																																		
de	db	39	02																																																																																		
d2	96	87	53																																																																																		
89	f1	1a	3b																																																																																		
49	45	7f	77																																																																																		
db	39	02	de																																																																																		
87	53	d2	96																																																																																		
3b	89	f1	1a																																																																																		
58	1b	db	1b																																																																																		
4d	4b	e7	6b																																																																																		
ca	5a	ca	b0																																																																																		
f1	ac	a8	e5																																																																																		
f2	7a	59	73																																																																																		
c2	96	35	59																																																																																		
95	b9	80	f6																																																																																		
f2	43	7a	7f																																																																																		
3	<table border="1"> <tr><td>aa</td><td>61</td><td>82</td><td>68</td></tr> <tr><td>8f</td><td>dd</td><td>d2</td><td>32</td></tr> <tr><td>5f</td><td>e3</td><td>4a</td><td>46</td></tr> <tr><td>03</td><td>ef</td><td>d2</td><td>9a</td></tr> </table>	aa	61	82	68	8f	dd	d2	32	5f	e3	4a	46	03	ef	d2	9a	<table border="1"> <tr><td>ac</td><td>ef</td><td>13</td><td>45</td></tr> <tr><td>73</td><td>c1</td><td>b5</td><td>23</td></tr> <tr><td>cf</td><td>11</td><td>d6</td><td>5a</td></tr> <tr><td>7b</td><td>df</td><td>b5</td><td>b8</td></tr> </table>	ac	ef	13	45	73	c1	b5	23	cf	11	d6	5a	7b	df	b5	b8	<table border="1"> <tr><td>ac</td><td>ef</td><td>13</td><td>45</td></tr> <tr><td>c1</td><td>b5</td><td>23</td><td>73</td></tr> <tr><td>d6</td><td>5a</td><td>cf</td><td>11</td></tr> <tr><td>b8</td><td>7b</td><td>df</td><td>b5</td></tr> </table>	ac	ef	13	45	c1	b5	23	73	d6	5a	cf	11	b8	7b	df	b5	<table border="1"> <tr><td>75</td><td>20</td><td>53</td><td>bb</td></tr> <tr><td>ec</td><td>0b</td><td>c0</td><td>25</td></tr> <tr><td>09</td><td>63</td><td>cf</td><td>d0</td></tr> <tr><td>93</td><td>33</td><td>7c</td><td>dc</td></tr> </table>	75	20	53	bb	ec	0b	c0	25	09	63	cf	d0	93	33	7c	dc	<table border="1"> <tr><td>3d</td><td>47</td><td>1e</td><td>6d</td></tr> <tr><td>80</td><td>16</td><td>23</td><td>7a</td></tr> <tr><td>47</td><td>fe</td><td>7e</td><td>88</td></tr> <tr><td>7d</td><td>3e</td><td>44</td><td>3b</td></tr> </table>	3d	47	1e	6d	80	16	23	7a	47	fe	7e	88	7d	3e	44	3b
aa	61	82	68																																																																																		
8f	dd	d2	32																																																																																		
5f	e3	4a	46																																																																																		
03	ef	d2	9a																																																																																		
ac	ef	13	45																																																																																		
73	c1	b5	23																																																																																		
cf	11	d6	5a																																																																																		
7b	df	b5	b8																																																																																		
ac	ef	13	45																																																																																		
c1	b5	23	73																																																																																		
d6	5a	cf	11																																																																																		
b8	7b	df	b5																																																																																		
75	20	53	bb																																																																																		
ec	0b	c0	25																																																																																		
09	63	cf	d0																																																																																		
93	33	7c	dc																																																																																		
3d	47	1e	6d																																																																																		
80	16	23	7a																																																																																		
47	fe	7e	88																																																																																		
7d	3e	44	3b																																																																																		
4	<table border="1"> <tr><td>48</td><td>67</td><td>4d</td><td>d6</td></tr> <tr><td>6c</td><td>1d</td><td>e3</td><td>5f</td></tr> <tr><td>4e</td><td>9d</td><td>b1</td><td>58</td></tr> <tr><td>ee</td><td>0d</td><td>38</td><td>e7</td></tr> </table>	48	67	4d	d6	6c	1d	e3	5f	4e	9d	b1	58	ee	0d	38	e7	<table border="1"> <tr><td>52</td><td>85</td><td>e3</td><td>f6</td></tr> <tr><td>50</td><td>a4</td><td>11</td><td>cf</td></tr> <tr><td>2f</td><td>5e</td><td>c8</td><td>6a</td></tr> <tr><td>28</td><td>d7</td><td>07</td><td>94</td></tr> </table>	52	85	e3	f6	50	a4	11	cf	2f	5e	c8	6a	28	d7	07	94	<table border="1"> <tr><td>52</td><td>85</td><td>e3</td><td>f6</td></tr> <tr><td>a4</td><td>11</td><td>cf</td><td>50</td></tr> <tr><td>c8</td><td>6a</td><td>2f</td><td>5e</td></tr> <tr><td>94</td><td>28</td><td>d7</td><td>07</td></tr> </table>	52	85	e3	f6	a4	11	cf	50	c8	6a	2f	5e	94	28	d7	07	<table border="1"> <tr><td>0f</td><td>60</td><td>6f</td><td>5e</td></tr> <tr><td>d6</td><td>31</td><td>c0</td><td>b3</td></tr> <tr><td>da</td><td>38</td><td>10</td><td>13</td></tr> <tr><td>a9</td><td>bf</td><td>6b</td><td>01</td></tr> </table>	0f	60	6f	5e	d6	31	c0	b3	da	38	10	13	a9	bf	6b	01	<table border="1"> <tr><td>ef</td><td>a8</td><td>b6</td><td>db</td></tr> <tr><td>44</td><td>52</td><td>71</td><td>0b</td></tr> <tr><td>a5</td><td>5b</td><td>25</td><td>ad</td></tr> <tr><td>41</td><td>7f</td><td>3b</td><td>00</td></tr> </table>	ef	a8	b6	db	44	52	71	0b	a5	5b	25	ad	41	7f	3b	00
48	67	4d	d6																																																																																		
6c	1d	e3	5f																																																																																		
4e	9d	b1	58																																																																																		
ee	0d	38	e7																																																																																		
52	85	e3	f6																																																																																		
50	a4	11	cf																																																																																		
2f	5e	c8	6a																																																																																		
28	d7	07	94																																																																																		
52	85	e3	f6																																																																																		
a4	11	cf	50																																																																																		
c8	6a	2f	5e																																																																																		
94	28	d7	07																																																																																		
0f	60	6f	5e																																																																																		
d6	31	c0	b3																																																																																		
da	38	10	13																																																																																		
a9	bf	6b	01																																																																																		
ef	a8	b6	db																																																																																		
44	52	71	0b																																																																																		
a5	5b	25	ad																																																																																		
41	7f	3b	00																																																																																		

Appendix B (continued)

