A PROJECT REPORT

ON

**DESIGN AND SIMULATION OF ADVANCED ENCRYPTION STANDARD**

Submitted in partial fulfillment of the requirement for

the award of the Degree of

**BACHELOR OF TECHNOLOGY**

In

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

By

P. Gowri Priya (14JN1A04B1)

M. Kavya (14JN1A0484)

N. Usha Rani (14JN1A04A0)

M. Arathi Priya (14JN1A0481)

P. Sireesha (14JN1A04B0)

**Under the Guidance of**

**Mr. L.Bhargav Kumar, M.Tech.**

**Assistant Professor**

Dept. of Electronics & Communication Engineering



**Department of Electronics and Communication Engineering**

**SREE VENKATESWARA COLLEGE OF ENGINEERING::NELLORE**

**(Approved by AICTE, New Delhi and Affiliated to JNTU, ANANTAPURAMU, 515 002)**

**2014 -18**

**SREE VENKATESWARA COLLEGE OF ENGINEERING::NELLORE**

**North Rajupalem, Kodavalur, Nellore**.

(Approved by AICTE, New Delhi and Affiliated to JNTU, ANANTAPURAMU, 515 002)

**CERTIFICATE**

****

This is to certify that the project report entitled “**DESIGN AND SIMULATION OF ADVANCED ENCRYPTION STANDARD”** being submitted by **P.GowriPriya** **(14JN1A04B1), M.Kavya(14JN1A0484), N.UshaRani(14JN1A04A0), M.ArathiPriya (14JN1A0481), P.Sireesha (14JN1A04B0)** in partial fulfillment for the award of the Degree of **BACHELOR OF TECHNOLOGY** in **ELECTRONICS AND COMMUNICATION ENGINEERING** to the Jawaharlal Nehru Technological University Ananthapuramu, is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree.

Signature of Project supervisor Signature of the Head of Department

Mr.L.BHARGAV KUMAR Mr. V.SUDHEER

M.Tech. M.Tech.

ASSISTANT PROFESSOR HEAD OF THE DEPARTMENT

Department of E.C.E Department of E.C.E

Sree Venkateswara College of Sree Venkateswara College of

Engineering, Nellore. Engineering, Nellore.

Signature of External Examiner

**ACKNOWLEDGEMENTS**

We take this opportunity to express our cordial, gratitude and deep sense of indebtness to our project guide Mr. L. BHARGAV KUMAR, Asst.Prof, for his valuable guidance and whole hearted support to us. We feel thankful to him for his innovative ideas, which lead to the successful completion of our project work.

We are thankful to our beloved head of the department of “ELECTRONICS AND COMMUNICATION ENGINEERING” Mr.V.SUDHEER, M.Tech for his timely help, encouragement and interest in this work.

We owe our guidance to our beloved principal Dr. S.V.PADMAJARANI, M.Tech, PH.D., for her timely help, encouragement and interest in this work.

We are thankful to our beloved chairman Dr. P.BABU NAIDU who took keen interest and encouraged us in every effort throughout this course.

In conclusion, we express my sincere thanks to teaching staff, lab assistants, classmates and friends who have helped us either directly or indirectly in carrying out this project work and especially to our parents who helped us thoroughly for shaping out the things well in order.

**PROJECT ASSOCIATES**

P. Gowri Priya (14JN1A04B1)

M. Kavya (14JN1A0484)

N. Usha Rani (14JN1A04A0)

M. Arathi Priya (14JN1A0481)

P.Sireesha (14JN1A04B0)

**DECLARATION**

We hereby declare that the project report entitled “DESIGN AND SIMULATION OF ADVANCED ENCRYPTION STANDARD” completed and written by us has not been previously formed the basis for the awards of any degree certificate.

Place:

Date:

P. Gowri Priya (14JNA04B1)

M. Kavya (14JN1A0484)

N. Usha Rani (14JN1A04A0)

M. Arathi Priya (14JN1A0481)

P.Sireesha (14JN1A04B0)

**ABSTRACT**

In today’s world most of the communication is done using electronic media. Data security plays a vital role in such communication. Hence, there is need to protect data from malicious attacks. This can be achieved by **Cryptography**. The earlier encryption algorithm is Data Encryption Standard (DES) which has several loopholes such as small key size and sensible to brute force attack etc. and it can’t provide high level, efficient and exportable security. These loopholes overcome by a new algorithm called as **Advanced Encryption Standard (AES).**

In this project work, the plain text of 128 bits is given as input to encryption of data is made and the cipher text of 128 bits is throughout as output. The key length of 128 bits, 192 bits, or 256 bits is used in process of encryption. The AES algorithm is a block cipher that uses the same binary key for both encryption and decryption of data blocks. Hence it is called a symmetric key cryptography. The rounds in decryption are exact inverse of encryption. There are four rounds in encryption viz. Sub Bytes, Shift Rows, Mix Columns and Add Round Key. Similarly for Decryption we have Inv Sub Bytes, Inv Shift Rows, Inv Mix Columns and Inv Add Round Key. The number of times operation performed is depend on key length i.e. for 128 bits we have 10 rounds. In this project we are using VHDL or Verilog HDL to implement it.

Since operations in AES are difficulty. There exists no attack better than key exhaustion to read an encrypted message. Ultimately, anyone can use AES encryption methods, and it is free for public or private, commercial or non-commercial use. The simplest version encrypts and decrypts each 128-bit block individually. It gives better security than DES versions and also better throughput.

# LIST OF CONTENTS

CHAPTER-1 INTRODUCTION 1

* 1. Background of the algorithm 1
  2. Notations and conventions 2

1.2.1.Inputs and Outputs 2

1.2.2. Bytes 2

1.2.3. Array of Bytes 3

1.2.4. The State 3

1.2.5. The State as an Array of Columns 4

1.3. Mathematical Background 5

1.3.1. Addition 5

1.3.2. Multiplication 5

1.3.3. Multiplication by x 6

1.3.4. Polynomials with Coefficients in GF(28) 7

CHAPTER-2 LITERATURE SURVEY 9

2.1. Brief history of DES 10

2.2. Disadvantages of DES 11

2.3. Importance of AES 11

CHAPTER-3 VLSI 12

3.1. History and evolution 12

3.2. Integrated Circuit 13

3.3. Moore’s Law 13

3.4. Importance of VLSI 14

3.5. VLSI Design Flow 14

3.6. Advantages of VLSI 15

3.7. Applications of VLSI 15

3.8. FPGA 16

3.8.1 Introduction 16

3.8.2. Architecture of FPGA 16

3.8.3. Advantages of FPGA 17

3.8.4. Applications of FPGA 17

CHAPTER-4 ENCRYPTION 19

4.1. Encryption Process 19

4.2. Bytes Substitution Transformation 20

4.3. Shift Rows Transformation 21

4.4. Mixing of Columns Transformation 22

4.5. Addition of Round Key Transformation 23

4.6. Key Schedule Generation 24

CHAPTER-5 DECRYPTION 25

5.1. Decryption Process 25

5.2. Inverse Bytes Substitution Transformation 26

5.3. Inverse Shift Rows Transformation 27

5.4. Inverse Mixing of Columns Transformation 27

CHAPTER-6 XILINX AND VHDL 29

6.1. Xilinx 29

6.2. Simulation on Xilinx 29

6.2.1. Introduction 29

6.2.2. Getting Started 30

6.2.3. Behavioral Simulation Using ISim 30

6.3. VHDL 39

CHAPTER-7 SIMULATION RESULTS 41

7.1. AES Encryption 41

7.1.1. Byte Substitution 41

7.1.2. Shift Rows 41

7.1.3. Mix Columns 42

7.1.4. Add Round Key 42

7.2. AES Decryption 43

7.2.1. Inverse Byte Substitution 43

7.2.2. Inverse Shift Rows 43

7.2.3. Inverse Mix Column 44

7.2.4. Inverse Add Round Key 44

7.3. Final Round Of Encryption And Decryption Process 45

7.4. Synthesis Report 46

7.5. RTL Schematic 47

**LIST OF FIGURES**

Figure 1.1 Hexa decimal representation of bit patterns 3 Figure 1.3 state array input and output 4 Figure 2.1 secret key expansion 10 Figure 3.1 Moore’s law 14 Figure 3.2 Design flow of VLSI 15 Figure 3.3 Architecture of FPGA 18 Figure 4.1 Encryption process 20 Figure 4.2 Application of S-Box to the each byte of the state 21 Figure 4.3 S-Box values for all 256 combinations in hexadecimal format 22 Figure 4.4 Cyclic shift of the last rows of the state 23 Figure 4.5 Mixing of columns of state 24 Figure 4.6 Exclusive – OR operation of state and cipher keywords 24 Figure 4.7 Key block –Round combinations 25 Figure 5.1 Decryption process 26 Figure 5.2 Application of Inverse S-Box to each byte of state 27 Figure 5.3 Inverse S-Box values for all combinations in Hexadecimal format 27 Figure 5.4 Inverse cyclic shift of last three rows of state 28 Figure 5.5 Inverse mix column operations on state 29

**LIST OF ABBREVIATIONS**

AES - Advanced Encryption Standard

DES - Data Encryption Standard

CBC - Cipher Block Chaining

EDE - Encrypt-Decrypt-Encrypt

FIPS - Federal Information Processing Standard

HTTP - Hyper Text Transfer Protocol

IDEA - International Data Encryption Algorithm

NIST - National Institute Of Standards And Technology

NSA - National Security Agency

OTP - One Time Pad

P-Box - Permutation Box

S-Box - Substitution Box

TDES - Tripple Data Encryption Standard

XML - Extensible Markup Language

XOR - Exclusive OR Operation

**CHAPTER 1**

**INTRODUCTION**

* 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 2nd 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. **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 ≤ i < 128, 0 ≤ i < 192 or 0 ≤ 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, an 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 ≤ n < 16. For a key length of 192 bits, n lies in the range 0 ≤ n < 24. For a key length of 256 bits, n lies in the range 0 ≤ 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 {b7, b6, b5, b4, b3, b2, b1, b0}.These bytes are interpreted as finite field elements using a polynomial representation.

+=…………….(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.1.1.

|  |  |
| --- | --- |
| 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 |

Figure1.1: Hexadecimal Representations of Bit Patterns

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 {b8} to the left of an 8-bit byte. When the b8 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 a0a1a2···a15. The bytes and the bit ordering within bytes are derived from the 128-bit input sequence, input0input1input2···input126input127 as a0 = {input0, input1, ···, input7}, a1 = {input8, input9, ···, input15} with the pattern continuing up to a15 = {input120, input121, ···, input127}. The pattern can be extended to longer sequences associated with 192 and 256 bit keys. In general,an = {input8n, input8n+1, ···, input8n+7}.

**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 Nb numbers of bytes, where Nb is the block length divided by 32. In the State array, which is denoted by the symbol ***S***, each individual byte has two indices. The first byte index is the row number **r**, which lies in the range 0 ≤ r ≤ 3 and the second byte index is the column number **c**, which lies in the range 0 ≤ c ≤ Nb−1. Such indexing allows an individual byte of the State to be referred to as Sr,c or S[r,c]. For the AES Nb = 4, which means that 0 ≤ c ≤ 3. At the beginning of the Encryption and Decryption the input, which is the array of bytes symbolized by in0in1···in15 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 out0out1···out15.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Input and output arrays:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Figure1.2: State Array Input and Output

At the start of the Encryption or Decryption the input array is copied to the State array with where 0 ≤ r ≤3 and 0 ≤ c ≤ Nb−1 copied to the output array with where 0 ≤ r ≤ 3 and 0 ≤ c ≤ Nb−1.S[r, c] = in[r + 4c]. At the end of the Encryption and Decryption the State is out[r + 4c] =S[r,c]

**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 w0...w3. 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,

w0 = S0,0 S1,0 S2,0 S3,0,

w1 = S0,1 S1,1 S2,1 S3,1,

w2 = S0,2 S1,2 S2,2 S3,2,

w3 = S0,3 S1,3 S2,3 S3,3.

