DESIGN OF A HUMMINGBIRD CRYPTO ASIC IMPLEMENTING BIST TECHNIQUE

By

Md Azizul Haque

MASTER OF ENGINEERING

IN

INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology

Bangladesh University of Engineering and Technology (BUET)

2016

This project titled, **"Design of a HUMMINGBIRD Crypto ASIC Implementing BIST Technique"** submitted by Md. Azizul Haque, Roll No. M1009312003, Session: 2009-2010 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Engineering in Information and Communication Technology on 19-03-2016.

AUTHOR’S DELARATION

This is hereby declared that this project or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Md Azizul Haque

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS OF TECHNICAL SYMBOLS AND TERMS

3DES	Triple-DES (Data Encryption Standard)
AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
GF	Galois Field
LUT	Look-Up Table
M512	Memory Block of 512 bits
M-RAM	Mega RAM
PLD	Programmable Logic Device
RAM	Random Access Memory
ROM	Read Only Memory
S-Box	A lookup table that holds non-linear substitute byte values
VHDL	Very High Speed Integrated Circuit Hardware Description Language
XOR	Exclusive-OR
$\oplus$	Exclusive-OR Operator
$\boxplus$	Modulo 2^{16} addition Operator
$\boxminus$	Modulo 2^{16} Subtraction Operators

Abstract

There are several emerging areas in which highly resource constrained devices are interconnected, typically communicating wirelessly with one another, and working in concert to accomplish some task. Examples of these areas include: sensor networks, healthcare, distributed control systems, the Internet of Things, cyber physical systems, and the smart grid. Security and privacy can be very important in all of these areas. Because the majority of current cryptographic algorithms were designed for desktop/server environments, many of these algorithms do not fit into the constrained resources. If current algorithms can be made to fit into the limited resources of constrained environments, their performance is typically not acceptable. Here comes in the Lightweight Cryptography. Many research works have been going on this topic. Among them, Hummingbird is a new ultra-lightweight cryptographic algorithm targeted for resource-constrained devices like RFID tags, smart cards, and wireless sensor nodes. The efficiency of the algorithm has been verified in software solution for a wide range of embedded applications. In this paper, we describe FPGA implementation with both software simulation as well as HW implementation of Hummingbird algorithm on Altera Platform.

Again testability of a complex chip is of prime concern now a days. It consists of IC design techniques that add the features to test the designed hardware and ensure the chip is free from defects and will function correctly. Because of its convenience and less expensiveness over ATE, Built-In-Self-Test (BIST) is a widely used technique for this purpose. In this project design of a Hummingbird Crypto ASIC implementing BIST technique is proposed. In this design LFSR is used to generate pseudorandom test pattern and Signature Analysis is used as a Data Compression Technique to implement BIST for multiple test counts. The proposed Crypto ASIC is simulated in Quartus II simulation software in the Altera Cyclone II family device as well as hardware implementation done in Altera DE2 board and performance is analyzed and compared with the other research works on Hummingbird implementation.

Acknowledgments

At first I would like to thanks Almighty Allah (SWT) for providing His divine blessing, unlimited mercy to me. During the time, I was performing my Master's Project, I came across many people who have supported and assisted me. First, I want to express my heartiest thanks to my supervisor, Professor Dr. Md. Liakot Ali for giving me the opportunity to do my Master's Project under his supervision. I would like to express many thanks for his invaluable advice and ideas on the project and also for his devotion of time during this program. His support and expertise resolved many hurdles that I encountered throughout the research. Without his continuous support, this project could not have been completed.

I also gratefully acknowledge the valued advice and support from Associate Professor Dr. Rubaiyat Hossain Mondol and Assistant Professor Mohammad Imam Hasan Bin Asad. My special thanks go to all the teachers and staffs of IICT, BUET.

Finally I would like to thank my beloved wife Kulsum Jamila for her endless support in all respects which helped me to devote to the work.

Chapter 1

1.1 Introduction

This is an age of Information and Communication Technology (ICT). The rapid growth in computer systems and their interconnections via networks has increased the dependence of both organizations and individuals on the information stored and its communication using these systems. Information security is now a burning issue in this era of Information and Communication Technology (ICT) [1]. The security has involves in many applications such as mobile networks, internet of things, automated teller machines (ATMs), copy protection (especially protection against reverse engineering and software piracy), internet e-commerce, internet banking, military and government to facilitate secret communication and many more. Cryptography plays an important role in security system. In cryptography the secret data is encrypted by a secret key and the encrypted data can only be deciphered if one has the key. The encryption algorithm is asset of well-defined steps to transform data from a readable format to an encoded format using the key. This set of well-defined steps is called cipher. A number of algorithms on cryptography have been presented in the literatures [2-5]. At present AES has been proved as the strongest encryption algorithm declared by USA Govt. [5].

However, every application has different requirements such as the speed at which the security operations must be performed, the physical area for embedded hardware, or its power budget. Now a days there are lot of applications coming in the market where an increasing number of battery powered embedded systems like PDAs, cell phones, networked sensors, smart cards, RFID are used to store, access, manipulate or communicate sensitive data. It makes security an important issue. Since those devices are resource constrained and battery powered, low power and small area are the mandatory requirements. It has been found AES is not suitable for low cost embedded devices such as RFID tags, smart cards, wireless sensor nodes etc. due to their resource constraints [6]. To overcome this problem Hummingbird has been proposed as ultra-lightweight cryptographic algorithm suitable for embedded system and shown that it is resistant to a number of attacks such as algebraic attacks, cube attacks, differential power attacks etc. The efficiency of the algorithm has been verified in software solution for a wide range of embedded applications [7-8]. To verify its effectiveness in hardware platform, its FPGA implementation on Xilinx devices has been presented [6]. However its implementation on Altera platform has not been reported yet although there are a major group of researchers who use this platform.

Again testability of a complex chip is of prime concern in today's IC design. Testing a VLSI chip to guarantee its functionality is extremely complex, time consuming as well as expensive [9]. To deal with the testing problem at the chip level incorporating built-in self-test (BIST) capability inside a chip is a widely accepted approach [10-14]. When chip is complex then Built-In-Self-Test (BIST) is a norms of this day because external testing using ATE is not cost effective and less convenient in this case. BIST in Hummingbird Crypto ASIC is not reported yet. This project focuses on implementing BIST in Hummingbird Crypto ASIC in Altera platform. In this design LFSR is used to generate pseudorandom test pattern and Signature Analysis is used as a Data Compression Technique to implement BIST for multiple test counts. The proposed Crypto ASIC is simulated in Quartus II simulation software in the Altera Cyclone II family device and performance is analyzed in terms of power consumption and compared with the other similar research works.