5	<table border="1"><tr><td>e0</td><td>c8</td><td>d9</td><td>85</td></tr><tr><td>92</td><td>63</td><td>b1</td><td>b8</td></tr><tr><td>7f</td><td>63</td><td>35</td><td>be</td></tr><tr><td>e8</td><td>c0</td><td>50</td><td>01</td></tr></table>	e0	c8	d9	85	92	63	b1	b8	7f	63	35	be	e8	c0	50	01	<table border="1"><tr><td>e1</td><td>e8</td><td>35</td><td>97</td></tr><tr><td>4f</td><td>fb</td><td>c8</td><td>6c</td></tr><tr><td>d2</td><td>fb</td><td>96</td><td>ae</td></tr><tr><td>9b</td><td>ba</td><td>53</td><td>7c</td></tr></table>	e1	e8	35	97	4f	fb	c8	6c	d2	fb	96	ae	9b	ba	53	7c	<table border="1"><tr><td>e1</td><td>e8</td><td>35</td><td>97</td></tr><tr><td>fb</td><td>c8</td><td>6c</td><td>4f</td></tr><tr><td>96</td><td>ae</td><td>d2</td><td>fb</td></tr><tr><td>7c</td><td>9b</td><td>ba</td><td>53</td></tr></table>	e1	e8	35	97	fb	c8	6c	4f	96	ae	d2	fb	7c	9b	ba	53	<table border="1"><tr><td>25</td><td>bd</td><td>b6</td><td>4c</td></tr><tr><td>d1</td><td>11</td><td>3a</td><td>4c</td></tr><tr><td>a9</td><td>d1</td><td>33</td><td>c0</td></tr><tr><td>ad</td><td>68</td><td>8e</td><td>b0</td></tr></table>	25	bd	b6	4c	d1	11	3a	4c	a9	d1	33	c0	ad	68	8e	b0	$\oplus$	<table border="1"><tr><td>d4</td><td>7c</td><td>ca</td><td>11</td></tr><tr><td>d1</td><td>83</td><td>f2</td><td>f9</td></tr><tr><td>c6</td><td>9d</td><td>b8</td><td>15</td></tr><tr><td>f8</td><td>87</td><td>bc</td><td>bc</td></tr></table>	d4	7c	ca	11	d1	83	f2	f9	c6	9d	b8	15	f8	87	bc	bc	=
e0	c8	d9	85																																																																																				
92	63	b1	b8																																																																																				
7f	63	35	be																																																																																				
e8	c0	50	01																																																																																				
e1	e8	35	97																																																																																				
4f	fb	c8	6c																																																																																				
d2	fb	96	ae																																																																																				
9b	ba	53	7c																																																																																				
e1	e8	35	97																																																																																				
fb	c8	6c	4f																																																																																				
96	ae	d2	fb																																																																																				
7c	9b	ba	53																																																																																				
25	bd	b6	4c																																																																																				
d1	11	3a	4c																																																																																				
a9	d1	33	c0																																																																																				
ad	68	8e	b0																																																																																				
d4	7c	ca	11																																																																																				
d1	83	f2	f9																																																																																				
c6	9d	b8	15																																																																																				
f8	87	bc	bc																																																																																				
6	<table border="1"><tr><td>f1</td><td>c1</td><td>7c</td><td>5d</td></tr><tr><td>00</td><td>92</td><td>c8</td><td>b5</td></tr><tr><td>6f</td><td>4c</td><td>8b</td><td>d5</td></tr><tr><td>55</td><td>ef</td><td>32</td><td>0c</td></tr></table>	f1	c1	7c	5d	00	92	c8	b5	6f	4c	8b	d5	55	ef	32	0c	<table border="1"><tr><td>a1</td><td>78</td><td>10</td><td>4c</td></tr><tr><td>63</td><td>4f</td><td>e8</td><td>d5</td></tr><tr><td>a8</td><td>29</td><td>3d</td><td>03</td></tr><tr><td>fc</td><td>df</td><td>23</td><td>fe</td></tr></table>	a1	78	10	4c	63	4f	e8	d5	a8	29	3d	03	fc	df	23	fe	<table border="1"><tr><td>a1</td><td>78</td><td>10</td><td>4c</td></tr><tr><td>4f</td><td>e8</td><td>d5</td><td>63</td></tr><tr><td>3d</td><td>03</td><td>a8</td><td>29</td></tr><tr><td>fe</td><td>fc</td><td>df</td><td>23</td></tr></table>	a1	78	10	4c	4f	e8	d5	63	3d	03	a8	29	fe	fc	df	23	<table border="1"><tr><td>4b</td><td>2c</td><td>33</td><td>37</td></tr><tr><td>86</td><td>4a</td><td>9d</td><td>d2</td></tr><tr><td>8d</td><td>89</td><td>f4</td><td>18</td></tr><tr><td>6d</td><td>80</td><td>e8</td><td>d8</td></tr></table>	4b	2c	33	37	86	4a	9d	d2	8d	89	f4	18	6d	80	e8	d8	$\oplus$	<table border="1"><tr><td>6d</td><td>11</td><td>db</td><td>ca</td></tr><tr><td>88</td><td>0b</td><td>f9</td><td>00</td></tr><tr><td>a3</td><td>3e</td><td>86</td><td>93</td></tr><tr><td>7a</td><td>fd</td><td>41</td><td>fd</td></tr></table>	6d	11	db	ca	88	0b	f9	00	a3	3e	86	93	7a	fd	41	fd	=
f1	c1	7c	5d																																																																																				
00	92	c8	b5																																																																																				
6f	4c	8b	d5																																																																																				
55	ef	32	0c																																																																																				
a1	78	10	4c																																																																																				
63	4f	e8	d5																																																																																				
a8	29	3d	03																																																																																				
fc	df	23	fe																																																																																				
a1	78	10	4c																																																																																				
4f	e8	d5	63																																																																																				
3d	03	a8	29																																																																																				
fe	fc	df	23																																																																																				
4b	2c	33	37																																																																																				
86	4a	9d	d2																																																																																				
8d	89	f4	18																																																																																				
6d	80	e8	d8																																																																																				
6d	11	db	ca																																																																																				
88	0b	f9	00																																																																																				
a3	3e	86	93																																																																																				
7a	fd	41	fd																																																																																				
7	<table border="1"><tr><td>26</td><td>3d</td><td>e8</td><td>fd</td></tr><tr><td>0e</td><td>41</td><td>64</td><td>d2</td></tr><tr><td>2e</td><td>b7</td><td>72</td><td>8b</td></tr><tr><td>17</td><td>7d</td><td>a9</td><td>25</td></tr></table>	26	3d	e8	fd	0e	41	64	d2	2e	b7	72	8b	17	7d	a9	25	<table border="1"><tr><td>f7</td><td>27</td><td>9b</td><td>54</td></tr><tr><td>ab</td><td>83</td><td>43</td><td>b5</td></tr><tr><td>31</td><td>a9</td><td>40</td><td>3d</td></tr><tr><td>f0</td><td>ff</td><td>d3</td><td>3f</td></tr></table>	f7	27	9b	54	ab	83	43	b5	31	a9	40	3d	f0	ff	d3	3f	<table border="1"><tr><td>f7</td><td>27</td><td>9b</td><td>54</td></tr><tr><td>83</td><td>43</td><td>b5</td><td>ab</td></tr><tr><td>40</td><td>3d</td><td>31</td><td>a9</td></tr><tr><td>3f</td><td>f0</td><td>ff</td><td>d3</td></tr></table>	f7	27	9b	54	83	43	b5	ab	40	3d	31	a9	3f	f0	ff	d3	<table border="1"><tr><td>14</td><td>46</td><td>27</td><td>34</td></tr><tr><td>15</td><td>16</td><td>46</td><td>2a</td></tr><tr><td>b5</td><td>15</td><td>56</td><td>d8</td></tr><tr><td>bf</td><td>ec</td><td>d7</td><td>43</td></tr></table>	14	46	27	34	15	16	46	2a	b5	15	56	d8	bf	ec	d7	43	$\oplus$	<table border="1"><tr><td>4e</td><td>5f</td><td>84</td><td>4e</td></tr><tr><td>54</td><td>5f</td><td>a6</td><td>a6</td></tr><tr><td>f7</td><td>c9</td><td>4f</td><td>dc</td></tr><tr><td>0e</td><td>f3</td><td>b2</td><td>4f</td></tr></table>	4e	5f	84	4e	54	5f	a6	a6	f7	c9	4f	dc	0e	f3	b2	4f	=
26	3d	e8	fd																																																																																				
0e	41	64	d2																																																																																				
2e	b7	72	8b																																																																																				
17	7d	a9	25																																																																																				
f7	27	9b	54																																																																																				
ab	83	43	b5																																																																																				
31	a9	40	3d																																																																																				
f0	ff	d3	3f																																																																																				
f7	27	9b	54																																																																																				
83	43	b5	ab																																																																																				
40	3d	31	a9																																																																																				
3f	f0	ff	d3																																																																																				
14	46	27	34																																																																																				
15	16	46	2a																																																																																				
b5	15	56	d8																																																																																				
bf	ec	d7	43																																																																																				
4e	5f	84	4e																																																																																				
54	5f	a6	a6																																																																																				
f7	c9	4f	dc																																																																																				
0e	f3	b2	4f																																																																																				
8	<table border="1"><tr><td>5a</td><td>19</td><td>a3</td><td>7a</td></tr><tr><td>41</td><td>49</td><td>e0</td><td>8c</td></tr><tr><td>42</td><td>dc</td><td>19</td><td>04</td></tr><tr><td>b1</td><td>1f</td><td>65</td><td>0c</td></tr></table>	5a	19	a3	7a	41	49	e0	8c	42	dc	19	04	b1	1f	65	0c	<table border="1"><tr><td>be</td><td>d4</td><td>0a</td><td>da</td></tr><tr><td>83</td><td>3b</td><td>e1</td><td>64</td></tr><tr><td>2c</td><td>86</td><td>d4</td><td>f2</td></tr><tr><td>c8</td><td>c0</td><td>4d</td><td>fe</td></tr></table>	be	d4	0a	da	83	3b	e1	64	2c	86	d4	f2	c8	c0	4d	fe	<table border="1"><tr><td>be</td><td>d4</td><td>0a</td><td>da</td></tr><tr><td>3b</td><td>e1</td><td>64</td><td>83</td></tr><tr><td>d4</td><td>f2</td><td>2c</td><td>86</td></tr><tr><td>fe</td><td>c8</td><td>c0</td><td>4d</td></tr></table>	be	d4	0a	da	3b	e1	64	83	d4	f2	2c	86	fe	c8	c0	4d	<table border="1"><tr><td>00</td><td>b1</td><td>54</td><td>fa</td></tr><tr><td>51</td><td>c8</td><td>76</td><td>1b</td></tr><tr><td>2f</td><td>89</td><td>6d</td><td>99</td></tr><tr><td>d1</td><td>ff</td><td>cd</td><td>ea</td></tr></table>	00	b1	54	fa	51	c8	76	1b	2f	89	6d	99	d1	ff	cd	ea	$\oplus$	<table border="1"><tr><td>ea</td><td>b5</td><td>31</td><td>7f</td></tr><tr><td>d2</td><td>8d</td><td>2b</td><td>8d</td></tr><tr><td>73</td><td>ba</td><td>f5</td><td>29</td></tr><tr><td>21</td><td>d2</td><td>60</td><td>2f</td></tr></table>	ea	b5	31	7f	d2	8d	2b	8d	73	ba	f5	29	21	d2	60	2f	=
5a	19	a3	7a																																																																																				
41	49	e0	8c																																																																																				
42	dc	19	04																																																																																				
b1	1f	65	0c																																																																																				
be	d4	0a	da																																																																																				
83	3b	e1	64																																																																																				
2c	86	d4	f2																																																																																				
c8	c0	4d	fe																																																																																				
be	d4	0a	da																																																																																				
3b	e1	64	83																																																																																				
d4	f2	2c	86																																																																																				
fe	c8	c0	4d																																																																																				
00	b1	54	fa																																																																																				
51	c8	76	1b																																																																																				
2f	89	6d	99																																																																																				
d1	ff	cd	ea																																																																																				
ea	b5	31	7f																																																																																				
d2	8d	2b	8d																																																																																				
73	ba	f5	29																																																																																				
21	d2	60	2f																																																																																				
9	<table border="1"><tr><td>ea</td><td>04</td><td>65</td><td>85</td></tr><tr><td>83</td><td>45</td><td>5d</td><td>96</td></tr><tr><td>5c</td><td>33</td><td>98</td><td>b0</td></tr><tr><td>f0</td><td>2d</td><td>ad</td><td>c5</td></tr></table>	ea	04	65	85	83	45	5d	96	5c	33	98	b0	f0	2d	ad	c5	<table border="1"><tr><td>87</td><td>f2</td><td>4d</td><td>97</td></tr><tr><td>ec</td><td>6e</td><td>4c</td><td>90</td></tr><tr><td>4a</td><td>c3</td><td>46</td><td>e7</td></tr><tr><td>8c</td><td>d8</td><td>95</td><td>a6</td></tr></table>	87	f2	4d	97	ec	6e	4c	90	4a	c3	46	e7	8c	d8	95	a6	<table border="1"><tr><td>87</td><td>f2</td><td>4d</td><td>97</td></tr><tr><td>6e</td><td>4c</td><td>90</td><td>ec</td></tr><tr><td>46</td><td>e7</td><td>4a</td><td>c3</td></tr><tr><td>a6</td><td>8c</td><td>d8</td><td>95</td></tr></table>	87	f2	4d	97	6e	4c	90	ec	46	e7	4a	c3	a6	8c	d8	95	<table border="1"><tr><td>47</td><td>40</td><td>a3</td><td>4c</td></tr><tr><td>37</td><td>d4</td><td>70</td><td>9f</td></tr><tr><td>94</td><td>e4</td><td>3a</td><td>42</td></tr><tr><td>ed</td><td>a5</td><td>a6</td><td>bc</td></tr></table>	47	40	a3	4c	37	d4	70	9f	94	e4	3a	42	ed	a5	a6	bc	$\oplus$	<table border="1"><tr><td>ac</td><td>19</td><td>28</td><td>57</td></tr><tr><td>77</td><td>fa</td><td>d1</td><td>5c</td></tr><tr><td>66</td><td>dc</td><td>29</td><td>00</td></tr><tr><td>f3</td><td>21</td><td>41</td><td>6e</td></tr></table>	ac	19	28	57	77	fa	d1	5c	66	dc	29	00	f3	21	41	6e	=
ea	04	65	85																																																																																				
83	45	5d	96																																																																																				
5c	33	98	b0																																																																																				
f0	2d	ad	c5																																																																																				
87	f2	4d	97																																																																																				
ec	6e	4c	90																																																																																				
4a	c3	46	e7																																																																																				
8c	d8	95	a6																																																																																				
87	f2	4d	97																																																																																				
6e	4c	90	ec																																																																																				
46	e7	4a	c3																																																																																				
a6	8c	d8	95																																																																																				
47	40	a3	4c																																																																																				
37	d4	70	9f																																																																																				
94	e4	3a	42																																																																																				
ed	a5	a6	bc																																																																																				
ac	19	28	57																																																																																				
77	fa	d1	5c																																																																																				
66	dc	29	00																																																																																				
f3	21	41	6e																																																																																				
10	<table border="1"><tr><td>eb</td><td>59</td><td>8b</td><td>1b</td></tr><tr><td>40</td><td>2e</td><td>a1</td><td>c3</td></tr><tr><td>f2</td><td>38</td><td>13</td><td>42</td></tr><tr><td>1e</td><td>84</td><td>e7</td><td>d2</td></tr></table>	eb	59	8b	1b	40	2e	a1	c3	f2	38	13	42	1e	84	e7	d2	<table border="1"><tr><td>e9</td><td>cb</td><td>3d</td><td>af</td></tr><tr><td>09</td><td>31</td><td>32</td><td>2e</td></tr><tr><td>89</td><td>07</td><td>7d</td><td>2c</td></tr><tr><td>72</td><td>5f</td><td>94</td><td>b5</td></tr></table>	e9	cb	3d	af	09	31	32	2e	89	07	7d	2c	72	5f	94	b5	<table border="1"><tr><td>e9</td><td>cb</td><td>3d</td><td>af</td></tr><tr><td>31</td><td>32</td><td>2e</td><td>09</td></tr><tr><td>7d</td><td>2c</td><td>89</td><td>07</td></tr><tr><td>b5</td><td>72</td><td>5f</td><td>94</td></tr></table>	e9	cb	3d	af	31	32	2e	09	7d	2c	89	07	b5	72	5f	94	<table border="1"><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr></table>																	$\oplus$	<table border="1"><tr><td>d0</td><td>c9</td><td>e1</td><td>b6</td></tr><tr><td>14</td><td>ee</td><td>3f</td><td>63</td></tr><tr><td>f9</td><td>25</td><td>0c</td><td>0c</td></tr><tr><td>a8</td><td>89</td><td>c8</td><td>a6</td></tr></table>	d0	c9	e1	b6	14	ee	3f	63	f9	25	0c	0c	a8	89	c8	a6	=
eb	59	8b	1b																																																																																				
40	2e	a1	c3																																																																																				
f2	38	13	42																																																																																				
1e	84	e7	d2																																																																																				
e9	cb	3d	af																																																																																				
09	31	32	2e																																																																																				
89	07	7d	2c																																																																																				
72	5f	94	b5																																																																																				
e9	cb	3d	af																																																																																				
31	32	2e	09																																																																																				
7d	2c	89	07																																																																																				
b5	72	5f	94																																																																																				
d0	c9	e1	b6																																																																																				
14	ee	3f	63																																																																																				
f9	25	0c	0c																																																																																				
a8	89	c8	a6																																																																																				

**Appendix B (continued)**

output	39	02	dc	19
	25	dc	11	6a
	84	09	85	0b
	1d	fb	97	32



## Appendix C: Example Vectors

This appendix contains example vectors, including intermediate values – for all three AES key lengths ( $N_k = 4, 6,$  and  $8$ ), for the Encryption and Decryption. All vectors are in hexadecimal notation, with each pair of characters giving a byte value in which the left character of each pair provides the bit pattern for the 4 bit group containing the higher numbered bits using the notation explained in Section 1.22, while the right character provides the bit pattern for the lower-numbered bits. The array index for all bytes (groups of two hexadecimal digits) within these test vectors starts at zero and increases from left to right.

