AN EFFICIENT VLSI DESIGN OF AES CRYPTOGRAPHY BASED ON DNA DESIGN

ABSTRACT

This paper describes a novel Sub bytes approach for implementation of the advanced encryption standard (AES) algorithm, which provides a significantly improved strength of Cryptography. Our method is based on randomization in composite field arithmetic, which entails a low implementation cost while does not alter the algorithm, does not reduce the working frequency, and keeps perfect compatibility with the published standard. The DNA Encoder, suitable to be integrated in AES Cryptographer, is presented. The DNA Technique is to improve the Security of AES Design. This Proposed AES System with DNA TRNG Implemented using Verilog HDL and Simulated by Modelsim 6.4 c and Synthesized by Xilinx tool. The proposed system implemented in FPGA Vertex or Spartan 3. The proposed AES System has been made into an IP and successfully applied in encryption application.

CHAPTER	TITLE	PAGE NO

CHAPTER 1

INTRODUCTION

1.1 GENERAL:

Cryptography, often called encryption, is the practice of creating and using a cryptosystem or cipher to prevent all but the intended recipient(s) from reading or using the information or application encrypted. A cryptosystem is a technique used to encode a message. The recipient can view the encrypted message only by decoding it with the correct algorithm and keys. Cryptography is used primarily for communicating sensitive material across computer networks. The process of encryption takes a clear-text document and applies a key and a mathematical algorithm to it, converting it into crypto-text. In crypto-text, the document is unreadable unless the reader possesses the key that can undo the encryption. In 1997 the National Institute of Standards and TECHNOLOGY (NIST), a branch of the US government, started a process to identify a replacement for the Data Encryption Standard (DES). It was generally recognized that DES was not secure because of advances in computer processing power. The goal of NIST was to define a replacement for DES that could be used for non-military information security applications by US government agencies. Of course, it was recognized that commercial and other non-government users would benefit from the work of NIST and that the work would be generally adopted as a commercial standard. The NIST invited cryptography and data security specialists from around the world to participate in the discussion and selection process. Five encryption algorithms were adopted for study. Through a process of consensus the encryption algorithm proposed by the Belgium cryptographers Joan Daeman and Vincent Rijmen was selected. Prior to selection Daeman and Rijnmen used the name Rijndael (derived from their names) for the algorithm. After adoption the encryption algorithm was given the name Advanced Encryption Standard (AES) which is in common use today. In 2000 the NIST formally adopted the AES encryption algorithm and published it as a federal standard under the designation FIPS-197. The full FIPS-197 standard is available on the NIST web site (see the Resources section below). As expected, many providers of encryption software and hardware have incorporated AES encryption into their products.

The AES encryption algorithm is a block cipher that uses an encryption key and a several rounds of encryption. A block cipher is an encryption algorithm that works on a single block of data at

a time. In the case of standard AES encryption the block is 128 bits, or 16 bytes, in length. The term "rounds" refers to the way in which the encryption algorithm mixes the data re-encrypting it ten to fourteen times depending on the length of the key. This is described in the Wikipedia article on AES encryption. The AES algorithm itself is not a computer program or computer source code. It is a mathematical description of a process of obscuring data. A number of people have created source code implementations of AES encryption, including the original authors.

AES encryption uses a single key as a part of the encryption process. The key can be 128 bits (16 bytes), 192 bits (24 bytes), or 256 bits (32 bytes) in length. The term 128-bit encryption refers to the use of a 128-bit encryption key. With AES both the encryption and the decryption are performed using the same key. This is called a symmetric encryption algorithm. Encryption algorithms that use two different keys, a public and a private key, are called asymmetric encryption algorithms. An encryption key is simply a binary string of data used in the encryption process. Because the same encryption key is used to encrypt and decrypt data, it is important to keep the encryption key a secret and to use keys that are hard to guess. Some keys are generated by software used for this specific task. Another method is to derive a key from a pass phrase. Good encryption systems never use a pass phrase alone as an encryption key.

Side channel Attacks are attacks on the implementation of AES, not on the input or the AES cipher text. It attempts to correlate various measurements of the encrypting tool with time in an attempt to guess the key. A professor at MIT,[9] encoded an AES algorithm on his computer, an 850MHz, Pentium III running FreeBSD 4.8 and by measuring time delays between the CPU and memory was able to successfully guess the key in under 100 minutes. There is a correlation between the index of an array and the time it takes to get the results back. This is due to the physical location of the data in the memory device. Data closer to the output lead will not take as much time to be retrieved as data further away, because it will not take as long for the signal to propagate its way out of the chip.

He feels he can improve on this time. After running a few thousand encryptions he spent about an hour studying the results of his measurements. After studying the data, there were many repetitions to avoid noise, he concluded the key was one of several possibilities. By trying each one, he was able to find the key. He believes this analysis process can be programmed, cutting the time down to just a few minutes.

The method of measuring time delays in memory requests are called timing attacks. Power attacks attempt to measure power consumption by the CPU. It takes more power to switch 8 bits than it takes to switch 1 bit. Some are also now measuring radiation levels from CPU's and gaining knowledge of its inner workings.

There are several techniques which can greatly frustrate side channel attacks. 1) Avoid use of arrays. Compute values of SBOX and RCon to avoid timing attacks. 2) Design algorithms and devices to work with constant time intervals. (independent of key and plaintext.) Study your device spec sheets, and insist on accurate data. For example you should know which takes longer, XOR or shift operations. 3) Use same memory throughout, remember, cache is faster than DRAM. 4) Compute Key Expansion on the fly. Don't compute the Key Expansion and then reference it from memory. 5) Utilize pipelining to stabilize CPU power consumption. 6) Use specialized chips whenever possible, right now they are significantly faster than CPU's and require extremely expensive equipment for side channel attack measurements.

NIST was aware of side channel attacks when evaluating all the finalists. Comparing the Rijndael algorithm security against side channel attacks to the other four finalists considered by NIST they concluded:

- Rijndael and Serpent use only Boolean operations, table lookups, and fixed shifts/rotations. These operations are the easiest to defend against attacks.
- Two fish uses addition, which is somewhat more difficult to defend against attacks.
- MARS and RC6 use multiplication/division/squaring and/or variable shift/rotation. These operations are the most difficult to defend.

1.2 OBJECTIVE:

This paper describes a novel DNA approach for implementation of the advanced encryption standard (AES) algorithm and Key expansion another contribution of this paper is that it designs Encryption Design using Shit rows, Mixed Column, Add Round Key and We Will Design a Decryption Part also. Finally with the help of DNA Block We Implement a New AES DNA Encryption & Decryption. The results of this paper can be served for protecting the Data with the High Security.

1.3 EXISTING SYSTEM:

This paper presents novel high-speed architectures for the hardware implementation of the Advanced Encryption Standard (AES) algorithm. Unlike previous works which rely on look-up tables to implement the Sub Bytes and Inv Sub Bytes transformations of the AES algorithm, the proposed design employs combinational logic only. As a direct consequence, the unbreakable delay incurred by look-up tables in the conventional approaches is eliminated, and the advantage of sub pipelining can be further explored. Furthermore, composite field arithmetic is employed to reduce the area requirements, and here Key expansion method was implemented for Random Key Generation. In this method, area and delay was Increased due to Round Key Expansion Process.

DISADVANTAGE:

- Area is more
- Complex Key Generation Process
- Manual Key Process

1.4 LITERATURE SURVEY:

Title: Differential power analysis: A serious threat for FPGA security
Authors: M. Masoumi
Publication: Int. J. Internet Technol. Secured Trans., vol. 4, no. 1, pp. 12–25

Year: 2012

Although cryptosystem designers frequently assume that secret parameters will be manipulated in closed reliable computing environments, Kocher et al. reported in 1998 that microchips leak information correlated with the data handled and introduced a new kind of attacks which were radically different from software and algorithmic attacks. These attacks use leaking or side-channel information, like power consumption data, electromagnetic emanations or computing time to recover the secret key. While FPGAs are becoming increasingly popular for cryptographic applications, there are only a few articles that assess their vulnerability to such attacks. This paper describes the principles of differential power analysis (DPA) attack and also illustrates a practical and successful implementation of this attack against an FPGA implementation of the advanced encryption standard (AES) algorithm. The results obtained in this work clearly demonstrate that DPA is a serious threat against realization of encryption algorithms on SRAM-based FPGAs without effective countermeasures.

Title: The research of DPA attacks against AES implementations
Authors: H. Yu, Z. Xue-Cheng, L. Zheng-Lin, and C. Yi-Chen
Publication: J. China Univ. Posts Telecommun. vol. 15, no. 4, pp. 101–106,
Year: Dec. 2008

This article examines vulnerabilities to power analysis attacks between software and hardware implementations of cryptographic algorithms. A simulation-based experimental environment is built to acquire power data, and single-bit differential power analysis (DPA), and multi-bit DPA and correlation power analysis (CPA) attacks are conducted on two implementations respectively. The experimental results show that the hardware implementation has less data-dependent power leakages to resist power attacks. Furthermore, an improved DPA approach is proposed. It adopts hamming distance of intermediate results as power model and arranges plaintext inputs to differentiate power traces to the maximal probability. Compared with the original power attacks, our improved DPA performs a successful attack on AES hardware implementations with acceptable power measurements and fewer computations.

Title: AES against first and second-order differential power analysis Applied Cryptography and Network Security

Authors: J. Zhou and M. Yung, Eds.

Publication: vol. 6123, Springer-Verlag, pp. 168–185. Berlin, Germany

Year: 2010

Differential Power Analysis (DPA) is a powerful and practical technique used to attack a cryptographic implementation in a resource limited application environment. In this paper, they show that some intermediate values from the inner rounds can be exploited by deploying techniques such as fixing certain plaintext/cipher text bytes. We give five general principles on DPA vulnerability of unprotected AES implementations, and give several general principles on DPA vulnerability of protected AES implementations. These principles specify which operations of AES are vulnerable to first and second-order DPA. To illustrate the principles, we attack the two AES implementations [1, 2] that use two kinds of countermeasures to achieve a high resistance against power analysis, and demonstrate that they are even vulnerable to DPA. Finally, we conclude that at least the first two and a half rounds and the last three rounds of AES should be secured for an AES software implementation to be resistant against first and second-order DPA in practice.

Title: High-speed VLSI architectures for the AES algorithm

Authors: X. Zhang and K. K. Parhi

Publication: IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 12, no. 9, pp. 957–967

Year: Sep. 2004

This paper presents novel high-speed architectures for the hardware implementation of the Advanced Encryption Standard (AES) algorithm. Unlike previous works which rely on look-up tables to implement the Sub Bytes and Inv Sub Bytes transformations of the AES algorithm, the proposed design employs combinational logic only. As a direct consequence, the unbreakable delay incurred by look-up tables in the conventional approaches is eliminated, and the advantage of sub pipelining can be further explored. Furthermore, composite field arithmetic is employed to reduce the area requirements, and different implementations for the inversion in subfield GF(2/sup 4/) are compared.

In addition, efficient key expansion architecture suitable for the sub pipelined round units is also presented.

Title: Differential power analysis: A serious threat for FPGA security

Authors: M. Masoumi

Publication: Int. J. Internet Technol. Secured Trans., vol. 4, no. 1, pp. 12–25

Year: 2012

Although cryptosystem designers frequently assume that secret parameters will be manipulated in closed reliable computing environments, Kocher et al. reported in 1998 that microchips leak information correlated with the data handled and introduced a new kind of attacks which were radically different from software and algorithmic attacks. These attacks use leaking or side-channel information, like power consumption data, electromagnetic emanations or computing time to recover the secret key. While FPGAs are becoming increasingly popular for cryptographic applications, there are only a few articles that assess their vulnerability to such attacks. This paper describes the principles of differential power analysis (DPA) attack and also illustrates a practical and successful implementation of this attack against an FPGA implementation of the advanced encryption standard (AES) algorithm. The results obtained in this work clearly demonstrate that DPA is a serious threat against realization of encryption algorithms on SRAM-based FPGAs without effective countermeasures.

Title: Differential power analysis: A serious threat for FPGA security

Authors: M. Masoumi

Publication: Int. J. Internet Technol. Secured Trans., vol. 4, no. 1, pp. 12–25

Year: 2012

Although cryptosystem designers frequently assume that secret parameters will be manipulated in closed reliable computing environments, Kocher et al. reported in 1998 that microchips leak information correlated with the data handled and introduced a new kind of attacks which were radically different from software and algorithmic attacks. These attacks use leaking or side-channel information, like power consumption data,

electromagnetic emanations or computing time to recover the secret key. While FPGAs are becoming increasingly popular for cryptographic applications, there are only a few articles that assess their vulnerability to such attacks. This paper describes the principles of differential power analysis (DPA) attack and also illustrates a practical and successful implementation of this attack against an FPGA implementation of the advanced encryption standard (AES) algorithm. The results obtained in this work clearly demonstrate that DPA is a serious threat against realization of encryption algorithms on SRAM-based FPGAs without effective countermeasures.

Title: The research of DPA attacks against AES implementations

Authors: H. Yu, Z. Xue-Cheng, L. Zheng-Lin, and C. Yi-Chen

Publication: J. China Univ. Posts Telecommun. vol. 15, no. 4, pp. 101–106,

Year: Dec. 2008

This article examines vulnerabilities to power analysis attacks between software and hardware implementations of cryptographic algorithms. A simulation-based experimental environment is built to acquire power data, and single-bit differential power analysis (DPA), and multi-bit DPA and correlation power analysis (CPA) attacks are conducted on two implementations respectively. The experimental results show that the hardware implementation has less data-dependent power leakages to resist power attacks. Furthermore, an improved DPA approach is proposed. It adopts hamming distance of intermediate results as power model and arranges plaintext inputs to differentiate power traces to the maximal probability.

Title: AES against first and second-order differential power analysis Applied Cryptography and Network Security

Authors: J. Zhou and M. Yung, Eds.

Publication: vol. 6123, Springer-Verlag, pp. 168–185. Berlin, Germany

Year: 2010

In this paper, they show that some intermediate values from the inner rounds can be exploited by deploying techniques such as fixing certain plaintext/cipher text bytes. We give five general principles on DPA vulnerability of unprotected AES implementations, and give several general principles on DPA vulnerability of protected AES implementations. These principles specify which operations of AES are vulnerable to first and second-order DPA. To illustrate the principles, we attack the two AES implementations that use two kinds of countermeasures to achieve a high resistance against power analysis, and demonstrate that they are even vulnerable to DPA.

Title: Differential fault analysis on lightweight block ciphers with statistical cryptanalysis techniques

Authors: D. Gu, J. Li, S. Li, Z. Ma, Z. Guo, and J. Liu

Year: September 2012

Publication: Fault Diagnosis and Tolerance in Cryptography (FDTC)

Differential fault analysis is one of the most efficient side channel attack techniques that threat the security of block cipher. However, it often requires a penultimate or an antepenultimate round faulty encryption and is not suitable for middle round fault. This paper presents attacks combining differential fault analysis with statistical cryptanalysis techniques Against lightweight ciphers. The analysis makes use of statistical cryptanalysis techniques in practice rather than theoretically, and exploits the weakness of bit-permutation adopted by many lightweight block ciphers under fault attack. Specific attacks against PRESENT and PRINTCIPHER are given to prove the validity.

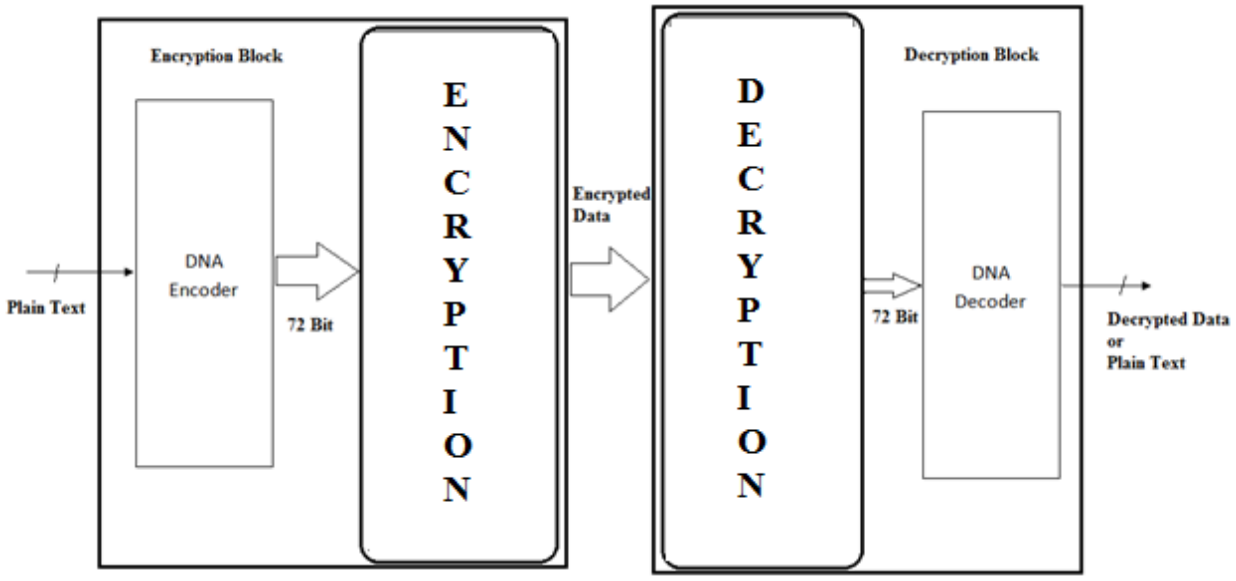
1.5 PROPOSED SYSTEM:

The proposed method is based on DNA Encoding in the underlying Galois Field of the AES and unlike other countermeasure does not add any additional operation to the algorithm execution flow, does not decrease the working frequency, does not alter the algorithm and keeps perfect compatibility with the published standard. In addition, it does not require pre-computed tables for storing masked values in ROM. This is specifically important for constrained security tokens such as smart cards. We focused on the optimization strategies for 128-b data path architecture to achieve

High Security, High-throughput hardware AES encryption module providing multiple levels of security with small area footprint. The area is saved by reorganizing the encryption data path to minimize the number of data registers and combinational logics.

1.5.1 PROPOSED SYSTEM BLOCK DIAGRAM:

PROPOSED BLOCK DIAGRAM:



1.5.2 PROPOSED SYSTEM TECHNIQUE:

- AES with S-Box Design with DNA Encoder

1.5.3 PROPOSED SYSTEM ADVANTAGES:

- Low Complexity
- Low Power
- Feasible to use for different performance
- Implementation objectives of sensitive smart applications

CHAPTER 2

NOVEL APPROACH TO PROTECT ADVANCED ENCRYPTION STANDARD ALGORITHM IMPLEMENTATION

2.1 GENERAL:

Cryptographic architectures provide protection for sensitive and smart infrastructures such as secure healthcare, smart grid, fabric, and home. Cryptography is closely related to the disciplines of cryptology and cryptanalysis. Cryptography includes techniques such as microdots, merging words with images, and other ways to hide information in storage or transit. However, in today's computer-centric world, cryptography is most often associated with scrambling plaintext (ordinary text, sometimes referred to as clear text) into cipher text (a process called encryption), then back again (known as decryption). Individuals who practice this field are known as cryptographers. Nonetheless, the use of cryptographic architectures does not guarantee immunity against faults occurring in these infrastructures. Defects in VLSI systems may cause smart usage models to malfunction. Extensive research has been done for detecting such faults in the cryptographic algorithms such as elliptic curve cryptography and the Advanced Encryption Standard. Design for reliability and fault immunity ensures that with the presence of faults, reliability is provided for the aforementioned sensitive cryptographic architectures.