**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 ⊕. Such addition is performed modulo-2. In modulo-2 addition,

1 ⊕ 1 = 0,

1 ⊕ 0 = 1,

0 ⊕ 1 = 1

0 ⊕ 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 {a7a6a5a4a3a2a1a0} and {b7b6b5b4b3b2b1b0}, the sum is {c7c6c5c4c3c2c1c0}, where each ci = ai ⊕ bi where i represents corresponding bits. For example, the following expressions are equivalent to one another.

(x6 + x 4 + x 2 + x + 1) + (x7 + *x* + 1) = x 7 + x 6 + x 4 + x 2 (Polynomial notation) {01010111} ⊕ {10000011} = {11010100} (Binary notation)

{57} ⊕ {83} = {*d* 4} (Hexadecimal notation)

**1.3.2. Multiplication:**

In the polynomial representation, multiplication in Galois Field GF (28) (denoted by • ) 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).

………………….(2)

For example, {57}•{83} = {*c*1} because

(*x*6 + *x* 4 + *x* 2 + *x* + 1)(*x*7 + *x* + 1) = *x*13 + *x*11 + *x*9 + *x*8 + *x*7 +*x*7+ *x*5+ *x*3+ *x* 2+ *x* +*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+1Modulo (*x*8+ *x* 4+ *x*3+ *x* +1)=*x*7+ *x*6+1.

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…………………………(3)

Hence, a(x).b(x) mod m(x)=1, which means

………………………..(4)

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

*a*(*x*)•(*b*(*x*)+ *c*(*x*))= *a*(*x*)• *b*(*x*)+ *a*(*x*)• *c*(*x*) ……………………(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 GF (28).

**1.3.3. Multiplication by x:**

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

*b*7 *x*8+ *b*6 *x*7+ *b*5 *x*6+ *b*4 *x*5+ *b*3 *x* 4+ *b*2 *x*3+ *b*1 *x* 2+ *b*0 *x* …………………… (6)

The result x • b(x) is obtained by reducing the above result modulo m(*x*). If b7 equals zero the result is already in reduced form. If b7 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} • {13} = {fe} because

{57} • {02} = xtime ({57}) = {ae}

{57} • {04} = xtime ({ae}) = {47}

{57} • {08}= xtime ({47}) = {8e}

{57} • {10} = xtime ({8e}) = {07},

{57} • {13} = {57} • ({01} • {02} • {10})

={57} • {ae} • {07}

={fe}.

**1.3.4. Polynomials with Coefficients in GF (28):**

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

…………..(7)

Which will be denoted as a word in the form [*a*0 , *a*1 , *a*2 , *a*3 ]. 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 the equation is,

……………..(8)

Define asecond 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),

() +()…………(9)

Multiplication is achieved in two steps. In the first step, the polynomial product *c*(*x*) = *a*(*x*)• *b*(*x*) is algebraically expanded, and like powers are collected to give

c(x)=……………..(10)

Where,

*c*0= *a*0• *b*0

*c*1= *a*1• *b*0⊕ *a*0• *b*1

*c*2= *a*2• *b*0⊕ *a*1• *b*1⊕ *a*0• *b*2

*c*3= *a*3• *b*0⊕ *a*2• *b*1⊕ *a*1• *b*2⊕ *a*0• *b*3

*c*4= *a*3• *b*1⊕ *a*2• *b*2⊕ *a*1• *b*3

*c*5= *a*3• *b*2⊕ *a*2• *b*3

*c*6= *a*3• *b*3

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

*xi* mod(*x* 4+1)= *xi* mod 4

**CHAPTER 2**

**LITERATURE SURVEY**

There are two main types of cryptography in using today - symmetric or secret key cryptography and asymmetric or public key cryptography. Symmetric key cryptography is the oldest type whereas asymmetric cryptography is only being used publicly since the late 1970’s. Asymmetric cryptography was a major milestone in the search for a perfect encryption scheme.

Secret key cryptography goes back to at least Egyptian times and is of concern here. It involves the use of only one key which is used for both encryption and decryption (hence the use of the term symmetric). Figure 2.1 depicts this idea. It is necessary for security purposes that the secret key never be revealed

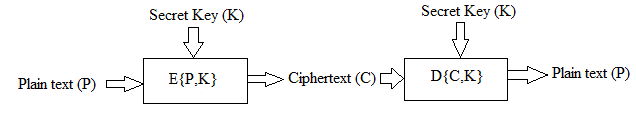


Figure 2.1: Secret key encryption.

To accomplish encryption, most secret key algorithms use two main techniques known as substitution and permutation. Substitution is simply a mapping of one value to another whereas permutation is a reordering of the bit positions for each of the inputs. These techniques are used a number of times in iterations called rounds. Generally, the more rounds there are, the more secure the algorithm. A non-linearity is also introduced into the encryption so that decryption will be computationally infeasible without the secret key. This is achieved with the use of S-boxes which are basically non-linear substitution tables where either the output is smaller than the input or vice versa.

One of the main problems with secret key cryptography is key distribution. For this form of cryptography to work, both parties must have a copy of the secret key. This would have to be communicated over some secure channel which, unfortunately, is not that easy to achieve. As will be seen later, public key cryptography provides a solution to this.

**2.1 Brief history of DES:**

Up until recently, the main standard for encrypting data was a symmetric algorithm known as the Data Encryption Standard (DES). However, this has now been replaced by a new standard known as the Advanced Encryption Standard (AES) which we will look at later. DES is a 64 bit block cipher which means that it encrypts data 64 bits at a time. This is contrasted to a stream cipher in which only one bit at a time (or sometimes small groups of bits such as a byte) is encrypted. DES was the result of a research project set up by International Business Machines (IBM) Corporation in the late 1960’s which resulted in a cipher known as LUCIFER. In the early 1970’s it was decided to commercialise LUCIFER and a number of significant changes were introduced. IBM was not the only one involved in these changes as they sought technical advice from the National Security Agency (NSA) (other outside consultants were involved but it is likely that the NSA were the major contributors from a technical point of view). The altered version of LUCIFER was put forward as a proposal for the new national encryption standard requested by the National Bureau of Standards (NBS). It was finally adopted in 1977 as the Data Encryption Standard - DES (FIPS PUB 46).

Some of the changes made to LUCIFER have been the subject of much controversy even to the present day. The most notable of these was the key size. LUCIFER used a key size of 128 bits however this was reduced to 56 bits for DES. Even though DES actually accepts a 64 bit key as input, the remaining eight bits are used for parity checking and have no effect on DES’s security. Outsiders were convinced that the 56 bit key was an easy target for a brute force attack due to its extremely small size. The need for the parity checking scheme was also questioned without satisfying answers. Another controversial issue was that the S-boxes used were designed under classified conditions and no reasons for their particular design were ever given. This led people to assume that the NSA had introduced a “trapdoor” through which they could decrypt any data encrypted by DES even without knowledge of the key. One startling discovery was that the S-boxes appeared to be secure against an attack known as Differential Cryptanalysis which was only publicly discovered by Bhīma and Shamir in 1990. This suggests that the NSA were aware of this attack in 1977; 13 years earlier. In fact the DES designers claimed that the reason they never made the design specifications for the S-boxes available was that they knew about a number of attacks that weren’t public knowledge at the time and they didn’t want them leaking - this is quite a plausible claim as differential cryptanalysis has shown. However, despite all this controversy, in 1994 NIST reaffirmed DES for government use for a further five years for use in areas other than “classified”.

DES of course isn’t the only symmetric cipher. There are many others, each with varying levels of complexity. Such ciphers include: IDEA, RC4, RC5, RC6 and the new Advanced Encryption Standard (AES). AES is an important algorithm and was originally meant to replace DES (and its more secure variant triple DES) as the standard algorithm for non-classified material. However as of 2003, AES with key sizes of 192 and 256 bits has been found to be secure enough to protect information up to top secret. Since its creation, AES had underdone intense scrutiny as one would expect for an algorithm that is to be used as the standard. To date it has withstood all attacks but the search is still on and it remains to be seen whether or not this will last. We will look at AES later in the course.

**2.2 Disadvantages of DES:**

* There is a weakness in the design of the cipher.
* S box creates same output with two chosen input.
* The initial and final permutation is not clear and seems confusing.

**2.3 Importance of AES:**

To achieve the requirements for secrecy, integrity and non-reproduction of exchanged information, transmission of secret data over the communication channel have emphasized the need for fast and secure digital communication networks. Cryptography is a method which secures and authenticates the transmission of information over unsafe channels. It helps us to store protected information or transmit it across unsafe networks so that any third person can’t read it. For the purpose of security and protection of data AES algorithm was designed.

**CHAPTER3**

**VLSI**

**3.1 History And Evolution:**

Invention of transistor was the driving factor of growth in the VLSI Technology. Before we get to know about the VLSI Technology, let us have a basic knowledge of [Electronics](http://www.techulator.com/articles/Electronics.aspx) evolution. Electronics deals with Circuits which involve various Active and Passive Components. These Circuits are used in various Electronic Devices and are called Electronic Circuits. Originally the components used in Electronic Circuits like diode were made up of vacuum tubes and were called discrete components. Later when the [Solid State Device (SSD)](http://www.techulator.com/articles/SSD.aspx) was invented, the components were made up of semiconductors. Vacuum tubes had the disadvantage of its size, power requirement and reliability.

The development of microelectronics spans a time which is even lesser than the average life expectancy of a human, and yet it has seen as many as four generations. Early 60’s saw the low density fabrication processes classified under Small Scale Integration(SSI) in which transistor count was limited to about 10. This rapidly gave way to Medium Scale Integration in the late 60’s when around 100 transistors could be placed on a single chip.

It was the time when the cost of research began to decline and private firms started entering the competition in contrast to the earlier years where the main burden was borne by the military. Transistor-Transistor logic (TTL) offering higher integration densities outlasted other IC families like ECL and became the basis of the first integrated circuit revolution.

It was the production of this family that gave impetus to semiconductor giants like Texas Instruments, Fairchild and National Semiconductors. Early seventies marked the growth of transistor count to about 1000 per chip called the Large Scale Integration.

By mid-eighties, the transistor count on a single chip had already exceeded 1000 and hence came the age of Very Large Scale Integration or VLSI. Though many improvements have been made and the transistor count is still rising, further names of generations like ULSI are generally avoided. It was during this time when TTL lost the battle to MOS family owing to the same problems that had pushed vacuum tubes into negligence, power dissipation and the limit it imposed on the number of gates that could be placed on a single die.

**3.2.IntegratedCircuit**  
 Integrated Circuit is the circuit in which all the Passive and Active components are fabricated onto a single chip. Initially the Integrated Chip could accommodate only a few components. As the days passed, the devices became more complex and required more number of circuits which made the devices look bulky. Instead of accommodating more circuits in the system, an Integration technologywas developed to increase the number of components that are to be placed on a single chip. This Technology not only helped to reduce the size of the devices but also improved their speed. Depending upon the number of components (Transistors) to be integrated, they were categorized as SSI, MSI, LSI, VLSI, ULSI & GSI.

**3.3. Moore’s Law:**   
 In 1965, Gordon Moore, an industry pioneer predicted that the number of Transistors on a chip doubles every 18 to 24 months. He also predicted that Semiconductor Technology will double its effectiveness every 18 months and many other factors grow exponentially. The graph is depicted in figure1.1

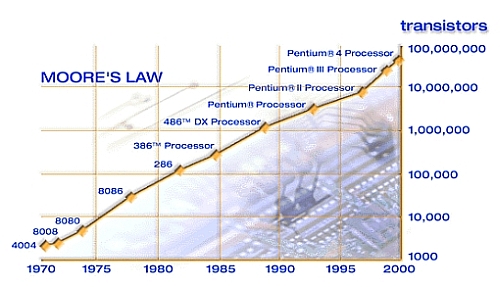


Figure 3.1: Moore’s Law

**3.4. Importance Of VLSI:**

Very-large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. Before the introduction of VLSI technology, most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip.