### AES-128 ( $N_k=4, N_r=10$ )

```
PLAINTEXT: 00112233445566778899aabbccddeeff
KEY: 000102030405060708090a0b0c0d0e0f
CIPHER (ENCRYPT):
round[ 0].input 00112233445566778899aabbccddeeff
round[ 0].k_sch 000102030405060708090a0b0c0d0e0f
round[ 1].start 00102030405060708090a0b0c0d0e0f0
round[ 1].s_box 63cab7040953d051cd60e0e7ba70e18c
round[ 1].s_row 6353e08c0960e104cd70b751bacad0e7
round[ 1].m_col 5f72641557f5bc92f7be3b291db9f91a
round[ 1].k_sch d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 2].start 89d810e8855ace682d1843d8cb128fe4
round[ 2].s_box a761ca9b97be8b45d8ad1a611fc97369
round[ 2].s_row a7be1a6997ad739bd8c9ca451f618b61
round[ 2].m_col ff87968431d86a51645151fa773ad009
round[ 2].k_sch b692cf0b643dbdf1be9bc5006830b3fe
round[ 3].start 4915598f55e5d7a0daca94fa1f0a63f7
```

## Appendix C (continued)

round[ 3].s\_box 3b59cb73fcd90ee05774222dc067fb68  
round[ 3].s\_row 3bd92268fc74fb735767cbe0c0590e2d  
round[ 3].m\_col 4c9c1e66f771f0762c3f868e534df256  
round[ 3].k\_sch b6ff744ed2c2c9bf6c590cbf0469bf41  
round[ 4].start fa636a2825b339c940668a3157244d17  
round[ 4].s\_box 2dfb02343f6d12dd09337ec75b36e3f0  
round[ 4].s\_row 2d6d7ef03f33e334093602dd5bfb12c7  
round[ 4].m\_col 6385b79ffc538df997be478e7547d691  
round[ 4].k\_sch 47f7f7bc95353e03f96c32bcfd058dfd  
round[ 5].start 247240236966b3fa6ed2753288425b6c  
round[ 5].s\_box 36400926f9336d2d9fb59d23c42c3950  
round[ 5].s\_row 36339d50f9b539269f2c092dc4406d23  
round[ 5].m\_col f4bcd45432e554d075f1d6c51dd03b3c  
round[ 5].k\_sch 3caaa3e8a99f9deb50f3af57adf622aa  
round[ 6].start c81677bc9b7ac93b25027992b0261996  
round[ 6].s\_box e847f56514dadde23f77b64fe7f7d490  
round[ 6].s\_row e8dab6901477d4653ff7f5e2e747dd4f  
round[ 6].m\_col 9816ee7400f87f556b2c049c8e5ad036  
round[ 6].k\_sch 5e390f7df7a69296a7553dc10aa31f6b  
round[ 7].start c62fe109f75eedc3cc79395d84f9cf5d  
round[ 7].s\_box b415f8016858552e4bb6124c5f998a4c  
round[ 7].s\_row b458124c68b68a014b99f82e5f15554c  
round[ 7].m\_col c57e1c159a9bd286f05f4be098c63439  
round[ 7].k\_sch 14f9701ae35fe28c440adf4d4ea9c026  
round[ 8].start d1876c0f79c4300ab45594add66ff41f  
round[ 8].s\_box 3e175076b61c04678dfc2295f6a8bfc0  
round[ 8].s\_row 3e1c22c0b6fcbf768da85067f6170495

## Appendix C (continued)

```
round[ 8].m_col baa03de7a1f9b56ed5512cba5f414d23
round[ 8].k_sch 47438735a41c65b9e016baf4aebf7ad2
round[ 9].start fde3bad205e5d0d73547964ef1fe37f1
round[ 9].s_box 5411f4b56bd9700e96a0902fa1bb9aa1
round[ 9].s_row 54d990a16ba09ab596bbf40ea111702f
round[ 9].m_col e9f74eec023020f61bf2ccf2353c21c7
round[ 9].k_sch 549932d1f08557681093ed9cbe2c974e
round[10].start bd6e7c3df2b5779e0b61216e8b10b689
round[10].s_box 7a9f102789d5f50b2beffd9f3dca4ea7
round[10].s_row 7ad5fda789ef4e272bca100b3d9ff59f
round[10].k_sch 13111d7fe3944a17f307a78b4d2b30c5
round[10].output 69c4e0d86a7b0430d8cdb78070b4c55a
```

INVERSE CIPHER (DECRYPT):

```
round[ 0].iinput 69c4e0d86a7b0430d8cdb78070b4c55a
round[ 0].ik_sch 13111d7fe3944a17f307a78b4d2b30c5
round[ 1].istart 7ad5fda789ef4e272bca100b3d9ff59f
round[ 1].is_row 7a9f102789d5f50b2beffd9f3dca4ea7
round[ 1].is_box bd6e7c3df2b5779e0b61216e8b10b689
round[ 1].ik_sch 549932d1f08557681093ed9cbe2c974e
round[ 1].ik_add e9f74eec023020f61bf2ccf2353c21c7
round[ 2].istart 54d990a16ba09ab596bbf40ea111702f
round[ 2].is_row 5411f4b56bd9700e96a0902fa1bb9aa1
round[ 2].is_box fde3bad205e5d0d73547964ef1fe37f1
round[ 2].ik_sch 47438735a41c65b9e016baf4aebf7ad2
round[ 2].ik_add baa03de7a1f9b56ed5512cba5f414d23
round[ 3].istart 3e1c22c0b6fcbf768da85067f6170495
```

## Appendix C (continued)

round[ 3].is\_row 3e175076b61c04678dfc2295f6a8bfc0  
round[ 3].is\_box d1876c0f79c4300ab45594add66ff41f  
round[ 3].ik\_sch 14f9701ae35fe28c440adf4d4ea9c026  
round[ 3].ik\_add c57e1c159a9bd286f05f4be098c63439  
round[ 4].istart b458124c68b68a014b99f82e5f15554c  
round[ 4].is\_row b415f8016858552e4bb6124c5f998a4c  
round[ 4].is\_box c62fe109f75eedc3cc79395d84f9cf5d  
round[ 4].ik\_sch 5e390f7df7a69296a7553dc10aa31f6b  
round[ 4].ik\_add 9816ee7400f87f556b2c049c8e5ad036  
round[ 5].istart e8dab6901477d4653ff7f5e2e747dd4f  
round[ 5].is\_row e847f56514dadde23f77b64fe7f7d490  
round[ 5].is\_box c81677bc9b7ac93b25027992b0261996  
round[ 5].ik\_sch 3caaa3e8a99f9deb50f3af57adf622aa  
round[ 5].ik\_add f4bcd45432e554d075f1d6c51dd03b3c  
round[ 6].istart 36339d50f9b539269f2c092dc4406d23  
round[ 6].is\_row 36400926f9336d2d9fb59d23c42c3950  
round[ 6].is\_box 247240236966b3fa6ed2753288425b6c  
round[ 6].ik\_sch 47f7f7bc95353e03f96c32bcfd058dfd  
round[ 6].ik\_add 6385b79ffc538df997be478e7547d691  
round[ 7].istart 2d6d7ef03f33e334093602dd5bfb12c7  
round[ 7].is\_row 2dfb02343f6d12dd09337ec75b36e3f0  
round[ 7].is\_box fa636a2825b339c940668a3157244d17  
round[ 7].ik\_sch b6ff744ed2c2c9bf6c590cbf0469bf41  
round[ 7].ik\_add 4c9c1e66f771f0762c3f868e534df256  
round[ 8].istart 3bd92268fc74fb735767cbe0c0590e2d  
round[ 8].is\_row 3b59cb73fcd90ee05774222dc067fb68  
round[ 8].is\_box 4915598f55e5d7a0daca94fa1f0a63f7

## Appendix C (continued)

```
round[ 8].ik_sch b692cf0b643dbdf1be9bc5006830b3fe
round[ 8].ik_add ff87968431d86a51645151fa773ad009
round[ 9].istart a7be1a6997ad739bd8c9ca451f618b61
round[ 9].is_row a761ca9b97be8b45d8ad1a611fc97369
round[ 9].is_box 89d810e8855ace682d1843d8cb128fe4
round[ 9].ik_sch d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 9].ik_add 5f72641557f5bc92f7be3b291db9f91a
round[10].istart 6353e08c0960e104cd70b751bacad0e7
round[10].is_row 63cab7040953d051cd60e0e7ba70e18c
round[10].is_box 00102030405060708090a0b0c0d0e0f0
round[10].ik_sch 000102030405060708090a0b0c0d0e0f
round[10].ioutput 00112233445566778899aabbccddeeff
```

### AES-192 (Nk=6, Nr=12)

```
PLAINTEXT: 00112233445566778899aabbccddeeff
KEY: 000102030405060708090a0b0c0d0e0f1011121314151617
CIPHER (ENCRYPT):
round[ 0].input 00112233445566778899aabbccddeeff
round[ 0].k_sch 000102030405060708090a0b0c0d0e0f
round[ 1].start 00102030405060708090a0b0c0d0e0f0
round[ 1].s_box 63cab7040953d051cd60e0e7ba70e18c
round[ 1].s_row 6353e08c0960e104cd70b751bacad0e7
round[ 1].m_col 5f72641557f5bc92f7be3b291db9f91a
round[ 1].k_sch 10111213141516175846f2f95c43f4fe
round[ 2].start 4f63760643e0aa85aff8c9d041fa0de4
round[ 2].s_box 84fb386f1ae1ac977941dd70832dd769
round[ 2].s_row 84e1dd691a41d76f792d389783fbac70
round[ 2].m_col 9f487f794f955f662afc86abd7f1ab29
```

## Appendix C (continued)

round[ 2].k\_sch 544afef55847f0fa4856e2e95c43f4fe  
round[ 3].start cb02818c17d2af9c62aa64428bb25fd7  
round[ 3].s\_box 1f770c64f0b579deaaac432c3d37cf0e  
round[ 3].s\_row 1fb5430ef0accf64aa370cde3d77792c  
round[ 3].m\_col b7a53ecbbf9d75a0c40efc79b674cc11  
round[ 3].k\_sch 40f949b31cbabd4d48f043b810b7b342  
round[ 4].start f75c7778a327c8ed8cfefbfc1a6c37f53  
round[ 4].s\_box 684af5bc0acce85564bb0878242ed2ed  
round[ 4].s\_row 68cc08ed0abbd2bc642ef555244ae878  
round[ 4].m\_col 7a1e98bdacb6d1141a6944dd06eb2d3e  
round[ 4].k\_sch 58e151ab04a2a5557effb5416245080c  
round[ 5].start 22ffc916a81474416496f19c64ae2532  
round[ 5].s\_box 9316dd47c2fa92834390alde43e43f23  
round[ 5].s\_row 93faa123c2903f4743e4dd83431692de  
round[ 5].m\_col aaa755b34cffe57cef6f98e1f01c13e6  
round[ 5].k\_sch 2ab54bb43a02f8f662e3a95d66410c08  
round[ 6].start 80121e0776fd1d8a8d8c31bc965d1fee  
round[ 6].s\_box cdc972c53854a47e5d64c765904cc028  
round[ 6].s\_row cd54c7283864c0c55d4c727e90c9a465  
round[ 6].m\_col 921f748fd96e937d622d7725ba8ba50c  
round[ 6].k\_sch f501857297448d7ebdf1c6ca87f33e3c  
round[ 7].start 671ef1fd4e2a1e03dfdcblf3d789b30  
round[ 7].s\_box 8572a1542fe5727b9e86c8df27bc1404  
round[ 7].s\_row 85e5c8042f8614549ebca17b277272df  
round[ 7].m\_col e913e7b18f507d4b227ef652758acbcc  
round[ 7].k\_sch e510976183519b6934157c9ea351f1e0  
round[ 8].start 0c0370d00c01e622166b8accd6db3a2c

## Appendix C (continued)

```
round[ 8].s_box fe7b5170fe7c8e93477f7e4bf6b98071
round[ 8].s_row fe7c7e71fe7f807047b95193f67b8e4b
round[ 8].m_col 6cf5edf996eb0a069c4ef21cbfc25762
round[ 8].k_sch 1ea0372a995309167c439e77ff12051e
round[ 9].start 7255dad30fb80310e00d6c6b40d0527c
round[ 9].s_box 40fc5766766c7bcae1d7507f09700010
round[ 9].s_row 406c501076d70066e17057ca09fc7b7f
round[ 9].m_col 7478bcdce8a50b81d4327a9009188262
round[ 9].k_sch dd7e0e887e2fff68608fc842f9dcc154
round[10].start a906b254968af4e9b4bdb2d2f0c44336
round[10].s_box d36f3720907ebf1e8d7a37b58c1c1a05
round[10].s_row d37e3705907a1a208d1c371e8c6fbfb5
round[10].m_col 0d73cc2d8f6abe8b0cf2dd9bb83d422e
round[10].k_sch 859f5f237a8d5a3dc0c02952beefd63a
round[11].start 88ec930ef5e7e4b6cc32f4c906d29414
round[11].s_box c4cedcabe694694e4b23bfdd6fb522fa
round[11].s_row c494bffae62322ab4bb5dc4e6fce69dd
round[11].m_col 71d720933b6d677dc00b8f28238e0fb7
round[11].k_sch de601e7827bcdcf2ca223800fd8aeda32
round[12].start afb73eeb1cd1b85162280f27fb20d585
round[12].s_box 79a9b2e99c3e6cd1aa3476cc0fb70397
round[12].s_row 793e76979c3403e9aab7b2d10fa96ccc
round[12].k_sch a4970a331a78dc09c418c271e3a41d5d
round[12].output dda97ca4864cdf06eaf70a0ec0d7191
```

INVERSE CIPHER (DECRYPT):

```
round[ 0].iinput dda97ca4864cdf06eaf70a0ec0d7191
```

## Appendix C (continued)

round[ 0].ik\_sch a4970a331a78dc09c418c271e3a41d5d  
round[ 1].istart 793e76979c3403e9aab7b2d10fa96ccc  
round[ 1].is\_row 79a9b2e99c3e6cd1aa3476cc0fb70397  
round[ 1].is\_box afb73eeb1cd1b85162280f27fb20d585  
round[ 1].ik\_sch de601e7827bcd2ca223800fd8aeda32  
round[ 1].ik\_add 71d720933b6d677dc00b8f28238e0fb7  
round[ 2].istart c494bffa62322ab4bb5dc4e6fce69dd  
round[ 2].is\_row c4cedcabe694694e4b23bfdd6fb522fa  
round[ 2].is\_box 88ec930ef5e7e4b6cc32f4c906d29414  
round[ 2].ik\_sch 859f5f237a8d5a3dc0c02952beefd63a  
round[ 2].ik\_add 0d73cc2d8f6abe8b0cf2dd9bb83d422e  
round[ 3].istart d37e3705907a1a208d1c371e8c6fbfb5  
round[ 3].is\_row d36f3720907ebf1e8d7a37b58c1c1a05  
round[ 3].is\_box a906b254968af4e9b4bdb2d2f0c44336  
round[ 3].ik\_sch dd7e0e887e2fff68608fc842f9dcc154  
round[ 3].ik\_add 7478bcdce8a50b81d4327a9009188262  
round[ 4].istart 406c501076d70066e17057ca09fc7b7f  
round[ 4].is\_row 40fc5766766c7bcaeld7507f09700010  
round[ 4].is\_box 7255dad30fb80310e00d6c6b40d0527c  
round[ 4].ik\_sch 1ea0372a995309167c439e77ff12051e  
round[ 4].ik\_add 6cf5edf996eb0a069c4ef21cbfc25762  
round[ 5].istart fe7c7e71fe7f807047b95193f67b8e4b  
round[ 5].is\_row fe7b5170fe7c8e93477f7e4bf6b98071  
round[ 5].is\_box 0c0370d00c01e622166b8accd6db3a2c  
round[ 5].ik\_sch e510976183519b6934157c9ea351f1e0  
round[ 5].ik\_add e913e7b18f507d4b227ef652758acbcc  
round[ 6].istart 85e5c8042f8614549ebca17b277272df