MODULES SEPERATION:

- Substitution Box (S- Box)
- Multiplier in GF(2^4)
- Multiplier in GF(2^2)
- Squarer in GF(2^4)
- Constant multiplier ($\times\lambda$)
- Multiplier ($\times\phi$)
- Addition in GF (2^4)
- Shift Rows

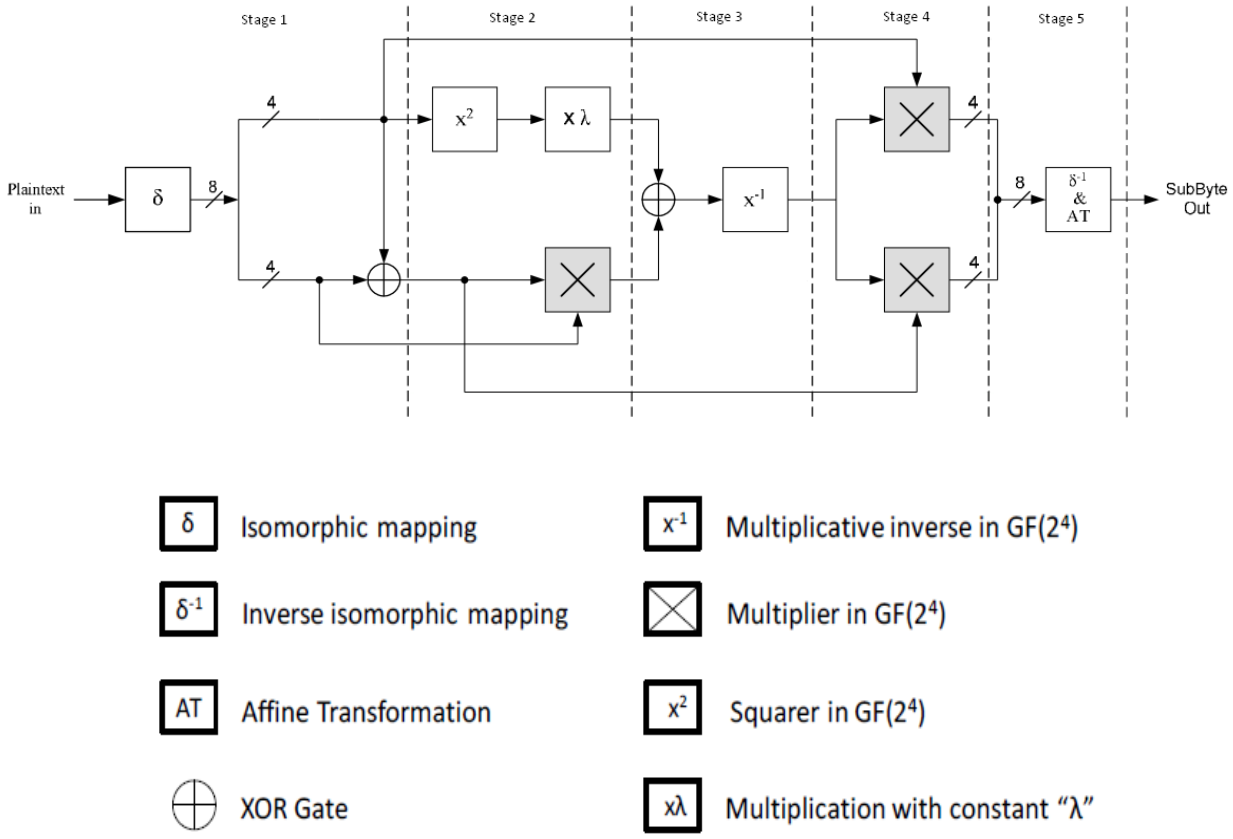
- Mix Columns
- Add Round Key
- Transformation matrix & inverse transformation matrix
- Key Expansion Unit
- DNA Coder

MODULE DESCRIPTION:

Substitution Box (S- Box):

The Sub Bytes operation is a nonlinear byte substitution. Each byte from the input state is replaced by another byte according to the substitution box (called the S-box). The S-box is computed based on a multiplicative inverse in the finite field GF (2^8) and a bitwise affine transformation.

In this module The implementation of the composite field S-BOX is accomplished using combinational logic circuits rather than using pre-stored S-BOX values.



ADDITION IN GF (2^4):

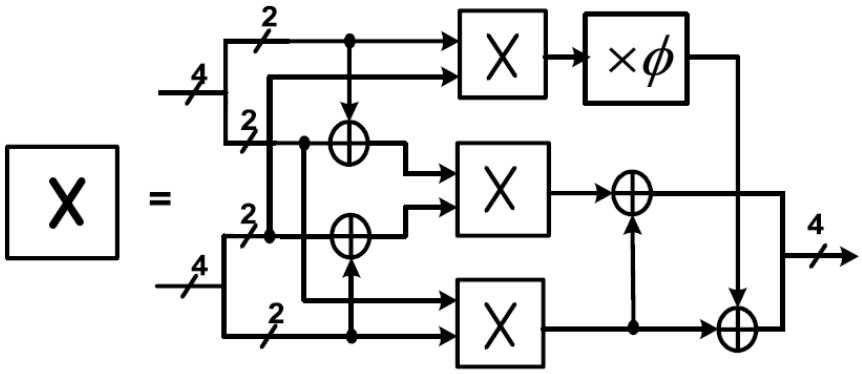
Addition of 2 elements in Galois Field can be translated to simple bitwise XOR operation Addition of 2 elements in Galois Field can be translated to simple bitwise XOR operation.

GF (2^4) MULTIPLIER

Sub Bytes is a nonlinear transformation that uses 16 byte substitution tables (S-Boxes). An S-Box is the multiplicative inverse of a Galois field GF (2^4) followed by an affine transformation**.** Although two Galois Fields of the same order are isomorphic, the complexity of the field operations may heavily depend on the representations of the field elements. Composite field arithmetic can be employed to reduce the hardware complexity.

Three multipliers in GF (2^4) are required as a part of finding the multiplicative inverse in GF (2^8). Fig. shows the GF (2^4) multiplier circuit. As can be seen from the figure the GF (2^4) multipliers consist of 3 GF (2^2) multipliers with 4 XOR Gates and with constant multiplier θ. This

constant multiplier which has 2 bits input extracts the lower bit output as the higher bit input, while the higher output bit will be the result of XOR operation between the 2 input bits. Full derivation of this multiplier circuit can be found.

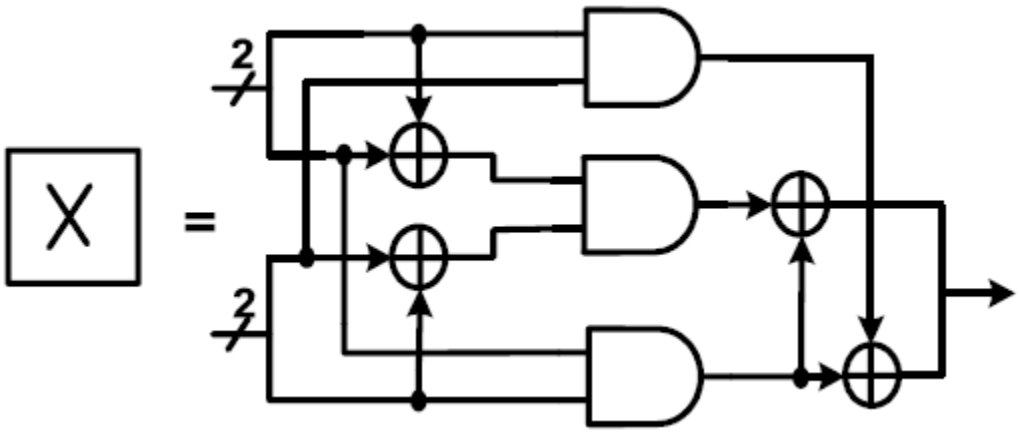


GF (2^4) Multiplier

GF (2^2) MULTIPLIER

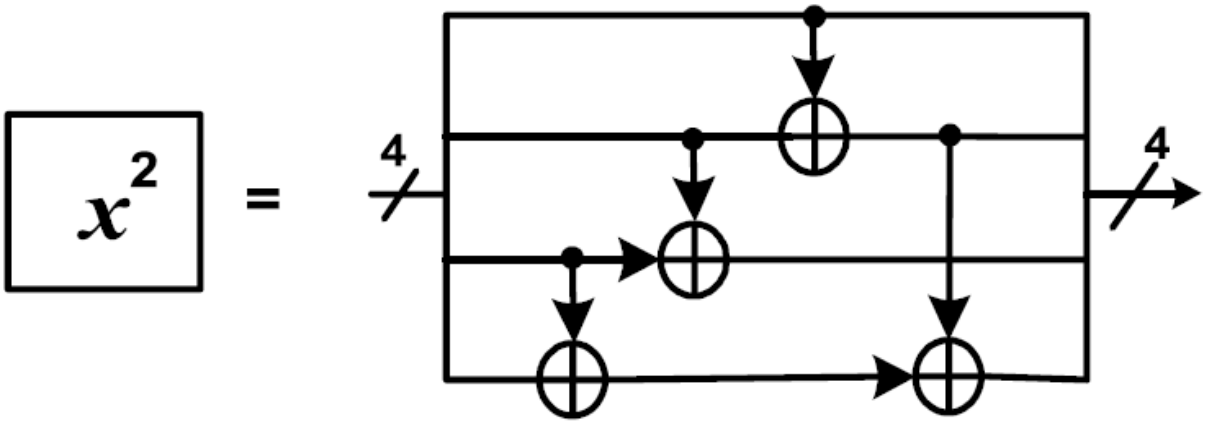
While each finite field is itself not infinite, there are infinitely many different finite fields; their number of elements (which is also called cardinality) is necessarily of the form p^n where p is a prime number and n is a positive integer.

IMPLEMENTATION OF THE GF (2^2) MULTIPLIER:



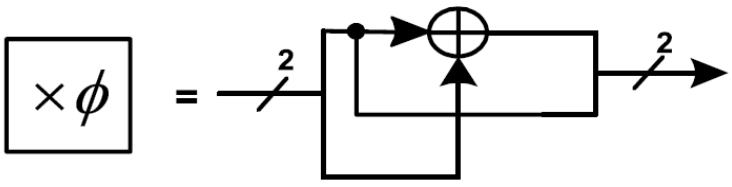
GF (2^4) SQUARER:

It consists of bitwise xor operation. A bitwise operation operates on one or more bit atterns or binary numerals at the level of their individual bits. It is a fast, primitive action directly supported by the processor, and is used to manipulate values for comparisons and calculations. On simple low-cost processors, typically, bitwise operations are substantially faster than division, several times faster than multiplication, and sometimes significantly faster than addition. While modern high-performance processors usually perform addition and multiplication as fast as bitwise operations, the latter may still be optimal for overall power/performance due to lower resource use.



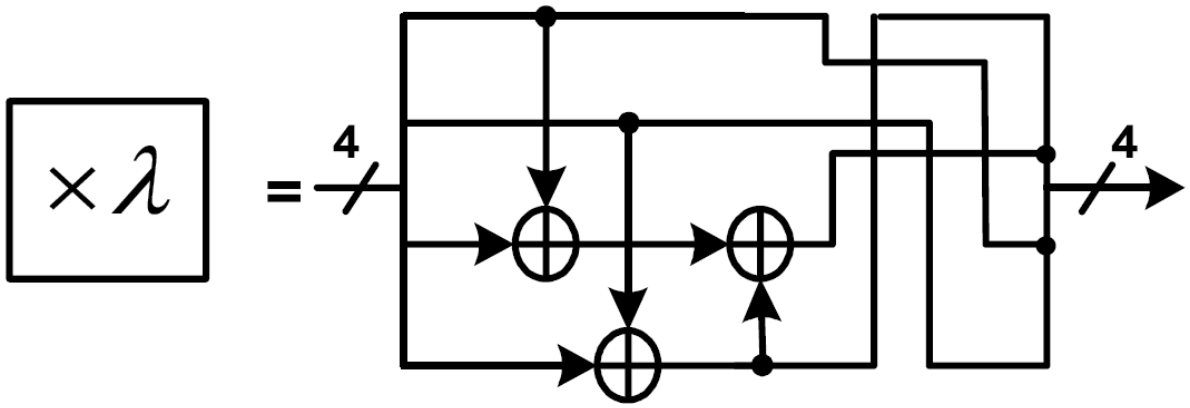
Hardware diagram for Squarer in GF (2^4)

CONSTANT MULTIPLIER (Xϕ):



HARDWARE IMPLEMENTATION OF MULTIPLICATION WITH CONSTANT Φ

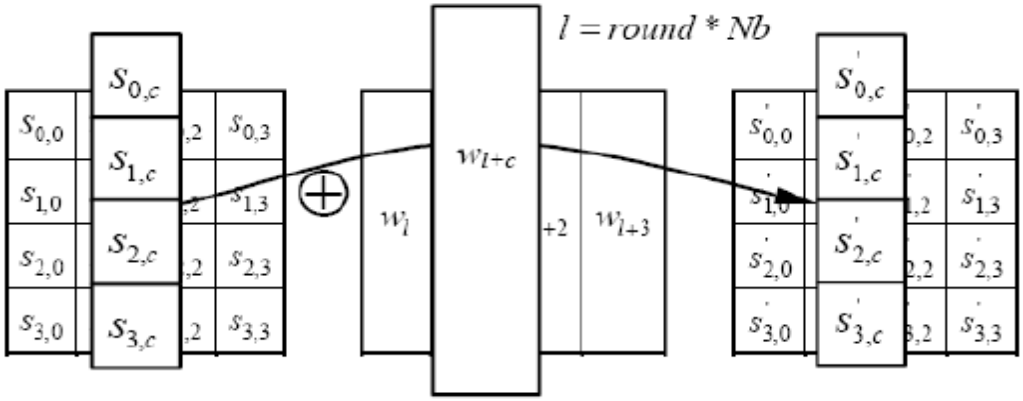
CONSTANT MULTIPLIER (Lambda):



HARDWARE DIAGRAM FOR MULTIPLICATION WITH CONSTANT

ADDROUNDKEY TRANSFORMATION:

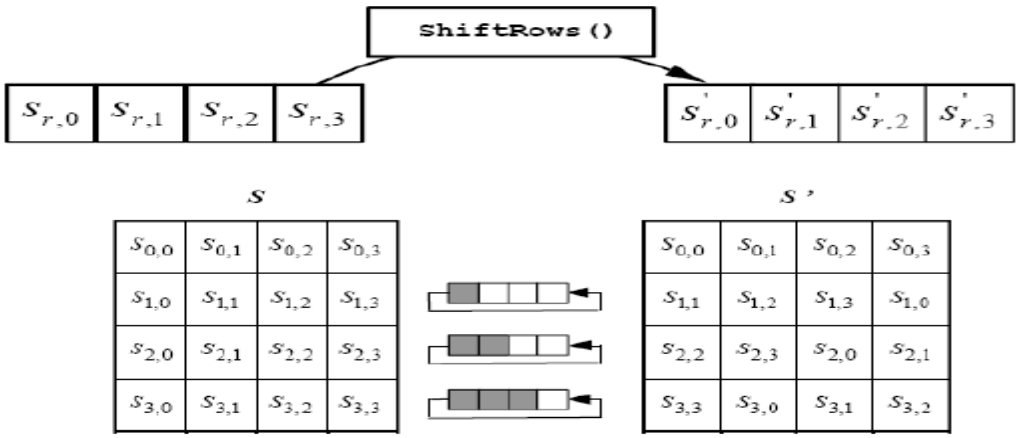
In the AddRoundKey transformation, a round key is added to the state by Bitwise Exclusive-OR (XOR) operation. Figure below illustrates the AddRoundKey. This transformation is the same for both encryption and decryption.



SHIFT ROWS TRANSFORMATION (INV SHIFT ROWS):

Shift Rows is a cyclic shift operation in each row of the State. In this operation, the bytes in the first row of the state do not change. The second, third, and fourth rows shift cyclically to the left one

byte, two bytes, three bytes, respectively, as illustrated in Figure. The reverse process, inv Shift Row, operates in reverse order to Shift Rows.



MIX COLUMN TRANSFORMATION (INV MIX COLUMN):

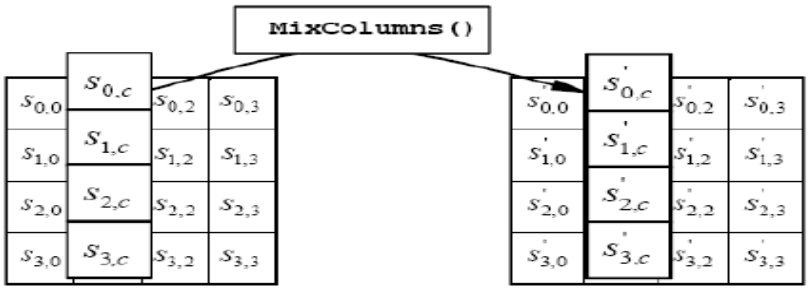
The Mix Column transformation is performed independently on the state Column-by-column. Each column is considered as four term polynomial over GF (2^8) and multiplied by

$$a(x) \text{ modulo } (x4 + 1) \text{ where } a(x) = \{03\}x3 + \{01\}x2 + \{01\}x + \{02\}$$

This transformation can be expressed in matrix form as

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

For *invMixColumn()*, replace $a(x) = \{0E\}x^3 + \{09\}x^2 + \{0D\}x + \{0B\}$.



TRANSFORMATION MATRIX & INVERSE TRANSFORMATION MATRIX:

A transformation matrix (M) transforms the elements in the binary field to the composite field GF $((2^3)^3)$. Then, the operations are done in composite fields to achieve the inverse which is then retransformed to binary field using an inverse transformation matrix (M^{-1}). Eventually, the two most and least significant bits are discarded to get to the uneven structure of the substitution box.

$$\delta \times q = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{pmatrix} \quad \delta^{-1} \times q = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} q_7 \\ q_6 \\ q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{pmatrix}$$

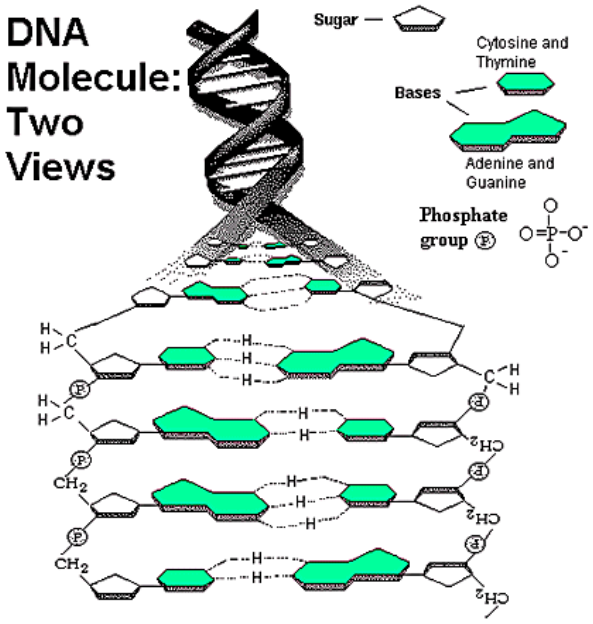
S BOX Truth Table:

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

DNA CODER:

DNA

One of the most essential components required for the functioning of all living organisms is DNA. DNA stands for Deoxyribonucleic acid and it has many properties like vast parallelism, exceptional energy storage capability. There are four classes of nucleotides, Adenine, Guanine, and Cytosine, Thymine (A, C, G, and T). These nucleotides are strung into polymer chains (DNA strands). DNA is basically used to store genetic information. This information cannot be duplicated or copied. The basic structure of a DNA molecule is shown in bellow Fig.



The concept of biotechnology, where DNA strands are used as a carrier for conveying message from the sender to receiver are used in DNA cryptography. The existing cryptographic schemes like RSA and DES is prone to many attacks in the near future and has been broken. This is one of the main reason for using DNA concept. This concept is still in the early stages and further developments are going on in this area. One end orientation of DNA is 5' and the other end is 3'. DNA which usually exists in the form of double helix structure and these structures are formed with the help of hydrogen bonds between them. This structure is called as Watson – Crick. An amplification method which is used to amplify DNA strands which are generally very small in size is called PCR amplification. PCR stands for Polymer Chain Reaction and it is a fast DNA amplification technology based on Watson-Crick complementarily. This being the most fragile method of amplification, after just 20 cycles the DNA molecule can be amplified to 106.

A=CGA	K=AAG	U=CTG	0=ACT
B=CCA	L=TGC	V=CCT	1=ACC
C=GTT	M=TCC	W=CCG	2=TAG
D=TTG	N=TCT	X=CTA	3=GCA
E=GGC	O=GGA	Y=AAA	4=GAG
F=GGT	P=GTG	Z=CTT	5=AGA
G=TTT	Q=AAC	=ATA	6=TTA
H=CGC	R=TCA	,=GAT	7=ACA
I=ATG	S=ACG	.=GAT	8=AGG
J-AGT	T=TTC	;=GCT	9=GCG

DNA Coding Technology:

DNA coding technology is explained with the help of a flow diagram shown in bellow figure. It can be seen that the input message that has to be encrypted contains characters, and these characters are given as an input to the processing unit. The processing unit generates a triplet code. The Encoded Data generated contains combination of three bases for each character. This is shown with the help of Table.

Based on ASCII CODE Table

Alphabet	DNA Conversion	Decimal Code	Binary Code
A	CGA	67 71 65	01000011 01000111 01000001
B	CCA	67 67 65	01000011 01000011 01000001
C	GTT	71 84 84	01000111 01010100 01010100
D	TTG	84 84 71	01010100 01010100 01000111
E	GGC	71 71 67	01000111 01000111 01000011
F	GGT	71 71 84	01000111 01000111 01010100
0	ACT	65 67 84	01000001 01000011 01010100
1	ACG	65 67 71	01000001 01000011 01000111
2	TAG	84 65 71	01010100 01000001 01000111

3	GCA	71 67 65	01000111 01000011 01000001
4	GAG	71 65 71	01000111 01000001 01000111
5	AGA	65 71 65	01000001 01000111 01000001
6	TTA	84 84 65	01010100 01010100 01000001
7	ACA	65 67 65	01000001 01000011 01000001
8	AGG	65 71 71	01000001 01000111 01000111
9	GCG	71 67 71	01000111 01000011 01000111

CHAPTER 3

AES ALGORITHM

3.1 GENERAL:

AES is a symmetric block cipher. This means that it uses the same key for both encryption and decryption. However, AES is quite different from DES in a number of ways. The algorithm Rijindael allows for a variety of block and key sizes and not just the 64 and 56 bits of DES' block and key size. The block and key can in fact be chosen independently from 128, 160, 192, 224, 256 bits and need not be the same. However, the AES standard states that the algorithm can only accept a block size of 128 bits and a choice of three keys - 128, 192, 256 bits. Depending on which version is used, the name of the standard is modified to AES-128, AES-192 or AES- 256 respectively. As well as these differences AES differs from DES in that it is not a feistel structure. Recall that in a feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. In this case the entire data block is processed in parallel during each round using substitutions and permutations.