1.2 Motivation

For many years, the cryptographic engineering communities had worked on the problem of implementing various cryptographic primitives as fast as possible. Typical examples were high-speed RSA and Advanced Encryption Standard (AES) engines. However, the upcoming pervasive computing era that features myriads of small, inexpensive, robust networked processing devices has put forward the new challenge to the implementation of security mechanisms for embedded applications. Ultra low-cost smart devices such as RFID tags, smart cards, and wireless sensor nodes usually have extremely constrained resources in terms of computational capabilities, memory, and power supply. Consequently, classical cryptographic primitives designed for full-fledged computers might not be suited for resource-constrained pervasive devices and it is often desirable to have cryptographic primitives as small as possible. As a response to the aforementioned issue, lightweight cryptography, which focuses on designing new cryptographic primitives with small footprint in hardware and low average and peak power consumption, has received a lot of attention from both academia and industry in recent years.

The key issue of designing lightweight cryptographic algorithms is to deal with the trade-off among security, cost, and performance and find an optimal cost-performance ratio [15]. Quite a few lightweight symmetric ciphers that particularly target resource-constrained smart devices have been published in the past few years and those ciphers can be utilized as basic building blocks to design security mechanisms for embedded applications. All the previous proposals can be roughly divided into the following three categories. The first category consists of highly optimized and compact hardware implementations for standardized block ciphers such as AES [16-18], IDEA [19] and XTEA [20], whereas the proposals in the second category involve slight modifications of a classical block cipher like DES [21] for lightweight applications. Finally, the third category features new low-cost designs, including lightweight block ciphers HIGHT [22], mCrypton [23], SEA [24], PRESENT [25] and KATAN and KTANTAN [26], as well as lightweight stream ciphers Grain [27], Trivium [28] and MICKEY [29]. A good survey covering recently published lightweight cryptographic implementations can be found in [30]. Hummingbird is a recently proposed ultra-lightweight cryptographic algorithm targeted for low-cost smart devices [7, 31]. It has a hybrid structure of block cipher and stream cipher and was developed with both lightweight software and lightweight hardware implementations for constrained devices in mind. The hybrid model can provide the designed security with small block size and is therefore expected to meet the stringent response time and power consumption requirements for a large variety of embedded applications. Moreover, Hummingbird has been shown to be resistant to the most common attacks to block ciphers and stream ciphers including birthday attack, differential and linear cryptanalysis, structure attacks, algebraic attacks, cube attacks, etc. [7].

Implementation on FPGA is a good choice to verify effectiveness of a cryptographic algorithm in hardware platform. Field Programmable Gate Arrays (FPGAs) are programmable logic devices which have proven to be highly feasible implementation platforms for cryptographic algorithms because they provide both speed and programmability. FPGAs consist of reconfigurable functional units, reconfigurable interconnections, and flexible interface. Reconfigurable functional units are used for implementing the logic needed in a design and they are connected with the reconfigurable interconnections. Interfacing is used for communication with the rest of a system. FPGA implementation of Hummingbird cryptographic algorithm on Xilinx devices has been presented [6]. There is another platform of FPGA i.e. Altera which is used by a major group of researchers. Altera FPGAs are ideal for a wide variety of applications, from high-volume applications to state-of-the-art products. Each series of FPGA includes different features, such as embedded memory, digital signal processing (DSP) blocks, high-speed transceivers, or high-speed I/O pins, to cover a broad range of end products. Altera FPGAs offer a wide variety of configurable embedded SRAM, high-speed transceivers, high-speed I/Os, logic blocks, and routing. Built in intellectual property (IP) combined with outstanding software tools lower FPGA development time, power and cost. In this project Hummingbird Cryptographic algorithm implemented on Altera platform. Altera FPGAs have a number of family/series of products Stratix, Arria, Cyclone, MAX. In this work, we used Device of Cyclone II family for Low Cost.

The testability of the cryptographic cores brings in an extra dimension to the process of digital circuits testing – security. When chip is complex then Built-In-Self-Test (BIST) is a norm of this day because external testing using ATE is not cost effective in this case. BIST in Hummingbird crypto core is not reported yet. In this project we proposes and implements BIST in Hummingbird Crypto core. Here to detect circuit faults, a set of pseudorandom test vectors is applied to a CUT and then the output responses of the CUT are compared with that of a fault-free CUT. Fault-free output data is stored in the memory of the tester. If they are identical then the CUT is certified as fault-free otherwise as faulty. With the increase of complexities in ICs, bit-by-bit comparison is being more difficult and time consuming. Moreover, if the circuit size is large and complex, it takes much memory to store the output response data of a fault-free circuit and thereby increases the cost of IC testing. In order to overcome the problem, data compression techniques are usually used for test response evaluation. The choice of a compression scheme is influenced by two factors such as (a) the amount of circuitry required to implement the scheme and (b) the loss of data due to aliasing errors. Some of the important compression schemes such as (a) one's counting, (b) transition counting, (c) parity checking and (d) signature analysis. In this work, we used Signature analysis as the compression schemes for its simplicity in hardware implementation and good test coverage.

1.3 Contributions of this Project

The aim of this project is to develop a prototype ASIC to implement Hummingbird Cryptographic Algorithm with BIST.

To meet this goal, the following objectives have been identified:

- To design the modules of the Hummingbird Crypto ASIC using Verilog HDL
- To simulate the design on Altera FPGA platform
- To perform the performance analysis in terms power, speed and hardware resources
- To compare the results with that of other researchers

1.4 Project Outline

The rest of the project is organized as follows: Chapter 2 provides the background information on basic mathematics of Hummingbird algorithm which is required for understanding the fundamental operations of different states of Hummingbird algorithm. This chapter also presents brief overview of the algorithm including its cipher and deciphers parts. Chapter 3 discusses about the FPGA implementation of the proposed design. The design components are also provided in this chapter. Chapter 4 discusses about the experimental results and discussion on power analysis and measurement of the proposed design.