## Appendix C (continued)

round[ 6].is\_row 8572a1542fe5727b9e86c8df27bc1404  
round[ 6].is\_box 671ef1fd4e2a1e03dfdcbléf3d789b30  
round[ 6].ik\_sch f501857297448d7ebdf1c6ca87f33e3c  
round[ 6].ik\_add 921f748fd96e937d622d7725ba8ba50c  
round[ 7].istart cd54c7283864c0c55d4c727e90c9a465  
round[ 7].is\_row cdc972c53854a47e5d64c765904cc028  
round[ 7].is\_box 80121e0776fd1d8a8d8c31bc965d1fee  
round[ 7].ik\_sch 2ab54bb43a02f8f662e3a95d66410c08  
round[ 7].ik\_add aaa755b34cffe57cef6f98e1f01c13e6  
round[ 8].istart 93faa123c2903f4743e4dd83431692de  
round[ 8].is\_row 9316dd47c2fa92834390alde43e43f23  
round[ 8].is\_box 22ffc916a81474416496f19c64ae2532  
round[ 8].ik\_sch 58e151ab04a2a5557effb5416245080c  
round[ 8].ik\_add 7ale98bdacb6d1141a6944dd06eb2d3e  
round[ 9].istart 68cc08ed0abbd2bc642ef555244ae878  
round[ 9].is\_row 684af5bc0acce85564bb0878242ed2ed  
round[ 9].is\_box f75c7778a327c8ed8cfefbfc1a6c37f53  
round[ 9].ik\_sch 40f949b31cbabd4d48f043b810b7b342  
round[ 9].ik\_add b7a53ecbbf9d75a0c40efc79b674cc11  
round[10].istart 1fb5430ef0accf64aa370cde3d77792c  
round[10].is\_row 1f770c64f0b579deaaac432c3d37cf0e  
round[10].is\_box cb02818c17d2af9c62aa64428bb25fd7  
round[10].ik\_sch 544afef55847f0fa4856e2e95c43f4fe  
round[10].ik\_add 9f487f794f955f662afc86abd7f1ab29  
round[11].istart 84e1dd691a41d76f792d389783fbac70  
round[11].is\_row 84fb386f1aelac977941dd70832dd769  
round[11].is\_box 4f63760643e0aa85aff8c9d041fa0de4

## Appendix C (continued)

```
round[11].ik_sch 10111213141516175846f2f95c43f4fe
round[11].ik_add 5f72641557f5bc92f7be3b291db9f91a
round[12].istart 6353e08c0960e104cd70b751bacad0e7
round[12].is_row 63cab7040953d051cd60e0e7ba70e18c
round[12].is_box 00102030405060708090a0b0c0d0e0f0
round[12].ik_sch 000102030405060708090a0b0c0d0e0f
round[12].ioutput 00112233445566778899aabbccddeeff
```

AES-256 (Nk=8, Nr=14)

PLAINTEXT: 00112233445566778899aabbccddeeff

KEY: 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

CIPHER (ENCRYPT):

```
round[ 0].input 00112233445566778899aabbccddeeff
round[ 0].k_sch 000102030405060708090a0b0c0d0e0f
round[ 1].start 00102030405060708090a0b0c0d0e0f0
round[ 1].s_box 63cab7040953d051cd60e0e7ba70e18c
round[ 1].s_row 6353e08c0960e104cd70b751bacad0e7
round[ 1].m_col 5f72641557f5bc92f7be3b291db9f91a
round[ 1].k_sch 101112131415161718191a1b1c1d1e1f
round[ 2].start 4f63760643e0aa85efa7213201a4e705
round[ 2].s_box 84fb386f1aelac97df5cfd237c49946b
round[ 2].s_row 84e1fd6b1a5c946fdf4938977cfbac23
round[ 2].m_col bd2a395d2b6ac438d192443e615da195
round[ 2].k_sch a573c29fa176c498a97fce93a572c09c
round[ 3].start 1859fbc28a1c00a078ed8aad42f6109
round[ 3].s_box adcb0f257e9c63e0bc557e951c15ef01
round[ 3].s_row ad9c7e017e55ef25bc150fe01ccb6395
round[ 3].m_col 810dce0cc9db8172b3678c1e88a1b5bd
```

## Appendix C (continued)

round[ 3].k\_sch 1651a8cd0244bedala5da4c10640bade  
round[ 4].start 975c66c1cb9f3fa8a93a28df8ee10f63  
round[ 4].s\_box 884a33781fdb75c2d380349e19f876fb  
round[ 4].s\_row 88db34fb1f807678d3f833c2194a759e  
round[ 4].m\_col b2822d81abe6fb275faf103a078c0033  
round[ 4].k\_sch ae87dff00ff11b68a68ed5fb03fc1567  
round[ 5].start 1c05f271a417e04ff921c5c104701554  
round[ 5].s\_box 9c6b89a349f0e18499fda678f2515920  
round[ 5].s\_row 9cf0a62049fd59a399518984f26be178  
round[ 5].m\_col aeb65ba974e0f822d73f567bdb64c877  
round[ 5].k\_sch 6de1f1486fa54f9275f8eb5373b8518d  
round[ 6].start c357aae11b45b7b0a2c7bd28a8dc99fa  
round[ 6].s\_box 2e5bacf8af6ea9e73ac67a34c286ee2d  
round[ 6].s\_row 2e6e7a2dafc6eef83a86ace7c25ba934  
round[ 6].m\_col b951c33c02e9bd29ae25cdb1efa08cc7  
round[ 6].k\_sch c656827fc9a799176f294cec6cd5598b  
round[ 7].start 7f074143cb4e243ec10c815d8375d54c  
round[ 7].s\_box d2c5831a1f2f36b278fe0c4cec9d0329  
round[ 7].s\_row d22f0c291ffe031a789d83b2ecc5364c  
round[ 7].m\_col ebb19e1c3ee7c9e87d7535e9ed6b9144  
round[ 7].k\_sch 3de23a75524775e727bf9eb45407cf39  
round[ 8].start d653a4696ca0bc0f5acaab5db96c5e7d  
round[ 8].s\_box f6ed49f950e06576be74624c565058ff  
round[ 8].s\_row f6e062ff507458f9be50497656ed654c  
round[ 8].m\_col 5174c8669da98435a8b3e62ca974a5ea  
round[ 8].k\_sch 0bdc905fc27b0948ad5245a4c1871c2f  
round[ 9].start 5aa858395fd28d7d05e1a38868f3b9c5

## Appendix C (continued)

round[ 9].s\_box bec26a12cfb55dff6bf80ac4450d56a6  
round[ 9].s\_row beb50aa6cff856126b0d6aff45c25dc4  
round[ 9].m\_col 0f77ee31d2ccadc05430a83f4ef96ac3  
round[ 9].k\_sch 45f5a66017b2d387300d4d33640a820a  
round[10].start 4a824851c57e7e47643de50c2af3e8c9  
round[10].s\_box d61352d1a6f3f3a04327d9fee50d9bdd  
round[10].s\_row d6f3d9dda6279bd1430d52a0e513f3fe  
round[10].m\_col bd86f0ea748fc4f4630f11c1e9331233  
round[10].k\_sch 7ccff71cbeb4fe5413e6bbf0d261a7df  
round[11].start c14907f6ca3b3aa070e9aa313b52b5ec  
round[11].s\_box 783bc54274e280e0511eacc7e200d5ce  
round[11].s\_row 78e2acce741ed5425100c5e0e23b80c7  
round[11].m\_col af8690415d6e1dd387e5fbedd5c89013  
round[11].k\_sch f01afafee7a82979d7a5644ab3afe640  
round[12].s\_box cfde0208f4b418ac5309db5c338538ed  
round[12].s\_row cfb4dbedf4093808538502ac33de185c  
round[12].start 5f9c6abfbac634aa50409fa766677653  
round[12].m\_col 7427fae4d8a695269ce83d315be0392b  
round[12].k\_sch 2541fe719bf500258813bbd55a721c0a  
round[13].start 516604954353950314fb86e401922521  
round[13].s\_box d133f22a1aed2a7bfa0f44697c4f3ffd  
round[13].s\_row d1ed44fd1a0f3f2afa4ff27b7c332a69  
round[13].m\_col 2c21a820306f154ab712c75eee0da04f  
round[13].k\_sch 4e5a6699a9f24fe07e572baacdf8cdea  
round[14].start 627bceb9999d5aaac945ecf423f56da5  
round[14].s\_box aa218b56ee5ebeacdd6ecef26e63c06  
round[14].s\_row aa5ece06ee6e3c56dde68bac2621bebf

## Appendix C (continued)

round[14].k\_sch 24fc79ccbf0979e9371ac23c6d68de36  
round[14].output 8ea2b7ca516745bfeafc49904b496089

INVERSE CIPHER (DECRYPT):

round[ 0].iinput 8ea2b7ca516745bfeafc49904b496089  
round[ 0].ik\_sch 24fc79ccbf0979e9371ac23c6d68de36  
round[ 1].istart aa5ece06ee6e3c56dde68bac2621bebf  
round[ 1].is\_row aa218b56ee5ebeacdd6ecef26e63c06  
round[ 1].is\_box 627bceb9999d5aaac945ecf423f56da5  
round[ 1].ik\_sch 4e5a6699a9f24fe07e572baacdf8cdea  
round[ 1].ik\_add 2c21a820306f154ab712c75eee0da04f  
round[ 2].istart d1ed44fd1a0f3f2afa4ff27b7c332a69  
round[ 2].is\_row d133f22a1aed2a7bfa0f44697c4f3ffd  
round[ 2].is\_box 516604954353950314fb86e401922521  
round[ 2].ik\_sch 2541fe719bf500258813bbd55a721c0a  
round[ 2].ik\_add 7427fae4d8a695269ce83d315be0392b  
round[ 3].istart cfb4dbedf4093808538502ac33de185c  
round[ 3].is\_row cfde0208f4b418ac5309db5c338538ed  
round[ 3].is\_box 5f9c6abfbac634aa50409fa766677653  
round[ 3].ik\_sch f01afafee7a82979d7a5644ab3afe640  
round[ 3].ik\_add af8690415d6e1dd387e5fbedd5c89013  
round[ 4].istart 78e2acce741ed5425100c5e0e23b80c7  
round[ 4].is\_row 783bc54274e280e0511eacc7e200d5ce  
round[ 4].is\_box c14907f6ca3b3aa070e9aa313b52b5ec  
round[ 4].ik\_sch 7ccff71cbeb4fe5413e6bbf0d261a7df  
round[ 4].ik\_add bd86f0ea748fc4f4630f11c1e9331233  
round[ 5].istart d6f3d9dda6279bd1430d52a0e513f3fe

## Appendix C (continued)

round[ 5].is\_row d61352d1a6f3f3a04327d9fee50d9bdd  
round[ 5].is\_box 4a824851c57e7e47643de50c2af3e8c9  
round[ 5].ik\_sch 45f5a66017b2d387300d4d33640a820a  
round[ 5].ik\_add 0f77ee31d2ccadc05430a83f4ef96ac3  
round[ 6].istart beb50aa6cff856126b0d6aff45c25dc4  
round[ 6].is\_row bec26a12cfb55dff6bf80ac4450d56a6  
round[ 6].is\_box 5aa858395fd28d7d05e1a38868f3b9c5  
round[ 6].ik\_sch 0bdc905fc27b0948ad5245a4c1871c2f  
round[ 6].ik\_add 5174c8669da98435a8b3e62ca974a5ea  
round[ 7].istart f6e062ff507458f9be50497656ed654c  
round[ 7].is\_row f6ed49f950e06576be74624c565058ff  
round[ 7].is\_box d653a4696ca0bc0f5acaab5db96c5e7d  
round[ 7].ik\_sch 3de23a75524775e727bf9eb45407cf39  
round[ 7].ik\_add ebb19e1c3ee7c9e87d7535e9ed6b9144  
round[ 8].istart d22f0c291ffe031a789d83b2ecc5364c  
round[ 8].is\_row d2c5831a1f2f36b278fe0c4cec9d0329  
round[ 8].is\_box 7f074143cb4e243ec10c815d8375d54c  
round[ 8].ik\_sch c656827fc9a799176f294cec6cd5598b  
round[ 8].ik\_add b951c33c02e9bd29ae25cdb1efa08cc7  
round[ 9].istart 2e6e7a2dafc6eef83a86ace7c25ba934  
round[ 9].is\_row 2e5bacf8af6ea9e73ac67a34c286ee2d  
round[ 9].is\_box c357aae11b45b7b0a2c7bd28a8dc99fa  
round[ 9].ik\_sch 6de1f1486fa54f9275f8eb5373b8518d  
round[ 9].ik\_add aeb65ba974e0f822d73f567bdb64c877  
round[10].istart 9cf0a62049fd59a399518984f26be178  
round[10].is\_row 9c6b89a349f0e18499fda678f2515920  
round[10].is\_box 1c05f271a417e04ff921c5c104701554