The Rijndael algorithm is a symmetric iterated block cipher. The block and key lengths can be 128, 192, or 256 bits. The NIST requested that the AES must implement a symmetric block cipher with a block size of 128 bits. Due to this requirement, variations of Rijndael that can operate on larger block sizes will not be included in the actual standard. Rijndael also has a variable number of iterations or rounds: 10, 12, and 14 when the key lengths are 128, 192, and 256 respectively. The transformations in Rijndael consider the data block as a four-column rectangular array of 4-byte vectors. The key is also considered to be a rectangular array of 4-byte vectors—the number of columns is dependent on key length.

Rijndael decryption comprises the inverse of the transformations that encryption uses, performed in reverse order. Decryption commences with the inverse of the final round, followed by the inverses of the rounds, and finishes with the initial data/key addition, which is its own inverse.

AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

A number of AES parameters depend on the key length. For example, if the key size used is 128 then the number of rounds is 10 whereas it is 12 and 14 for 192 and 256 bits respectively. At present the most common key size likely to be used is the 128 bit key. This description of the AES algorithm therefore describes this particular implementation.

Rijindael was designed to have the following characteristics:

- Resistance against all known attacks.
- Speed and code compactness on a wide range of platforms.
- Design Simplicity

The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages. This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of it's counterpart in the encryption algorithm. The four stages are as follows:

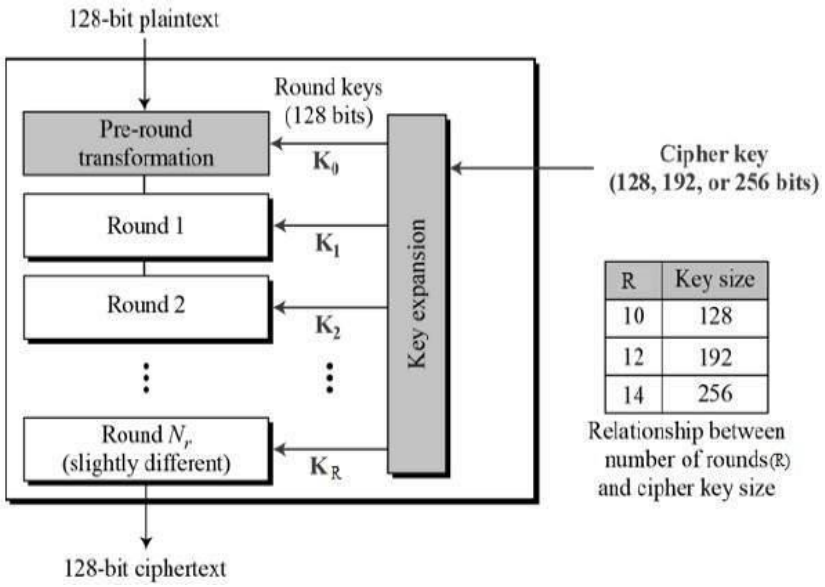
1. Substitute bytes
2. Shift rows
3. Mix Columns
4. Add Round Key

The tenth round simply leaves out the Mix Columns stage. The first nine rounds of the decryption algorithm consist of the following:

1. Inverse Shift rows
2. Inverse Substitute bytes
3. Inverse Add Round Key

4. Inverse Mix Columns

The schematic of AES structure is given in the following illustration

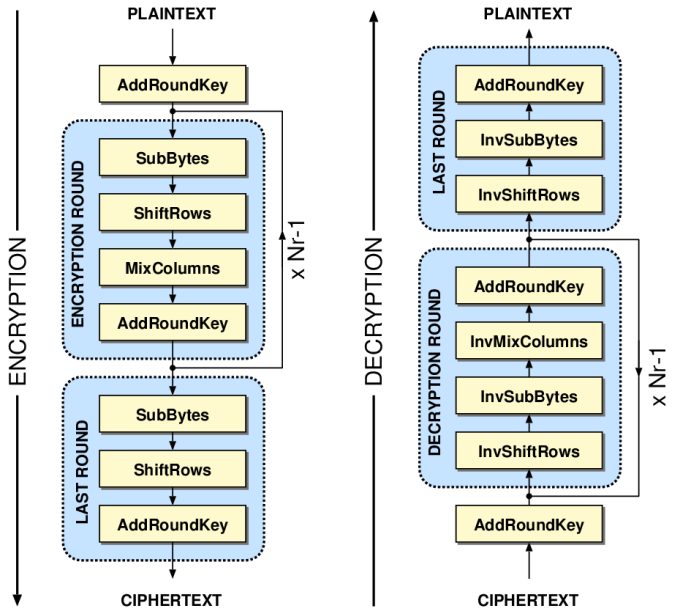


R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

The AES Encryption and Decryption process is mentioned in bellow Diagram. The process of decryption of an AES cipher text is similar to the encryption process in the reverse order. Each round consists of the four processes

- Add round key
- Mix columns
- Shift rows
- Byte substitution



3.2 APPLICATIONS:

- Data encryption and decryption
- Security system
- Digital information security
- Computer/network security

CHAPTER 4

HARDWARE REQUIREMENT

4.1 GENERAL

INTEGRATED CIRCUITS:

The term integrated circuit is used to describe a wide variety of devices ranging from simple logic gates through to complex state-of- the-art microprocessors. Integrated circuits basically consist of a circuit, typically made up from a number of transistors and their interconnections, fabricated from a single semiconductor chip or die.

a) Analogue integrated circuits

Analogue integrated circuits include a wide range of applications, many of which are highly specific. Some examples are the simple operational amplifiers and timers, and the more complex FM stereo decoders and single-chip FM radios.

There has been a trend towards fabricating the more commonly used analogue circuits into single chip form. An example of this is FM radio receiver, which is a fairly complex circuit when fabricated from discrete components. A FM radio receiver can now be constructed from a FM radio chip, an audio amplifier chip and a few discrete passive components.

b) Digital integrated circuits

Digital integrated circuits are devices, which are functionally based on logic gates (AND & OR gates). They are commercially available in families of devices which take their name from the fabrication method used to manufacture the devices from different families are not readily compatible in the same circuit. The more common types of logic integrated circuits are typically represented in each family of devices like TTL, Schottky TTL, CMOS and the new high speed CMOS. The CMOS family devices have a very low power consumption that makes them very popular for many applications where very high speeds are not required.

c) Computer integrated circuits

Computer integrated circuits are devices, which form the active components of a computer system. They are often used in conjunction with digital integrated circuits, which provide a 'glue logic' function. Computer integrated circuits can be functionally divided into microprocessors, memory devices and peripheral control devices.

HISTORY OF IC's:

The first silicon chip or integrated circuit consisting of many transistors fabricated on the same piece of silicon was made at Fairchild in 1959. More recent developments have been towards miniaturization-packing more and more active components or gates on to a single chip of silicon. The level of device complexity is usually referred to as a scale of integration. The evolution from scale integration (SSI), through large-scale integration (LSI), to very large scale integration (VLSI) has already occurred, and the scale is running out of adjectives. The scale of integration is based on the number of logic elements that constitute a device.

INTRODUCTION TO VLSI:

VLSI stands for "Very Large Scale Integrated Circuits". It's a classification of ICs. An IC of common VLSI includes about millions active devices. Typical functions of VLSI include Memories, computers, and signal processors, etc. A semiconductor process technology is a method by which working circuits can be manufactured from designed specifications. There are many such technologies, each of which creates a different environment or style of design. In integrated circuit design, the specification consists of polygons of conducting and semiconducting material that will be layered on top of each other to produce a working chip. When a chip is custom-designed for a specific use, it is called an application-specific integrated circuit (ASIC). Printed-circuit (PC) design also results in precise positions of conducting materials, as they will appear on a circuit board; in addition, PC design aggregates the bulk of the electronic activity into standard IC packages, the position and interconnection of which are essential to the final circuit. Printed circuitry may be easier to debug than integrated circuitry is, but it is slower, less compact, more expensive, and unable to take advantage of specialized silicon layout structures that make VLSI systems so attractive. The design of these electronic circuits can be achieved at many different refinement levels from the most detailed layout to

the most abstract architectures. Given the complexity that is demanded at all levels, computers are increasingly used to aid this design at each step. It is no longer reasonable to use manual design techniques, in which each layer is hand etched or composed by laying tape on film. Thus the term computer-aided design or CAD is a most accurate description of this modern way and seems more broad in its scope than the recently popular term computer-aided engineering (CAE)

APPLICATION AREAS OF VLSI:

PLAs:

Combinational circuit elements are an important part of any digital design. Three common methods of implementing a combinational block are random logic, read-only memory (ROM), and programmable logic array (PLA). In random-logic designs, the logic description of the circuit is directly translated into hardware structures such as AND and OR gates. The PLA occupies less area on the silicon due to reduced interconnection wire space; however, it may be slower than purely random logic. A PLA can also be used as a compact finite state machine by feeding back part of its outputs to the inputs and clocking both sides. Normally, for high-speed applications, the PLA is not implemented as two NOR arrays. The inputs and outputs are inverted to preserve the AND-OR structure.

Gate-Arrays:

The gate-array is a popular technique used to design IC chips. Like the PLA, it contains a fixed mesh of unfinished layout that must be customized to yield the final circuit. Gate-arrays are more powerful, however, because the contents of the mesh are less structured so the interconnection options are more flexible. Gate-arrays exist in many forms with many names, eg: uncommitted logic arrays and master-slice. The disadvantage of gate-arrays is that they are not optimal for any task.

Gate Matrices:

The gate matrix is the next step in the evolution of automatically generated layout from high-level specification. Like the PLA, this layout has no fixed size; a gate matrix grows according to its complexity. Like all regular forms of layout, this one has its fixed aspects and its customizable aspects. In gate matrix layout the fixed design consists of vertical columns of polysilicon gating material. The customizable part is the metal and diffusion wires that run horizontally to interconnect and form gates with the columns.

APPLICATIONS OF VLSI

Electronic systems now perform a wide variety of tasks in daily life. Electronic systems in some cases have replaced mechanisms that operated mechanically, hydraulically, or by other means; electronics are usually smaller, more flexible, and easier to service. In other cases electronic systems have created totally new applications. Electronic systems perform a variety of tasks, some of them visible, some more hidden:

- Personal entertainment systems such as portable MP3 players and DVD players perform sophisticated algorithms with remarkably little energy.
- Electronic systems in cars operate stereo systems and displays; they also control fuel injection systems, adjust suspensions to varying terrain, and perform the control functions required for anti-lock braking (ABS) systems.
- Digital electronics compress and decompress video, even at high definition data rates, on-the-fly in consumer electronics.
- Low-cost terminals for Web browsing still require sophisticated electronics, despite their dedicated function.
- Personal computers and workstations provide word-processing, financial analysis, and games. Computers include both central processing units (CPUs) and special-purpose hardware for disk access, faster screen display, etc.

ADVANTAGES OF VLSI

While we will concentrate on integrated circuits in this book, the properties of integrated circuits what we can and cannot efficiently put in an integrated circuit—largely determine the architecture of the entire system. Integrated circuits improve system characteristics in several critical ways. ICs have three key advantages over digital circuits built from discrete components:

• **Size.** Integrated circuits are much smaller—both transistors and wires are shrunk to micrometer sizes, compared to the millimeter or centimeter scales of discrete components. Small size leads to advantages in speed and power consumption, since smaller components have smaller parasitic resistances, capacitances, and inductances.

• **Speed.** Signals can be switched between logic 0 and logic 1 much quicker within a chip than they can between chips. Communication within a chip can occur hundreds of times faster than communication between chips on a printed circuit board. The high speed of circuits on-chip is due to their small size—smaller components and wires have smaller parasitic capacitances to slow down the signal

• **Power consumption.** Logic operations within a chip also take much less power. Once again, lower power consumption is largely due to the small size of circuits on the chip—smaller parasitic capacitances and resistances require less power to drive them.

4.2 VLSI AND SYSTEMS

These advantages of integrated circuits translate into advantages at the system level:

• **Smaller physical size**. Smallness is often an advantage in itself—consider portable televisions or handheld cellular telephones.

• **Lower power consumption.** Replacing a handful of standard parts with a single chip reduces total power consumption. Reducing power consumption has a ripple effect on the rest of the system: a smaller, cheaper power supply can be used; since less power consumption means less heat, a fan may no longer be necessary; a simpler cabinet with less shielding for electromagnetic shielding may be feasible, too.

• **Reduced cost.** Reducing the number of components, the power supply requirements, cabinet costs, and so on, will inevitably reduce system cost. The ripple effect of integration is such that the cost of a system built from custom ICs can be less, even though the individual ICs cost more than the standard parts they replace. Understanding why integrated circuit technology has such profound influence on the design of digital systems requires understanding both the technology of IC manufacturing and the economics of ICs and digital systems.

4.3 INTRODUCTION TO ASICS AND PROGRAMMABLE LOGIC:

The last 15 years have witnessed the demise in the number of cell-based ASIC designs as a means for developing customized SoCs. Rising NREs, development times and risk have mostly

restricted the use of cell-based ASICs to the highest volume applications; applications that can withstand the multi-million dollar development costs associated with 1-2 design re-spins. Analysts estimate that the number of cell based ASIC design starts per year is now only between 2000-3000 compared to ~10,000 in the late 1990s. The FPGA has emerged as a technology that fills some of the gap left by cell-based ASICs. Yet even after 20+ years of existence and 40X more design starts per year than cell-based ASICs, the size of the FPGA market in dollar terms remains only a fraction that of cell-based ASICs. This suggests that there are many FPGA designs that never make it into production and that for the most part; the FPGA is still seen by many as a vehicle for prototyping or college education and has perhaps even succeeded in actually stifling industry innovation. This paper introduces a new technology, the second generation Structured ASIC that is tipped to reenergize the path to innovation within the electronics industry. It brings together some of the key advantages of FPGA technology (i.e. fast turnaround, no mask charges, no minimum order quantity) and of cell-based ASIC (i.e. low unit cost and power) to deliver a new platform for SoC design. This document defines requirements for development of Application Specific Integrated Circuits (ASICs). It is intended to be used as an appendix to a Statement of Work. The document complements the ESA ASIC Design and Assurance Requirements (AD1), which is a precursor to a future ESA PSS document on ASIC design.

4.3.1 Moore's Law

In the 1960s Gordon Moore predicted that the number of transistors that could be manufactured on a chip would grow exponentially. His prediction, now known as **Moore's Law**, was remarkably prescient. Moore's ultimate prediction was that transistor count would double every two years, an estimate that has held up remarkably well. Today, an industry group maintains the International Technology Roadmap for Semiconductors (ITRS), that maps out strategies to maintain the pace of Moore's Law. (The ITRS roadmap can be found at http://www.itrs.net.)

4.3.2 APPLICATIONS FOR NEXTREME STRUCTURED ASICS:

Embedded Processing

Nextreme Structured ASICs are ideally suited for embedded processing applications. The availability of a firm, 150MHz ARM926EJT™ processor and AMBA peripherals backed by industry

standard development tools from ARM and its Connected Community™ partners, designers have the option to implement control circuits in software. A major benefit of using Nextreme for implementing embedded systems is that designers are able to make performance, area and feature tradeoffs using both hardware and software allowing for highly differentiated yet cost-optimized systems.

Signal, Video and Image Processing

Having to deal with programmable metal interconnect and its associated carry chain delays ultimately forced FPGA vendors to develop dedicated DSP blocks and slices to overcome performance bottlenecks. With Nextreme Structured ASICs, the elimination of massive amounts of metal interconnect means that these devices are not subject to unacceptable carry chain delays and many signal processing structured can be implemented, at speed, using logic fabric alone. Another capability with in Nextreme that makes them particularly suitable for signal processing is memories. eRAM blocks are particularly suited for distributed applications such as semi-parallel filters and video processing. As these blocks are located very close together, they can be connected to form larger blocks up to 4Kbits per eUnit.

4.3.3 FIELD-PROGRAMMABLE GATE ARRAY (FPGA)

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC) (circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design and the low non-recurring engineering costs relative to an ASIC design (notwithstanding the generally higher unit cost), offer advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational

functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow. Another relatively common analog feature is differential comparators on input pins designed to be connected to differential signaling channels. A few "mixed signal FPGAs" have integrated peripheral Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system-on-a-chip. Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA), which carries analog values on its internal programmable interconnect fabric.

Definitions of Relevant Terminology

The most important terminology used below.

Field-Programmable Device (FPD)

A general term that refers to any type of integrated circuit used for implementing digital hardware, where the chip can be configured by the end user to realize different designs. Programming of such a device often involves placing the chip into a special programming unit, but some chips can also be configured “in-system”. Another name for FPDs is programmable logic devices (PLDs); although PLDs encompass the same types of chips as FPDs, we prefer the term FPD because historically the word PLD has referred to relatively simple types of devices.

Programmable Logic Array (PLA)

A Programmable Logic Array (PLA) is a relatively small FPD that contains two levels of logic, an AND-plane and an OR-plane, where both levels are programmable (note: although PLA structures are sometimes embedded into full-custom chips, we refer here only to those PLAs that are provided as separate integrated circuits and are user-programmable).

Programmable Array Logic (PAL)

A Programmable Array Logic (PAL) is a relatively small FPD that has a programmable AND-plane followed by a fixed OR-plane.

Simple PLD

Refers to any type of Simple PLD, usually either a PLA or PAL.

Complex PLD

A more Complex PLD that consists of an arrangement of multiple SPLD-like blocks on a single chip. Alternative names (that will not be used in this paper) sometimes adopted for this style of chip are Enhanced PLD (EPLD), Super PAL, Mega PAL, and others.

Field-Programmable Gate Array (FPGA)

A Field-Programmable Gate Array is an FPD featuring a general structure that allows very high logic capacity. Whereas CPLDs feature logic resources with a wide number of inputs (AND planes), FPGAs offer more narrow logic resources. FPGAs also offer a higher ratio of flip-flops to logic resources than do CPLDs.

High-Capacity PLDs (HCPLD):

high-capacity PLDs: a single acronym that refers to both CPLDs and FPGAs. This term has been coined in trade literature for providing an easy way to refer to both types of devices. PAL is a trademark of Advanced Micro Devices.

- **Interconnect -** the wiring resources in an FPD.
- **Programmable Switch-** a user-programmable switch that can connect a logic element to an interconnect wire, or one interconnect wire to another
- **Logic Block** - a relatively small circuit block that is replicated in an array in an FPD. When a circuit is implemented in an FPD, it is first decomposed into smaller sub-circuits that can each be mapped into a logic block. The term logic block is mostly used in the context of FPGAs, but it could also refer to a block of circuitry in a CPLD.
- **Logic Capacity** - the amount of digital logic that can be mapped into a single FPD. This is usually measured in units of “equivalent number of gates in a traditional gate array”. In other words, the

capacity of an FPD is measured by the size of gate array that it is comparable to. In simpler terms, logic capacity can be thought of as "number of 2-input NAND gates".

- **Logic Density** - the amount of logic per unit area in an FPD.
- **Speed-Performance** - measures the maximum operable speed of a circuit when implemented in an FPD. For combinational circuits, it is set by the longest delay through any path, and for sequential circuits it is the maximum clock frequency for which the circuit functions properly. In the remainder of this section, to provide insight into FPD development the evolution of FPDs over the past two decades is described. Additional background information is also included on the semiconductor technologies used in the manufacture of FPDs.

Evolution of Programmable Logic Devices:

The first type of user-programmable chip that could implement logic circuits was the Programmable Read-Only Memory (PROM), in which address lines can be used as logic circuit inputs and data lines as outputs. Logic functions, however, rarely require more than a few product terms, and a PROM contains a full decoder for its address inputs. PROMS are thus an inefficient architecture for realizing logic circuits, and so are rarely used in practice for that purpose. The first device developed later specifically for implementing logic circuits was the Field-Programmable Logic Array (FPLA), or simply PLA for short. A PLA consists of two levels of logic gates: a programmable "wired" AND-plane followed by a programmable "wired" OR-plane. A PLA is structured so that any of its inputs (or their complements) can be AND'ed together in the AND-plane; each AND-plane output can thus correspond to any product term of the inputs. Similarly, each OR plane output can be configured to produce the logical sum of any of the AND-plane outputs. With this structure, PLAs are well-suited for implementing logic functions in sum-of-products form. They are also quite versatile, since both the AND terms and OR terms can have many inputs (this feature is often referred to as wide AND and OR gates). When PLAs were introduced in the early 1970s, by Philips, their main drawbacks were that they were expensive to manufacture and offered somewhat poor speed-performance.