The electronics industry has achieved a phenomenal growth over the last few decades, mainly due to the rapid advances in large scale integration technologies and system design applications. With the advent of very large scale integration (VLSI) designs, the number of applications of integrated circuits (ICs) in high-performance computing, controls, telecommunications, image and video processing, and consumer electronics has been rising at a very fast pace.

The current cutting-edge technologies such as high resolution and low bit-rate video and cellular communications provide the end-users a marvelous amount of applications, processing power and portability. This trend is expected to grow rapidly, with very important implications on VLSI design and systems design.

## 3.5 VLSI Design Flow:

The VLSI IC circuits design flow is shown in the figure below. The various levels of design are numbered and the blocks show processes in the design flow. Specifications comes first, they describe abstractly, the functionality, interface, and the architecture of the digital IC circuit to be designed. The flowchart of VLSI design flow is depicted in figure 1.2.

Behavioral description is then created to analyze the design in terms of functionality, performance, compliance to given standards, and other specifications.RTL description is done using HDLs. This RTL description is simulated to test functionality. From here onwards we need the help of EDA tools. RTL description is then converted to a gate-level net list using logic synthesis tools. A gate level net list is a description of the circuit in terms of gates and connections between them, which are made in such a way that they meet the timing, power and area specifications. Finally, a physical layout is made, which will be verified and then sent to fabrication.

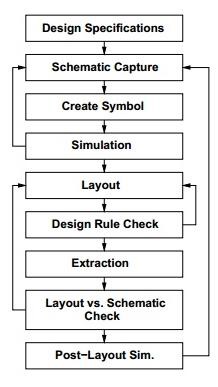


Fig3.2:design flow of vlsi

**3.6. Advantages Of VLSI:**

VLSI has many advantages:

* Reduces the Size of Circuits.
* Reduces the effective cost of the devices.
* Increases the Operating speed of circuits
* Requires less power than Discrete components.
* Higher Reliability
* Occupies a relatively smaller area.

**3.7. Applications Of VLSI:**  
In today's world VLSI chips are widely used in various branches of Engineering like:

* Voice and Data Communication networks
* Digital Signal Processing
* Commercial Electronics
* Medical field

**3.8. FPGA:**

**3.8.1. Introduction:**

Field Programmable Gate Arrays (FPGAs) are a step in the continuum of evolution of Integrated Circuits (IC). FPGAs are reprogrammable silicon chips and are one of the Programmable Logic Devices (PLDs) that can be configured to implement customized hardware functionality of any digital circuit. Due to their flexibility, programmability, capacity for various applications and low end product cycle, FPGAs are highly desirable for implementation of digital circuits.

The main difference between FPGAs and conventional fixed logic implementations, such as Application Specific Integrated Circuits (ASICs), is that the designer can program the FPGA on-site. Using an FPGA instead of a fixed logic implementation eliminates the non-recurring engineering (NRE) costs and significantly reduces time-to-market. FPGA chips adoption across all industries is driven by the fact that FPGAs combine the best parts of ASICs and processor-based systems. These reprogrammable silicon chips also have the same flexibility of software running on a processor-based system, but it is not limited by the number of processing cores available. The software tools provide the programming environment, whereas FPGA circuitry is truly a “hard” implementation of program execution.

**3.8.2. Architecture Of FPGA:**

The FPGA-architecture consists of many logic modules which are placed in an array structure and these modules are configurable at site and are therefore called as Configurable Logic Blocks (CLBs). The channels between the CLBs are used for routing. The arrays of the CLBs are surrounded by programmable I/O modules and connected via programmable interconnects.

CLB performs the logic operation given to the module. The inter connection between CLB and I/O blocks are made with the help of horizontal routing channels, vertical routing channels and PSM. The number of CLB it contains only decides the complexity of FPGA. The functionality of CLB’s and PSM are designed by VHDL or any other hardware descriptive language. After programming, CLB and PSM are placed on chip and connected with each other with routing channels. The Architecture of FPGA is shown in figure 1.3.

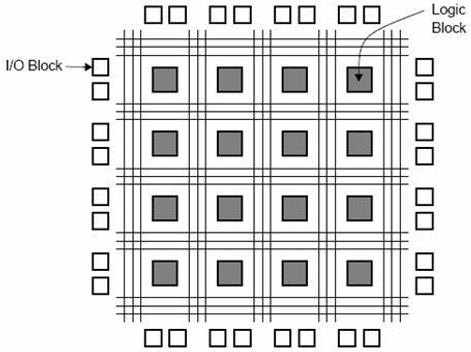
****

Figure 3.3: Architecture of FPGA

**3.8.3 Advantages OF FPGA:**

The advantages of using FPGA based design can thus be summarized with following five Indicators:

* Performance
* Time to Market
* Cost
* Reliability

**3.8.4 Applications Of FPGAs:**

Due their field-programmability, FPGAs are an ideal fit for many different markets. Many industry leaders in the field of FPGAs specially Xilinx provides comprehensive solutions of FPGA devices, advanced solutions, configurable and ready to use Intellectual Property (IP) cores for various applications and markets such as:

* **AEROSPACE AND DEFENCE:**

FPGA vendors provide radiation-tolerant FPGAs along with intellectual property for image processing, waveform generation, and partial reconfiguration for SDRs.

* **AUDIO:**

Xilinx FPGAs and targeted design platforms enable higher degrees of flexibility, faster time-to-market, and lower overall non-recurring engineering costs (NRE) for a wide range of audio, communications, and multimedia applications.

* **AUTOMATIVE SYSTEMS :**

Xilinx and other vendors provide automotive silicon and IP solutions for gateway and driver assistance systems, comfort, convenience, and in-vehicle infotainment.

* **BROADCAST:**

FPGAs adapt to changing requirements faster and lengthen product life cycles with Broadcast Targeted Design Platforms and provide solutions for high end professional broadcast systems.

* **CONSUMER ELECTRONICS:**

Cost-effective solutions enabling next generation, full featured consumer applications, such as converged handsets, digital flat panel displays information appliances, home networking, and residential set top boxes make uses of FPGAs based designs.

* **VIDEO AND IMAGE PROCESSING:**

Xilinx FPGAs and targeted design platforms enable higher degrees of flexibility, faster time-to-market, and lower overall NRE for a wide range of video and imaging applications

* Data Centers designed for high-bandwidth, low-latency servers, networking, and storage applications to bring higher value into cloud deployments also make use of FPGAs based systems.
* High Performance Computing and Data Storage systems uses FPGA based systems for Solutions to Network Attached Storage (NAS), Storage Area Network (SAN), servers, and storage appliances.
* Most Industrial Imaging and Surveillance, Industrial Automation makes use of FPGAs.
* The Virtex FPGA and Spartan FPGA families can be used to meet a range of processing, Display, and I/O interface requirements in Medical Imaging Equipment for diagnostic, monitoring, and therapy applications.
* FPGAS are used in wired and wireless Communication for end-to-end solutions for the Reprogrammable Networking Line Card Packet Processing, Framer/MAC, serial backplanes, and more. RF, base band, connectivity, transport and networking solutions for wireless equipment, addressing standards such as WCDMA, HSDPA, WiMAX and others also use FPGA based system.

**CHAPTER 4**

**ENCRYPTION**

**4.1. Encryption Process:**

The Encryption process of Advanced Encryption Standard algorithm is presented below, in figure 4.1.

Add Round Key (State, Round Key)

ByteSub(State)

ShiftRow(State)

MixColumn(State)

Add Round Key (State, Round Key)

i=i+1

i< Nr

ByteSub (State)

ShiftRow(State)

Add Round Key (State, Round Key)

Key Schedule

Yes

No

W[0]

W[i\*Nb]

W[Nr\*Nb]

Figure 4.1: Encryption Process

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.

**4.2. Bytes Substitution Transformation:**

The bytes substitution transformation Byte sub (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 (28), described in Section

1.3.2The element {00} is mapped to itself.

2. Apply the following affine transformation (over GF (2))

…………(16)

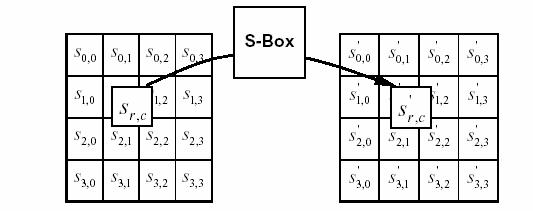
for 0≤ i ≤ 8 , where bi is the ith bit of the byte, and ci is the *i*th bit of a byte *c* with the value {63} or {01100011}. Here and elsewhere, a prime on a variable (e.g., *b*′ ) indicates that the variable is to be updated with the value on the right. In matrix form, the affine transformation element of the S-box can be expressed as

Figure 4.2: Application of S-box to the Each Byte of the State

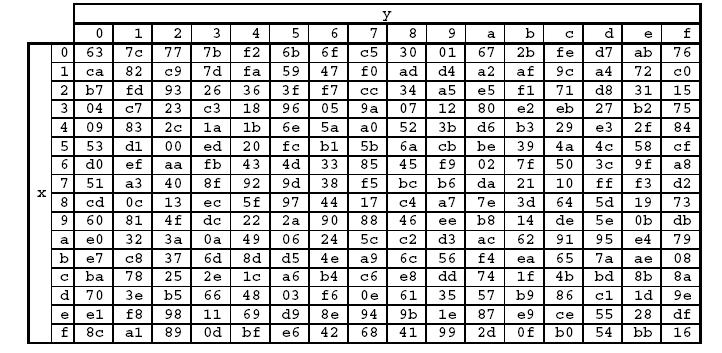
The S-box used in the Sub Bytes transformation is presented in hexadecimal form in figure 7. For example, if =S1,1= {53}, then the substitution value would be determined by the intersection of the row with index ‘5’ and the column with index ‘3’ in figure 7. This would result in S'1, 1 having a value of {ed}.

Figure 4.3: S-box Values for All 256 Combinations in Hexadecimal Format

**4.3. Shift Rows Transformation:**

In the Shift Rows transformation Shift Rows( ), 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 Shift Rows( ) transformation proceeds as follows

*sr*',*c* = *sr*,(*c*+*shift*(*r*,*Nb*)) mod *Nb* for 0< r < 4 and 0≤c≤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 Shift Rows( )transformation.