## Appendix C (continued)

```
round[10].ik_sch ae87dff00ff11b68a68ed5fb03fc1567
round[10].ik_add b2822d81abe6fb275faf103a078c0033
round[11].istart 88db34fb1f807678d3f833c2194a759e
round[11].is_row 884a33781fdb75c2d380349e19f876fb
round[11].is_box 975c66c1cb9f3fa8a93a28df8ee10f63
round[11].ik_sch 1651a8cd0244bedala5da4c10640bade
round[11].ik_add 810dce0cc9db8172b3678c1e88a1b5bd
round[12].istart ad9c7e017e55ef25bc150fe01ccb6395
round[12].is_row adcb0f257e9c63e0bc557e951c15ef01
round[12].is_box 1859fbc28a1c00a078ed8aad42f6109
round[12].ik_sch a573c29fa176c498a97fce93a572c09c
round[12].ik_add bd2a395d2b6ac438d192443e615da195
round[13].istart 84e1fd6b1a5c946fdf4938977cfbac23
round[13].is_row 84fb386f1aelac97df5cfd237c49946b
round[13].is_box 4f63760643e0aa85efa7213201a4e705
round[13].ik_sch 101112131415161718191a1b1c1d1e1f
round[13].ik_add 5f72641557f5bc92f7be3b291db9f91a
round[14].istart 6353e08c0960e104cd70b751bacad0e7
round[14].is_row 63cab7040953d051cd60e0e7ba70e18c
round[14].is_box 00102030405060708090a0b0c0d0e0f0
round[14].ik_sch 000102030405060708090a0b0c0d0e0f
round[14].ioutput 00112233445566778899aabbccddeeff
```

## Appendix D: VHDL Design Code

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

package rijndael_package is

    subtype SLV_8 is std_logic_vector(7 downto 0);
    subtype STATE_TYPE is std_logic_vector(127 downto 0);
    subtype SLV_128 is std_logic_vector(127 downto 0);
    subtype SLV_32 is std_logic_vector(31 downto 0);
    subtype round_type is integer range 0 to 16;

    function SBOX_LOOKUP (a: SLV_8) return SLV_8;

    function INV_SBOX_LOOKUP (a: SLV_8) return SLV_8;

    function BYTE_SUB_FUNCT (state: STATE_TYPE) return STATE_TYPE;

    function INV_BYTE_SUB_FUNCT (state: STATE_TYPE) return
        STATE_TYPE;

    function SHIFT_ROW_FUNCT (state: STATE_TYPE) return STATE_TYPE;

    function INV_SHIFT_ROW_FUNCT (state: STATE_TYPE) return
        STATE_TYPE;

    function MIX_COLUMN_FUNCT (state: STATE_TYPE) return
        STATE_TYPE;

    function POLY_MULTE_FUNCT (a: SLV_8; b: SLV_8) return SLV_8;

    function POLY_MULTD_FUNCT (a: SLV_8; b: SLV_8) return SLV_8;

    function INV_MIX_COLUMN_FUNCT (state: STATE_TYPE) return
        STATE_TYPE;

    function ADD_ROUNDKEY_FUNCT (roundkey, state: STATE_TYPE) return
        STATE_TYPE;
    function ROUNDKEY_GEN (roundkey: STATE_TYPE; round: round_type)
        return STATE_TYPE;

end package rijndael_package;
```



## Appendix D (continued)

**package body rijndael\_package is**

**function SBOX\_LOOKUP (a: SLV\_8) return SLV\_8 is**

**variable temp: SLV\_8;**

**begin**

**case a is**

**when x"00" => temp := x"63";**  
**when x"01" => temp := x"7c";**  
**when x"02" => temp := x"77";**  
**when x"03" => temp := x"7b";**  
**when x"04" => temp := x"f2";**  
**when x"05" => temp := x"6b";**  
**when x"06" => temp := x"6f";**  
**when x"07" => temp := x"c5";**  
**when x"08" => temp := x"30";**  
**when x"09" => temp := x"01";**  
**when x"0a" => temp := x"67";**  
**when x"0b" => temp := x"2b";**  
**when x"0c" => temp := x"fe";**  
**when x"0d" => temp := x"d7";**  
**when x"0e" => temp := x"ab";**  
**when x"0f" => temp := x"76";**

**when x"10" => temp := x"ca";**  
**when x"11" => temp := x"82";**  
**when x"12" => temp := x"c9";**  
**when x"13" => temp := x"7d";**  
**when x"14" => temp := x"fa";**  
**when x"15" => temp := x"59";**  
**when x"16" => temp := x"47";**  
**when x"17" => temp := x"f0";**  
**when x"18" => temp := x"ad";**  
**when x"19" => temp := x"d4";**  
**when x"1a" => temp := x"a2";**  
**when x"1b" => temp := x"af";**  
**when x"1c" => temp := x"9c";**  
**when x"1d" => temp := x"a4";**  
**when x"1e" => temp := x"72";**  
**when x"1f" => temp := x"c0";**

## Appendix D (continued)

```
when x"20" => temp := x"b7";  
when x"21" => temp := x"fd";  
when x"22" => temp := x"93";  
when x"23" => temp := x"26";  
when x"24" => temp := x"36";  
when x"25" => temp := x"3f";  
when x"26" => temp := x"f7";  
when x"27" => temp := x"cc";  
when x"28" => temp := x"34";  
when x"29" => temp := x"a5";  
when x"2a" => temp := x"e5";  
when x"2b" => temp := x"f1";  
when x"2c" => temp := x"71";  
when x"2d" => temp := x"d8";  
when x"2e" => temp := x"31";  
when x"2f" => temp := x"15";
```

```
when x"30" => temp := x"04";  
when x"31" => temp := x"c7";  
when x"32" => temp := x"23";  
when x"33" => temp := x"c3";  
when x"34" => temp := x"18";  
when x"35" => temp := x"96";  
when x"36" => temp := x"05";  
when x"37" => temp := x"9a";  
when x"38" => temp := x"07";  
when x"39" => temp := x"12";  
when x"3a" => temp := x"80";  
when x"3b" => temp := x"e2";  
when x"3c" => temp := x"eb";  
when x"3d" => temp := x"27";  
when x"3e" => temp := x"b2";  
when x"3f" => temp := x"75";
```

## Appendix D (continued)

```
when x"40" => temp := x"09";  
when x"41" => temp := x"83";  
when x"42" => temp := x"2c";  
when x"43" => temp := x"1a";  
when x"44" => temp := x"1b";  
when x"45" => temp := x"6e";  
when x"46" => temp := x"5a";  
when x"47" => temp := x"a0";  
when x"48" => temp := x"52";  
when x"49" => temp := x"3b";  
when x"4a" => temp := x"d6";  
when x"4b" => temp := x"b3";  
when x"4c" => temp := x"29";  
when x"4d" => temp := x"e3";  
when x"4e" => temp := x"2f";  
when x"4f" => temp := x"84";
```

```
when x"50" => temp := x"53";  
when x"51" => temp := x"d1";  
when x"52" => temp := x"00";  
when x"53" => temp := x"ed";  
when x"54" => temp := x"20";  
when x"55" => temp := x"fc";  
when x"56" => temp := x"b1";  
when x"57" => temp := x"5b";  
when x"58" => temp := x"6a";  
when x"59" => temp := x"cb";  
when x"5a" => temp := x"be";  
when x"5b" => temp := x"39";  
when x"5c" => temp := x"4a";  
when x"5d" => temp := x"4c";  
when x"5e" => temp := x"58";  
when x"5f" => temp := x"cf";
```

## Appendix D (continued)

```
when x"60" => temp := x"d0";  
when x"61" => temp := x"ef";  
when x"62" => temp := x"aa";  
when x"63" => temp := x"fb";  
when x"64" => temp := x"43";  
when x"65" => temp := x"4d";  
when x"66" => temp := x"33";  
when x"67" => temp := x"85";  
when x"68" => temp := x"45";  
when x"69" => temp := x"f9";  
when x"6a" => temp := x"02";  
when x"6b" => temp := x"7f";  
when x"6c" => temp := x"50";  
when x"6d" => temp := x"3c";  
when x"6e" => temp := x"9f";  
when x"6f" => temp := x"a8";
```

```
when x"70" => temp := x"51";  
when x"71" => temp := x"a3";  
when x"72" => temp := x"40";  
when x"73" => temp := x"8f";  
when x"74" => temp := x"92";  
when x"75" => temp := x"9d";  
when x"76" => temp := x"38";  
when x"77" => temp := x"f5";  
when x"78" => temp := x"bc";  
when x"79" => temp := x"b6";  
when x"7a" => temp := x"da";  
when x"7b" => temp := x"21";  
when x"7c" => temp := x"10";  
when x"7d" => temp := x"ff";  
when x"7e" => temp := x"f3";  
when x"7f" => temp := x"d2";
```

## Appendix D (continued)

```
when x"80" => temp := x"cd";  
when x"81" => temp := x"0c";  
when x"82" => temp := x"13";  
when x"83" => temp := x"ec";  
when x"84" => temp := x"5f";  
when x"85" => temp := x"97";  
when x"86" => temp := x"44";  
when x"87" => temp := x"17";  
when x"88" => temp := x"c4";  
when x"89" => temp := x"a7";  
when x"8a" => temp := x"7e";  
when x"8b" => temp := x"3d";  
when x"8c" => temp := x"64";  
when x"8d" => temp := x"5d";  
when x"8e" => temp := x"19";  
when x"8f" => temp := x"73";
```

```
when x"90" => temp := x"60";  
when x"91" => temp := x"81";  
when x"92" => temp := x"4f";  
when x"93" => temp := x"dc";  
when x"94" => temp := x"22";  
when x"95" => temp := x"2a";  
when x"96" => temp := x"90";  
when x"97" => temp := x"88";  
when x"98" => temp := x"46";  
when x"99" => temp := x"ee";  
when x"9a" => temp := x"b8";  
when x"9b" => temp := x"14";  
when x"9c" => temp := x"de";  
when x"9d" => temp := x"5e";  
when x"9e" => temp := x"0b";  
when x"9f" => temp := x"db";
```

## Appendix D (continued)

```
when x"a0" => temp := x"e0";  
when x"a1" => temp := x"32";  
when x"a2" => temp := x"3a";  
when x"a3" => temp := x"0a";  
when x"a4" => temp := x"49";  
when x"a5" => temp := x"06";  
when x"a6" => temp := x"24";  
when x"a7" => temp := x"5c";  
when x"a8" => temp := x"c2";  
when x"a9" => temp := x"d3";  
when x"aa" => temp := x"ac";  
when x"ab" => temp := x"62";  
when x"ac" => temp := x"91";  
when x"ad" => temp := x"95";  
when x"ae" => temp := x"e4";  
when x"af" => temp := x"79";
```

```
when x"b0" => temp := x"e7";  
when x"b1" => temp := x"c8";  
when x"b2" => temp := x"37";  
when x"b3" => temp := x"6d";  
when x"b4" => temp := x"8d";  
when x"b5" => temp := x"d5";  
when x"b6" => temp := x"4e";  
when x"b7" => temp := x"a9";  
when x"b8" => temp := x"6c";  
when x"b9" => temp := x"56";  
when x"ba" => temp := x"f4";  
when x"bb" => temp := x"ea";  
when x"bc" => temp := x"65";  
when x"bd" => temp := x"7a";  
when x"be" => temp := x"ae";  
when x"bf" => temp := x"08";
```

## Appendix D (continued)

```
when x"c0" => temp := x"ba";  
when x"c1" => temp := x"78";  
when x"c2" => temp := x"25";  
when x"c3" => temp := x"2e";  
when x"c4" => temp := x"1c";  
when x"c5" => temp := x"a6";  
when x"c6" => temp := x"b4";  
when x"c7" => temp := x"c6";  
when x"c8" => temp := x"e8";  
when x"c9" => temp := x"dd";  
when x"ca" => temp := x"74";  
when x"cb" => temp := x"1f";  
when x"cc" => temp := x"4b";  
when x"cd" => temp := x"bd";  
when x"ce" => temp := x"8b";  
when x"cf" => temp := x"8a";
```

```
when x"d0" => temp := x"70";  
when x"d1" => temp := x"3e";  
when x"d2" => temp := x"b5";  
when x"d3" => temp := x"66";  
when x"d4" => temp := x"48";  
when x"d5" => temp := x"03";  
when x"d6" => temp := x"f6";  
when x"d7" => temp := x"0e";  
when x"d8" => temp := x"61";  
when x"d9" => temp := x"35";  
when x"da" => temp := x"57";  
when x"db" => temp := x"b9";  
when x"dc" => temp := x"86";  
when x"dd" => temp := x"c1";  
when x"de" => temp := x"1d";  
when x"df" => temp := x"9e";
```

## Appendix D (continued)

```
when x"e0" => temp := x"e1";
when x"e1" => temp := x"f8";
when x"e2" => temp := x"98";
when x"e3" => temp := x"11";
when x"e4" => temp := x"69";
when x"e5" => temp := x"d9";
when x"e6" => temp := x"8e";
when x"e7" => temp := x"94";
when x"e8" => temp := x"9b";
when x"e9" => temp := x"1e";
when x"ea" => temp := x"87";
when x"eb" => temp := x"e9";
when x"ec" => temp := x"ce";
when x"ed" => temp := x"55";
when x"ee" => temp := x"28";
when x"ef" => temp := x"df";
```

```
when x"f0" => temp := x"8c";
when x"f1" => temp := x"a1";
when x"f2" => temp := x"89";
when x"f3" => temp := x"0d";
when x"f4" => temp := x"bf";
when x"f5" => temp := x"e6";
when x"f6" => temp := x"42";
when x"f7" => temp := x"68";
when x"f8" => temp := x"41";
when x"f9" => temp := x"99";
when x"fa" => temp := x"2d";
when x"fb" => temp := x"0f";
when x"fc" => temp := x"b0";
when x"fd" => temp := x"54";
when x"fe" => temp := x"bb";
when x"ff" => temp := x"16";
```