Finally, Chapter 5 offers suggestions for future work along with concluding remarks.

Chapter 2

Fundamentals of Hummingbird Cryptographic Algorithm

2.1 Introduction

A cryptographic algorithm is the mathematical function used for encrypting and decrypting messages. A modern cryptographic algorithm always includes a key. A cryptographic algorithm, plain texts, cipher texts, and keys are referred to as cryptosystem. The message, which is to be kept in secret, is referred to as plain text. The process of hiding its content is called encryption and the encrypted message is referred to as cipher text. The process of receiving the content of plain text back from cipher text is decryption.

The techniques used were Secret-Key Cryptography, Public-Key Cryptography and Elliptic Curve cryptography which were based on ASIC and used General purpose processors for the task. Then came FPGA which worked by combining them.

2.2 Basic Mathematics for Hummingbird

All the operations performed in Hummingbird Cryptographic algorithm are based on modulo-2 operations. These operations are not the same operations used in general number system. The basic operations based on which the entire math of the Hummingbird developed are Addition and Subtraction of modulo operation. These operations are explained in the subsequent sections.

2.3 Addition

In modulo-2 additions, two elements are added by adding the coefficients of the corresponding powers in the polynomial [5]. The addition operation here is the XOR operation denoted by the symbol „^". Subtraction of the polynomial is exactly the same as addition. Like other cryptographic algorithm, in the Hummingbird also the finite field algorithm of Galois Field, GF (2^n) is used for addition/subtraction.

The following are equivalent representations of the same value in a characteristic 2 finite field:

Polynomial: $x^6 + x^4 + x + 1$

Binary: {01010011}

Hexadecimal: {53}

Addition and subtraction are performed by adding or subtracting two of these polynomials together, and reducing the result modulo the characteristic. In a finite field with characteristic 2, addition modulo 2, subtraction modulo 2, and XOR are identical. Thus,

Polynomial: $(x^6 + x^4 + x + 1) + (x^7 + x^6 + x^3 + x) = x^7 + x^4 + x^3 + 1$

Binary: {01010011} + {11001010} = {10011001}

Hexadecimal: {53} + {CA} = {99}

Notice that under regular addition of polynomials, the sum would contain a term $2x^6$, but that this term becomes $0x^6$ and is dropped when the answer is reduced modulo 2.

Here is a table with both the normal algebraic sum and the characteristic 2 finite field sum of a few polynomials:

TABLE 2.1: COMPARISON OF NORMAL AND FINITE FIELD ALGEBRA

p_1	p_2	$p_1 + p_2$ **(normal algebra)**	$p_1 + p_2$ **in GF(2^n)**
$x^3 + x + 1$	$x^3 + x^2$	$2x^3 + x^2 + x + 1$	$x^2 + x + 1$
$x^4 + x^2$	$x^6 + x^2$	$x^6 + x^4 + 2x^2$	$x^6 + x^4$
$x + 1$	$x^2 + 1$	$x^2 + x + 2$	$x^2 + x$
$x^3 + x$	$x^2 + 1$	$x^3 + x^2 + x + 1$	$x^3 + x^2 + x + 1$
$x^2 + x$	$x^2 + x$	$2x^2 + 2x$	0

2.4 Modular arithmetic

In mathematics, modular arithmetic is a system of arithmetic for integers, where numbers "wrap around" upon reaching a certain value—the modulus. The modern approach to modular arithmetic was developed by Carl Friedrich Gauss in his book *Disquisitiones Arithmeticae*, published in 1801.

The foundations of modular arithmetic were introduced in the third century BCE, by Euclid, in the 7th book of his Elements.

2.4.1 Congruence relation

Modular arithmetic can be handled mathematically by introducing a congruence relation on the integers that is compatible with the operations on integers: addition, subtraction, and multiplication. For a positive integer n, two integers a and b are said to be congruent modulo n, written:

$a \equiv b \pmod{n}$

if their difference $a-b$ is an integer multiple of n (or n divides $a-b$). The number n is called the modulus of the congruence.

For example,

$38 \equiv 14 \pmod{12}$

because $38-14=24$, which is a multiple of 12.

The same rule holds for negative values:

$-8 \equiv 7 \pmod{5}$

$2 \equiv -3 \pmod{5}$

$-3 \equiv -8 \pmod{5}$

Equivalently, $a \equiv b \pmod{n}$ can also be thought of as asserting that the remainders of the division of both a and b by n are the same. For instance:

$38 \equiv 14 \pmod{12}$

Because both 38 and 14 have the same remainder 2 when divided by 12, it is also the case that $38-14=24$ is an integer multiple of 12, which agrees with the prior definition of the congruence relation.

2.4.2 Remainders

The notion of modular arithmetic is related to that of the remainder in Euclidean division. The operation of finding the remainder is sometimes referred to as the modulo operation, and denoted with "mod" used as an infix operator. For example, the remainder of the division of 14 by 12 is denoted by 14 mod 12; as this remainder is 2, we have 14 mod 12 = 2.

The congruence, indicated by "≡" followed by "mod" between parentheses, means that the operator "mod", applied to both members, gives the same result. That is

$A \equiv B \pmod{n}$

is equivalent to

$A \bmod n \equiv B \bmod n$

The fundamental property of multiplication in modular arithmetic may thus be written

$(a \bmod n)(b \bmod n) \equiv ab \pmod{n}$

or, equivalently,

$((a \bmod n)(b \bmod n)) \bmod n \equiv (ab) \bmod n$

In computer science, it is the remainder operator that is usually indicated by either "%" (e.g., in C, C++, Java, JavaScript, Perl and Python) or "mod" (e.g., in Pascal, BASIC, SQL, Haskell, ABAP), with exceptions (e.g., Excel). These operators are commonly pronounced as "mod", but it is specifically a remainder that is computed. The function modulo instead of mod, like 38 ≡ 14 (modulo 12) is sometimes used to indicate the common residue rather than a remainder.

2.5 Overview of Hummingbird Cryptographic Algorithm

The design of Hummingbird is based on an elegant combination of a block cipher and stream cipher with 16-bit block size, 256-bit key size, and 80-bit internal state. Figure 1(a) and Figure 1(b) illustrate the initialization and encryption processes of the Hummingbird cryptographic algorithm, respectively.