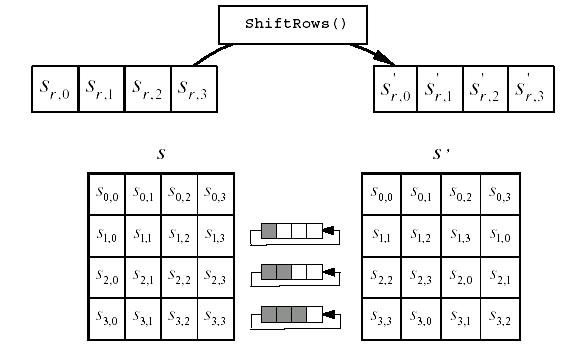


Figure 4.4:.Cyclic Shift of the Last Three Rows of the State

**4.4. 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 Mix Columns( ) 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 (28) 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*)⊗ *S* (*x*)

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

*S*0',*c* =({02}• *S*0,*c* )⊕({03}• *S*1,*c* )⊕ *S*2,*c* ⊕ *S*3,*c*

*S*1',*c* = *S*0,*c* ⊕({02}• *S*1,*c* )⊕({03}• *S*2,*c* )⊕ *S*3,*c*

*S*2',*c* = *S*0,*c* ⊕ *S*1,*c* ⊕({02}• *S*2,*c* )⊕({03}• *S*3,*c* )

*S*3’,*c* =({03}• *S*0,*c* )⊕ *S*1,*c* ⊕ *S*2,*c* ⊕({02}• *S*3,*c* )

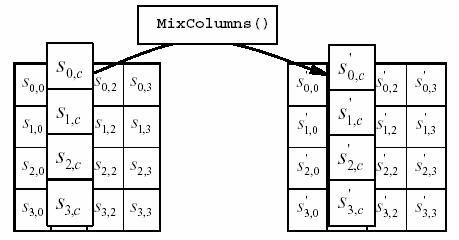


Figure 4.5: Mixing of Columns of the State

**4.5. 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

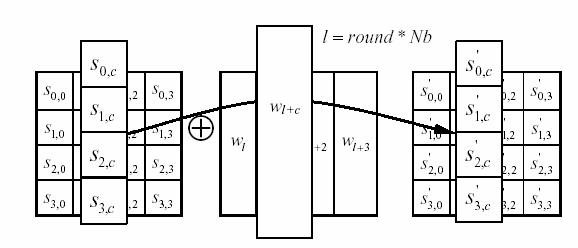
]=[⊕[………….(16)

Figure 4.6: Exclusive-OR Operation of State and Cipher Key Words

where [wi] 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 Add RoundKey ( ) transformation to the Nr rounds of the encryption occurs when 1 ≤ round ≤ *Nr*. The action of this transformation is illustrated in figure10, where l = round \* Nb. The byte address within words of the key schedule was described in Section1.2.1.

**4.6. 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. The number of rounds required for three different key lengths is presented in figure11.

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

Figure 4.7: Key-Block- Round Combinations

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.

**CHAPTER 5**

**DECRYPTION**

**5.1. Decryption Process:**

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

Add Round Key (State,Inv Round Key)

InvByteSub(State)

InvShiftRow(State)

InvMixColumn(State)

Add Round Key (State, InvRound Key)

i=i-1

i>1

InvByteSub (State)

InvShiftRow(State)

Add Round Key (State, InvRound Key)

Key Schedule

Yes

No

W[0]

W[i\*Nb]

W[Nr\*Nb]

Figure 5.1: Decryption Process

sThis 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.

**5.2. Inverse Bytes Substitution Transformation:**

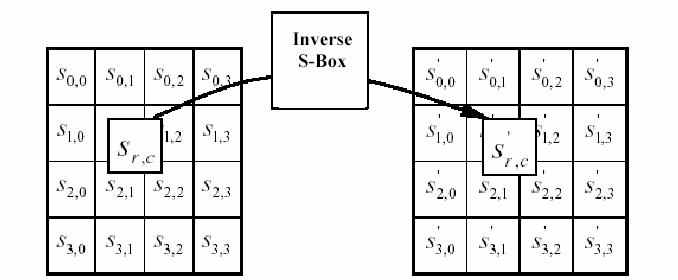
Inverse Byte Substitution Transformation InvSubBytes( ) is the inverse of the byte substitution transformation, in which the inverse S-Box (figure14) is applied to each byte of the State. This is obtained by applying the inverse of the affine transformation to the equation (16) followed by taking the multiplicative inverse in GF (28).

Figure 5.2: Application of the Inverse S-box to Each Byte of the State

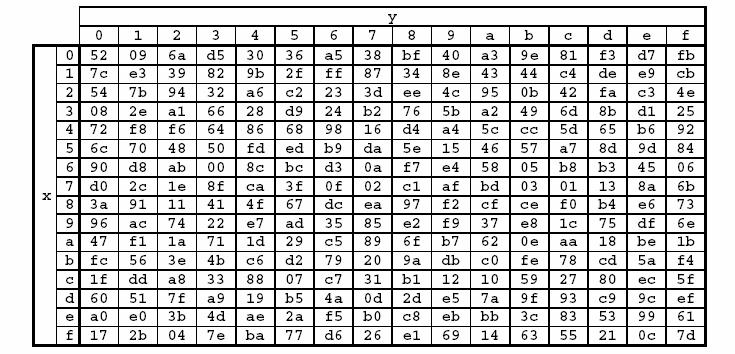
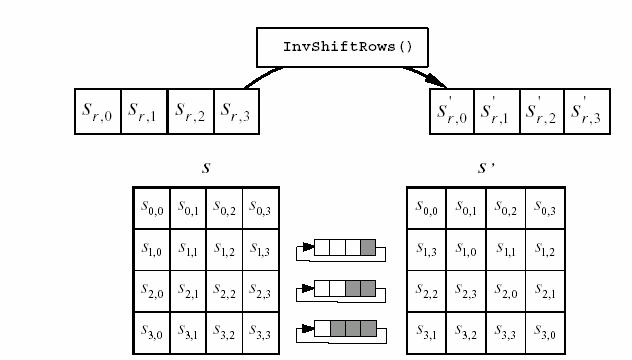


Figure 5.3: Inverse S-box Values for All 256 Combinations in Hexadecimal Format

**5.3. 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.

*Sr*',(*c*+*shift* (*r*,*Nb*)) mod *Nb* = *Sr*,*c* for 0 ≤ r<4 and 0 ≤ c<Nb

.

Figure 5.4: Inverse Cyclic Shift of the Last Three Rows of the State

**5.4. Inverse Mixing of Columns Transformation:**

Inverse Mixing of Columns Transformation InvMixColumns( ) is the inverse of the MixColumns ( ) transformation) presented in chapter2. 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 GF (28) and multiplied modulo*x*4 + 1 with a fixed polynomial *a*-1(*x*), given by

*a* −1(*x*)={0*b*}*x*3+{0*d*}*x*2+{09}*x* +{0*e*}.

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

*S* '(*x*)= *a* −1(*x*)⊗ *S*(*x*)

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

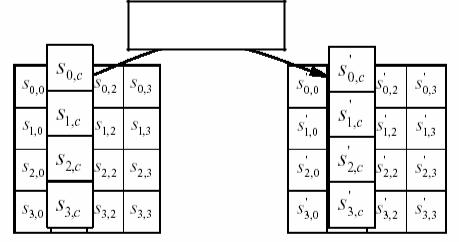
*S*0',*c* =({0*e*}• *S*0,*c* )⊕({0*b*}• *S*1,*c* )⊕({0*d*}• *S*2,*c* )⊕({09}• *S*3,*c* )

*S*1',*c* =({09}• *S*0,*c* )⊕({0*e*}• *S*1,*c* )⊕({0*b*}• *S*2,*c* )⊕({0*d*}• *S*3,*c* )

*S*2',*c* =({0*d*}• *S*0,*c* )⊕({09}• *S*1,*c* )⊕({0*e*}• *S*2,*c* )⊕({0*b*}• *S*3,*c* )

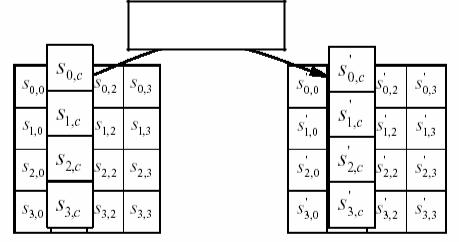
*S*3',*c* =({0*b*}• *S*0,*c* )⊕({0*d*}• *S*1,*c* )⊕({09}• *S*2,*c* )⊕({0*e*}• *S*3,*c* )

Hence state can be represented as,



InvMixColumn( )

Figure 5.5: Inverse Mix Column Operation on State

**CHAPTER 6**

**XILINX AND VHDL**

**6.1. Xilinx:**

Xilinx Inc. is the world's largest supplier of programmable logic devices, the inventor of FIELD PROGRAMMABLE LOGIC ARRAY (FPGA) and the first semi-conductor company with a fabulous manufacturing model. Founded in Silicon Valley in 1984 and head quartered in San Jose, California, USA, the company has corporate offices throughout North America and Europe. The programmable logic device market has been led by Xilinx since the late 1990s. Over the years, Xilinx has fuelled an aggressive expansion to India, Asia and Europe- regions. The company has expanded its product portfolio since its founding. Xilinx sells a broad range of FPGAs, complex programmable logic devices (CPLDs), design tools, intellectual property and reference designs. Xilinx also has a global services and training program.

Xilinx ISE (Integrated Software Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE) to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test bench.

**6.2. Simulation on Xilinx:**

**6.2.1. Introduction:**

The simulator is a tool that allows us to test a circuits using software. Traditionally, a digital circuit had to be built on a breadboard using TTL chips in order to test for logical correctness. This had many drawbacks. It was costly to equip a lab with enough hardware for a class full of students. The cost was augmented further because many chips where hooked up incorrectly and destroyed. Hookup errors often occurred which were hard to detect on a breadboard. Software design solves these problems and allows designs to be re-used and built upon. The objective of this tutorial is to familiarize the student with the Xilinx ISE Design Suite 14.6 ISim Simulator. In this tutorial you will learn the following topics:

1. How to use the diﬀerent capabilities of the simulator.

2. How to create bench test (VHDL) to be used by the simulator.

3. How to verify the functionality of your design (Half Adder) using the Xilinx ISim Simulator.

**6.2.2. Getting Started:**

The following sections outline the requirements for performing behavioral simulation.

Required Files: The behavioral simulation flow requires Design Files, A Test Bench File, and Xilinx Simulation Libraries.

1. Design Files (VHDL or Schematics): This tutorial assumes that you have completed the design entry tutorial in either Schematic-Based Design or VHDL-Based Design. After you have completed one of these, your design includes the required design files and is ready for simulation.

2. Test Bench File: To simulate the design, a bench file is required to provide stimulus to the design. A VHDL test bench file is available with this tutorial. You may also create your own test bench file.

3. Simulation Libraries: Xilinx simulation libraries are required when a Xilinx primitive or IP core is instantiated in the design. Even though we do not use any IP cores or any Xilinx primitives (i.e., digital clock manager) it is useful to know that these libraries are essential for future simulations since you might be using some IP core in your design.

**6.2.3. Behavioral Simulation Using ISim:**

In previous tutorials you learned how to enter your design using schematic capture.In this case a schematic is available in the project. Here are the steps to be taken to simulate your design.