```
when others => null;
```

```
end case;
```

```
return temp;
```

```
end function SBOX_LOOKUP;
```



## Appendix D (continued)

```
function INV_SBOX_LOOKUP (a: SLV_8) return SLV_8 is
```

```
    variable temp: SLV_8;
```

```
    begin
```

```
        case a is
```

```
            when x"00" => temp := x"52";  
            when x"01" => temp := x"09";  
            when x"02" => temp := x"6a";  
            when x"03" => temp := x"d5";  
            when x"04" => temp := x"30";  
            when x"05" => temp := x"36";  
            when x"06" => temp := x"a5";  
            when x"07" => temp := x"38";  
            when x"08" => temp := x"bf";  
            when x"09" => temp := x"40";  
            when x"0a" => temp := x"a3";  
            when x"0b" => temp := x"9e";  
            when x"0c" => temp := x"81";  
            when x"0d" => temp := x"f3";  
            when x"0e" => temp := x"d7";  
            when x"0f" => temp := x"fb";
```

```
            when x"10" => temp := x"7c";  
            when x"11" => temp := x"e3";  
            when x"12" => temp := x"39";  
            when x"13" => temp := x"82";  
            when x"14" => temp := x"9b";  
            when x"15" => temp := x"2f";  
            when x"16" => temp := x"ff";  
            when x"17" => temp := x"87";  
            when x"18" => temp := x"34";  
            when x"19" => temp := x"8e";  
            when x"1a" => temp := x"43";  
            when x"1b" => temp := x"44";  
            when x"1c" => temp := x"c4";  
            when x"1d" => temp := x"de";  
            when x"1e" => temp := x"e9";  
            when x"1f" => temp := x"cb";
```

## Appendix D (continued)

```
when x"20" => temp := x"54";  
when x"21" => temp := x"7b";  
when x"22" => temp := x"94";  
when x"23" => temp := x"32";  
when x"24" => temp := x"a6";  
when x"25" => temp := x"c2";  
when x"26" => temp := x"23";  
when x"27" => temp := x"3d";  
when x"28" => temp := x"ee";  
when x"29" => temp := x"4c";  
when x"2a" => temp := x"95";  
when x"2b" => temp := x"0b";  
when x"2c" => temp := x"42";  
when x"2d" => temp := x"fa";  
when x"2e" => temp := x"c3";  
when x"2f" => temp := x"49";
```

```
when x"30" => temp := x"08";  
when x"31" => temp := x"2e";  
when x"32" => temp := x"a1";  
when x"33" => temp := x"66";  
when x"34" => temp := x"28";  
when x"35" => temp := x"d9";  
when x"36" => temp := x"24";  
when x"37" => temp := x"b2";  
when x"38" => temp := x"76";  
when x"39" => temp := x"5b";  
when x"3a" => temp := x"a2";  
when x"3b" => temp := x"49";  
when x"3c" => temp := x"6d";  
when x"3d" => temp := x"8b";  
when x"3e" => temp := x"d1";
```

## Appendix D (continued)

```
when x"40" => temp := x"72";  
when x"41" => temp := x"f8";  
when x"42" => temp := x"f6";  
when x"43" => temp := x"64";  
when x"44" => temp := x"86";  
when x"45" => temp := x"68";  
when x"46" => temp := x"98";  
when x"47" => temp := x"16";  
when x"48" => temp := x"d4";  
when x"49" => temp := x"a4";  
when x"4a" => temp := x"5c";  
when x"4b" => temp := x"cc";  
when x"4c" => temp := x"5d";  
when x"4d" => temp := x"65";  
when x"4e" => temp := x"b6";  
when x"4f" => temp := x"92";
```

```
when x"50" => temp := x"6c";  
when x"51" => temp := x"70";  
when x"52" => temp := x"48";  
when x"53" => temp := x"50";  
when x"54" => temp := x"fd";  
when x"55" => temp := x"ed";  
when x"56" => temp := x"b9";  
when x"57" => temp := x"da";  
when x"58" => temp := x"5e";  
when x"59" => temp := x"15";  
when x"5a" => temp := x"46";  
when x"5b" => temp := x"57";  
when x"5c" => temp := x"a7";  
when x"5d" => temp := x"8d";  
when x"5e" => temp := x"9d";  
when x"5f" => temp := x"84";
```

## Appendix D (continued)

```
when x"60" => temp := x"90";  
when x"61" => temp := x"d8";  
when x"62" => temp := x"ab";  
when x"63" => temp := x"00";  
when x"64" => temp := x"8c";  
when x"65" => temp := x"bc";  
when x"66" => temp := x"d3";  
when x"67" => temp := x"0a";  
when x"68" => temp := x"f7";  
when x"69" => temp := x"e4";  
when x"6a" => temp := x"58";  
when x"6b" => temp := x"05";  
when x"6c" => temp := x"b8";  
when x"6d" => temp := x"b3";  
when x"6e" => temp := x"45";  
when x"6f" => temp := x"06";
```

```
when x"70" => temp := x"d0";  
when x"71" => temp := x"2c";  
when x"72" => temp := x"1e";  
when x"73" => temp := x"8f";  
when x"74" => temp := x"ca";  
when x"75" => temp := x"3f";  
when x"76" => temp := x"0f";  
when x"77" => temp := x"02";  
when x"78" => temp := x"c1";  
when x"79" => temp := x"af";  
when x"7a" => temp := x"bd";  
when x"7b" => temp := x"03";  
when x"7c" => temp := x"01";  
when x"7d" => temp := x"13";  
when x"7e" => temp := x"8a";  
when x"7f" => temp := x"6b";
```

## Appendix D (continued)

```
when x"80" => temp := x"3a";  
when x"81" => temp := x"91";  
when x"82" => temp := x"11";  
when x"83" => temp := x"41";  
when x"84" => temp := x"4f";  
when x"85" => temp := x"67";  
when x"86" => temp := x"dc";  
when x"87" => temp := x"ea";  
when x"88" => temp := x"97";  
when x"89" => temp := x"f2";  
when x"8a" => temp := x"cf";  
when x"8b" => temp := x"ce";  
when x"8c" => temp := x"f0";  
when x"8d" => temp := x"b4";  
when x"8e" => temp := x"e6";  
when x"8f" => temp := x"73";
```

```
when x"90" => temp := x"96";  
when x"91" => temp := x"ac";  
when x"92" => temp := x"74";  
when x"93" => temp := x"22";  
when x"94" => temp := x"e7";  
when x"95" => temp := x"ad";  
when x"96" => temp := x"35";  
when x"97" => temp := x"85";  
when x"98" => temp := x"e2";  
when x"99" => temp := x"f9";  
when x"9a" => temp := x"37";  
when x"9b" => temp := x"e8";  
when x"9c" => temp := x"1c";  
when x"9d" => temp := x"75";  
when x"9e" => temp := x"df";  
when x"9f" => temp := x"6e";
```

## Appendix D (continued)

```
when x"a0" => temp := x"47";  
when x"a1" => temp := x"f1";  
when x"a2" => temp := x"1a";  
when x"a3" => temp := x"71";  
when x"a4" => temp := x"1d";  
when x"a5" => temp := x"29";  
when x"a6" => temp := x"c5";  
when x"a7" => temp := x"89";  
when x"a8" => temp := x"6f";  
when x"a9" => temp := x"b7";  
when x"aa" => temp := x"62";  
when x"ab" => temp := x"0e";  
when x"ac" => temp := x"aa";  
when x"ad" => temp := x"18";  
when x"ae" => temp := x"be";  
when x"af" => temp := x"1b";
```

```
when x"b0" => temp := x"fc";  
when x"b1" => temp := x"56";  
when x"b2" => temp := x"3e";  
when x"b3" => temp := x"4b";  
when x"b4" => temp := x"c6";  
when x"b5" => temp := x"d2";  
when x"b6" => temp := x"79";  
when x"b7" => temp := x"20";  
when x"b8" => temp := x"9a";  
when x"b9" => temp := x"db";  
when x"ba" => temp := x"c0";  
when x"bb" => temp := x"fe";  
when x"bc" => temp := x"78";  
when x"bd" => temp := x"cd";  
when x"be" => temp := x"5a";  
when x"bf" => temp := x"f4";
```

## Appendix D (continued)

```
when x"c0" => temp := x"1f";  
when x"c1" => temp := x"dd";  
when x"c2" => temp := x"a8";  
when x"c3" => temp := x"33";  
when x"c4" => temp := x"88";  
when x"c5" => temp := x"07";  
when x"c6" => temp := x"c7";  
when x"c7" => temp := x"31";  
when x"c8" => temp := x"b1";  
when x"c9" => temp := x"12";  
when x"ca" => temp := x"10";  
when x"cb" => temp := x"59";  
when x"cc" => temp := x"27";  
when x"cd" => temp := x"80";  
when x"ce" => temp := x"ec";  
when x"cf" => temp := x"5f";
```

```
when x"d0" => temp := x"60";  
when x"d1" => temp := x"51";  
when x"d2" => temp := x"7f";  
when x"d3" => temp := x"a9";  
when x"d4" => temp := x"19";  
when x"d5" => temp := x"b5";  
when x"d6" => temp := x"4a";  
when x"d7" => temp := x"0d";  
when x"d8" => temp := x"2d";  
when x"d9" => temp := x"e5";  
when x"da" => temp := x"7a";  
when x"db" => temp := x"9f";  
when x"dc" => temp := x"93";  
when x"dd" => temp := x"c9";  
when x"de" => temp := x"9c";  
when x"df" => temp := x"ef";
```

## Appendix D (continued)

```
when x"e0" => temp := x"a0";  
when x"e1" => temp := x"e0";  
when x"e2" => temp := x"3b";  
when x"e3" => temp := x"4d";  
when x"e4" => temp := x"ae";  
when x"e5" => temp := x"2a";  
when x"e6" => temp := x"f5";  
when x"e7" => temp := x"b0";  
when x"e8" => temp := x"c8";  
when x"e9" => temp := x"eb";  
when x"ea" => temp := x"bb";  
when x"eb" => temp := x"3c";  
when x"ec" => temp := x"83";  
when x"ed" => temp := x"53";  
when x"ee" => temp := x"99";  
when x"ef" => temp := x"61";
```

```
when x"f0" => temp := x"17";  
when x"f1" => temp := x"2b";  
when x"f2" => temp := x"04";  
when x"f3" => temp := x"7e";  
when x"f4" => temp := x"ba";  
when x"f5" => temp := x"77";  
when x"f6" => temp := x"d6";  
when x"f7" => temp := x"26";  
when x"f8" => temp := x"e1";  
when x"f9" => temp := x"69";  
when x"fa" => temp := x"14";  
when x"fb" => temp := x"63";  
when x"fc" => temp := x"55";  
when x"fd" => temp := x"21";  
when x"fe" => temp := x"0c";  
when x"ff" => temp := x"7d";
```

```
when others => null;
```

```
end case;
```

```
return temp;
```

```
end function INV_SBOX_LOOKUP;
```



## Appendix D (continued)

```
function BYTE_SUB_FUNCT (state: STATE_TYPE) return STATE_TYPE is

    variable b: STATE_TYPE;
    variable temp: STATE_TYPE;

begin

    b(127 downto 120) := SBOX_LOOKUP(state(127 downto 120));
    b(119 downto 112) := SBOX_LOOKUP(state(119 downto 112));
    b(111 downto 104) := SBOX_LOOKUP(state(111 downto 104));
    b(103 downto 96) := SBOX_LOOKUP(state(103 downto 96));
    b(95 downto 88) := SBOX_LOOKUP(state(95 downto 88));
    b(87 downto 80) := SBOX_LOOKUP(state(87 downto 80));
    b(79 downto 72) := SBOX_LOOKUP(state(79 downto 72));
    b(71 downto 64) := SBOX_LOOKUP(state(71 downto 64));
    b(63 downto 56) := SBOX_LOOKUP(state(63 downto 56));
    b(55 downto 48) := SBOX_LOOKUP(state(55 downto 48));
    b(47 downto 40) := SBOX_LOOKUP(state(47 downto 40));
    b(39 downto 32) := SBOX_LOOKUP(state(39 downto 32));
    b(31 downto 24) := SBOX_LOOKUP(state(31 downto 24));
    b(23 downto 16) := SBOX_LOOKUP(state(23 downto 16));
    b(15 downto 8) := SBOX_LOOKUP(state(15 downto 8));
    b(7 downto 0) := SBOX_LOOKUP(state(7 downto 0));

    temp:=b;

    return temp;

end function BYTE_SUB_FUNCT;
```

## Appendix D (continued)

```
function INV_BYTE_SUB_FUNCT (state: STATE_TYPE) return
STATE_TYPE is

    variable b: STATE_TYPE;
    variable temp: STATE_TYPE;

begin
    b(127 downto 120) := INV_SBOX_LOOKUP(state(127 downto 120));
    b(119 downto 112) := INV_SBOX_LOOKUP(state(119 downto 112));
    b(111 downto 104) := INV_SBOX_LOOKUP(state(111 downto 104));
    b(103 downto 96) := INV_SBOX_LOOKUP(state(103 downto 96));
    b(95 downto 88) := INV_SBOX_LOOKUP(state(95 downto 88));
    b(87 downto 80) := INV_SBOX_LOOKUP(state(87 downto 80));
    b(79 downto 72) := INV_SBOX_LOOKUP state(79 downto 72));
    b(71 downto 64) := INV_SBOX_LOOKUP(state(71 downto 64));
    b(63 downto 56) := INV_SBOX_LOOKUP(state(63 downto 56));
    b(55 downto 48) := INV_SBOX_LOOKUP(state(55 downto 48));
    b(47 downto 40) := INV_SBOX_LOOKUP( state(47 downto 40));
    b(39 downto 32) := INV_SBOX_LOOKUP(state(39 downto 32));
    b(31 downto 24) := INV_SBOX_LOOKUP(state(31 downto 24));
    b(23 downto 16) := INV_SBOX_LOOKUP(state(23 downto 16));
    b(15 downto 8) := INV_SBOX_LOOKUP(state(15 downto 8));
    b(7 downto 0) := INV_SBOX_LOOKUP(state(7 downto 0));

    temp:=b;

    return temp;

end function INV_BYTE_SUB_FUNCT;
```

## Appendix D (continued)