Both initialization and encryption consist of four 16-bit block ciphers E_{ki} ($i = 1, 2, 3, 4$), four 16-bit internal state registers RS_i ($i = 1, 2, 3, 4$), and a 16-stage Linear Shift Feedback Register (LFSR). Moreover, the 256-bit secret key K is divided into four 64-bit subkeys $k1$, $k2$, $k3$ and $k4$ which are used in the four block ciphers, respectively.

TABLE 2.2: NOTATION

PT_i	the i-th 16-bit plaintext block, i = 1; 2; : : : ; n
CT_i	the i-th 16-bit ciphertext block, i = 1; 2; : : : ; n
K	the 256-bit secret key
$\mathbf{E}_K(\cdot)$	the encryption function of Hummingbird with 256-bit secret key K
$\mathbf{D}_K(\cdot)$	the decryption function of Hummingbird with 256-bit secret key K
k_i	the 64-bit subkey used in the i-th block cipher, i = 1; 2; 3; 4, such that K = k1‖k2‖k3‖k4
$E_{ki}(\cdot)$	a block cipher encryption algorithm with 16-bit input, 64-bit key ki, and 16-bit output, i.e., $E_{ki} : \{0; 1\}^{16} \times \{0; 1\}^{64} \rightarrow \{0; 1\}^{16}$; i = 1; 2; 3; 4
$D_{ki}(\cdot)$	a block cipher decryption algorithm with 16-bit input, 64-bit key ki, and 16-bit output, i.e., $D_{ki} : \{0; 1\}^{16} \times \{0; 1\}^{64} \rightarrow \{0; 1\}^{16}$; i = 1; 2; 3; 4
RSi	the i-th 16-bit internal state register, i = 1; 2; 3; 4
LFSR	a 16-stage Linear Feedback Shift Register with the characteristic polynomial $f(x) = x^{16} + x^{15} + x^{12} + x^{10} + x^7 + x^3 + 1$
$\boxplus$	modulo 2^{16} addition operator
$\boxminus$	modulo 2^{16} subtraction operator
$\oplus$	exclusive-OR (XOR) operator
$m \lll l$	left circular shift operator, which rotates all bits of m to the left by l bits, as if the left and the right ends of m were joined.
$K_i^{(i)}$	the j-th 16-bit key used in the i-th block cipher, j = 1; 2; 3; 4, such that $k_i = K_1^{(i)} \| K_2^{(i)} \| K_3^{(i)} \| K_4^{(i)} \|$
$S_i(x)$	the i-th 4-bit to 4-bit S-box used in the block cipher, $S_i(x) : F_2^4 \rightarrow F_2^4$
$NONCE_i$	the i-th nonce which is a 16-bit random number, i = 1, 2, 3, 4
IV	the 64-bit initial vector, such that IV = $NONCE_1 \| NONCE_2 \| NONCE_3 \| NONCE_4$

2.5.1 16-Bit Block Cipher:

Four identical 16-bit block ciphers are employed in a consecutive manner in the Hummingbird encryption scheme. The 16-bit block cipher is a typical substitution-permutation (SP) network with 16-bit block size and 64-bit key as shown in Figure 2. It consists of four regular rounds and a final round that only includes the key mixing and the S-box substitution steps. The 64-bit subkey k_i is split into four 16-bit round keys $K_1^{(i)}$, $K_2^{(i)}$, $K_3^{(i)}$, $K_4^{(i)}$, which are used in the four regular rounds, respectively. Moreover, the final round utilizes two keys $K_5^{(i)}$, $K_6^{(i)}$, directly derived from the four round keys (see Figure 2). Like any other SP network, one regular round comprises of three stages: a key mixing step, a substitution layer, and a permutation layer. For the key mixing, a simple exclusive-OR operation is used in this 16-bit block cipher for efficient implementation in both software and hardware. The final round only includes the key mixing and the S-box substitution steps. The key mixing step is implemented using a simple exclusive-OR operation, whereas the substitution layer is composed of four S-boxes with 4-bit inputs and 4-bit outputs as shown in Table 2.3.

TABLE 2.3
FOUR S-BOXES IN HEXADECIMAL NOTATION

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S1(x)	8	6	5	F	1	C	A	9	E	B	2	4	7	0	D	3
S2(x)	0	7	E	1	5	B	8	2	3	A	D	6	F	C	4	9
S3(x)	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D
S4(x)	0	7	3	4	C	1	A	F	D	E	6	B	2	8	9	5

The permutation layer in this 16-bit block cipher is given by the linear transform $L: \{0; 1\}^{16} \rightarrow \{0; 1\}^{16}$ defined as follows:

$L(m) = m \oplus (m \ll 6) \oplus (m \ll 10);$

where $m = (m0; m1; \cdots; m15)$ is a 16-bit data block.

Algorithm 4 16-bit Block Cipher Encryption E_{ki} (▪)

```
Input: A 16-bit data block m = ($m_0, m_1, \ldots, m_{15}$) and a 64-bit subkey $k_i$ such that
subkey $k_i = K_1^{(i)} \| K_2^{(i)} \| K_3^{(i)} \| K_4^{(i)} \|$
Output: A 16-bit date block $m' = (m_0', m_1', \ldots, m_{15}')$
1: for j = 1 to 4 do
2: $m \leftarrow m \oplus K_j^{(i)}$                    [key mixing step]
3: $A = m_0 \| m_1 \| m_2 \| m_3$; $B = m_4 \| m_5 \| m_6 \| m_7$
   $C = m_8 \| m_9 \| m_{10} \| m_{11}$; $D = m_{12} \| m_{13} \| m_{14} \| m_{15}$
4: $m \leftarrow S_1(A) \| S_2(B) \| S_3(C) \| S_4(D)$          [substitution layer]
5: $m \leftarrow m \oplus (m \ll 6) \oplus (m \ll 10)$   permutation layer]
6: end for
7: $m \leftarrow m \oplus K_1^{(i)} \oplus K_3^{(i)}$
8: $A = m_0 \| m_1 \| m_2 \| m_3$; $B = m_4 \| m_5 \| m_6 \| m_7$
```

```
C = $m_8 \| m_9 \| m_{10} \| m_{11}$; D = $m_{12} \| m_{13} \| m_{14} \| m_{15}$
9: $m \leftarrow S_1(A) \| S_2(B) \| S_3(C) \| S_4(D)$
10: $m' \leftarrow m \oplus K_2^{(i)} \oplus K_4^{(i)}$
11: return $m' = (m_0', m_1' \ldots. m_{15}')$
```