1. Opening your existing design project (Half Adder in our case).

2. Switching mode to Simulation

3. Adding an HDL Test Bench to your existing design.

4. Locating the Simulation Processes.

5. Specifying Simulation Properties.

6. Performing Simulation

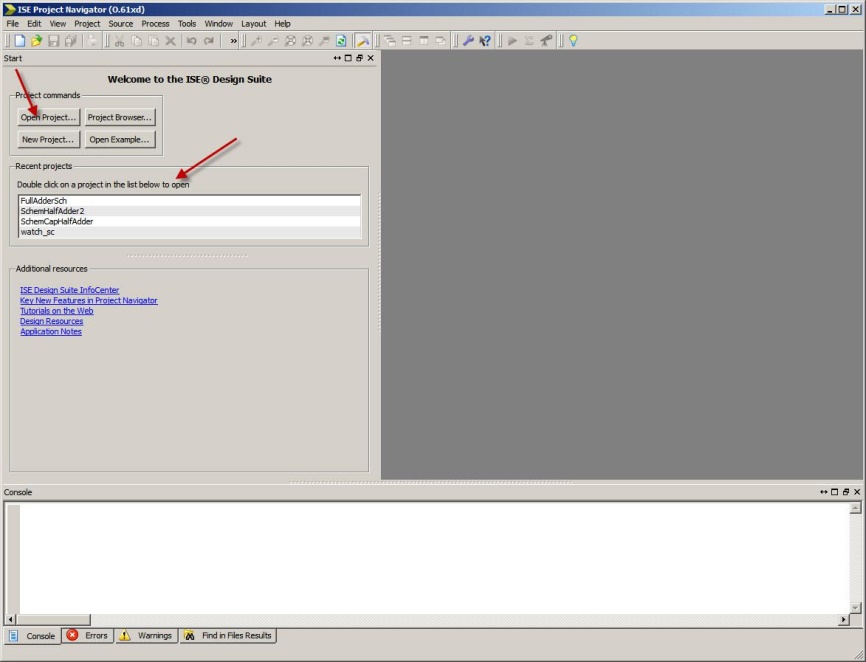
7. Adding Signals

8. Analyzing the Signals

**Step-1:** Opening your Existing Project

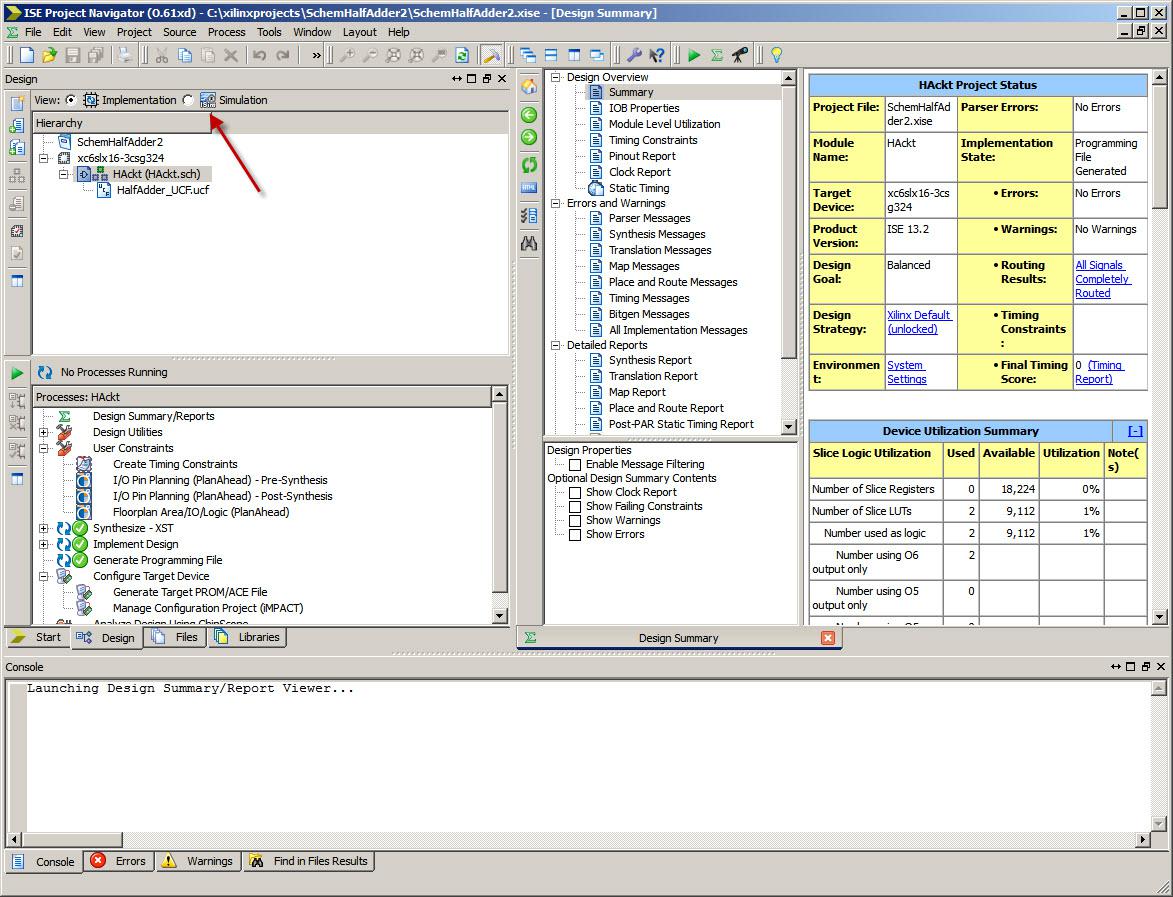
1. Load the Project Navigator from the  → All Programs → Xilinx ISE Design Suite 14.5 → ISE Design Tools → Project Navigator.

2. The Project Navigator window will appear.



3. Click on Open Project or double click on the existing project on the screen.

4. A new screen will appear that shows your previous Half Adder design.



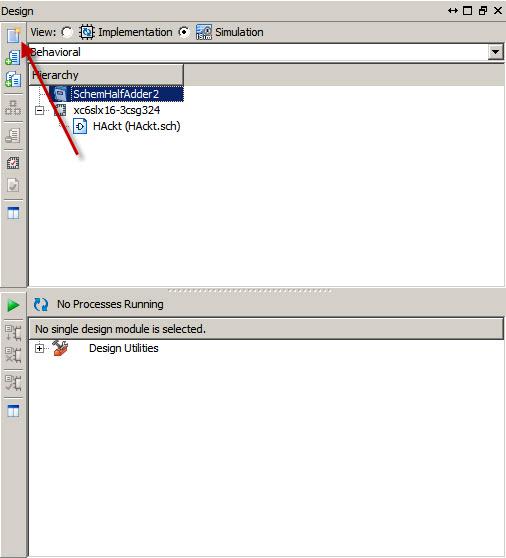
5. Notice in the view you have either Implementation or Simulation highlighted.

6. You should use the Implementation mode when you want to design a circuit using either schematic capture or VHDL. On the other hand you should switch to Simulation mode when you want to simulate your design.

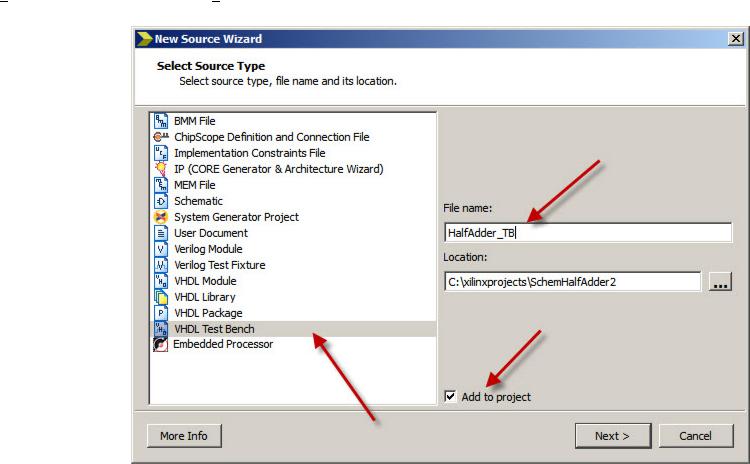
**Step-2: Adding an HDL Test Bench**

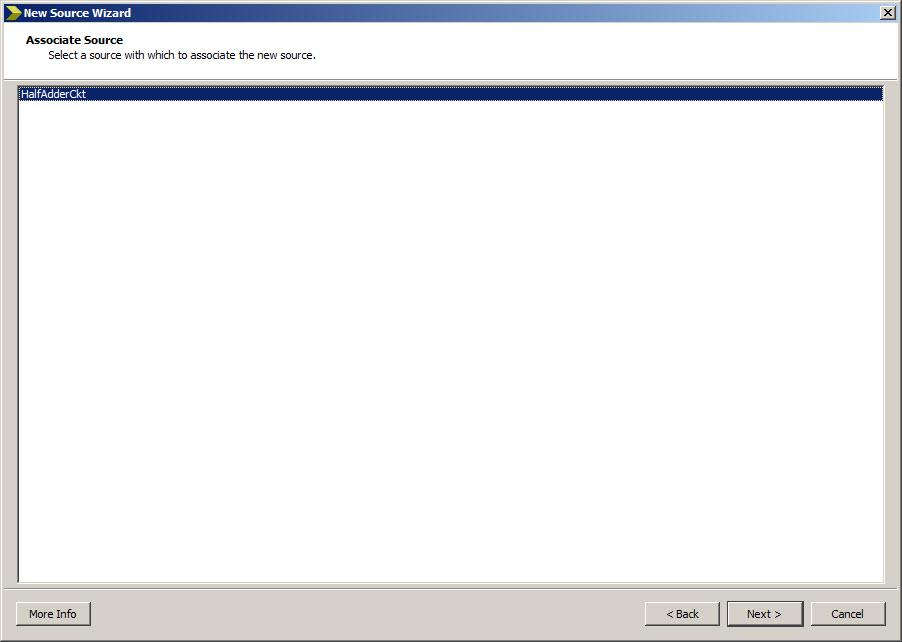
To add an HDL test bench to your design project, you can either add a test bench file provided with this tutorial, or create your own test bench file and add it to your project. This section demonstrates how to create a new test bench file and modify it by editing it with statements from an existing test bench found on the web site. Follow these steps:

1. Highlight your project as seen in the figure below and then click on the  New Source button.

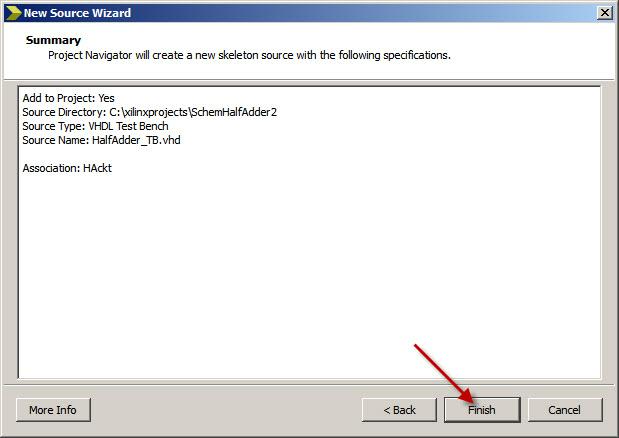


2. In the new dialog box that appears, select VHDL Test Bench from the list of file types and enter the file name (should match your schematic if possible).

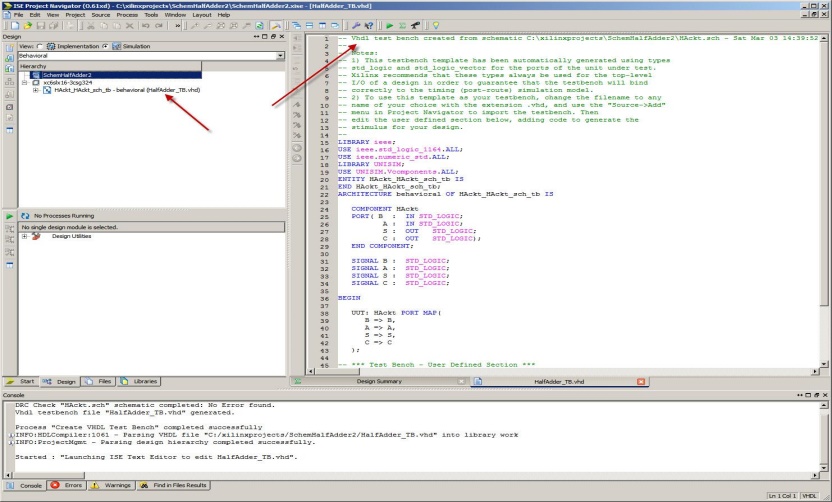


The default location is the current project directory and can be left as is. Ensure the Add to Project box is selected and click the Next button. You will then see a screen with your file name associated to the simulation (as seen in the figure below).

Verify the information in the next few dialog boxes and click Next then Finish.

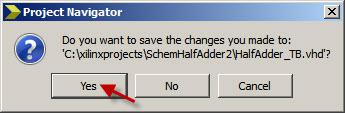


3. A new file will be associated with your project in the Hierarchy pane (HAckt TB.vhd) and a skeleton VHDL file with new information will be shown in the Workspace panel on the right hand side of the screen



4.You will edit the VHDL code by using the “Test Bench used in Behavioral Simulation for Half Adder Design” which is found on the web page of the course (LAB2). A Copy of this test bench is found in Appendix A of this tutorial. It is important to make sure that the name in the COM-PONENT Part of the Architecture matches the name of your Schematic (HAckt.sch or HalfAdderCkt.sch).