```
function SHIFT_ROW_FUNCT (state: STATE_TYPE) return STATE_TYPE is

    variable a: STATE_TYPE;
    variable temp: STATE_TYPE;

begin
    temp(127 downto 120) := state(127 downto 120);      --t0
    temp(119 downto 112) := state(87 downto 80);      --t1
    temp(111 downto 104) := state(47 downto 40);      --t2
    temp(103 downto 96) := state(7 downto 0);          --t3

    temp(95 downto 88) := state(95 downto 88);        --t4
    temp(87 downto 80) := state(55 downto 48);        --t5
    temp(79 downto 72) := state(15 downto 8);         --t6
    temp(71 downto 64) := state(103 downto 96);       --t7

    temp(63 downto 56) := state(63 downto 56);        --t8
    temp(55 downto 48) := state(23 downto 16);        --t9
    temp(47 downto 40) := state(111 downto 104);     --t10
    temp(39 downto 32) := state(71 downto 64);       --t11

    temp(31 downto 24) := state(31 downto 24);        --t12
    temp(23 downto 16) := state(119 downto 112);     --t13
    temp(15 downto 8) := state(79 downto 72);        --t14
    temp(7 downto 0) := state(39 downto 32);         --t15

    a:=temp;

    return a;

end function SHIFT_ROW_FUNCT;
```

## Appendix D (continued)

**function** INV\_SHIFT\_ROW\_FUNCT (state: STATE\_TYPE) **return**  
STATE\_TYPE **is**

**variable** a: STATE\_TYPE;  
**variable** temp: STATE\_TYPE;

**begin**

temp(127 **downto** 120) := state(127 **downto** 120);       --t0  
temp(119 **downto** 112) := state(23 **downto** 16);       --t1  
temp(111 **downto** 104) := state(47 **downto** 40);       --t2  
temp(103 **downto** 96) := state(71 **downto** 64);       --t3

temp(95 **downto** 88) := state(95 **downto** 88);       --t4  
temp(87 **downto** 80) := state(119 **downto** 112);       --t5  
temp(79 **downto** 72) := state(15 **downto** 8);       --t6  
temp(71 **downto** 64) := state(39 **downto** 32);       --t7

temp(63 **downto** 56) := state(63 **downto** 56);       --t8  
temp(55 **downto** 48) := state(87 **downto** 80);       --t9  
temp(47 **downto** 40) := state(111 **downto** 104);       --t10  
temp(39 **downto** 32) := state(7 **downto** 0);       --t11

temp(31 **downto** 24) := state(31 **downto** 24);       --t12  
temp(23 **downto** 16) := state(55 **downto** 48);       --t13  
temp(15 **downto** 8) := state(79 **downto** 72);       --t14  
temp(7 **downto** 0) := state(103 **downto** 96);       --t15

a := temp;

**return** a;

**end function** INV\_SHIFT\_ROW\_FUNCT;

## Appendix D (continued)

**function** MIX\_COLUMN\_FUNCT (state: STATE\_TYPE) **return**  
STATE\_TYPE **is**

**variable** t0: SLV\_8;  
**variable** t1: SLV\_8;  
**variable** t2: SLV\_8;  
**variable** t3: SLV\_8;  
**variable** t4: SLV\_8;  
**variable** t5: SLV\_8;  
**variable** t6: SLV\_8;  
**variable** t7: SLV\_8;  
**variable** t8: SLV\_8;  
**variable** t9: SLV\_8;  
**variable** t10: SLV\_8;  
**variable** t11: SLV\_8;  
**variable** t12: SLV\_8;  
**variable** t13: SLV\_8;  
**variable** t14: SLV\_8;  
**variable** t15: SLV\_8;  
**variable** DATAOUT: SLV\_128;  
**variable** temp: SLV\_128;

**begin**

t0 := state(127 **downto** 120);  
t1 := state(119 **downto** 112);  
t2 := state(111 **downto** 104);  
t3 := state(103 **downto** 96);

t4 := state(95 **downto** 88);  
t5 := state(87 **downto** 80);  
t6 := state(79 **downto** 72);  
t7 := state(71 **downto** 64);

t8 := state(63 **downto** 56);  
t9 := state(55 **downto** 48);  
t10 := state(47 **downto** 40);  
t11 := state(39 **downto** 32);

t12 := state(31 **downto** 24);  
t13 := state(23 **downto** 16);  
t14 := state(15 **downto** 8);  
t15 := state(7 **downto** 0);

## Appendix D (continued)

DATAOUT(127 **downto** 120) := POLY\_MULTE\_FUNCT("00000010", t0) **xor**  
POLY\_MULTE\_FUNCT("00000011", t1) **xor**  
t2 **xor** t3;

DATAOUT(119 **downto** 112) := t0 **xor** POLY\_MULTE\_FUNCT("00000010", t1)  
**xor** POLY\_MULTE\_FUNCT("00000011", t2)  
**xor** t3;

DATAOUT(111 **downto** 104) := POLY\_MULTE\_FUNCT("00000010", t2) **xor**  
POLY\_MULTE\_FUNCT("00000011", t3) **xor**  
t0 **xor** t1;

DATAOUT(103 **downto** 96) := POLY\_MULTE\_FUNCT("00000011", t0) **xor**  
POLY\_MULTE\_FUNCT("00000010", t3) **xor**  
t1 **xor** t2;

DATAOUT(95 **downto** 88) := POLY\_MULTE\_FUNCT("00000010", t4) **xor**  
POLY\_MULTE\_FUNCT("00000011", t5) **xor**  
t6 **xor** t7;

DATAOUT(87 **downto** 80) := POLY\_MULTE\_FUNCT("00000010", t5) **xor**  
POLY\_MULTE\_FUNCT("00000011", t6) **xor**  
t4 **xor** t7;

DATAOUT(79 **downto** 72) := POLY\_MULTE\_FUNCT("00000010", t6) **xor**  
POLY\_MULTE\_FUNCT("00000011", t7) **xor**  
t4 **xor** t5;

DATAOUT(71 **downto** 64) := POLY\_MULTE\_FUNCT("00000011", t4) **xor**  
POLY\_MULTE\_FUNCT("00000010", t7) **xor**  
t5 **xor** t6;

DATAOUT(63 **downto** 56) := POLY\_MULTE\_FUNCT("00000010", t8) **xor**  
POLY\_MULTE\_FUNCT("00000011", t9) **xor**  
t10 **xor** t11;

DATAOUT(55 **downto** 48) := POLY\_MULTE\_FUNCT("00000010", t9) **xor**  
POLY\_MULTE\_FUNCT("00000011", t10) **xor**  
t8 **xor** t11;

## Appendix D (continued)

```
DATAOUT(47 downto 40) := POLY_MULTE_FUNCT("00000010", t10) xor
                        POLY_MULTE_FUNCT("00000011", t11) xor
                        t8 xor t9;

DATAOUT(39 downto 32) := POLY_MULTE_FUNCT("00000011", t8) xor
                        POLY_MULTE_FUNCT("00000010", t11) xor
                        t9 xor t10;

DATAOUT(31 downto 24) := POLY_MULTE_FUNCT("00000010", t12) xor
                        POLY_MULTE_FUNCT("00000011", t13) xor
                        t14 xor t15;

DATAOUT(23 downto 16) := POLY_MULTE_FUNCT("00000010", t13) xor
                        POLY_MULTE_FUNCT("00000011", t14) xor
                        t12 xor t15;

DATAOUT(15 downto 8) := POLY_MULTE_FUNCT("00000010", t14) xor
                        POLY_MULTE_FUNCT("00000011", t15) xor
                        t12 xor t13;

DATAOUT(7 downto 0) := POLY_MULTE_FUNCT("00000011", t12) xor
                        POLY_MULTE_FUNCT("00000010", t15) xor
                        t13 xor t14;

temp:=DATAOUT;

return temp;

end function MIX_COLUMN_FUNCT;
```

## Appendix D (continued)

**function** INV\_MIX\_COLUMN\_FUNCT (state: STATE\_TYPE) **return**  
STATE\_TYPE is

```
variable t0: SLV_8;  
variable t1: SLV_8;  
variable t2: SLV_8;  
variable t3: SLV_8;  
variable t4: SLV_8;  
variable t5: SLV_8;  
variable t6: SLV_8;  
variable t7: SLV_8;  
variable t8: SLV_8;  
variable t9: SLV_8;  
variable t10: SLV_8;  
variable t11: SLV_8;  
variable t12: SLV_8;  
variable t13: SLV_8;  
variable t14: SLV_8;  
variable t15: SLV_8;  
variable b: STATE_TYPE;  
variable temp: STATE_TYPE;
```

**begin**

```
t0 := state(127 downto 120);  
t1 := state(119 downto 112);  
t2 := state(111 downto 104);  
t3 := state(103 downto 96);  
  
t4 := state(95 downto 88);  
t5 := state(87 downto 80);  
t6 := state(79 downto 72);  
t7 := state(71 downto 64);  
  
t8 := state(63 downto 56);  
t9 := state(55 downto 48);  
t10 := state(47 downto 40);  
t11 := state(39 downto 32);  
  
t12 := state(31 downto 24);  
t13 := state(23 downto 16);  
t14 := state(15 downto 8);  
t15 := state(7 downto 0);
```



## Appendix D (continued)

**b(127 downto 120) :=** POLY\_MULTD\_FUNCT("00001110", t0) **xor**  
POLY\_MULTD\_FUNCT("00001011", t1) **xor**  
POLY\_MULTD\_FUNCT("00001101", t2) **xor**  
POLY\_MULTD\_FUNCT("00001001", t3);

**b(119 downto 112) :=** POLY\_MULTD\_FUNCT("00001001", t0) **xor**  
POLY\_MULTD\_FUNCT("00001110", t1) **xor**  
POLY\_MULTD\_FUNCT("00001011", t2) **xor**  
POLY\_MULTD\_FUNCT("00001101", t3);

**b(111 downto 104) :=** POLY\_MULTD\_FUNCT("00001101", t0) **xor**  
POLY\_MULTD\_FUNCT("00001001", t1) **xor**  
POLY\_MULTD\_FUNCT("00001110", t2) **xor**  
POLY\_MULTD\_FUNCT("00001011", t3);

**b(103 downto 96) :=** POLY\_MULTD\_FUNCT("00001011", t0) **xor**  
POLY\_MULTD\_FUNCT("00001101", t1) **xor**  
POLY\_MULTD\_FUNCT("00001001", t2) **xor**  
POLY\_MULTD\_FUNCT("00001110", t3);

**b(95 downto 88) :=** POLY\_MULTD\_FUNCT("00001110", t4) **xor**  
POLY\_MULTD\_FUNCT("00001011", t5) **xor**  
POLY\_MULTD\_FUNCT("00001101", t6) **xor**  
POLY\_MULTD\_FUNCT("00001001", t7);

**b(87 downto 80) :=** POLY\_MULTD\_FUNCT("00001001", t4) **xor**  
POLY\_MULTD\_FUNCT("00001110", t5) **xor**  
POLY\_MULTD\_FUNCT("00001011", t6) **xor**  
POLY\_MULTD\_FUNCT("00001101", t7);

**b(79 downto 72) :=** POLY\_MULTD\_FUNCT("00001101", t4) **xor**  
POLY\_MULTD\_FUNCT("00001001", t5) **xor**  
POLY\_MULTD\_FUNCT("00001110", t6) **xor**  
POLY\_MULTD\_FUNCT("00001011", t7);

**b(71 downto 64) :=** POLY\_MULTD\_FUNCT("00001011", t4) **xor**  
POLY\_MULTD\_FUNCT("00001101", t5) **xor**  
POLY\_MULTD\_FUNCT("00001001", t6) **xor**  
POLY\_MULTD\_FUNCT("00001110", t7);

## Appendix D (continued)

```
b(63 downto 56) := POLY_MULTD_FUNCT("00001110", t8) xor  
POLY_MULTD_FUNCT("00001011", t9) xor  
POLY_MULTD_FUNCT("00001101", t10) xor  
POLY_MULTD_FUNCT("00001001", t11);
```

```
b(55 downto 48) := POLY_MULTD_FUNCT("00001001", t8) xor  
POLY_MULTD_FUNCT("00001110", t9) xor  
POLY_MULTD_FUNCT("00001011", t10) xor  
POLY_MULTD_FUNCT("00001101", t11);
```

```
b(47 downto 40) := POLY_MULTD_FUNCT("00001101", t8) xor  
POLY_MULTD_FUNCT("00001001", t9) xor  
POLY_MULTD_FUNCT("00001110", t10) xor  
POLY_MULTD_FUNCT("00001011", t11);
```

```
b(39 downto 32) := POLY_MULTD_FUNCT("00001011", t8) xor  
POLY_MULTD_FUNCT("00001101", t9) xor  
POLY_MULTD_FUNCT("00001001", t10) xor  
POLY_MULTD_FUNCT("00001110", t11);
```