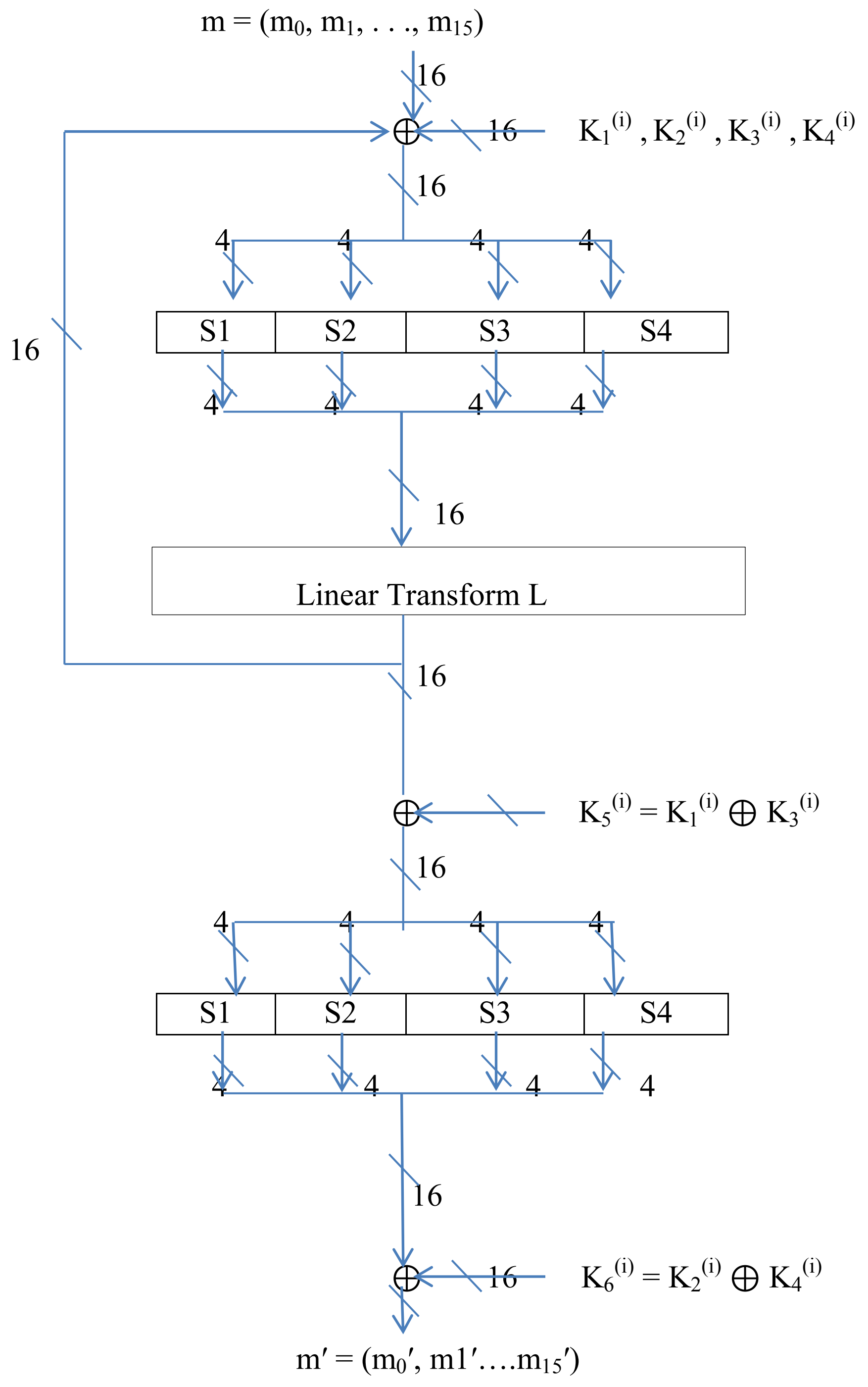
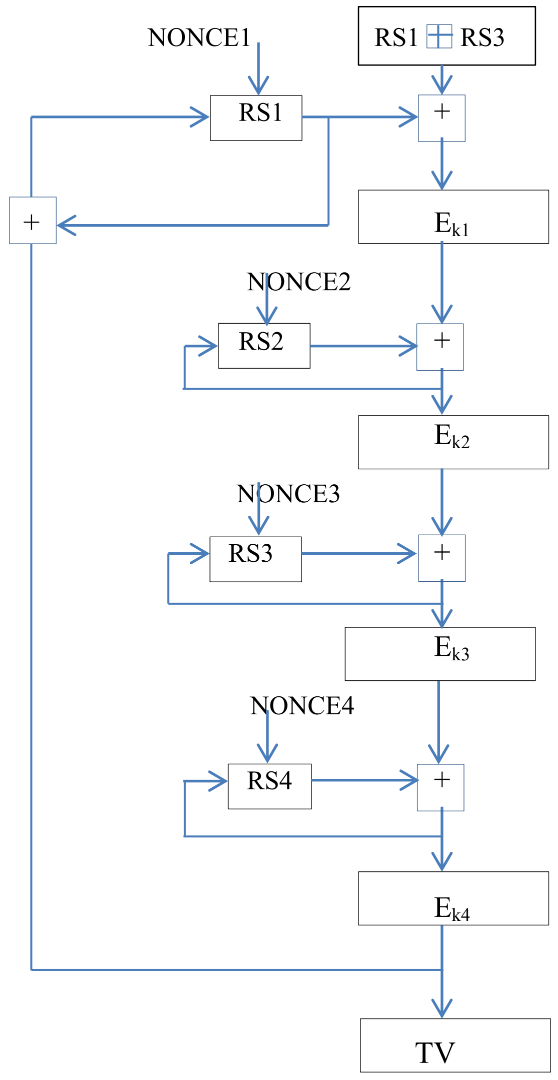
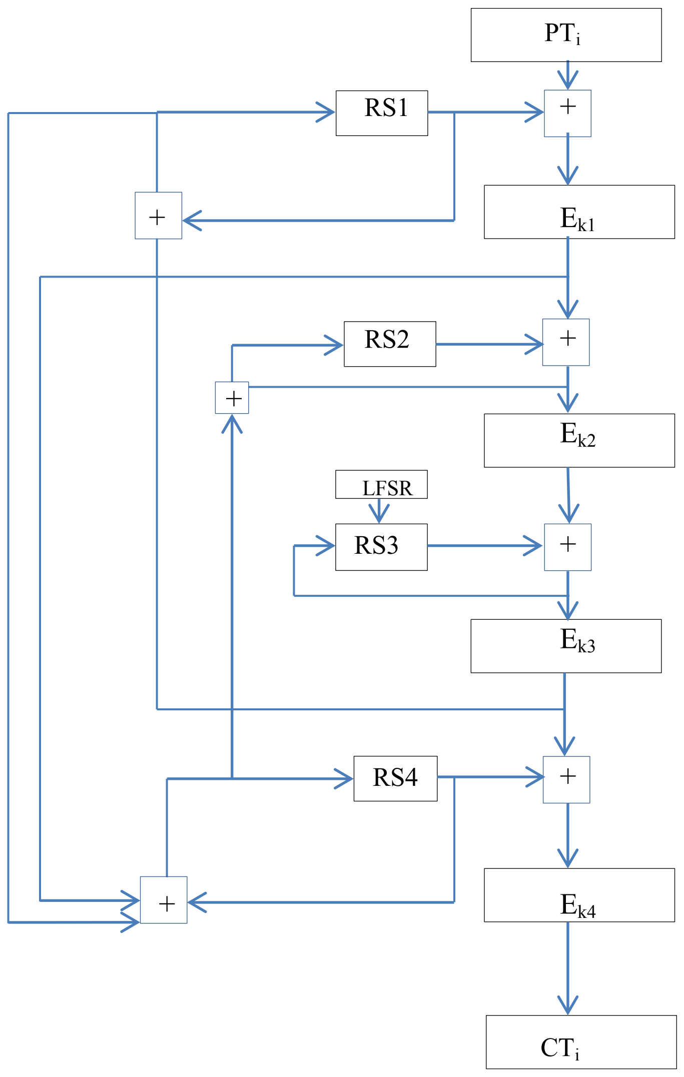