Both disadvantages were due to the two levels of configurable logic, because programmable logic planes were difficult to manufacture and introduced significant propagation delays. To overcome these weaknesses, Programmable Array Logic (PAL) devices were developed. PALs feature only a single level of programmability, consisting of a programmable "wired" AND plane that feeds fixed OR-gates. To compensate for lack of generality incurred because the OR- Outputs plane is fixed, several variants of PALs are produced, with different numbers of inputs and outputs, and various sizes of OR-

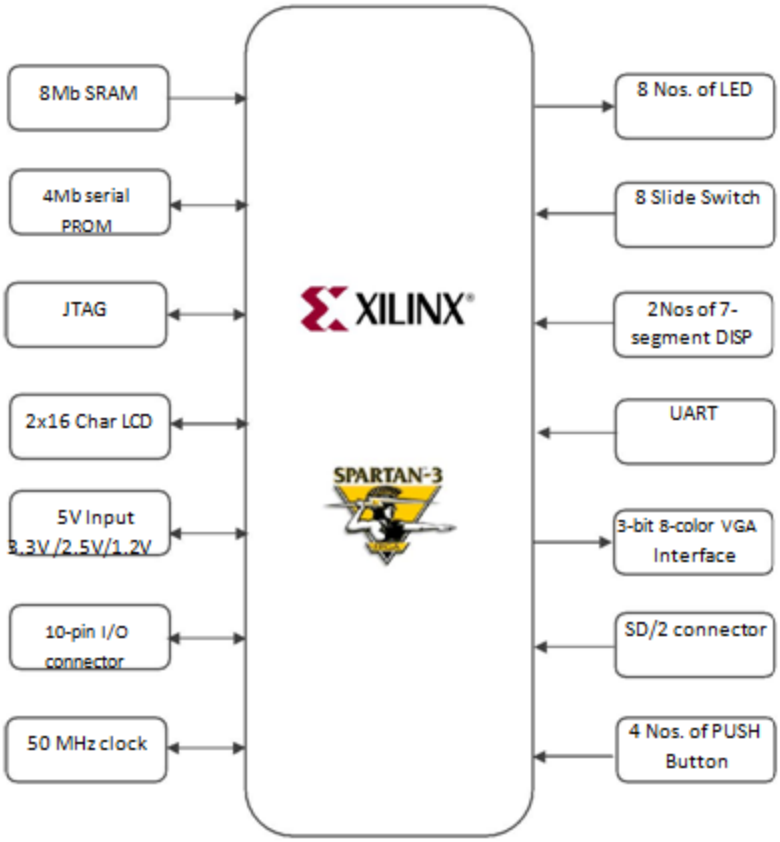
gates. PALs usually contain flip-flops connected to the OR-gate outputs so that sequential circuits can be realized.

SPARTAN 3 XC3S 200 TQ 144

The Spartan-3 EDK Board provides a powerful, self-contained development platform for designs targeting the new Spartan-3 FPGA from Xilinx. It features a 200K gate Spartan-3, on-board I/O devices, and 1MB fast asynchronous SRAM, making it the perfect platform to experiment with any new design, from a simple logic circuit to an embedded processor core. The board also contains a Platform Flash JTAG-programmable ROM, so designs can easily be made non-volatile.

Components placement

Figure 1. SD – TYRO PLUS SPARTAN3 (EDK) Board Components placement top view



Power Distribution

AC Wall Adapter

The Spartan3FPGA Lab Kit includes an international-ready AC wall adapter that produces a +5V DC output. Connect the AC wall adapter to the barrel connector along the left edge of the board, indicated as in Figure 3. To disconnect power, switch off the power switch. The power indicator LED, as shown in Figure 3, lights up when power is properly applied to the board. The AC wall adapter operates from 100V to 240V AC input, at 50 or 60 Hz.

Voltage Regulators

There are Overall, the 5V DC switching power adapter that connects to AC wall power powers the board. A 3.3V regulator, powered by the 5V DC supply, provides power to the inputs of the 2.5V and 1.2V regulators. Similarly, the 3.3V regulator feeds all the VCCO voltage supply inputs to the FPGA's I/O banks and powers most of the components on the board. The 2.5V regulator supplies power to the FPGA's VCCAUX supply inputs. The VCCAUX voltage input supplies power to Digital Clock Managers (DCMs) within the FPGA and supplies some of the I/O structures. In specific, all of the FPGA's dedicated configuration pins, such as DONE, PROG_B, CCLK, and the FPGA's JTAG pins, are powered by VCCAUX. The FPGA configuration interface on the board is powered by 3.3V. Consequently, the 2.5V supply has a current shunt resistor to prevent

reverse current. Finally, a 1.2V regulator supplies power to the FPGA's VCCINT voltage inputs, which power the FPGA's core logic. The board uses three discrete regulators to generate the necessary oltages. However, various power supply vendors are developing integrated solutions specifically for Spartan-3 FPGAs.

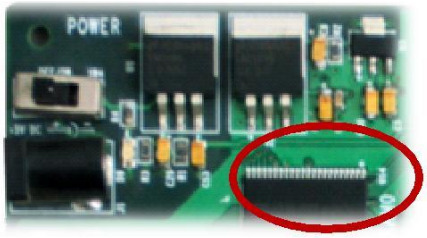


Figure 3. Power Supply

On-board Peripherals

The Spartan3FPGA Lab Kit comes with many interfacing options

2 Nos. of Seven-segment display

8-Nos. of Toggle switches (Digital Inputs)

4-Nos. of Push Button (Digital Inputs)

8-Nos. of Point LED's (Digital Outputs)

2x16 Character LCD

UART for serial port communication through PC

SD/2 keyboard Interface

3-Bit VGA Interface

Seven Segment Display

The Spartan3 FPGA Kit has a two-character, seven-segment LED display controlled by FPGA user-I/O pins, as shown in Figure 4. Each digit shares eight common control signals to light individual LED segments. Each individual character has a separate anode control input. The pin number for each FPGA pin connected to the LED display is shown in Table 1. To light an individual signal, drive the individual segment control signal Low along with the associated anode control signal for the individual character.

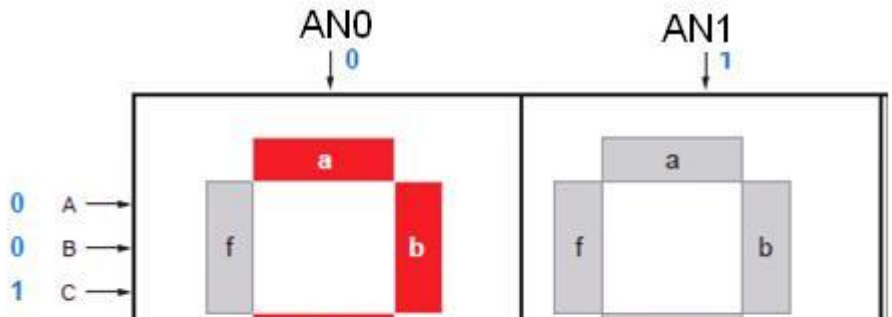


Figure 4. Seven-segment display connections from Spartan3FPGA Lab Kit

Table 1. Seven-segment display connections to the FPGA pins

Segment	FPGA PIN
A	P82
B	P83
C	P84
D	P85
E	P86
F	P87
G	P89
DP	P90

Table 2. Digit Enable (Anode Control) Signals (Active Low)

Anode Control	FPGA PIN
AN1	P76
AN0	P77

The LED control signals are time-multiplexed to display data on two characters. Present the value to be displayed on the segment control inputs and select the specified character by driving the associated anode control signal Low. Through persistence of vision, the human brain perceives that all four characters appear simultaneously, similar to the way the brain perceives a TV display.

This "scanning" technique reduces the number of I/O pins required for the four

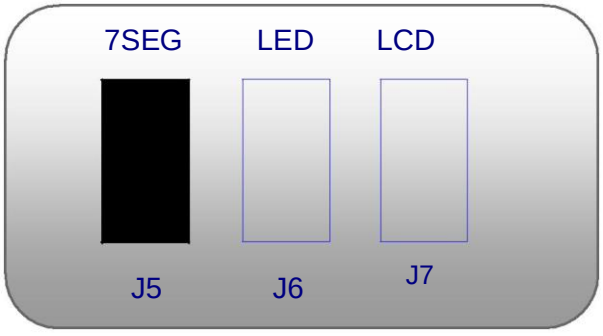
characters. In case an FPGA pin were dedicated for each individual segment, then 32 pins are required to drive four 7-segment LED characters. The scanning technique reduces the required I/O down to 12 pins. The drawback to this approach is that the FPGA logic must continuously scan data out to the displays—a small price to save 20 additional I/O pins.

Table 3. Display Characters and Resulting LED Segment Control Values

Character	a	b	c	d	e	F	g
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
B	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
D	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

Hardware Settings

Place Jumper at J5 (7SEG) to enable supply to seven segment display.



Digital Inputs Toggle Switch

The Spartan3FPGA Kit has eight slide switches, indicated as in Figure 5. The switches

connect to an associated FPGA pin, as shown in Table 4 Error! Reference source not found.**. A detailed schematic appears in Figure 5.**

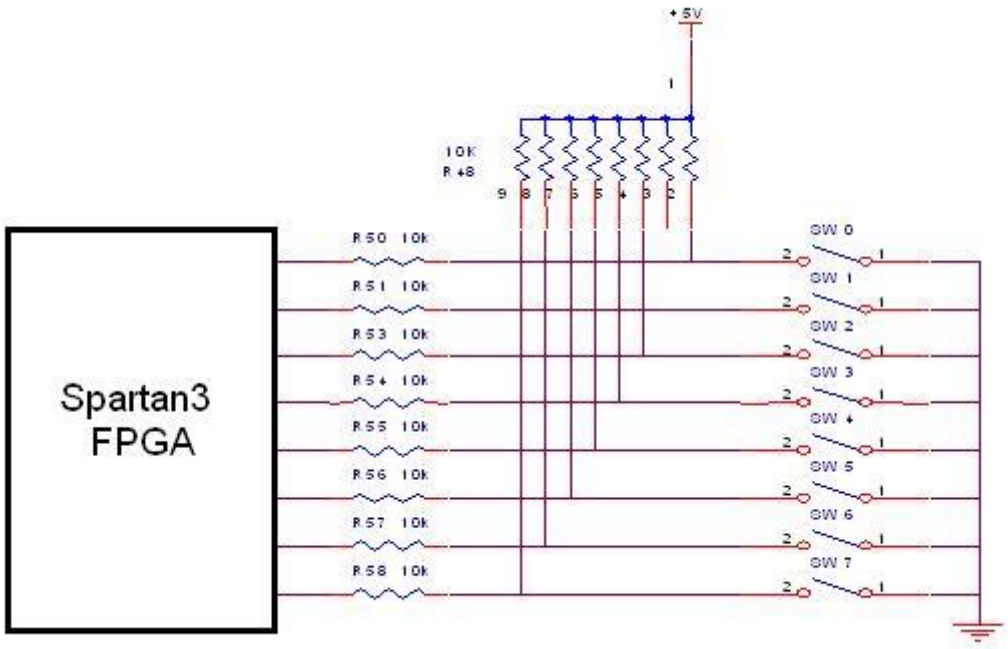


Figure 5. Slide switches connections from Spartan3FPGA Lab Kit

Table 4. FPGA Connections to Slide Switches

Switch	1	2	3	4	5	6	7	8
FPGA pin	P99	P100	P102	P103	P104	P105	P107	P108

When in the UP or ON position, a switch connects the FPGA pin to VCCO, a logic High. When DOWN or in the OFF position, the switch connects the FPGA pin to ground, a logic Low. The switches typically exhibit about 2 ms of mechanical bounce and there is no active debouncing circuitry, although such circuitry could easily be added to the FPGA design programmed on the board. A 10KΩ series resistor provides nominal input protection.

Light Emitting Diodes

Light Emitting Diodes (LEDs) are the most commonly used components, usually for displaying pin's digital states. The Spartan3FPGA Lab Kit has eight LEDs located above the push button switches, indicated by in Figure .

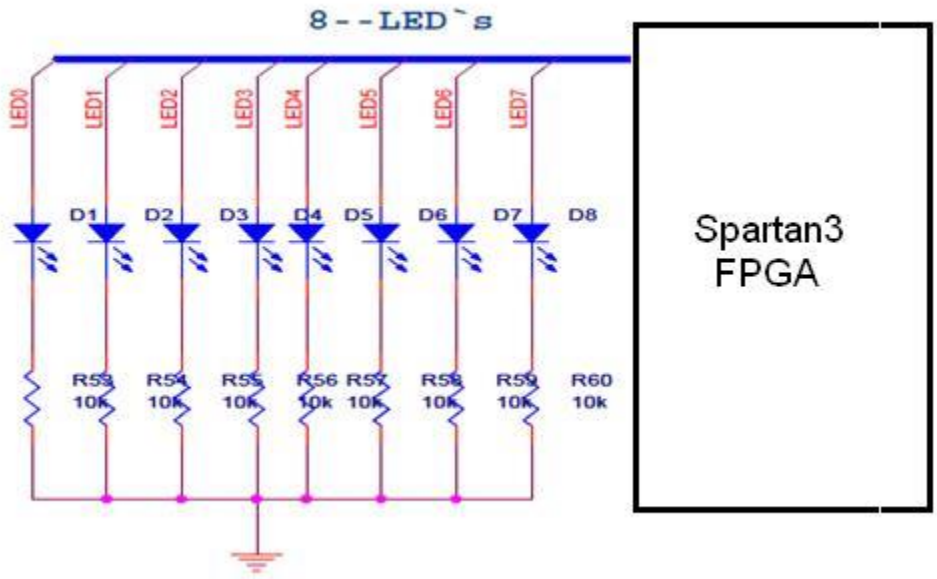


Figure 6. Point LED interface from Spartan3FPGA Lab Kit

Table 5. FPGA connections to the LEDs

LED	D1	D2	D3	D4	D5	D6	D7	D8
FPGA pin	P82	P83	P84	P85	P86	P87	P89	P90

The cathode of each LED connects to ground via a 220 ohm Ω resistor. To light an individual LED, drive the associated FPGA control signal High, which is the opposite polarity from lighting one of the 7-segment LEDs.

CHAPTER 5

SOFTWARE REQUIREMENT

VERIFICATION TOOL

- Modelsim 6.4a

SYNTHESIS TOOL

- Xilinx ISE 9.1/ Xilinx 13.2

5.1 MODELSIM

Modelsim SE - High Performance Simulation and Debug

ModelSim SE is our UNIX, Linux, and Windows-based simulation and debug environment, combining high performance with the most powerful and intuitive GUI in the industry.

What's New in ModelSim SE?

- Improved FSM debug options including control of basic information, transition table and warning messages. Added support of FSM Multi-state transitions coverage (i.e. coverage for all possible FSM state sequences).
- Improved debugging with hyperlinked navigation between objects and their declaration, and between visited source files.
- The dataflow window can now compute and display all paths from one net to another.
- Enhanced code coverage data management with fine grain control of information in the source window.
- Toggle coverage has been enhanced to support SystemVerilog types: structures, packed unions, fixed-size multi-dimensional arrays and real.
- Some IEEE VHDL 2008 features are supported including source code encryption. Added support of new VPI types, including packed arrays of struct nets and variables.

ModelSim SE Features:

- Multi-language, high performance simulation engine
- Verilog, VHDL, SystemVerilog Design
- Code Coverage
- SystemVerilog for Design
- Integrated debug
- JobSpy Regression Monitor
- Mixed HDL simulation option
- SystemC Option
- TCL/tk
- Solaris and Linux 32 & 64-bit
- Windows 32-bit

ModelSim SE Benefits:

- High performance HDL simulation solution for FPGA & ASIC design teams
- The best mixed-language environment and performance in the industry
- Intuitive GUI for efficient interactive or post-simulation debug of RTL and gate-level designs
- Merging, ranking and reporting of code coverage for tracking verification progress
- Sign-off support for popular ASIC libraries
- All ModelSim products are 100% standards based. This means your investment is protected, risk is lowered, reuse is enabled, and productivity is enhanced
- Award-winning technical support

High-Performance, Scalable Simulation Environment:

ModelSim provides seamless, scalable performance and capabilities. Through the use of a single compiler and library system for all ModelSim configurations, employing the right ModelSim configuration for project needs is as simple as pointing your environment to the appropriate installation directory.

ModelSim also supports very fast time-tenet-simulation turnarounds while maintaining high performance with its new black box use model, known as bbox. With bbox, non-changing elements can be compiled and optimized once and reused when running a modified version of the test bench. bbox delivers dramatic throughput improvements of up to 3X when running a large suite of test cases.

Easy-to-Use Simulation Environment:

An intelligently engineered graphical user interface (GUI) efficiently displays design data for analysis and debug. The default configuration of windows and information is designed to meet the needs of most users. However, the flexibility of the ModelSim SE GUI allows users to easily customize it to their preferences. The result is a feature-rich GUI that is easy to use and quickly mastered.

A message viewer enables simulation messages to be logged to the ModelSim results file in addition to the standard transcript file. The GUI's organizational and filtering capabilities allow design and simulation information to be quickly reduced to focus on areas of interest, such as possible causes of design bugs.

ModelSim SE allows many debug and analysis capabilities to be employed post-simulation on saved results, as well as during live simulation runs. For example, the coverage viewer analyzes and annotates source code with code coverage results, including FSM state and transition, statement, expression, branch, and toggle coverage. Signal values can be annotated in the source window and viewed in the waveform viewer. Race conditions, delta, and event activity can be analyzed in the list and wave windows. User-defined enumeration values can be easily defined for quicker understanding of simulation results. For improved debug productivity, ModelSim also has graphical and textual dataflow capabilities. The memory window identifies memories in the design and accommodates flexible viewing and modification of the memory contents. Powerful search, fill, load, and save functionalities are supported. The memory window allows memories to be pre-loaded with specific or randomly generated values, saving the time-consuming step of initializing sections of the simulation merely to load memories. All functions are available via the command line, so they can be used in scripting.

Advanced Code Coverage

The ModelSim advanced code coverage capabilities deliver high performance with ease of use. Most simulation optimizations remain enabled with code coverage. Code coverage metrics can be reported by-instance or by-design unit, providing flexibility in managing coverage data. All coverage information is now stored in the Unified Coverage Database (UCDB), which is used to collect and manage all coverage information in one highly efficient database. Coverage utilities that analyze code coverage data, such as merging and test ranking, are available.

The coverage types supported include:

- Statement coverage: number of statements executed during a run
- Branch coverage: expressions and case statements that affect the control flow of the HDL execution
- Condition coverage: breaks down the condition on a branch into elements that make the result true or false
- Expression coverage: the same as condition coverage, but covers concurrent signal assignments instead of branch decisions
- Focused expression coverage: presents expression coverage data in a manner that accounts for each independent input to the expression in determining coverage results
- Enhanced toggle coverage: in default mode, counts low-to-high and high-to-low transitions; in extended mode, counts transitions to and from X
- Finite State Machine coverage: state and state transition coverage

SYNTHESIS TOOL:

5.2 XILINX ISE

INTRODUCTION

For two-and-a-half decades, Xilinx has been at the forefront of the programmable logic revolution, with the invention and continued migration of FPGA platform technology. During that

time, the role of the FPGA has evolved from a vehicle for prototyping and glue-logic to a highly flexible alternative to ASICs and ASSPs for a host of applications and markets. Today, Xilinx® FPGAs have become strategically essential to world-class system companies that are hoping to survive and compete in these times of extreme global economic instability, turning what was once the programmable revolution into the "programmable imperative" for both Xilinx and our customers.

Programmable Imperative

When viewed from the customer's perspective, the programmable imperative is the necessity to do more with less, to remove risk wherever possible, and to differentiate in order to survive. In essence, it is the quest to simultaneously satisfy the conflicting demands created by ever-evolving product requirements (i.e., cost, power, performance, and density) and mounting business challenges (i.e., shrinking market windows, fickle market demands, capped engineering budgets, escalating ASIC and ASSP non-recurring engineering costs, spiraling complexity, and increased risk). To Xilinx, the programmable imperative represents a two-fold commitment. The first is to continue developing programmable silicon innovations at every process node that deliver industry-leading value for every key figure of merit against which FPGAs are measured: price, power, performance, density, features, and programmability. The second commitment is to provide customers with simpler, smarter, and more strategically viable design platforms for the creation of world-class FPGA-based solutions in a wide variety of industries—what Xilinx calls targeted design platforms.