5.If you attempt to close the testbench file in the Workspace panel on the right hand side of the screen a new window will pop as seen below. Press Yes to ensure you save the file.

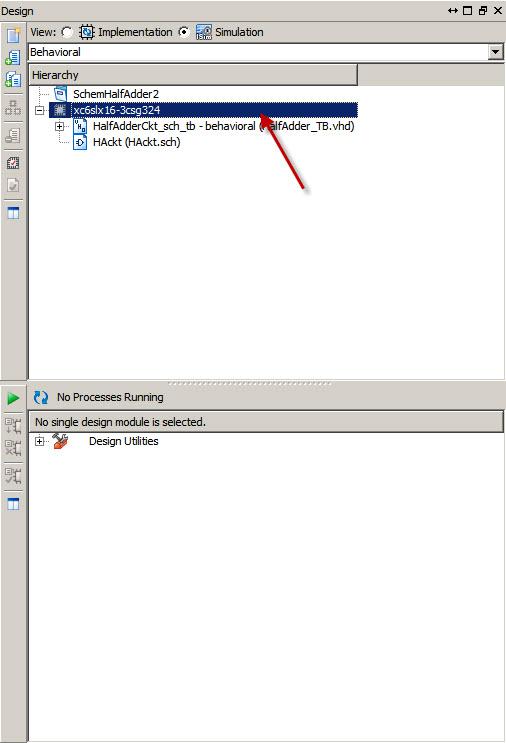


**Step 3: Behavioral Simulation Using ISim**

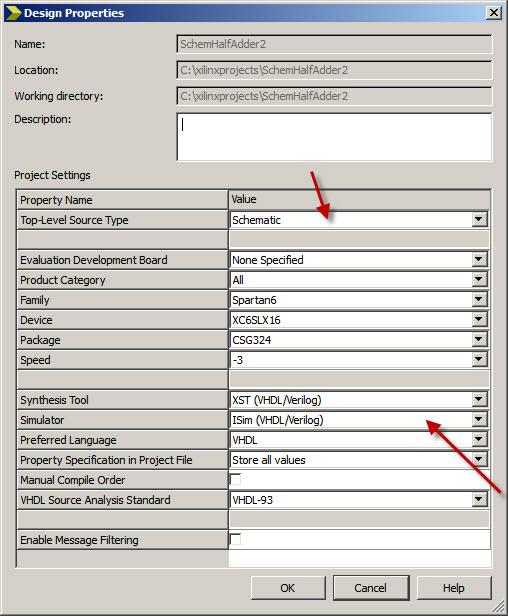
Now that you have a test bench in your project, you can perform behavioral simulation on the design using ISim. The ISE software has full integration with ISim. The ISE software enables ISim to create the work directory, compile the source files, load the design, and perform simulation based on simulation properties. Here are the basic steps that you have to follow:

1. Selecting ISim: To select ISim as your project simulator, do the following:

(a) In the Hierarchy pane of the Project Navigator Design panel, right-click the device line (xc6slx16-3csg324), and select Design Properties.



(b) In the Design Properties dialog box, set the Simulator field to ISim(VHDL/Verilog) and press OK.

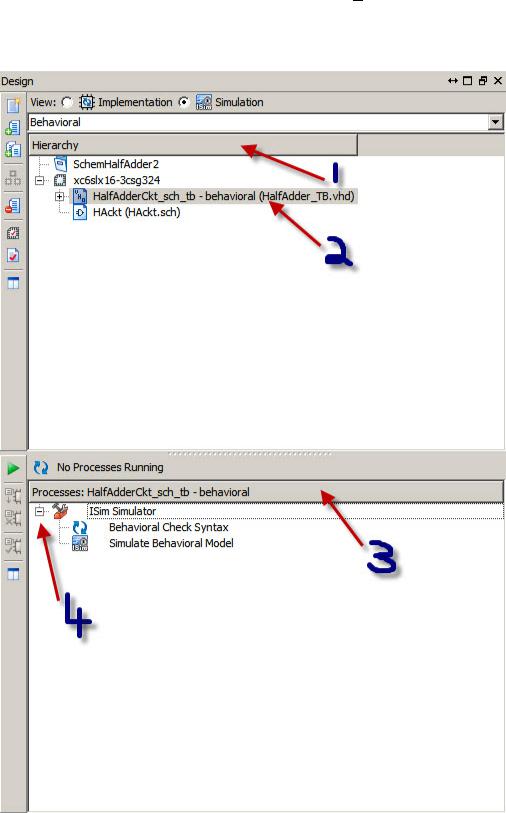


1. Locating the Simulation Processes: The simulation processes in the ISE software enable you to run simulation on the design using ISim. To locate the ISim processes, do the following:

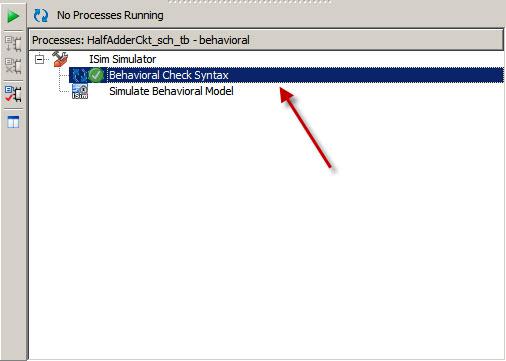
(a) In the View pane of the Project Navigator Design panel, select Simulation, and select Behavioral from the drop-down list.



(b) In the Hierarchy pane, select the test bench files (Hack TB).

(c) In the Processes pane, expand ISim Simulator to view the process hierarchy.

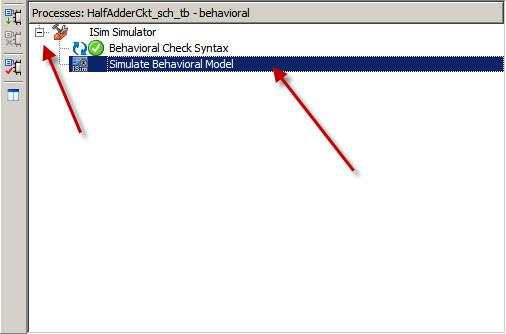
The following simulation processes are available: Check Syntax and Simulate Behavioral Model. Use the Check Syntax to make sure that your benchmark is free of any syntax errors. If all goes well you will see the following message (Process “Behavioral Check Syntax” Completed successfully).



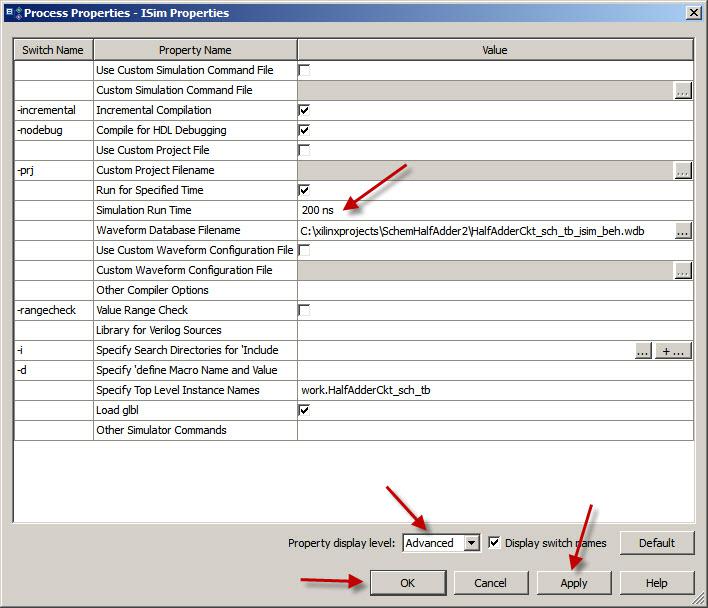
3. Specifying Simulation Properties: Now you will perform a behavioral simulation on the Half Adder circuit design after you set process properties for simulation. The ISE software allows you to set several ISim properties in addition to the simulation net-list properties. To see the behavioral simulation properties and to modify the properties for this tutorial, do the following:

(a) In the Hierarchy pane of the Project Navigator Design panel, select the test bench file (HAckt TB).

(b) In the processes pane, expand ISim Simulator, right-click Simulator Behavioral Model, and select Process Properties.



(c) In the Process Properties dialog box, set the Property display level to Advanced.



(d) Change the Simulation Run Time to say 200 ns.

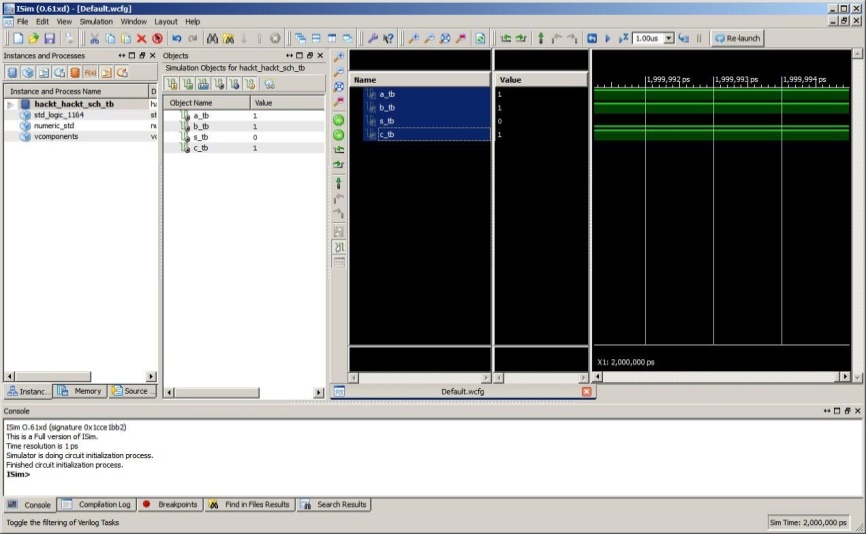
(e) Click Apply and then click OK.

4. Performing Simulation: After the process properties have been set, you are ready to run ISim to simulate the design. Follow the steps below to get waveforms for your design:

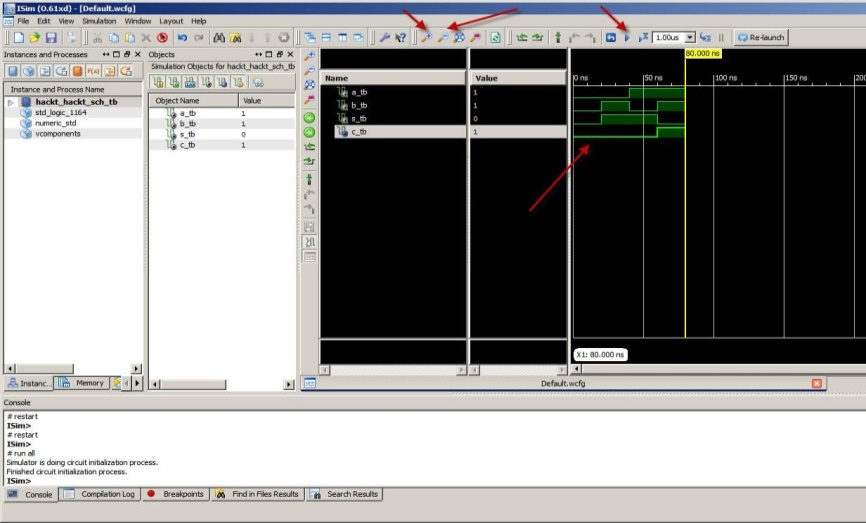
(a) To start the behavioral simulation, double-click Simulate Behavioral Model.