$m = (m_0, m_1, \ldots, m_{15})$
16
16
$K_1^{(i)}, K_2^{(i)}, K_3^{(i)}, K_4^{(i)}$
16
4
S1
S2
S3
S4
16
Linear Transform L
$K_5^{(i)} = K_1^{(i)} \oplus K_3^{(i)}$
$K_6^{(i)} = K_2^{(i)} \oplus K_4^{(i)}$
$m' = (m_0', m1' \ldots. m_{15}')$

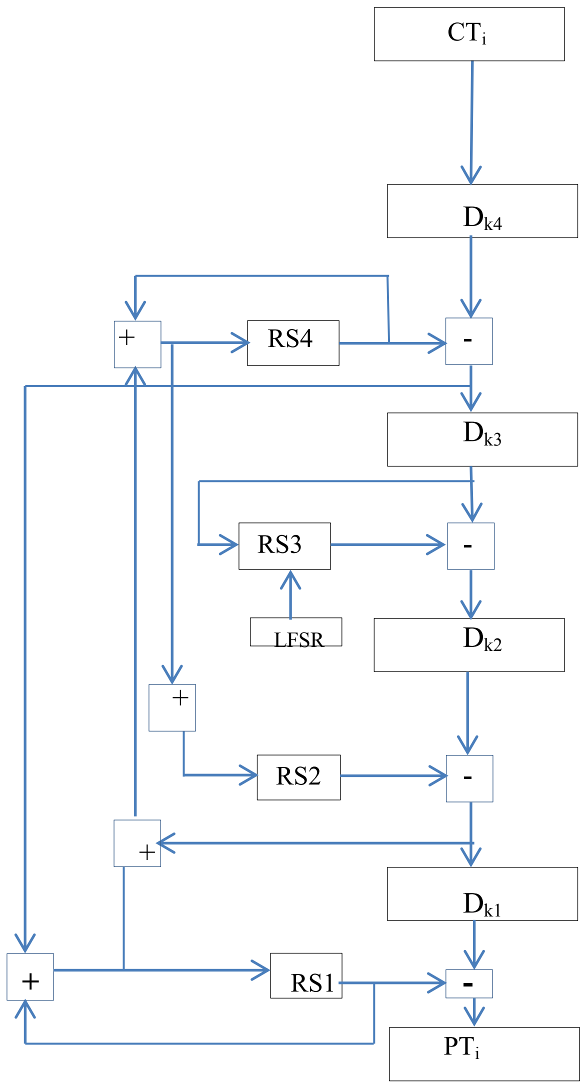
Fig. 2.1 Structure of Block Cipher in the Hummingbird Cryptographic Algorithm



a) Initialization Process



b) Encryption Process



c) Decryption Process

Fig. 2.2 A Top-Level Description of the Hummingbird Cryptographic Algorithm

2.5.2 Initialization Process

The overall structure of the Hummingbird initialization algorithm is shown in Figure 1(a).When using Hummingbird in practice, four 16-bit random nonces NONCEi are first chosen to initialize the four internal state registers RSi (i = 1; 2; 3; 4), respectively, followed by four consecutive encryptions on the message RS1 $\boxplus$ RS3 by Hummingbird running in initialization mode (see Figure 1(a)). The final 16-bit ciphertext *TV* is used to initialize the LFSR. Moreover, the 13th bit of the LFSR is always set to prevent a zero register. The LFSR is also stepped once before it is used to update the internal state register *RS*3.We can summarize the Hummingbird initialization process in the following Algorithm 1.