Base Platform

The base platform is both the delivery vehicle for all new silicon offerings from Xilinx and the foundation upon which all Xilinx targeted design platforms are built. As such, it is the most fundamental platform used to develop and run customer-specific software applications and hardware designs as production system solutions. Released at launch, the base platform comprises a robust set of well-integrated, tested, and targeted elements that enable customers to immediately start a design. These elements include:

• FPGA silicon

• ISE® Design Suite design environment

• Third-party synthesis, simulation, and signal integrity tools

• Reference designs common to many applications, such as memory interface and configuration designs.

• Development boards that run the reference designs

• A host of widely used IP, such as GigE, Ethernet, memory controllers, and PCIe.

Domain-Specific Platform

The next layer in the targeted design platform hierarchy is the domain-specific platform. Released from three to six months after the base platform, each domain specific platform targets one of the three primary Xilinx FPGA user profiles (domains):the embedded processing developer, the digital signal processing (DSP) developer, or the logic/connectivity developer. This is where the real power and intent of the targeted design platform begins to emerge. Domain-specific platforms augment the base platform with a predictable, reliable, and intelligently targeted set of integrated technologies, including:

• Higher-level design methodologies and tools

• Domain-specific embedded, DSP, and connectivity IP

• Domain-specific development hardware and daughter cards

• Reference designs optimized for embedded processing, connectivity, and DSP

• Operating systems (required for embedded processing) and software

Every element in these platforms is tested, targeted, and supported by Xilinx and/or our ecosystem partners. Starting a design with the appropriate domain-specific platform can cut weeks, if not months, off of the user's development time.

Market-Specific Platform

A market-specific platform is an integrated combination of technologies that enables software or hardware developers to quickly build and then run their specific application or solution. Built for use in specific markets such as Automotive, Consumer, Mil/Aero, Communications, AVB, or ISM,

market-specific platforms integrate both the base and domain-specific platforms and provide higher level elements that can be leveraged by customer-specific software and hardware designs. The market-specific platform can rely more heavily on third-party targeted IP than the base or domain-specific platforms. The market-specific platform includes: the base and domain-specific platforms, reference designs, and boards (or daughter cards) to run reference designs that are optimized for a particular market (e.g., lane departure early-warning systems, analytics, and display processing).Xilinx will begin releasing market-specific platforms three to six months after the domain-specific platforms, augmenting the domain-specific platforms with reference designs, IP, and software aimed at key growth markets. Initially, Xilinx will target markets such as Communications, Automotive, Video, and Displays with platform elements that abstract away the more mundane portions of the design, thereby further reducing the customer's development effort so they can focus their attention on creating differentiated value in their end solution. This systematic platform development and release strategy provides the framework for the consistent and efficient fulfillment of the programmable imperative—both by Xilinx and by its customers.

Platform Enablers

Xilinx has instituted a number of changes and enhancements that have contributed substantially to the feasibility and viability of the targeted design platform. These platform-enabling changes cover six primary areas:

• Design environment enhancements

• Socket able IP creation

• New targeted reference designs

• Scalable unified board and kit strategy

• Ecosystem expansion

• Design services supporting the targeted design platform approach

Design Environment Enhancements

With the breadth of advances and capabilities that the Virtex®-6 and Spartan®-6 programmable devices deliver coupled with the access provided by the associated targeted design platforms, it is no longer feasible for one design flow or environment to fit every designer's needs. System designers, algorithm designers, SW coders, and logic designers each represent a different user-profile, with unique requirements for a design methodology and associated design environment. Instead of addressing the problem in terms of individual fixed tools, Xilinx targets the required or preferred methodology for each user, to address their specific needs with the appropriate design flow. At this level, the design language changes from HDL (VHDL/Verilog) to C, C++, MATLAB® software, and other higher level languages which are more widely used by these designers, and the design abstraction moves up from the block or component to the system level. The result is a methodology and complete design flow tailored to each user profile that provides design creation, design implementation, and design verification. Indicative of the complexity of the problem, to fully understand the user profile of a "logic designer," one must consider the various levels of expertise represented by this demographic. The most basic category in this profile is the "push-button user" who wants to complete a design with minimum work or knowledge.

The push-button user just needs "good-enough" results. Contrastingly, more advanced users want some level of interactive capabilities to squeeze more value into their design, and the "power user" (the expert) wants full control over a vast array of variables. Add the traditional ASIC designers, tasked with migrating their designs to an FPGA (a growing trend, given the intolerable costs and risks posed by ASIC development these days), and clearly the imperative facing Xilinx is to offer targeted flows and tools that support each user's requirements and capabilities, on their terms. The most recent release of the ISE Design Suite includes numerous changes that fulfil requirements specifically pertinent to the targeted design platform. The new release features a complete tool chain for each top-level user profile (the domain-specific personas: the embedded, DSP, and logic/connectivity designers), including specific accommodations for everyone from the push-button user to the ASIC designer.

The tighter integration of embedded and DSP flows enables more seamless integration of designs that contain embedded, DSP, IP, and user blocks in one system. To further enhance productivity and help customers better manage the complexity of their designs, the new ISE Design Suite enables designers to target area, performance, or power by simply selecting a design goal in the setup. The tools then apply specific optimizations to help meet the design goal. In addition, the ISE Design Suite boasts substantially faster place-and-route and simulation run times, providing users with 2X faster compile times. Finally, Xilinx has adopted the FLEXnet Licensing strategy that provides a floating license to track and monitor usage.

XILINX ISE Design Tools:

Xilinx ISE is the design tool provided by Xilinx. Xilinx would be virtually identical for our purposes.

There are four fundamental steps in all digital logic design. These consist of:

1. Design – The schematic or code that describes the circuit.
2. Synthesis – The intermediate conversion of human readable circuit description to FPGA code (EDIF) format. It involves syntax checking and combining of all the separate design files into a single file.
3. Place & Route – Where the layout of the circuit is finalized. This is the translation of the EDIF into logic gates on the FPGA.
4. Program – The FPGA is updated to reflect the design through the use of programming (.bit) files.

Test bench simulation is in the second step. As its name implies, it is used for testing the design by simulating the result of driving the inputs and observing the outputs to verify your design.

ISE has the capability to do a variety of different design methodologies including: Schematic Capture, Finite State Machine and Hardware Descriptive Language (VHDL or Verilog).

CHAPTER 5

SOFTWARE REQUIREMENT

VERIFICATION TOOL

- Modelsim 6.4a

SYNTHESIS TOOL

- Xilinx ISE 9.1/ Xilinx 13.2

5.1 MODELSIM

Modelsim SE - High Performance Simulation and Debug

ModelSim SE is our UNIX, Linux, and Windows-based simulation and debug environment, combining high performance with the most powerful and intuitive GUI in the industry.

What's New in ModelSim SE?

- Improved FSM debug options including control of basic information, transition table and warning messages. Added support of FSM Multi-state transitions coverage (i.e. coverage for all possible FSM state sequences).
- Improved debugging with hyperlinked navigation between objects and their declaration, and between visited source files.
- The dataflow window can now compute and display all paths from one net to another.
- Enhanced code coverage data management with fine grain control of information in the source window.
- Toggle coverage has been enhanced to support SystemVerilog types: structures, packed unions, fixed-size multi-dimensional arrays and real.
- Some IEEE VHDL 2008 features are supported including source code encryption. Added support of new VPI types, including packed arrays of struct nets and variables.

ModelSim SE Features:

- Multi-language, high performance simulation engine
- Verilog, VHDL, SystemVerilog Design
- Code Coverage
- SystemVerilog for Design
- Integrated debug
- JobSpy Regression Monitor
- Mixed HDL simulation option
- SystemC Option
- TCL/tk
- Solaris and Linux 32 & 64-bit
- Windows 32-bit

ModelSim SE Benefits:

- High performance HDL simulation solution for FPGA & ASIC design teams
- The best mixed-language environment and performance in the industry
- Intuitive GUI for efficient interactive or post-simulation debug of RTL and gate-level designs
- Merging, ranking and reporting of code coverage for tracking verification progress
- Sign-off support for popular ASIC libraries
- All ModelSim products are 100% standards based. This means your investment is protected, risk is lowered, reuse is enabled, and productivity is enhanced
- Award-winning technical support

High-Performance, Scalable Simulation Environment:

ModelSim provides seamless, scalable performance and capabilities. Through the use of a single compiler and library system for all ModelSim configurations, employing the right ModelSim configuration for project needs is as simple as pointing your environment to the appropriate installation directory.

ModelSim also supports very fast time-tenet-simulation turnarounds while maintaining high performance with its new black box use model, known as bbox. With bbox, non-changing elements

can be compiled and optimized once and reused when running a modified version of the test bench. bbox delivers dramatic throughput improvements of up to 3X when running a large suite of test cases.

Easy-to-Use Simulation Environment:

An intelligently engineered graphical user interface (GUI) efficiently displays design data for analysis and debug. The default configuration of windows and information is designed to meet the needs of most users. However, the flexibility of the ModelSim SE GUI allows users to easily customize it to their preferences. The result is a feature-rich GUI that is easy to use and quickly mastered.

A message viewer enables simulation messages to be logged to the ModelSim results file in addition to the standard transcript file. The GUI's organizational and filtering capabilities allow design and simulation information to be quickly reduced to focus on areas of interest, such as possible causes of design bugs.

ModelSim SE allows many debug and analysis capabilities to be employed post-simulation on saved results, as well as during live simulation runs. For example, the coverage viewer analyzes and annotates source code with code coverage results, including FSM state and transition, statement, expression, branch, and toggle coverage. Signal values can be annotated in the source window and viewed in the waveform viewer. Race conditions, delta, and event activity can be analyzed in the list and wave windows. User-defined enumeration values can be easily defined for quicker understanding of simulation results. For improved debug productivity, ModelSim also has graphical and textual dataflow capabilities. The memory window identifies memories in the design and accommodates flexible viewing and modification of the memory contents. Powerful search, fill, load, and save functionalities are supported. The memory window allows memories to be pre-loaded with specific or randomly generated values, saving the time-consuming step of initializing sections of the simulation merely to load memories. All functions are available via the command line, so they can be used in scripting.

Advanced Code Coverage

The ModelSim advanced code coverage capabilities deliver high performance with ease of use. Most simulation optimizations remain enabled with code coverage. Code coverage metrics can be reported by-instance or by-design unit, providing flexibility in managing coverage data. All coverage information is now stored in the Unified Coverage Database (UCDB), which is used to collect and manage all coverage information in one highly efficient database. Coverage utilities that analyze code coverage data, such as merging and test ranking, are available.

The coverage types supported include:

- Statement coverage: number of statements executed during a run
- Branch coverage: expressions and case statements that affect the control flow of the HDL execution
- Condition coverage: breaks down the condition on a branch into elements that make the result true or false
- Expression coverage: the same as condition coverage, but covers concurrent signal assignments instead of branch decisions
- Focused expression coverage: presents expression coverage data in a manner that accounts for each independent input to the expression in determining coverage results
- Enhanced toggle coverage: in default mode, counts low-to-high and high-to-low transitions; in extended mode, counts transitions to and from X
- Finite State Machine coverage: state and state transition coverage

SYNTHESIS TOOL:

5.2 XILINX ISE

INTRODUCTION

For two-and-a-half decades, Xilinx has been at the forefront of the programmable logic revolution, with the invention and continued migration of FPGA platform technology. During that time, the role of the FPGA has evolved from a vehicle for prototyping and glue-logic to a highly flexible alternative to ASICs and ASSPs for a host of applications and markets. Today, Xilinx®

FPGAs have become strategically essential to world-class system companies that are hoping to survive and compete in these times of extreme global economic instability, turning what was once the programmable revolution into the “programmable imperative” for both Xilinx and our customers.

Programmable Imperative

When viewed from the customer's perspective, the programmable imperative is the necessity to do more with less, to remove risk wherever possible, and to differentiate in order to survive. In essence, it is the quest to simultaneously satisfy the conflicting demands created by ever-evolving product requirements (i.e., cost, power, performance, and density) and mounting business challenges (i.e., shrinking market windows, fickle market demands, capped engineering budgets, escalating ASIC and ASSP non-recurring engineering costs, spiraling complexity, and increased risk). To Xilinx, the programmable imperative represents a two-fold commitment. The first is to continue developing programmable silicon innovations at every process node that deliver industry-leading value for every key figure of merit against which FPGAs are measured: price, power, performance, density, features, and programmability. The second commitment is to provide customers with simpler, smarter, and more strategically viable design platforms for the creation of world-class FPGA-based solutions in a wide variety of industries—what Xilinx calls targeted design platforms.

Base Platform

The base platform is both the delivery vehicle for all new silicon offerings from Xilinx and the foundation upon which all Xilinx targeted design platforms are built. As such, it is the most fundamental platform used to develop and run customer-specific software applications and hardware designs as production system solutions. Released at launch, the base platform comprises a robust set of well-integrated, tested, and targeted elements that enable customers to immediately start a design. These elements include:

• FPGA silicon

• ISE® Design Suite design environment

• Third-party synthesis, simulation, and signal integrity tools

• Reference designs common to many applications, such as memory interface and configuration designs.

• Development boards that run the reference designs

• A host of widely used IP, such as GigE, Ethernet, memory controllers, and PCIe.

Domain-Specific Platform

The next layer in the targeted design platform hierarchy is the domain-specific platform. Released from three to six months after the base platform, each domain specific platform targets one of the three primary Xilinx FPGA user profiles (domains):the embedded processing developer, the digital signal processing (DSP) developer, or the logic/connectivity developer. This is where the real power and intent of the targeted design platform begins to emerge. Domain-specific platforms augment the base platform with a predictable, reliable, and intelligently targeted set of integrated technologies, including:

• Higher-level design methodologies and tools

• Domain-specific embedded, DSP, and connectivity IP

• Domain-specific development hardware and daughter cards

• Reference designs optimized for embedded processing, connectivity, and DSP

• Operating systems (required for embedded processing) and software

Every element in these platforms is tested, targeted, and supported by Xilinx and/or our ecosystem partners. Starting a design with the appropriate domain-specific platform can cut weeks, if not months, off of the user's development time.

Market-Specific Platform

A market-specific platform is an integrated combination of technologies that enables software or hardware developers to quickly build and then run their specific application or solution. Built for use in specific markets such as Automotive, Consumer, Mil/Aero, Communications, AVB, or ISM, market-specific platforms integrate both the base and domain-specific platforms and provide higher

level elements that can be leveraged by customer-specific software and hardware designs. The market-specific platform can rely more heavily on third-party targeted IP than the base or domain-specific platforms. The market-specific platform includes: the base and domain-specific platforms, reference designs, and boards (or daughter cards) to run reference designs that are optimized for a particular market (e.g., lane departure early-warning systems, analytics, and display processing).Xilinx will begin releasing market-specific platforms three to six months after the domain-specific platforms, augmenting the domain-specific platforms with reference designs, IP, and software aimed at key growth markets. Initially, Xilinx will target markets such as Communications, Automotive, Video, and Displays with platform elements that abstract away the more mundane portions of the design, thereby further reducing the customer's development effort so they can focus their attention on creating differentiated value in their end solution. This systematic platform development and release strategy provides the framework for the consistent and efficient fulfillment of the programmable imperative—both by Xilinx and by its customers.

Platform Enablers

Xilinx has instituted a number of changes and enhancements that have contributed substantially to the feasibility and viability of the targeted design platform. These platform-enabling changes cover six primary areas:

• Design environment enhancements

• Socket able IP creation

• New targeted reference designs

• Scalable unified board and kit strategy

• Ecosystem expansion

• Design services supporting the targeted design platform approach

Design Environment Enhancements

With the breadth of advances and capabilities that the Virtex®-6 and Spartan®-6 programmable devices deliver coupled with the access provided by the associated targeted design

platforms, it is no longer feasible for one design flow or environment to fit every designer's needs. System designers, algorithm designers, SW coders, and logic designers each represent a different user-profile, with unique requirements for a design methodology and associated design environment. Instead of addressing the problem in terms of individual fixed tools, Xilinx targets the required or preferred methodology for each user, to address their specific needs with the appropriate design flow. At this level, the design language changes from HDL (VHDL/Verilog) to C, C++, MATLAB® software, and other higher level languages which are more widely used by these designers, and the design abstraction moves up from the block or component to the system level. The result is a methodology and complete design flow tailored to each user profile that provides design creation, design implementation, and design verification. Indicative of the complexity of the problem, to fully understand the user profile of a "logic designer," one must consider the various levels of expertise represented by this demographic. The most basic category in this profile is the "push-button user" who wants to complete a design with minimum work or knowledge.

The push-button user just needs "good-enough" results. Contrastingly, more advanced users want some level of interactive capabilities to squeeze more value into their design, and the "power user" (the expert) wants full control over a vast array of variables. Add the traditional ASIC designers, tasked with migrating their designs to an FPGA (a growing trend, given the intolerable costs and risks posed by ASIC development these days), and clearly the imperative facing Xilinx is to offer targeted flows and tools that support each user's requirements and capabilities, on their terms. The most recent release of the ISE Design Suite includes numerous changes that fulfil requirements specifically pertinent to the targeted design platform. The new release features a complete tool chain for each top-level user profile (the domain-specific personas: the embedded, DSP, and logic/connectivity designers), including specific accommodations for everyone from the push-button user to the ASIC designer.

The tighter integration of embedded and DSP flows enables more seamless integration of designs that contain embedded, DSP, IP, and user blocks in one system. To further enhance productivity and help customers better manage the complexity of their designs, the new ISE Design

Suite enables designers to target area, performance, or power by simply selecting a design goal in the setup. The tools then apply specific optimizations to help meet the design goal. In addition, the ISE Design Suite boasts substantially faster place-and-route and simulation run times, providing users with 2X faster compile times. Finally, Xilinx has adopted the FLEXnet Licensing strategy that provides a floating license to track and monitor usage.

XILINX ISE Design Tools:

Xilinx ISE is the design tool provided by Xilinx. Xilinx would be virtually identical for our purposes.

There are four fundamental steps in all digital logic design. These consist of:

1. Design – The schematic or code that describes the circuit.
2. Synthesis – The intermediate conversion of human readable circuit description to FPGA code (EDIF) format. It involves syntax checking and combining of all the separate design files into a single file.
3. Place & Route – Where the layout of the circuit is finalized. This is the translation of the EDIF into logic gates on the FPGA.
4. Program – The FPGA is updated to reflect the design through the use of programming (.bit) files.

Test bench simulation is in the second step. As its name implies, it is used for testing the design by simulating the result of driving the inputs and observing the outputs to verify your design.

ISE has the capability to do a variety of different design methodologies including: Schematic Capture, Finite State Machine and Hardware Descriptive Language (VHDL or Verilog).

CHAPTER 6

SIMULATION IMPLEMENTATION

6.1 GENERAL:

Verilog HDL is a Hardware Description Language (HDL). A Hardware Description Language is a language used to describe a digital system, for example, a computer or a component of a computer. One may describe a digital system at several levels. For example, an HDL might describe the layout of the wires, resistors and transistors on an Integrated Circuit (IC) chip, i. e., the switch level. Or, it might describe the logical gates and flip flops in a digital system, i. e., the gate level. An even higher level describes the registers and the transfers of vectors of information between registers. This is called the Register Transfer Level (RTL). Verilog supports all of these levels. However, this handout focuses on only the portions of Verilog which support the RTL level.

6.2 VERILOG:

Verilog is one of the two major Hardware Description Languages (HDL) used by hardware designers in industry and academia. VHDL is the other one. The industry is currently split on which is better. Many feel that Verilog is easier to learn and use than VHDL. As one hardware designer puts it, "I hope the competition uses VHDL." VHDL was made an IEEE Standard in 1987 and Verilog in 1995. Verilog is very C-like and liked by electrical and computer engineers as most learn the C language in college. VHDL is very most engineers have no experience. Verilog was introduced in 1985 by Gateway Design System Corporation, now a part of Cadence Design Systems, Inc.'s Systems Division. Until May, 1990, with the formation of Open Verilog International (OVI), Verilog HDL was a proprietary language of Cadence. Cadence was motivated to open the language to the Public Domain with the expectation that the market for Verilog HDL-related software products would grow more rapidly with broader acceptance of the language. Cadence realized that Verilog HDL users wanted other software and service companies to embrace the language and develop Verilog-supported design tools.