(b) ISim creates the work directory, compiles the sources files, loads the design, and performs simulation for the time specified.

(c) The following screen will pop:



(d)To display the correct waveforms expected you will need to use the zoom tools as shown below (zoom in or out):



(e) If you need to rerun the simulation, click the Restart Simulation icon 

5. Your behavioral simulation is complete. You can print the waveforms and submit them along with your report by moving the mouse to the file section and then print or print preview.

**6.3. VHDL:**

VHDL ([VHSIC](https://en.wikipedia.org/wiki/VHSIC) Hardware Description Language) is a [hardware description language](https://en.wikipedia.org/wiki/Hardware_description_language) used in [electronic design automation](https://en.wikipedia.org/wiki/Electronic_design_automation) to describe [digital](https://en.wikipedia.org/wiki/Digital_electronics) and [mixed-signal](https://en.wikipedia.org/wiki/Mixed-signal_integrated_circuit) systems such as [field-programmable gate arrays](https://en.wikipedia.org/wiki/Field-programmable_gate_array) and [integrated circuits](https://en.wikipedia.org/wiki/Integrated_circuit). VHDL can also be used as a general purpose [parallel programming language](https://en.wikipedia.org/wiki/Parallel_programming_language). VHDL was originally developed at the behest of the [U.S Department of Defense](https://en.wikipedia.org/wiki/United_States_Department_of_Defense) in order to document the behavior of the [ASICs](https://en.wikipedia.org/wiki/Application-specific_integrated_circuit) that supplier.

VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. This collection of simulation models is commonly called a test bench. VHDL has file input and output capabilities, and can be used as a general-purpose language for text processing, but files are more commonly used by a simulation test bench for stimulus or verification data. There are some VHDL compilers which build executable binaries. In this case, it might be possible to use VHDL to write a test bench to verify the functionality of the design using files on the host computer to define stimuli, to interact with the user, and to compare results with those expected. However, most designers leave this job to the simulator. It is relatively easy for an inexperienced developer to produce code that simulates successfully but that cannot be synthesized into a real device, or is too large to be practical. One particular pitfall is the accidental production of [transparent latches](https://en.wikipedia.org/wiki/Transparent_latch) rather than [D-type flip-flops](https://en.wikipedia.org/wiki/Flip-flop_(electronics)#D_flip-flop) as storage elements.

One can design hardware in a VHDL IDE (for FPGA implementation such as Xilinx ISE, Altera Quartus, Synopsys Synplify or Mentor Graphics HDL Designer) to produce the [RTL](https://en.wikipedia.org/wiki/Register-transfer_level) schematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test bench. To generate an appropriate test bench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required. A final point is that when a VHDL model is translated into the "gates and wires" that are mapped onto a programmable logic device such as a [CPLD](https://en.wikipedia.org/wiki/CPLD) or [FPGA](https://en.wikipedia.org/wiki/FPGA), then it is the actual hardware being configured, rather than the VHDL code being "execute.

**Spartan family**

The Spartan series targets applications with a low-power footprint, extreme cost sensitivity and high-volume; e.g. [displays](https://en.wikipedia.org/wiki/Displays), [set-top boxes](https://en.wikipedia.org/wiki/Set-top_boxes), [wireless routers](https://en.wikipedia.org/wiki/Wireless_router) and other applications.

**CHAPTER 7**

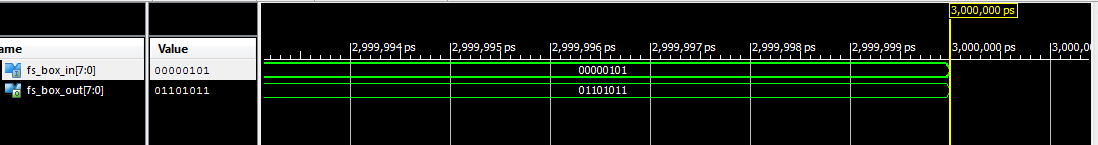
**SIMULATION RESULTS**

**7.1 AES Encryption:**

The wave forms generated by the each round of encryption process are shown below.

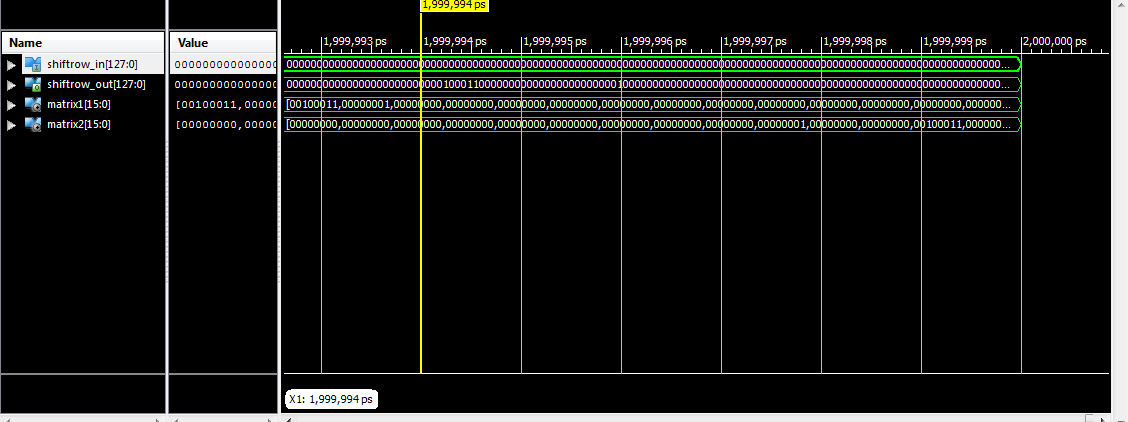
**7.1.1 Byte Substitution:**

The waveforms generated by the 128-bit byte substitution transformation. The inputs are clock of 100ns and active high reset and 128-bit state as a standard logic vector whose output is 128-bit S-box lookup substitution.

****

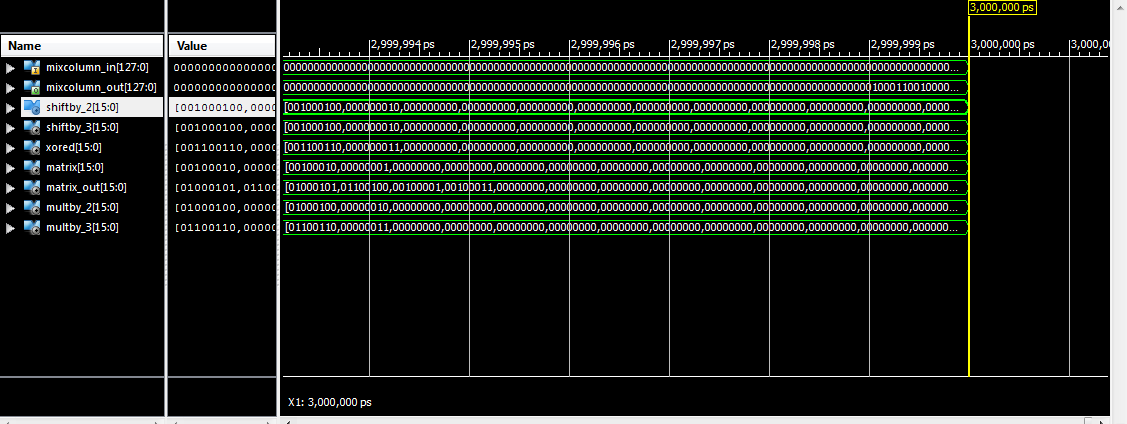
**7.1.2Shift Rows:**

The wave forms generated by the 128-bit shifrrow transformation as shown below.The inputs are clock of 100ns time period, active high reset and 128-bit state as a standard logic vector.



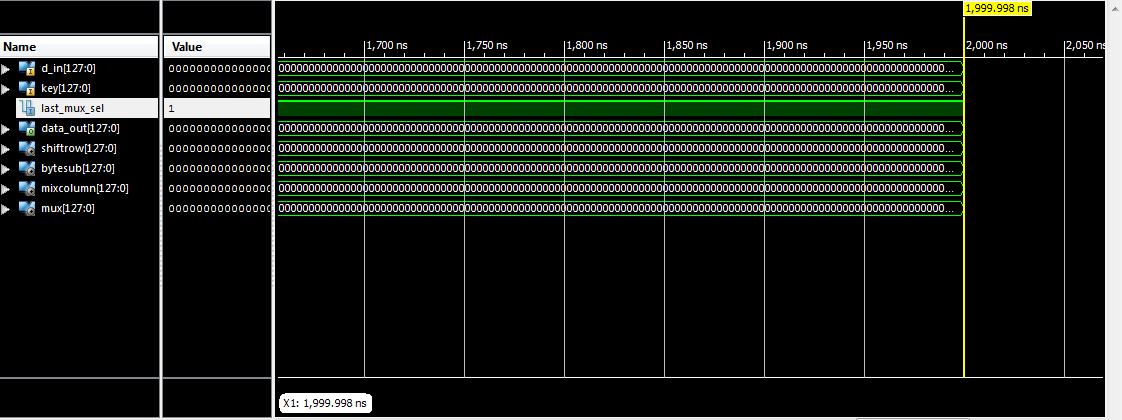
**7.1.3 Mix column:**

The wave forms generated by the 128-bit Mix Columns transformation. The inputs are clock of 100ns time period, active high reset and 128-bit state.

****

**7.1.4 Add Round Key:**

The wave forms generated by the 128-bit Add Round Key operation as shown in below figure. The inputs are of 100ns time period, 128-bit key and 128-bit state as a standard logic vector, whose output is the 128-bit which is EXOR operation of 128-bit key and 128-bit state.

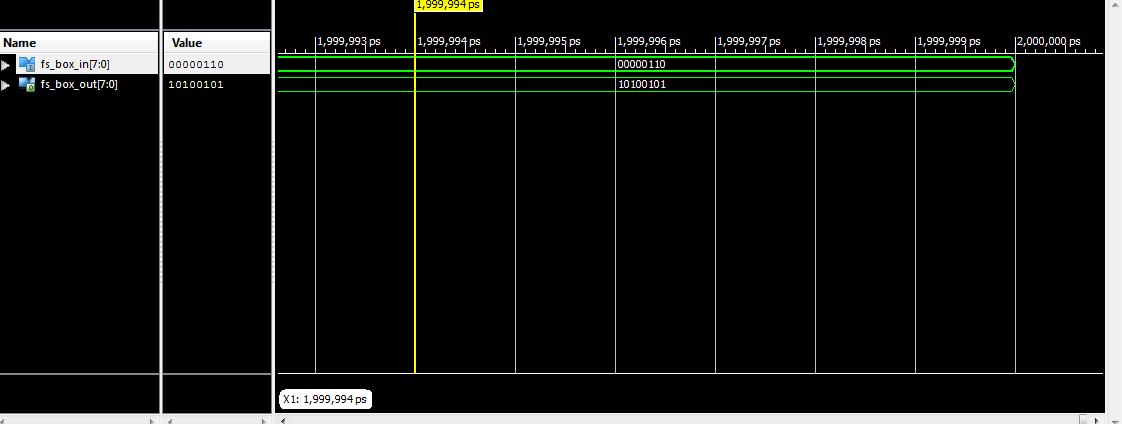


**7.2 AES Decryption:**

The waveforms generated by the each round of decryption process are shown below.