Algorithm 1 Hummingbird Initialization

```
Input: Four 16-bit random nonce $NONCE_i$ (i = 1; 2; 3; 4)
Output: Initialized four rotors $RSi_4$ (i = 1; 2; 3; 4) and LFSR
1: $RS1_0 = NONCE_1$                    [Nonce Initialization]
2: $RS2_0 = NONCE_2$
3: $RS3_0 = NONCE_3$
4: $RS4_0 = NONCE_4$
5: for t = 0 to 3 do
6: $V12_t = E_{k1} ((RS1_t \boxplus RS3_t) \boxplus RS1_t)$
7: $V23_t = E_{k2} (V12_t \boxplus RS2_t)$
8: $V34_t = E_{k3} (V23_t \boxplus RS3_t)$
9: $TV_t = E_{k4} (V34_t \boxplus RS4_t)$
10: $RS1_{t+1} = RS1_t \boxplus TV_t$
11: $RS2_{t+1} = RS2_t \boxplus V12_t$
12: $RS3_{t+1} = RS3_t \boxplus V23_t$
13: $RS4_{t+1} = RS4_t \boxplus V34_t$
14: end for
15: LFSR = TV3 | 0x1000                  [LFSR Initialization]
16: return $RSi_4$ (i = 1; 2; 3; 4) and LFSR
```

2.5.3 Encryption Process

The overall structure of the Hummingbird encryption algorithm is depicted in Figure 1(b).After a system initialization process, a 16-bit plaintext block PT_i is encrypted by first executing a modulo 2^{16} addition of PT_i and the content of the first internal state register *RS*1.
The result of the addition is then encrypted by the first block cipher E_{k1}. This procedure is

repeated in a similar manner for another three times and the output of E_{k4} is the corresponding ciphertext CT_i . Furthermore, the states of the four internal state registers will also be updated in an unpredictable way based on their current states, the outputs of the first three block ciphers, and the state of the LFSR. Algorithm 2 describes the detailed procedure of Hummingbird encryption.

Algorithm 2 Hummingbird Encryption

```
Input: A 16-bit plaintext $PT_i$ and four rotors $RSi_t$ (i = 1; 2; 3; 4)
Output: A 16-bit ciphertext $CT_i$
1: $V12_t = Ek1 (PTi \boxplus RS1_t)$            [Block Encryption]
2: $V23_t = Ek2 (V12_t \boxplus RS2_t)$
3: $V34t = Ek3 (V23_t \boxplus RS3_t)$
4: $CT_i = Ek4 (V34_t \boxplus RS4t)$
5: $LFSR_{t+1} \leftarrow LFSR_t$                [Internal State Updating]
6: $RS1_{t+1} = RS1t \boxplus V34t$
7: $RS3_{t+1} = RS3_t \boxplus V23t \boxplus LFSR_{t+1}$
8: $RS4_{t+1} = RS4_t \boxplus V12t \boxplus RS1_{t+1}$
9: $RS2_{t+1} = RS2_t \boxplus V12t \boxplus RS4_{t+1}$
10: return $CT_i$
```

2.5.4 Decryption Process

The overall structure of the Hummingbird decryption algorithm is illustrated in Figure 1(c). The decryption process follows the similar pattern as the encryption and a detailed description is shown in the following Algorithm 3.

Algorithm 3 Hummingbird Decryption

```
Input: A 16-bit ciphertext $CT_i$ and four rotors $RSi_t$ (i = 1; 2; 3; 4)
Output: A 16-bit plaintext $PT_i$
1: $V34_t = D_{k4} (CT_i) \boxminus RS4_t$          [Block Decryption]
2: $V23_t = D_{k3} (V34_t) \boxminus RS3_t$
3: $V12_t = D_{k2} (V23_t) \boxminus RS2_t$
4: $PT_i = Dk1 (V12_t) \boxminus RS1_t$
5: $LFSR_{t+1} \leftarrow LFSR_t$                [Internal State Updating]
6: $RS1_{t+1} = RS1_t \boxminus V34_t$
7: $RS3_{t+1} = RS3_t \boxminus V23t \boxminus LFSR_{t+1}$
8: $RS4_{t+1} = RS4_t \boxminus V12_t \boxminus RS1_{t+1}$
9: $RS2_{t+1} = RS2_t \boxminus V12t \boxminus RS4_{t+1}$
10: return $PT_i$
```

2.6 Linear Feedback Shift Register (LFSR)

LFSR is widely used for test pattern generation. It is considered as the simple and most efficient pseudo-random test pattern generator (Dufaza 1998). Figure 2.3 shows the basic structure of a standard LFSR. It consists of a set of storage elements (D-Flip-Flops) and modulo-2 adder (X-OR gate). The connection is in such a way that the state of each element is shifted to the next element with the application of clock signal.

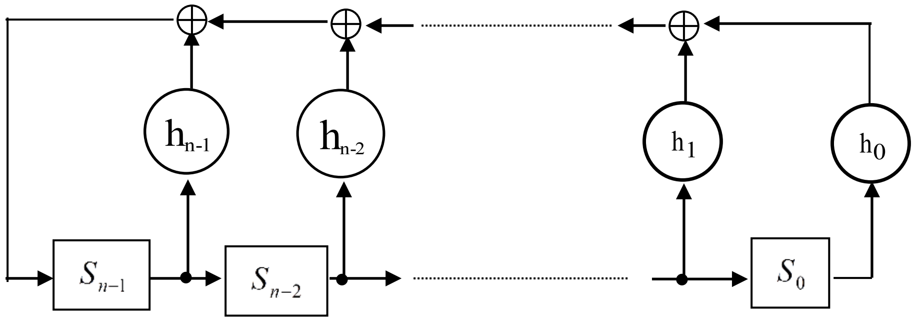


Fig. 2.3: General structure of an n-bit LFSR

In Figure 2.3, all the operations are in Galois Field GF (2). S= (S_0, S_1, ………., S_{n-1}), the binary n-tuples, represents the state of the LFSR. It can be represented in the polynomial form as follows:

$$S(x) = S_0 + S_1 x + \dots\dots\dots\dots + S_{n-2} x^{n-2} + S_{n-1} x^{n-1} \tag{2.1}$$

where x^i denotes the ith stage of the LFSR. For example, x^0 represents stage 0, x^1 represents stage 1 and so on. Feedback function of the LFSR is called the feedback polynomial or the generator polynomial and can be represented as follows:

$$h(x) = h_0 + h_1 x^1 + \dots\dots\dots + h_{n-1} x^{n-1} + x^n \tag{2.2}$$

where $h_i \in \{1,0\}$ denotes the feedback tap in the ith stage of the LFSR. $h_i = 0$ means there exists no feedback link in the ith stage whereas $h_i = 1$ means there exists feedback link in that stage.