```
b(31 downto 24) := POLY_MULTD_FUNCT("00001110", t12) xor  
POLY_MULTD_FUNCT("00001011", t13) xor  
POLY_MULTD_FUNCT("00001101", t14) xor  
POLY_MULTD_FUNCT("00001001", t15);
```

```
b(23 downto 16) := POLY_MULTD_FUNCT("00001001", t12) xor  
POLY_MULTD_FUNCT("00001110", t13) xor  
POLY_MULTD_FUNCT("00001011", t14) xor  
POLY_MULTD_FUNCT("00001101", t15);
```

```
b(15 downto 8) := POLY_MULTD_FUNCT("00001101", t12) xor  
POLY_MULTD_FUNCT("00001001", t13) xor  
POLY_MULTD_FUNCT("00001110", t14) xor  
POLY_MULTD_FUNCT("00001011", t15);
```

```
b(7 downto 0) := POLY_MULTD_FUNCT("00001011", t12) xor  
POLY_MULTD_FUNCT("00001101", t13) xor  
POLY_MULTD_FUNCT("00001001", t14) xor  
POLY_MULTD_FUNCT("00001110", t15);
```

```
temp:=b;  
return temp;
```

```
end function INV_MIX_COLUMN_FUNCT;
```

## Appendix D (continued)

**function** POLY\_MULTE\_FUNCT (a: SLV\_8; b : SLV\_8 ) **return** SLV\_8 **is**

```
variable temp    : SLV_8;
variable temp1   : SLV_8;
variable temp2   : SLV_8;
variable temp3   : SLV_8;
variable and_mask : SLV_8;

begin

and_mask := b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7);

case a(3 downto 0) is

    when "0001" => temp := b;

    when "0010" =>temp := b(6 downto 0) & '0' xor ("00011011")
                    and and_mask);

    when "0011"=> temp := b(6 downto 0) & '0' xor ("00011011")
                    and and_mask) xor b;

    when others => temp := (others => '0');

end case;

return temp;

end function POLY_MULTE_FUNCT;
```

## Appendix D (continued)

**function** POLY\_MULTD\_FUNCT (a: SLV\_8; b: SLV\_8) **return** SLV\_8 **is**

```
variable temp: SLV_8;  
variable temp1: SLV_8;  
variable temp2: SLV_8;  
variable temp3: SLV_8;  
variable and_mask: SLV_8;
```

**begin**

```
and_mask := b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7);
```

**case** a(3 **downto** 0) **is**

```
when "1001"=> temp1 := b(6 downto 0) & '0' xor ("00011011") and  
    and_mask);  
and_mask := temp1(7) & temp1(7) & temp1(7) & temp1(7) &  
    temp1(7) & temp1(7) & temp1(7) & temp1(7);  
temp2 := temp1(6 downto 0) & '0' xor ("00011011") and  
    and_mask);  
and_mask := temp2(7) & temp2(7) & temp2(7) & temp2(7) &  
    temp2(7) & temp2(7) & temp2(7) & temp2(7);  
temp3 := temp2(6 downto 0) & '0' xor ("00011011") and  
    and_mask);  
temp := temp3 xor b;
```

```
when "1011"=> temp1 := b(6 downto 0) & '0' xor ("00011011") and  
    and_mask);  
and_mask := temp1(7) & temp1(7) & temp1(7) & temp1(7) &  
    temp1(7) & temp1(7) & temp1(7) & temp1(7);  
temp2 := temp1(6 downto 0) & '0' xor ("00011011") and  
    and_mask);  
and_mask := temp2(7) & temp2(7) & temp2(7) &  
    temp2(7) & temp2(7) & temp2(7) & temp2(7) &  
    temp2(7);  
temp3 := temp2(6 downto 0) & '0' xor ("00011011")  
    and and_mask);  
temp := temp1 xor temp3 xor b;
```

## Appendix D (continued)

```
    when "1101" => temp1 := b(6 downto 0) & '0' xor ("00011011") and
        and_mask);
        and_mask := temp1(7) & temp1(7) & temp1(7) & temp1(7) &
            temp1(7) & temp1(7) & temp1(7) & temp1(7);
        temp2 := temp1(6 downto 0) & '0' xor ("00011011") and
            and_mask);
        and_mask := temp2(7) & temp2(7) & temp2(7) & temp2(7) &
            temp2(7) & temp2(7) & temp2(7) & temp2(7);
        temp3 := temp2(6 downto 0) & '0' xor ("00011011") and
            and_mask);
        temp := temp2 xor temp3 xor b;

    when "1110"=> temp1 := b(6 downto 0) & '0' xor ("00011011") and
        and_mask);
        and_mask := temp1(7) & temp1(7) & temp1(7) & temp1(7) &
            temp1(7) & temp1(7) & temp1(7) & temp1(7);
        temp2 := temp1(6 downto 0) & '0' xor ("00011011") and
            and_mask);
        and_mask := temp2(7) & temp2(7) & temp2(7) & temp2(7) &
            temp2(7) & temp2(7) & temp2(7) & temp2(7);
        temp3 := temp2(6 downto 0) & '0' xor ("00011011") and
            and_mask);
        temp := temp1 xor temp2 xor temp3;

    when others => temp := (others => '0');

end case;

return temp;

end function POLY_MULTD_FUNCT;
```

## Appendix D (continued)

```
function ADD_ROUNDKEY_FUNCT (roundkey: STATE_TYPE;  
state: STATE_TYPE) return STATE_TYPE is
```

```
    variable b: STATE_TYPE;
```

```
begin
```

```
    b := state xor roundkey;
```

```
    return b;
```

```
end function ADD_ROUNDKEY_FUNCT;
```

```
function ROUNDKEY_GEN (roundkey: STATE_TYPE; round: round_type) return  
STATE_TYPE is
```

```
    variable b: STATE_TYPE;
```

```
    variable b0:SLV_8;
```

```
    variable b1:SLV_8;
```

```
    variable b2:SLV_8;
```

```
    variable b3:SLV_8;
```

```
begin
```

```
    b0 := roundkey(31 downto 24);
```

```
    b1 := roundkey(23 downto 16);
```

```
    b2 := roundkey(15 downto 8);
```

```
    b3 := roundkey(7 downto 0);
```

```
    case round is
```

```
        when 1 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "00000001" xor  
            roundkey(127 downto 120);
```

```
        b(119 downto 112) := SBOX_LOOKUP(b2)xor  
            roundkey(119 downto 112);
```

```
        b(111 downto 104) := SBOX_LOOKUP(b3) xor  
            roundkey(111 downto 104);
```

```
        b(103 downto 96) := SBOX_LOOKUP(b0) xor  
            roundkey(103 downto 96);
```

```
        b(95 downto 64) := b(127 downto 96) xor roundkey(95 downto 64);
```

```
        b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);
```

```
        b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);
```

## Appendix D (continued)

```
when 2 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "00000010" xor  
    roundkey(127 downto 120);  
b(119 downto 112) := SBOX_LOOKUP(b2) xor  
    roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3)  
    xor roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0) xor  
    roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);
```

```
when 3 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "00000100" xor  
    roundkey(127 downto 120);  
b(119 downto 112) := SBOX_LOOKUP(b2) xor  
    roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3) xor  
    roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0) xor  
    roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);
```

```
when 4 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "00001000" xor  
    roundkey(127 downto 120);  
b(119 downto 112) := SBOX_LOOKUP(b2) xor  
    roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3) xor  
    roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0) xor  
    roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);
```

## Appendix D (continued)

```
when 5 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "00010000" xor  
    roundkey(127 downto 120);  
b(119 downto 112) := SBOX_LOOKUP(b2) xor  
    roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3) xor  
    roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0) xor  
    roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);  
  
when 6 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "00100000" xor  
    roundkey(127 downto 120);  
b(119 downto 112) := SBOX_LOOKUP(b2) xor  
    roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3) xor  
    roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0) xor  
    roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);  
  
when 7 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "01000000" xor  
    roundkey(127 downto 120);  
b(119 downto 112) := SBOX_LOOKUP(b2) xor  
    roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3) xor  
    roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0) xor  
    roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);
```



## Appendix D (continued)

```
when 8 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "10000000" xor  
roundkey(127 downto 120) ;  
b(119 downto 112) := SBOX_LOOKUP(b2) xor  
roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3) xor  
roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0)  
xor roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);
```

```
when 9 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "00011011" xor  
roundkey(127 downto 120);  
b(119 downto 112):= SBOX_LOOKUP(b2) xor  
roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3) xor  
roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0) xor  
roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor  
roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor roundkey(31 downto 0);
```

```
when 10 => b(127 downto 120) := SBOX_LOOKUP(b1) xor "00110110" xor  
roundkey(127 downto 120);  
b(119 downto 112) := SBOX_LOOKUP(b2) xor  
roundkey(119 downto 112);  
b(111 downto 104) := SBOX_LOOKUP(b3) xor  
roundkey(111 downto 104);  
b(103 downto 96) := SBOX_LOOKUP(b0) xor  
roundkey(103 downto 96);  
b(95 downto 64) := b(127 downto 96) xor  
roundkey(95 downto 64);  
b(63 downto 32) := b(95 downto 64) xor  
roundkey(63 downto 32);  
b(31 downto 0) := b(63 downto 32) xor  
roundkey(31 downto 0);
```

## Appendix D (continued)

```
        when others => null;

    end case;

    return b

end function ROUNDKEY_GEN;

end package body rijndael_package;
```

## Appendix D (continued)

### S-Box Transformation

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

entity byte_sub is

    port (state: in STD_LOGIC_VECTOR(7 downto 0);
          clk: in std_logic; rst: in std_logic;
          b: out STD_LOGIC_VECTOR(7 downto 0));

end entity byte_sub;

architecture top_aes_RTL of byte_sub is

begin

    process (clk) is

        begin

            if rst = '1' then
                b <= (others => '0');
            elsif (clk='1' and clk'event) then
                b <= SBOX_LOOKUP( state);
            end if;

        end process;

end architecture top_aes_RTL;
```

## Appendix D (continued)

### Shift Row Transformation

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

entity shift_row is

    port (state: in STD_LOGIC_VECTOR(127 downto 0);
          clk: in std_logic; rst: in std_logic;
          DATAOUT: out STD_LOGIC_VECTOR(127 downto 0));

end entity shift_row;

architecture top_aes_RTL of shift_row is

begin

    process (clk) is

        begin

            if rst = '1' then
                DATAOUT <= (others => '0');
            elsif (clk='1' and clk'event) then
                DATAOUT <= SHIFT_ROW_FUNCT(state);
            end if;

        end process;

end architecture top_aes_RTL;
```

## Appendix D (continued)

Mix Column Transformation

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

entity mix_column is

    port (state: in STD_LOGIC_VECTOR(127 downto 0);
          clk: in std_logic; rst: in std_logic;
          DATAOUT: out STD_LOGIC_VECTOR(127 downto 0));

end entity mix_column;

architecture top_aes_RTL of mix_column is

begin

    process (clk) is

        begin
            if rst = '1' then
                DATAOUT <= (others => '0');
            elsif (clk='1' and clk'event) then
                DATAOUT <= MIX_COLUMN_FUNCT(state);
            end if;

        end process;

end architecture top_aes_RTL;
```

## Appendix D (continued)

### Key Generation

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

entity key_gen is

    port (roundkey: in STD_LOGIC_VECTOR(127 downto 0);
          round: in round_type;
          DATAOUT: out STD_LOGIC_VECTOR(127 downto 0));

end entity key_gen;

architecture top_aes_RTL of key_gen is

begin

    process (roundkey, round) is

        begin

            DATAOUT <= ROUNDKEY_GEN(roundkey, round);

        end process;

end architecture top_aes_RTL;
```

## Appendix D (continued)

Inverse S-BOX

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

entity inv_byte_sub is

    port (state: IN STD_LOGIC_VECTOR(7 downto 0);
          clk: in std_logic; rst: in std_logic;
          b: out STD_LOGIC_VECTOR(7 downto 0));

end entity inv_byte_sub;

architecture top_aes_RTL of inv_byte_sub is

begin

    process (clk) is

        begin

            if rst = '1' then
                b <= ( others => '0' );
            elsif (clk='1' and clk'event) then
                b <= INV_SBOX_LOOKUP( state);
            end if;

        end process;

end architecture top_aes_RTL;
```

## Appendix D (continued)

Inverse Shift Row Transformation

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

entity inv_shift_row is

    port (state: in STD_LOGIC_VECTOR(127 downto 0);
          clk: in std_logic; rst: in std_logic;
          DATAOUT: out STD_LOGIC_VECTOR(127 downto 0));

end entity inv_shift_row;

architecture top_aes_RTL of inv_shift_row is

begin

    process (clk) is

        begin

            if rst = '1' then
                DATAOUT <= (others => '0');
            elsif (clk='1' and clk'event) then
                DATAOUT <= INV_SHIFT_ROW_FUNCT(state);
            end if;

        end process;

end architecture top_aes_RTL;
```



## Appendix D (continued)

Inverse Mix Column Transformation

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

entity inv_mix_column is

    port (state: in STD_LOGIC_VECTOR(127 downto 0);
          clk: in std_logic; rst: in std_logic;
          DATAOUT : out STD_LOGIC_VECTOR(127 downto 0));

end entity inv_mix_column;

architecture top_aes_RTL of inv_mix_column is

begin

    process (clk) is

        begin

            if rst = '1' then
                DATAOUT <= ( others => '0' );
            elsif (clk='1' and clk'event) then
                DATAOUT <= INV_MIX_COLUMN_FUNCT(state);
            end if;

        end process;

end architecture top_aes_RTL;
```