**7.2.1 Inv Byte Substitution:**

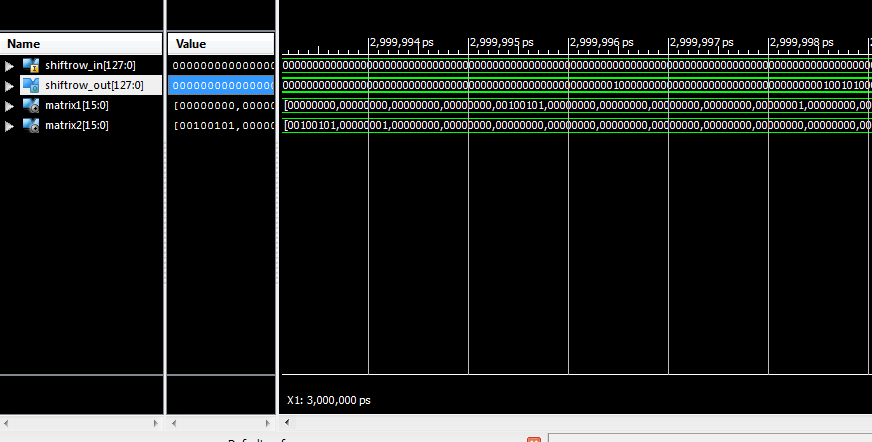
The wave forms generated by the 128-bit inverse byte substitution transformation as shown in below figure. The inputs are clock of 100ns time period, active high reset and 128-bit state as a standard logic vector, whose output is 128-bit inverse S-box lookup substitution.

****

**7.2.2 Inv Shift Rows:**

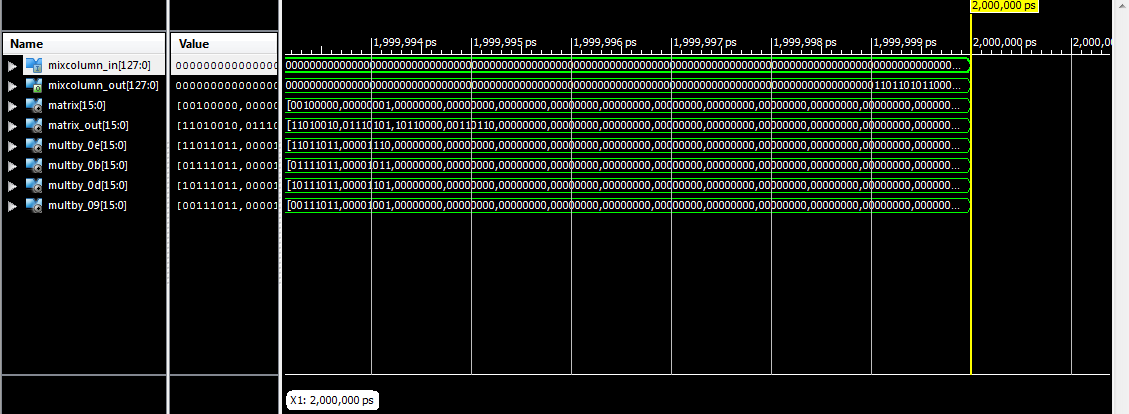
The wave forms generated by the 128-bit inverse shift row transformation are shown in below figure. The inputs are of 100ns time period, active high reset and 128-bit as standard logic vector.

.



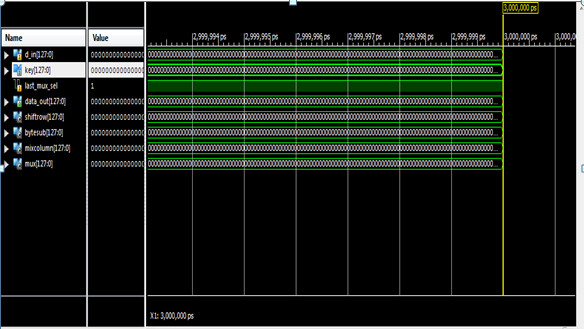
**7.2.3 Inv Mix Column:**

The wave forms generated by 128-bit Inverse Mix Columns transformation are shown in below figure. The inputs are clock of 100ns time period, active high reset and 128-bit state as a standard input logic vector.



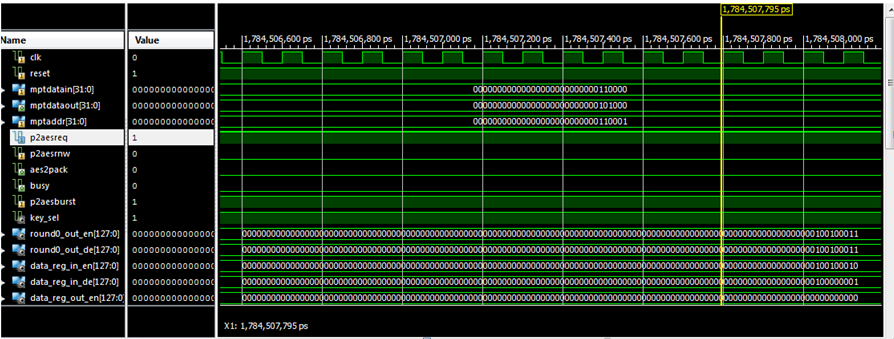
**7.2.4 Inv Add Round Key:**

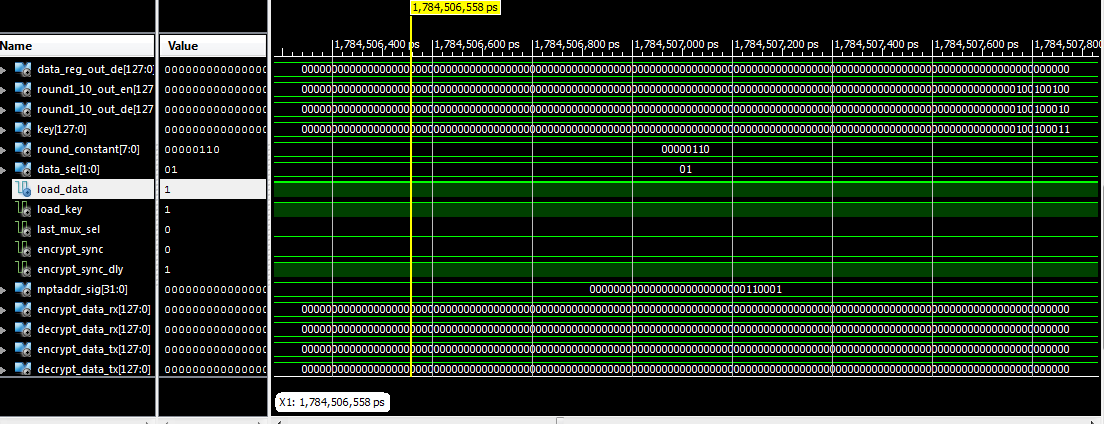
The wave forms generated by inverse add round key transformation as shown in below figure.The inputs are of 100ns time period, 128-bit key and 128-bit state as a standard logic vector whose output is 128-bit whis is produced by EXOR operation of 128-bit key and 128-bit state.

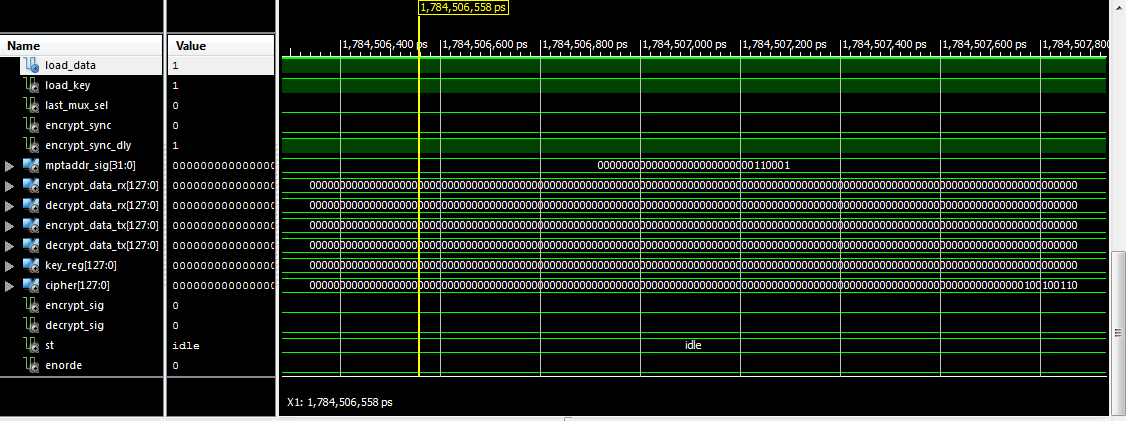


**7.3 Final Round of Encryption and Decryption Process:**

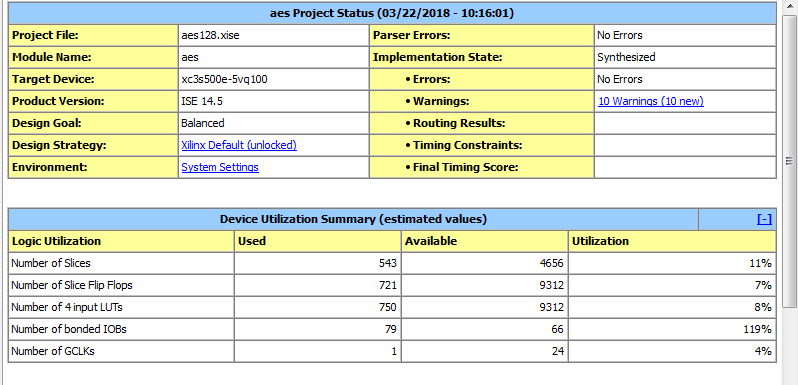
The wave forms generated by the 128-bit complete encryption and decryption process are shown in below figures. The inputs are of clock, active high reset 18-bit state and keys as standard logic vectors. The output produced by encryption is cipher text and the output produced by decryption process is plain text.





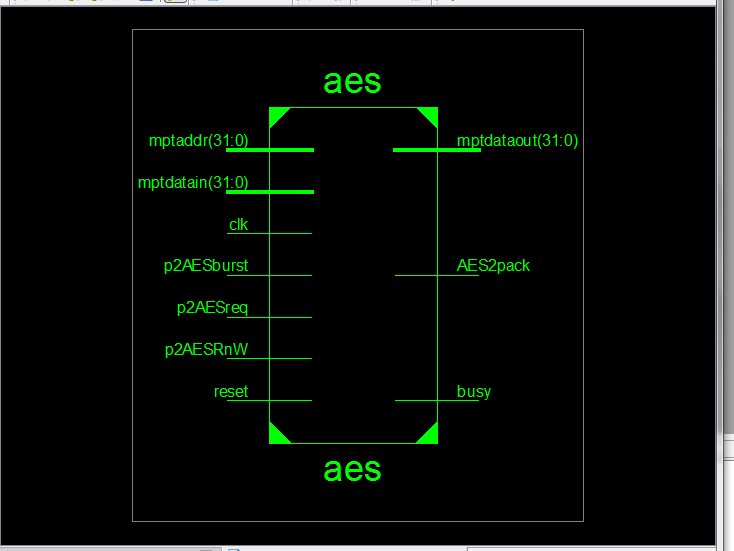


**7.4 Synthesis Report:**

The synthesis report for the aes is shown below.****

**7.5 RTL Schematic:**

The RTL schematic for advanced encryption standard is shown in below figure.

****

**CONCLUSION**

The advanced encryption standard algorithms is an iterative private key symmetric block cipher that can process data blocks of 128 bits through the use of cipher keys with length of 128, 192 and 256 bits. An effective VHDL implementation of 128 bit block and 128 key AES with S Box cryptosystem is presented in this project. An Optimized and Synthesizable VHDL code is developed for the implementation of both 128 bit data encryption and decryption process & description is verified using Xilinx.

**FUTURE SCOPE**

Side channel attacks are the way to test the security levels of cryptosystem. Therefore, future works of our project involves testing of the AES with S box design against the side channel attacks and thereby comparing AES and AES with s box in terms of better security and thereby proposed a better crypto algorithm. Also, other possible attacks are tested on the 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/rijndaeldocV2.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
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
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”, 2nd Edition, San Francisco, CA, Morgan Kaufmann, 2002