6.3 CODING IMPLEMENTATION:

2 BIT ADDER:

```
  xor x1(C[0],A[0],B[0]);

  xor x2(C[1],A[1],B[1]);
```

4 BIT ADDER:

```
  xor x1(C[0],A[0],B[0]);

  xor x2(C[1],A[1],B[1]);

  xor x3(C[2],A[2],B[2]);

  xor x4(C[3],A[3],B[3]);
```

DEL Iverse:

```
assign A[8]=D[8];

  assign A[7]=D[7]^D[6]^D[5]^D[1];

  assign A[6]=D[6]^D[2];

  assign A[5]=D[6]^D[5]^D[1];

  assign A[4]=D[6]^D[5]^D[4]^D[2]^D[1];

  assign A[3]=D[5]^D[4]^D[3]^D[2]^D[1];

  assign A[2]=D[7]^D[4]^D[3]^D[2]^D[1];

  assign A[1]=D[5]^D[4];

  assign A[0]=D[6]^D[5]^D[4]^D[2]^D[0];
```

AFFINE TRANSFORM:

```
  assign AT[8]=(b[8]);

  assign AT[7]=(b[7]^b[6]^b[5]^b[4]^b[3]);

  assign AT[6]=~(b[6]^b[5]^b[4]^b[3]^b[2]);

  assign AT[5]=~(b[5]^b[4]^b[3]^b[2]^b[1]);

  assign AT[4]=(b[4]^b[3]^b[2]^b[1]^b[0]);

  assign AT[3]=(b[7]^b[3]^b[2]^b[1]^b[0]);

  assign AT[2]=(b[7]^b[6]^b[2]^b[1]^b[0]);

  assign AT[1]=~(b[7]^b[6]^b[5]^b[1]^b[0]);

  assign AT[0]=~(b[7]^b[6]^b[5]^b[4]^b[0]);
```

X INVERSE:

```
M_X2Section VA1 (

    .A(A),

    .B(a)

    );



M_X2Section VA2 (

    .A(a),

    .B(b)
```

```
    );

M_X2Section VA3 (
    .A(b),
    .B(c)
    );

M_BlackXblock VA4 (
    .A(a),
    .B(b),
    .D(d)
    );

M_BlackXblock VA5 (
    .A(d),
    .B(c),
    .D(B)
    );
```

LAMDA:

```
xor x1(c,A[1],A[3]);
  xor x2(d,A[0],A[2]);
  xor x3(e,d,c);
  assign B[0]=A[2];
  assign B[1]=A[3];
  assign B[2]=e;
  assign B[3]=d;
```

BLACK X BOX:

```
M_XS V1 (
    .X(p),
    .X1(s)
    );
M_2XAdder V2 (
    .A(A[3:2]),
    .B(A[1:0]),
    .C(x)
    );
M_2XAdder V3 (
    .A(B[3:2]),
```

```
    .B(B[1:0]),
    .C(y)
    );
M_2XAdder V4 (
    .A(q),
    .B(r),
    .C(D[3:2])
    );
M_2XAdder V5 (
    .A(r),
    .B(s),
    .C(D[1:0])
    );
M_WX V6 (
    .A(A[3:2]),
    .B(B[3:2]),
    .O(p)
    );
M_WX V7 (
```

```
    .A(x),
    .B(y),
    .O(q)
    );

M_WX V8 (
    .A(A[1:0]),
    .B(B[1:0]),
    .O(r)
    );
```

SUBBYTES:

```
assign C=B[7:4];
  assign D=B[3:0];
  assign O[7:4]=M;
  assign O[3:0]=N;

   S_Isomorphic_Map S1(
                .X(in),
                .Y(B)
                 );

   S_Square_Block S2(
                 .X(C),
```

```
        .Y(E)
        );


  S_4bitAdd S3(
        .X(D),
        .Y(C),
        .Z(F)
        );


  S_4bitmulitiplication S4(
              .A(F),
              .B(D),
              .D(G)
              );


S_4bitAdd S5(
        .X(G),
        .Y(H),
        .Z(I)
        );


S_Multi_Constant_lambda S6(
              .A(E),
              .B(H)
              );
```

```
S_Multiplicative_Inv S7(
                .A(I),
                .B(J)
                );


S_4bitmulitiplication S8(
                .A(J),
                .B(C),
                .D(M)
                );

 S_4bitmulitiplication S9(
                .A(J),
                .B(F),
                .D(N)
                );

 S_Inv_Isomorphic_Map S10(
                .Y(O),
                .X(P)
                );

 S_affine S11(
        .i(P),
        .q(out)
        );
```

DNA Code Generation:

```
 always @ (In)
 begin
 case (In)
        4'b0000: Out=24'b010000010100001101010100;
        4'b0001: Out=24'b010000010100001101000111;
        4'b0010: Out=24'b010101000100000101000111;
        4'b0011: Out=24'b010001110100001101000001;
        4'b0100: Out=24'b010001110100000101000111;
        4'b0101: Out=24'b010000010100011101000001;
        4'b0110: Out=24'b010101000101010001000001;
        4'b0111: Out=24'b010000010100001101000001;
        4'b1000: Out=24'b010000010100011101000111;
        4'b1001: Out=24'b010001110100001101000111;
         4'b1010:Out=24'b010000110100011101000001;
        4'b1011: Out=24'b010000110100001101000001;
        4'b1100: Out=24'b010001110101010001010100;
        4'b1101: Out=24'b010101000101010001000111;
        4'b1110: Out=24'b010001110100011101000011;
        4'b1111: Out=24'b010001110100011101010100;
endcase
```

DNA Code Block:

```
 SD_DNA_Code_Generation M0 (
  .In(Key[3:0]),
  .Out(DNA_Key[23:0])
  );

 SD_DNA_Code_Generation M1 (
  .In(Key[7:4]),
  .Out(DNA_Key[47:24])
  );


 SD_DNA_Code_Generation M2 (
  .In(Key[11:8]),
  .Out(DNA_Key[71:48])
  );

 SD_DNA_Code_Generation M3 (
  .In(Key[15:12]),
  .Out(DNA_Key[95:72])
  );
```

```
  SD_DNA_Code_Generation M4 (
   .In(Key[19:16]),
   .Out(DNA_Key[119:96])
   );

  SD_DNA_Code_Generation M5 (
   .In(Key[23:20]),
   .Out(DNA_Key[143:120])
   );

assign Main_DNA_Key=DNA_Key[127:0];
```

TOP Module Crypto DNA:

```
  SD_DNA_Code_Block DNA_Coder (
    .Key(Key),
    .DNA_Key(DNA_Key)
    );

   S_Topmodule_AES_Encryption AES_Design (
    .Plain_Test(Plain_Text_In),
    .Key(DNA_Key[127:0]),
    .Cipher_Test(Cipher_Text)
    );

   S_AES_Decryption Decryption_Unit (
    .Cipher_Text(Cipher_Text),
    .Key(DNA_Key[127:0]),
    .Plain_Text(Plain_Text_Out)
    );
```

CHAPTER 7

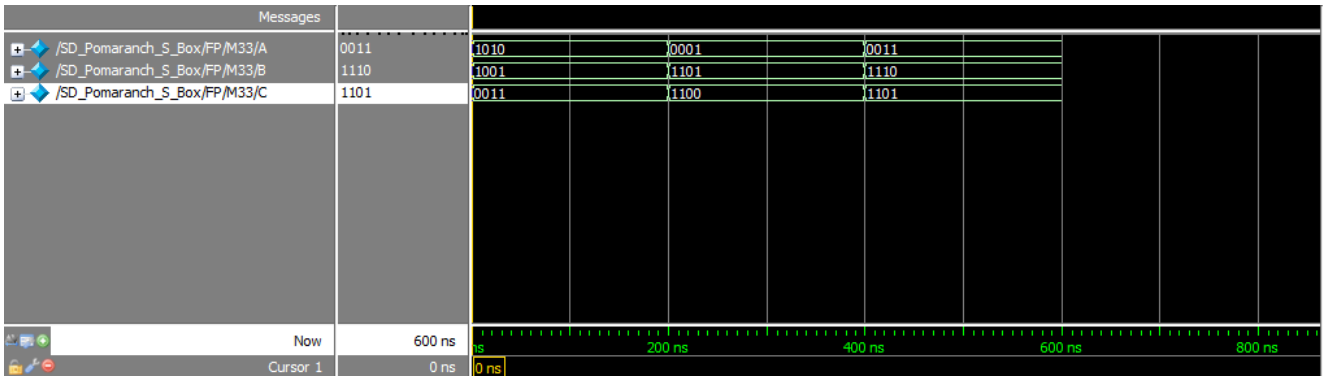
SNAPSHOTS

7.1 GENERAL:

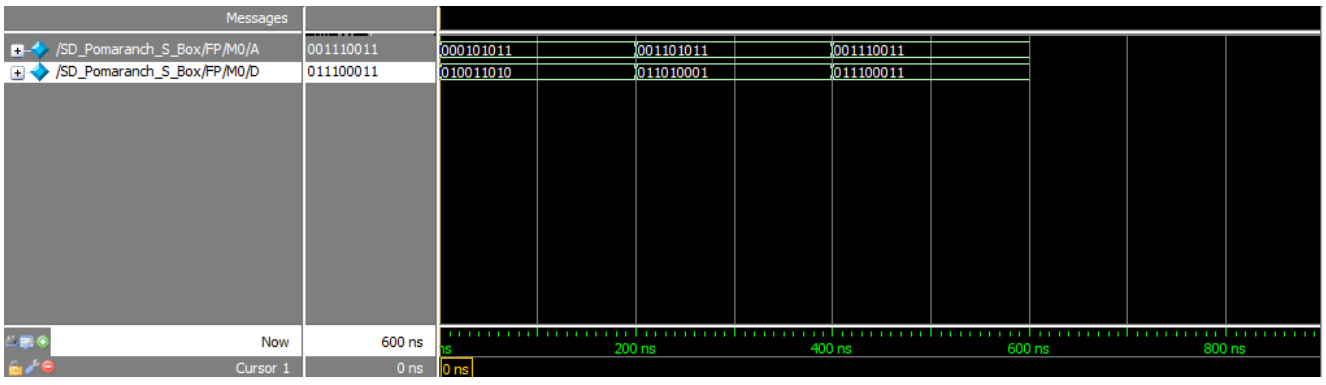
Snapshot is nothing but every moment of the application while running. It gives the clear elaborated of application. It will be useful for the new user to understand for the future steps.

7.2 VARIOUS SNAPSHOTS:

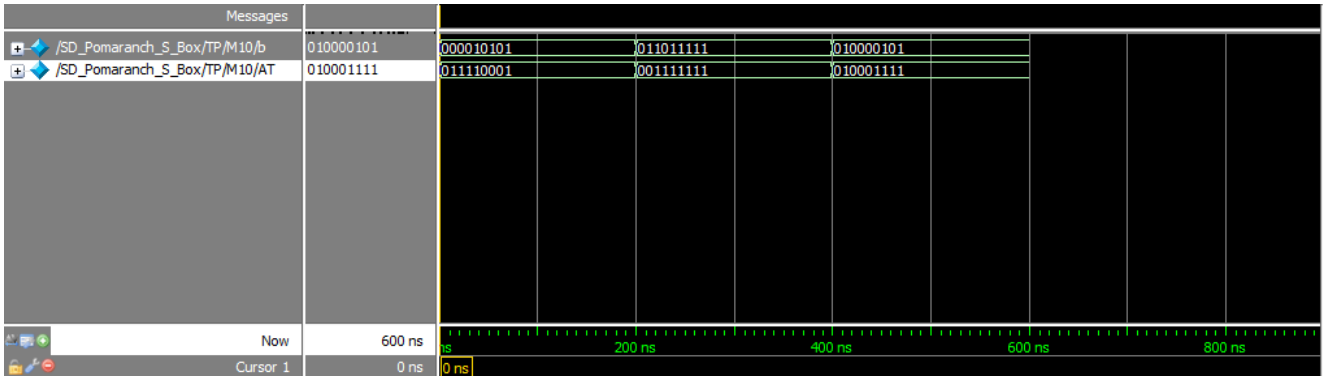
2 BIT ADDER:



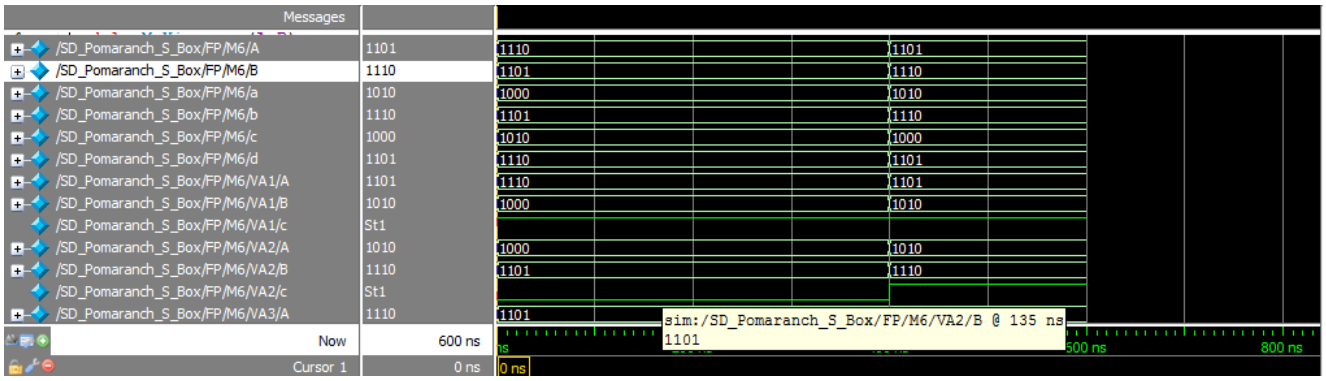
DEL Iverse:



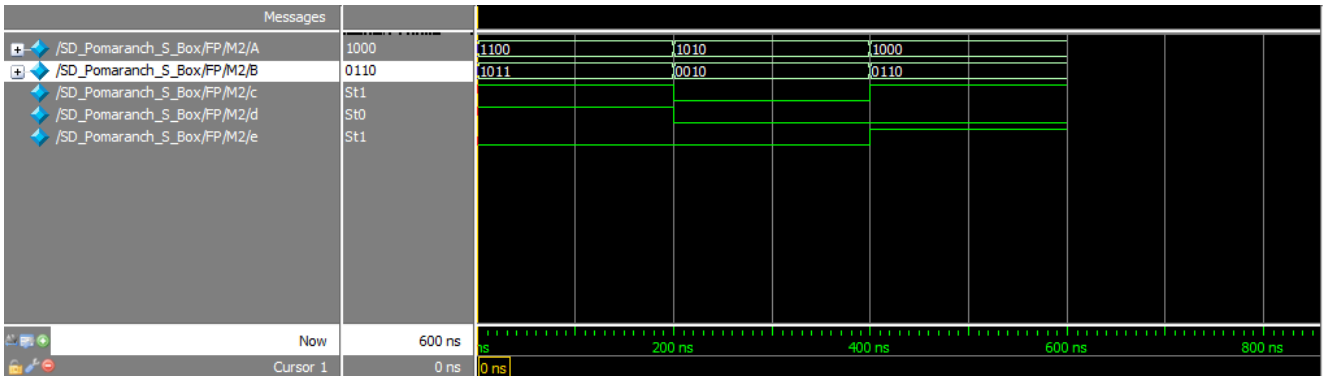
AFFINE TRANSFORM:



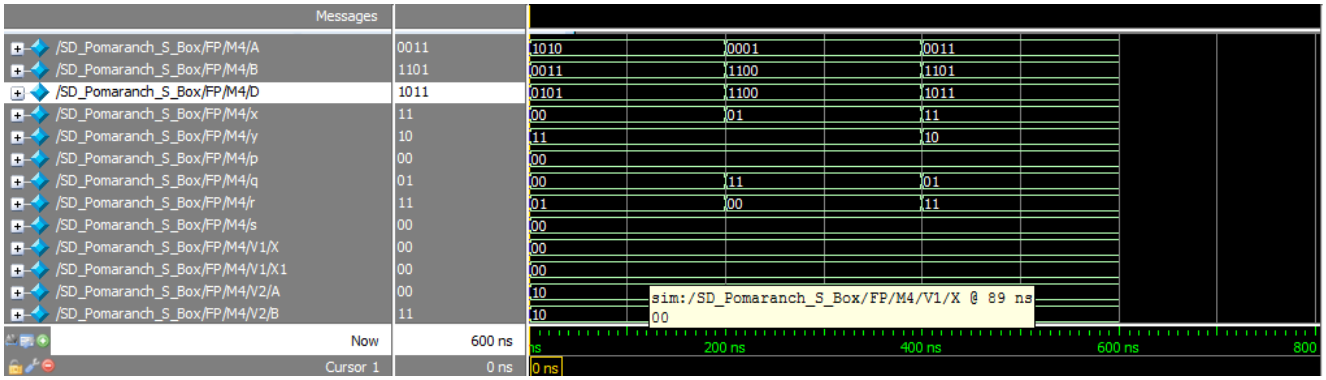
X INVERSE:



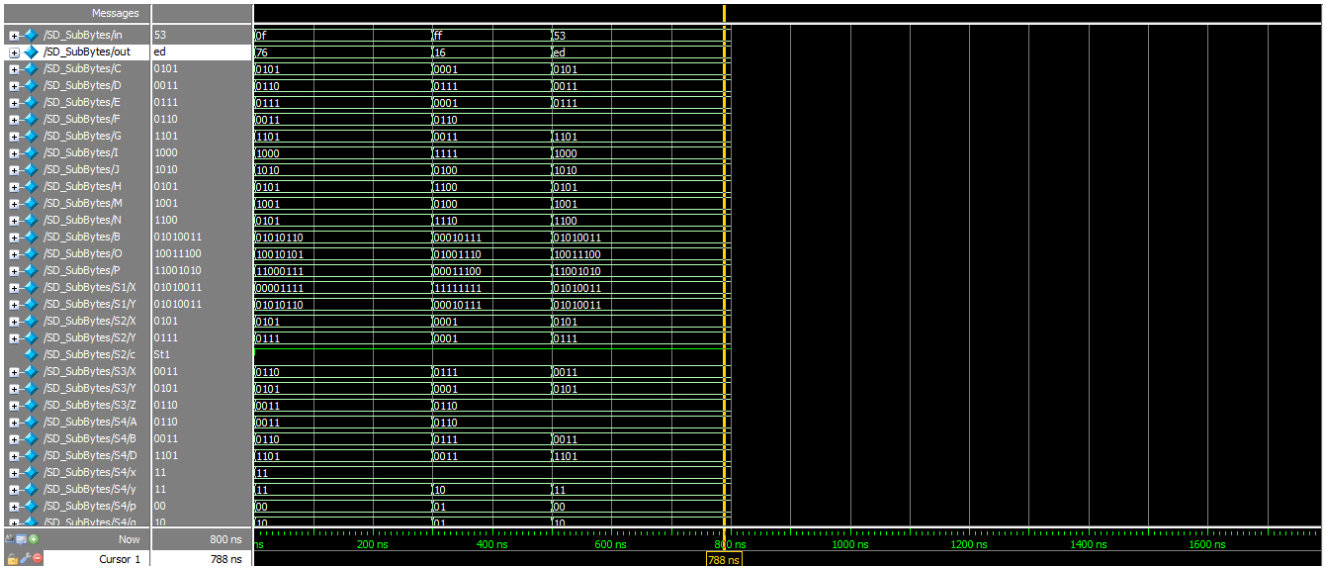
LAMDA:



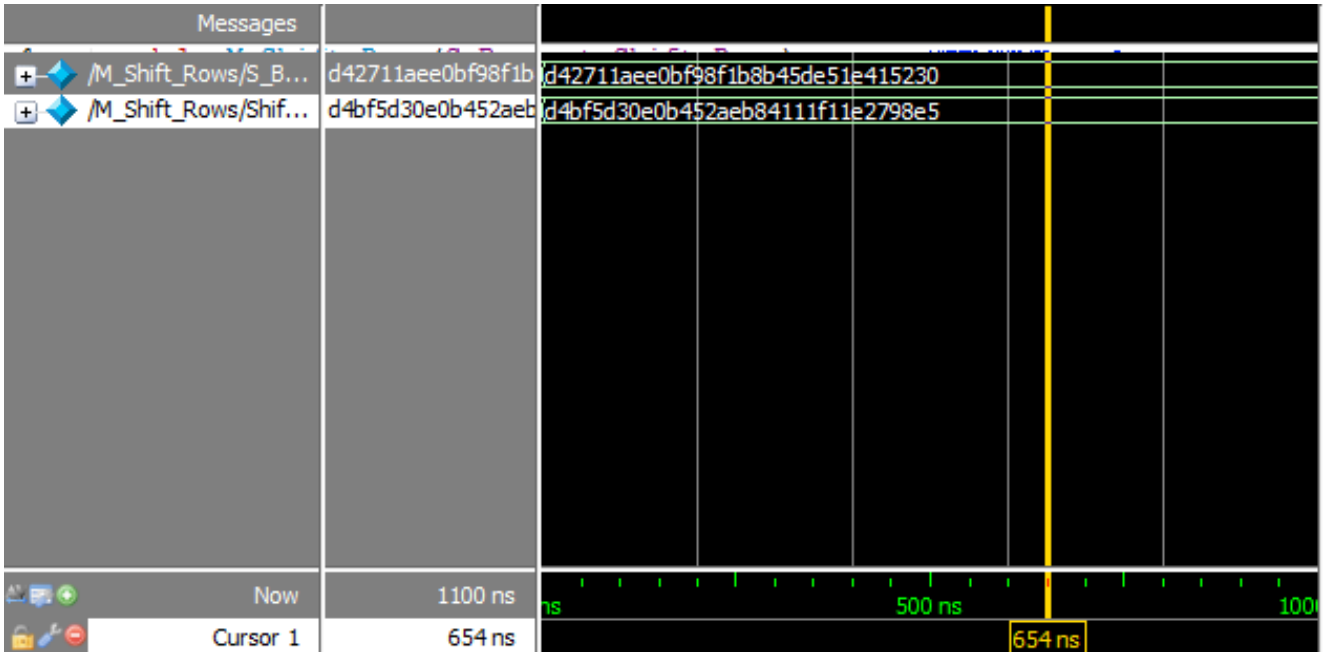
BLACK X BOX:



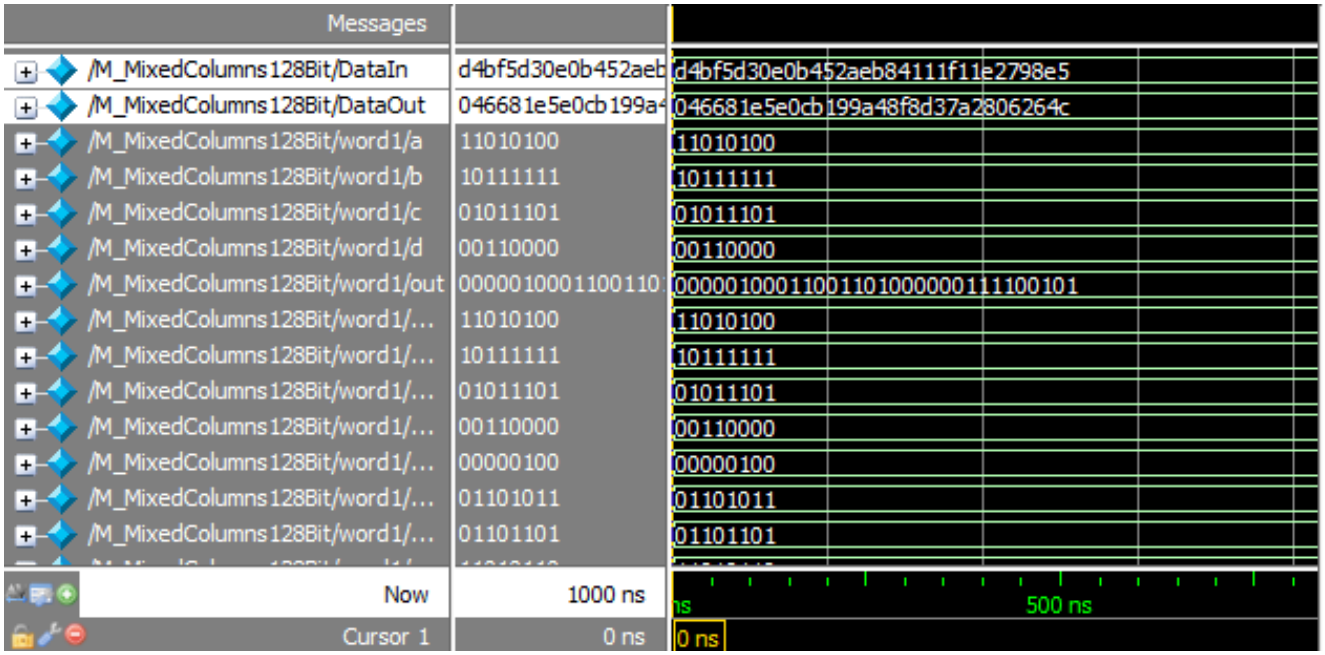
SUB BYTES:



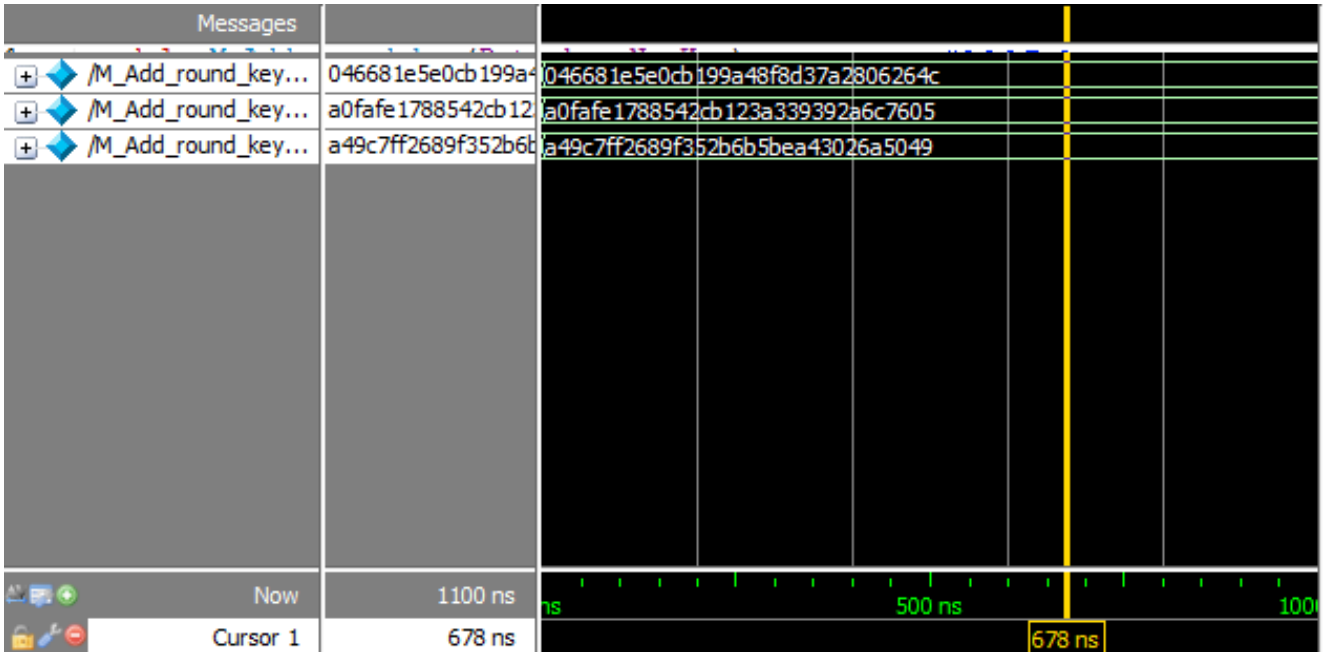
SHIFT ROWS:



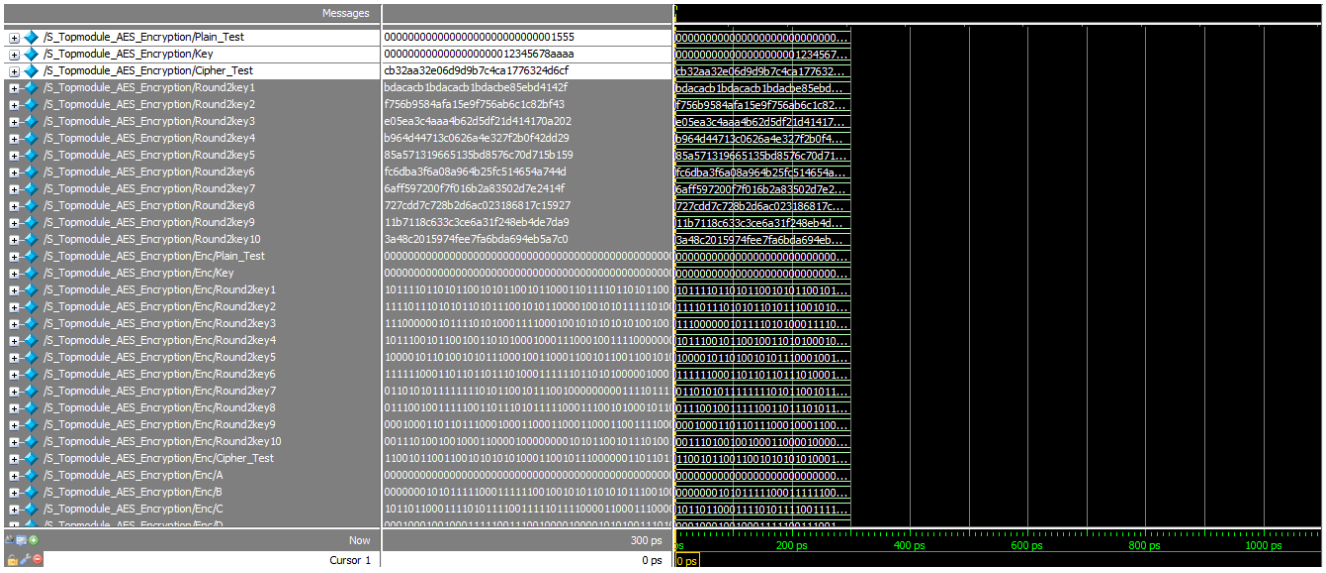
MIXED COLUMN:



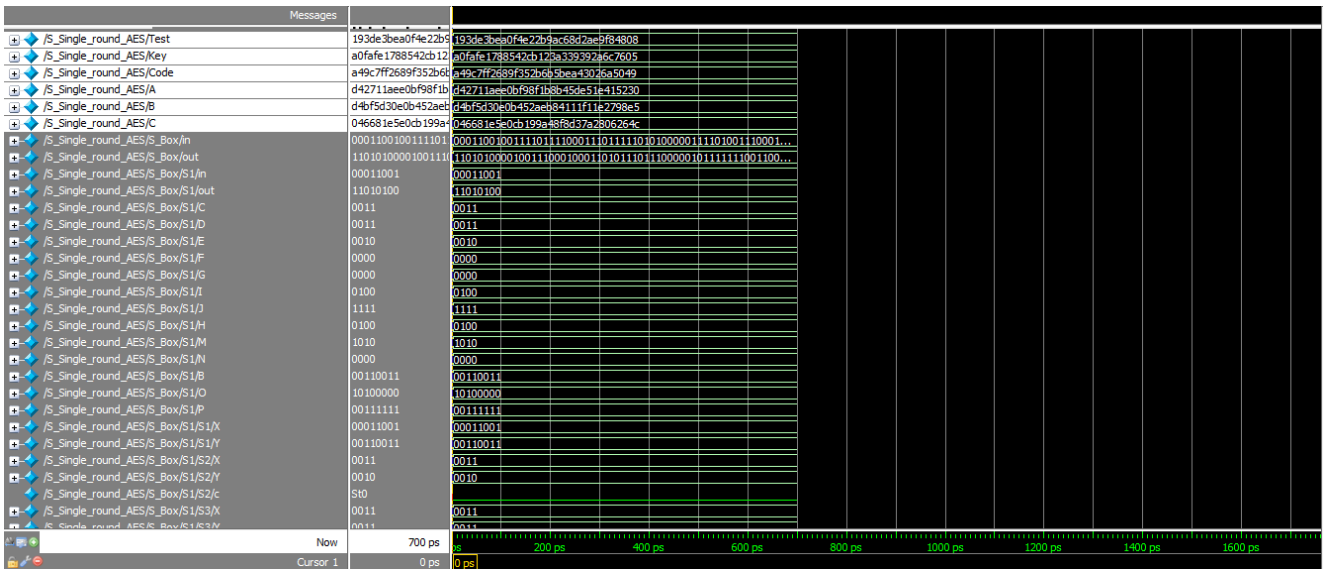
ADD ROUND KEY:



ENCRYPTION:

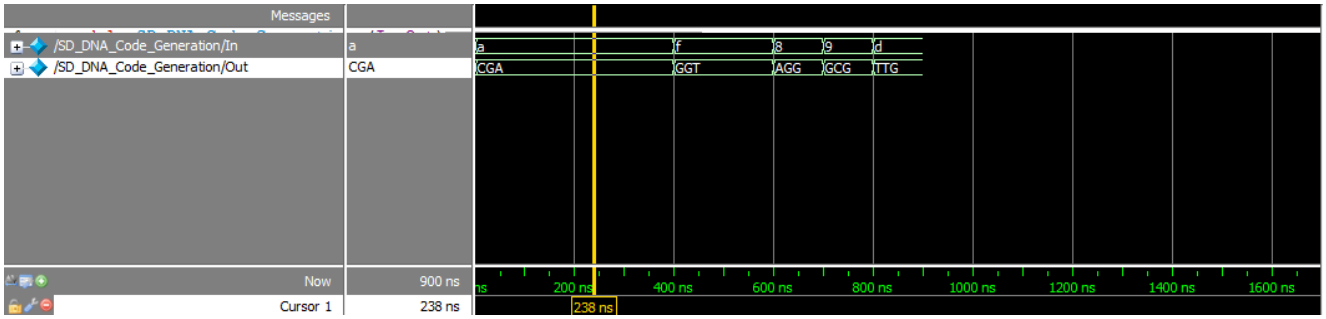


SINGLE ROUND OPERATION:

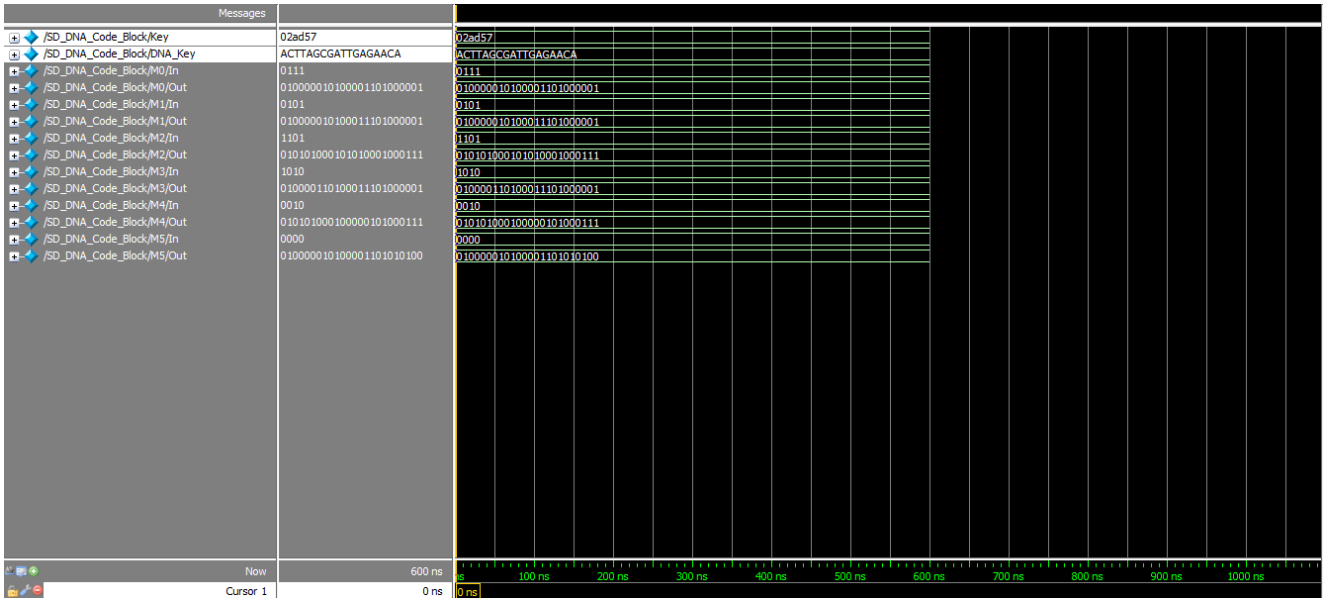


FINAL TOP MODULE:

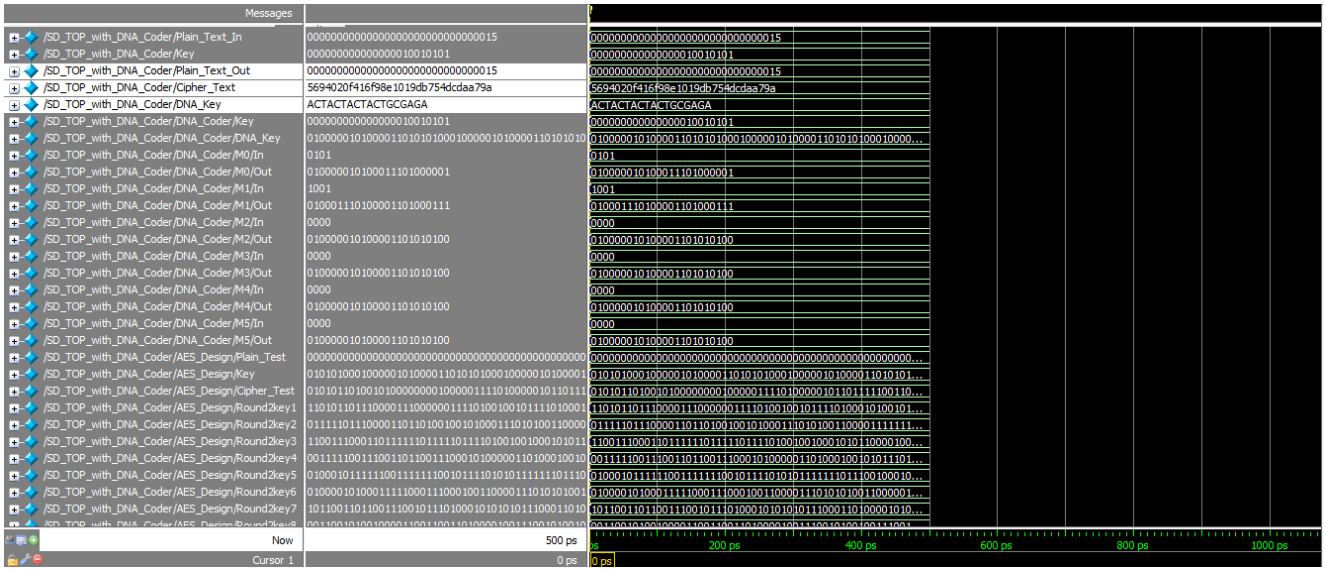
DNA CODE GENERATION:



DNA CODE BLOCK:



DNA BASED TOP MODULE CRYPTO:



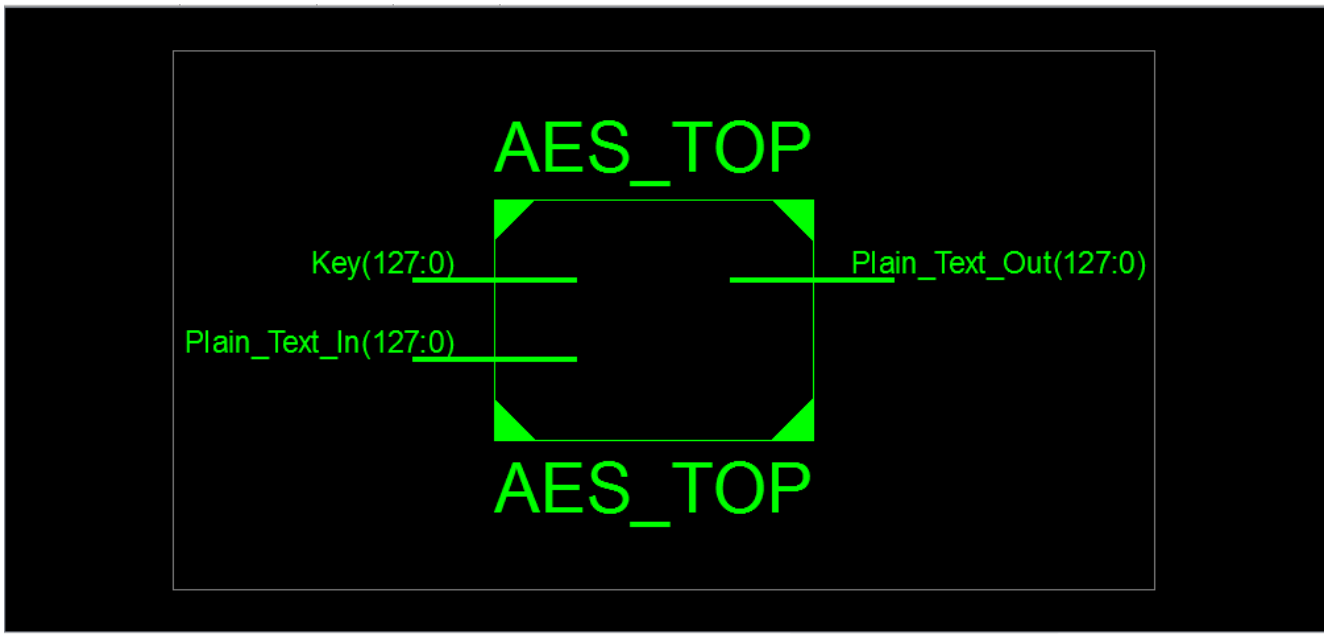
CHAPTER 8

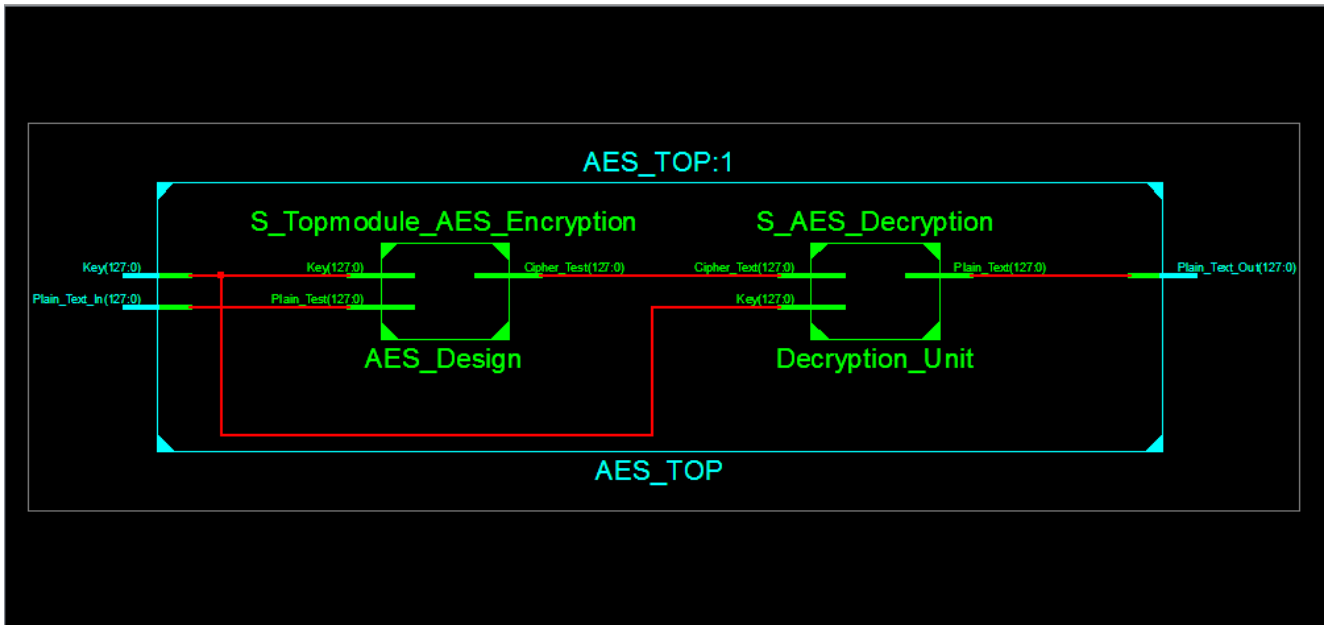
SYNTHESIS REPORT

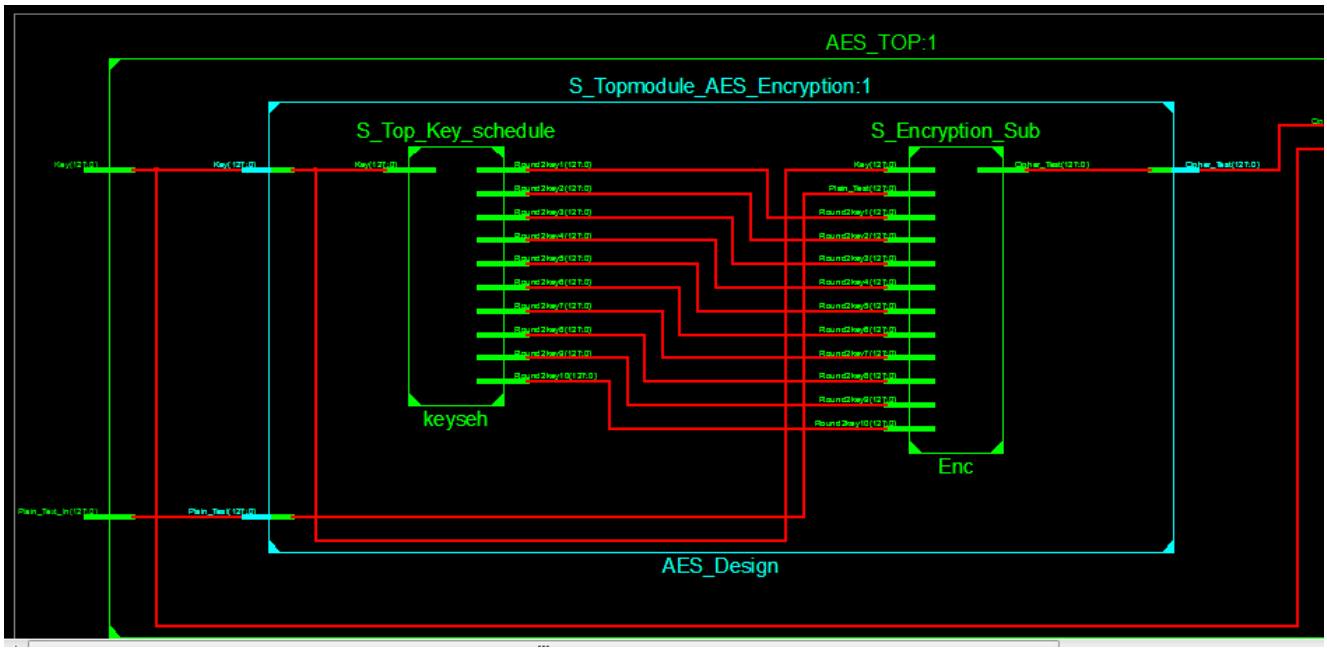
8.1 DEVICE UTILIZATION SUMMARY

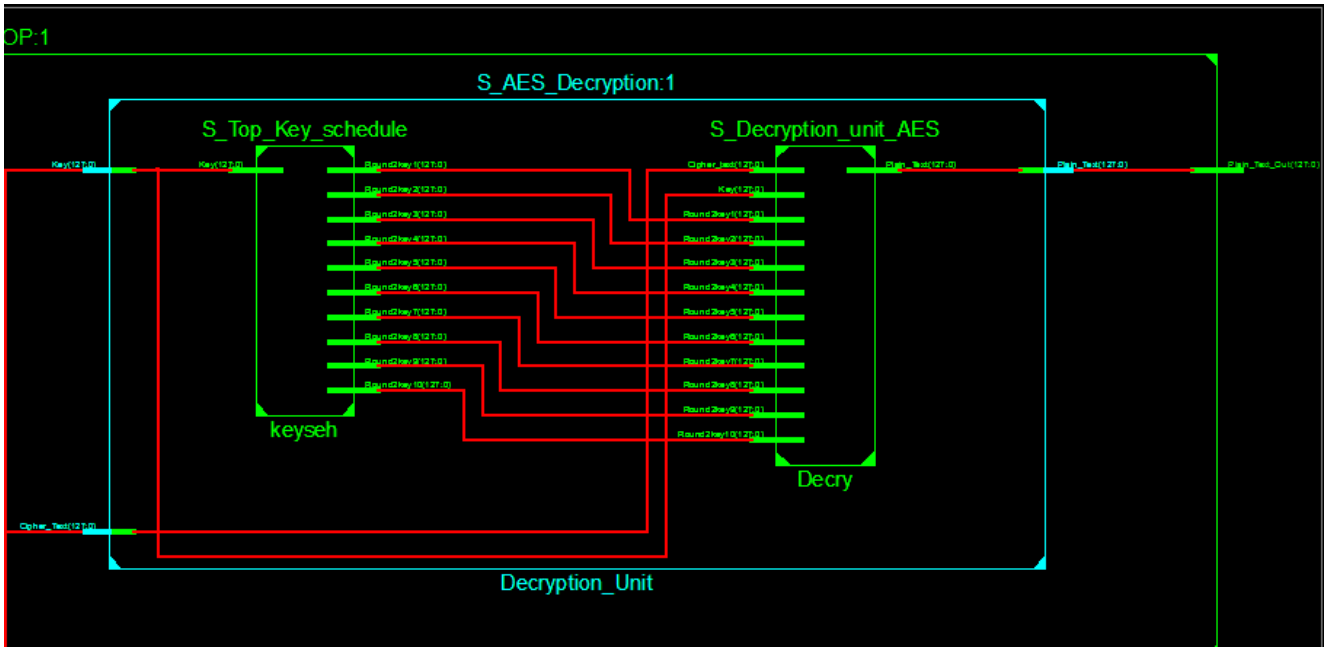
Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	42,933	55,296	77%	
Number of occupied Slices	22,510	27,648	81%	
Number of Slices containing only related logic	22,510	22,510	100%	
Number of Slices containing unrelated logic	0	22,510	0%	
Total Number of 4 input LUTs	42,933	55,296	77%	
Number of bonded IOBs	384	633	60%	
Average Fanout of Non-Clock Nets	4.17			

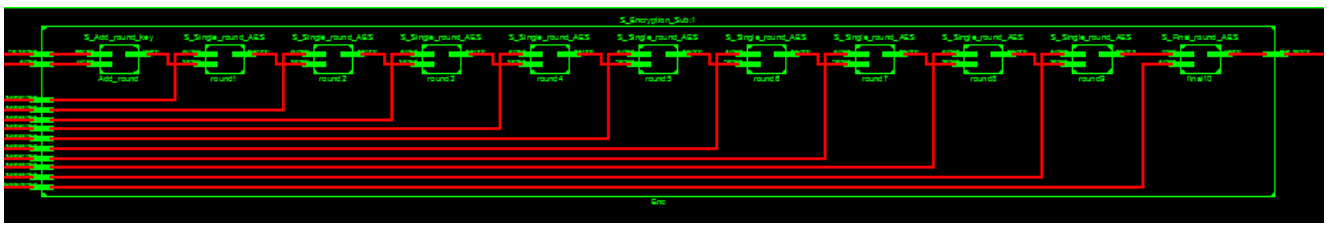
8.2 RTL SCHEMATIC


AES_TOP
Key(127:0)
Plain_Text_In(127:0)
Plain_Text_Out(127:0)
AES_TOP


AES_TOP:1
S_Topmodule_AES_Encryption
S_AES_Decryption
Key(127:0)
Plain_Text_In(127:0)
Key(127:0)
Plain_Test(127:0)
Cipher_Test(127:0)
Cipher_Text(127:0)
Key(127:0)
Plain_Text(127:0)
Plain_Text_Out(127:0)
AES_Design
Decryption_Unit
AES_TOP


AES_TOP:1
S_Topmodule_AES_Encryption:1
S_Top_Key_schedule
S_Encryption_Sub
keyseh
Enc
AES_Design


S_AES_Decryption:1
S_Top_Key_schedule
S_Decryption_unit_AES
keyseh
Decry
Decryption_Unit

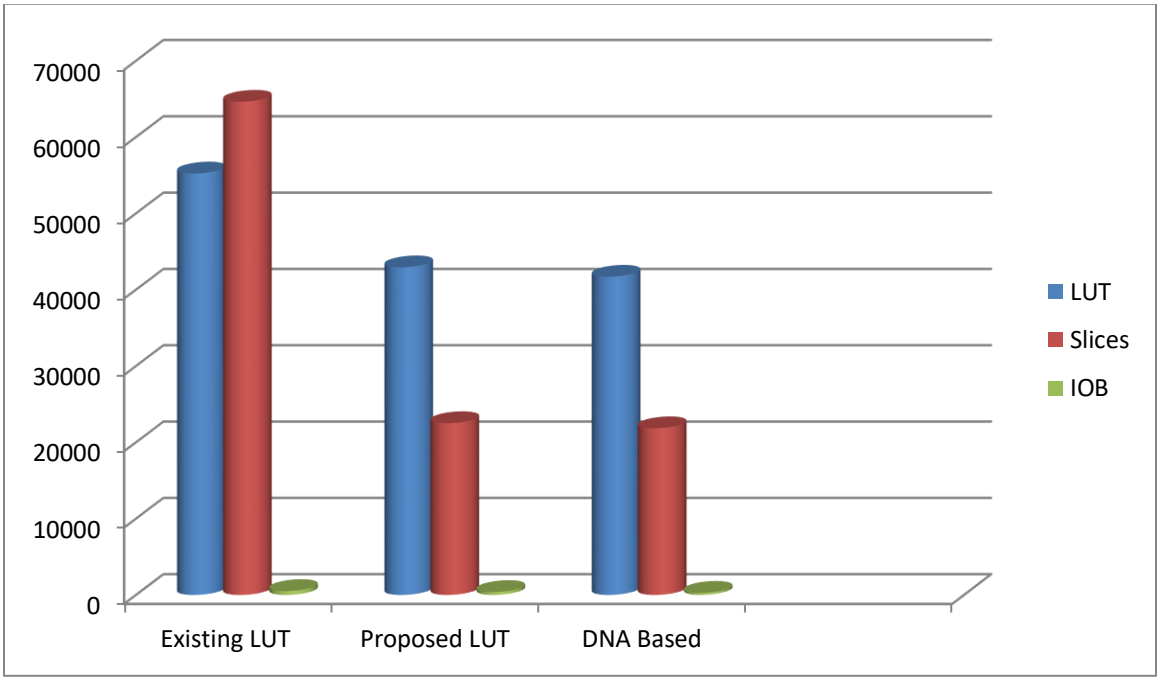


COMPARISON TABLE:

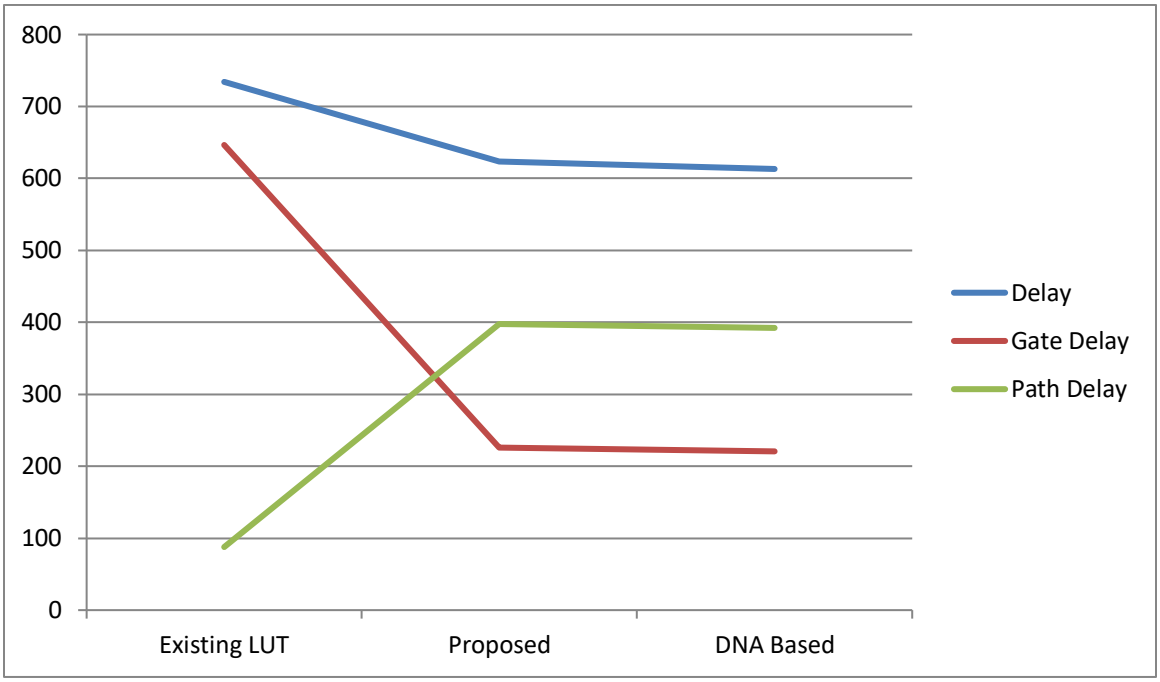
AREA AND DELAY COMMPARISON TABLE

Method Name	Area in Number of LUT			Delay		
Spartan 3 XC 3S 4000L-4FG900	**LUT**	**Slices**	**IOB**	**Delay**	**Gate or Logic Delay**	**Path or Route Delay**
Existing LUT	55226	64627	512	734.2ns	646.5ns logic	87.7ns route
Proposed Design	42933	22510	384	623.168ns	225.498ns	397.671ns
DNA BASED	41697	21844	280	613.140ns	220.653ns	392.487ns

AREA GRAPH:



DELAY GRAPH:


800
700
600
500
400
300
200
100
0
Existing LUT
Proposed
DNA Based
Delay
Gate Delay
Path Delay

CHAPTER 8

CONCLUSION AND REFERENCES

8.1 CONCLUSION:

A novel AES Design with DNA implementation with High Security Constrains. The Implementation is based on mathematical properties of Rijndael algorithm and remains based on DNA Coders. Another contribution of this paper is that it designs Encryption Design using Shit rows, Mixed Column, Add Round Key and We Will Design a Decryption Part also. Finally with the help of Genetic Algorithm based Encoding for Key Generation is used for Encryption and Decryption Process. The new design permits the construction of efficient area and speed characteristics, while still keeping a very high protection level. We conducted relevant AES Implementation with DNA for Key Generation Method. Nearly all the algorithms embedded in Cryptographer have been designed to resist at high level to linear, differential and high order differential attacks, whereas nothing has been done to make them inherently resistant attacks. However, this work will be implemented in the FPGA. It is possible to design algorithms, so when the next generation of cryptographic.

9.2 REFERENCES OR BIBLIOGRAPHY:

1. K. Fu and J. Blum, "Controlling for cybersecurity risks of medical device software," Commun. ACM, vol. 56, no. 10, pp. 35–37, Oct. 2013.

2. D. Halperin, T. Kohno, T. S. Heydt-Benjamin, K. Fu, and W. H. Maisel, "Security and privacy for implantable medical devices," IEEE Pervasive Comput., vol. 7, no. 1, pp. 30–39, Jan./Mar. 2008.

3. M. Rostami, W. Burleson, A. Jules, and F. Koushanfar, "Balancing security and utility in medical devices?" in Proc. 50th ACM/EDAC/IEEE Int. Conf. Design Autom., May/Jun. 2013, pp. 1–6.

4. M. Zhang, A. Raghunathan, and N. K. Jha, "Trustworthiness of medical devices and body area networks," Proc. IEEE, vol. 102, no. 8, pp. 1174–1188, Aug. 2014.

5. H. Khurana, M. Hadley, N. Lu, and D. A. Frincke, "Smart-grid security issues," IEEE Security Privacy, vol. 8, no. 1, pp. 81–85, Jan./Feb. 2010.

6. M. Mozaffari-Kermani, M. Zhang, A. Raghunathan, and N. K. Jha, "Emerging frontiers in embedded security," in Proc. 26th Int. Conf. VLSI Design, Jan. 2013, pp. 203–208.

7. R. Roman, P. Najera, and J. Lopez, "Securing the Internet of things," Computer, vol. 44, no. 9, pp. 51–58, Sep. 2011.

8. T. H.-J. Kim, L. Bauer, J. Newsome, A. Perrig, and J. Walker, “Challenges in access right assignment for secure home networks,” in Proc. USENIX Conf. Hot Topics Secur., 2010, pp. 1–6.

9. M. Mozaffari-Kermani and A. Reyhani-Masoleh, “Concurrent structure independent fault detection schemes for the Advanced Encryption Standard,” IEEE Trans. Comput., vol. 59, no. 5, pp. 608–622, May 2010.

10. M. Mozaffari-Kermani and A. Reyhani-Masoleh, “A low-power high performance concurrent fault detection approach for the composite field S-box and inverse S-box,” IEEE Trans. Comput., vol. 60, no. 9, pp. 1327–1340, Sep. 2011.

11. M. Mozaffari-Kermani and A. Reyhani-Masoleh, “A lightweight high performance fault detection scheme for the Advanced Encryption Standard using composite fields,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 19, no. 1, pp. 85–91, Jan. 2011.

12. A. Satoh, T. Sugawara, N. Homma, and T. Aoki, “High-performance concurrent error detection scheme for AES hardware,” in Proc. 10th Int. Workshop CHES, Aug. 2008, pp. 100–112.

13.P. Maistri and R. Leveugle, “Double-data-rate computation as a countermeasure against fault analysis,” IEEE Trans. Comput., vol. 57, no. 11, pp. 1528–1539, Nov. 2008.

14. X. Guo and R. Karri, “Recomputing with permuted operands: A concurrent error detection approach,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 32, no. 10, pp. 1595–1608, Oct. 2013.

15. M. Mozaffari-Kermani and R. Azarderakhsh, “Efficient fault diagnosis schemes for reliable lightweight cryptographic ISO/IEC standard CLEFIA benchmarked on ASIC and FPGA,” IEEE Trans. Ind. Electron., vol. 60, no. 12, pp. 5925–5932, Dec. 2013.

16. C. J. A. Jansen, T. Helleseth, and A. Kholosha, “Cascade jump controlled sequence generator (CJCSG),” in Proc. Workshop Symmetric Key Encryption, 2005, pp. 1–16. [Online]. Available: http://www.ecrypt.eu.org/stream/ciphers/pomaranch/pomaranch.pdf

17. C. J. A. Jansen, T. Helleseth, and A. Kholosha, “Cascade jump controlled sequence generator and Pomaranch stream cipher (version 3),” Dept. Informat., Univ. Bergen, Bergen, Norway, Tech. Rep. 2006/006, 2006. [Online]. Available: http://www.ecrypt.eu.org/stream/papers.html