With the application of clock signal, the LFSR goes into autonomous mode. The past state of the LFSR (S(x)) changes to a new state and generates pseudo-random patterns (Chen 1988; Lin and Costello 1983). If the period of the LFSR is u then the LFSR returns to the initial state after u number of shifts. The period (u) of the LFSR depends on the feedback polynomial and initial state. If the initial state is all zero then u will be 1 meaning that the state remains unchanged. Again if the initial state of the LFSR is non-zero and the feedback polynomial is primitive then u becomes near exhaustive (Bardell et al. 1987).

The states of an LFSR can be represented in matrix form. It is helpful in computation and analysis of the pseudo-random sequences generated from it. Matrix representation of the states of the LFSR is illustrated as follows:

Matrix Representation of LFSR:

Let"s assume the present state of an n-stage LFSR (at time t) is $X_0(t), X_1(t), X_2(t), ..., X_{n-1}(t)$ then the next state of the LFSR (at time t+1) $X_0(t+1), X_1(t+1), X_2(t+1), ..., X_{n-1}(t+1)$ can be written as follows (Wang and McCluskey 1988):

$$\begin{bmatrix} X_0(t+1) \\ X_1(t+1) \\ \cdot \\ \cdot \\ \cdot \\ X_{n-3}(t+1) \\ X_{n-2}(t+1) \\ X_{n-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & ... & 0 & 0 \\ 0 & 0 & 1 & ... & 0 & 0 \\ \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & \cdot & ... & \cdot & \cdot \\ \cdot & \cdot & \cdot & & \cdot & \cdot \\ 0 & 0 & 0 & ... & 1 & 0 \\ 0 & 0 & 0 & ... & 0 & 1 \\ h_0 & h_1 & h_2 & ... & h_{n-2} & h_{n-1} \end{bmatrix} \begin{bmatrix} X_0(t) \\ X_1(t) \\ \cdot \\ \cdot \\ \cdot \\ X_{n-3}(t) \\ X_{n-2}(t) \\ X_{n-1}(t) \end{bmatrix} \tag{2.3}$$

$$\text{i.e. } X(t+1) = CX(t) \tag{2.4}$$

where X(t), X(t+1) are n-by-1 state matrix representing the present state and the next state of the LFSR respectively and C is n-by-n companion matrix of the LFSR. In the companion matrix, $h_i (0 \le i \le n-1)$ is either 1 or 0, depending on the existence or absence of the feedback path in the LFSR. If companion matrix of the LFSR is known then the states traveled by the LFSR starting from a non-zero initial state X(t) can be calculated as follows: $X, CX, C^2X, C^3X, ...$ If the period of the LFSR is u then $XC^u = X$, where u is the smallest integer for which $C^u = I$ (I is an n-by-n identity matrix). However, if the initial state of the

LFSR is zero then u is always 1 and the state of the LFSR remains unchanged and independent of C.

The companion matrix can be derived from the generator polynomial of the LFSR. It can be proven in the following way:

The characteristic polynomial of the companion matrix (C), say f(x), can be determined from the determinant of C-IX which is given by:

$$f(x) = |C - IX| \tag{2.5}$$

Since in modulo-2 arithmetic, addition and subtraction are equivalent, determinant of C-IX is same as the determinant of C+IX. Hence the Equation 2.7 can be rewritten as:

$$f(x) = | C - IX | = | C + IX | = h_0 + h_1 x^1 + \ldots\ldots\ldots\ldots\ldots + h_{n-1} x^{n-1} + x^n \tag{2.6}$$

Equation 2.6 and Equation 2.2 prove that companion matrix can be obtained from the generator polynomial of an LFSR.

Again the same characteristic polynomial, f(x), as in Equation 2.6, can be obtained by substituting the transpose of C (C^t) in Equation 2.5 where

$$C^t = \begin{bmatrix} 0 & 0 & \ldots & 0 & 0 & h_0 \\ 1 & 0 & \ldots & 0 & 0 & h_1 \\ 0 & 1 & \ldots & 0 & 0 & h_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \ldots & 1 & 0 & h_{n-2} \\ 0 & 0 & \ldots & 0 & 1 & h_{n-1} \end{bmatrix} \tag{2.7}$$

Hence Equation 2.4 can be rewritten as:

$$X(t+1) = CX(t) = C^t X(t) \tag{2.8}$$

Hence the states of a LFSR can be written in the matrix form by obtaining the companion matrix or its transpose from the generator polynomial of the LFSR.

2.7 Theory of Data Compression Technique

To detect circuit faults, a set of test vectors is applied to a CUT and then the output responses of the CUT are compared with that of a fault-free CUT. Fault-free output data is stored in the memory of the tester. If they are identical then the CUT is certified as fault-free otherwise as faulty. With the increase of complexities in ICs, bit-by-bit comparison is being more difficult and time consuming. Moreover, if the circuit size is large and complex, it takes much memory to store the output response data of a fault-free circuit and thereby increases the cost of IC testing. In order to overcome the problem, data compression techniques are usually used for test response evaluation. Figure 2.4 shows a generalized block diagram of data compression scheme.

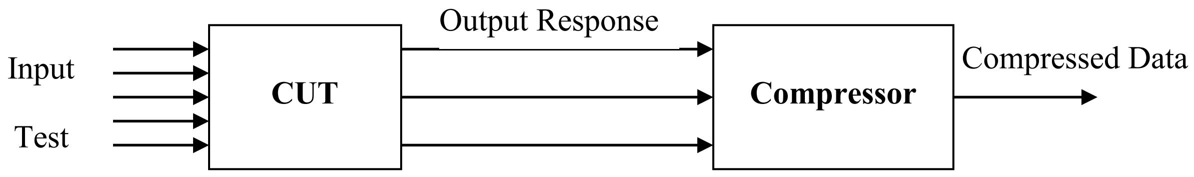


Fig. 2.4: Generalized data compression scheme

The choice of a compression scheme is influenced by two factors such as (a) the amount of circuitry required to implement the scheme and (b) the loss of data due to aliasing errors. Some of the important compression schemes are (a) one‟s counting, (b) transition counting, (c) parity checking and (d) signature analysis.

In our project, we have used signature analysis which is described in below:

2.7.1 Signature Analysis

Signature analysis is a widely used data compression technique. It is popular because of its simplicity in hardware implementation and good test coverage (David 1986; Bardel et al. 1987; Ivanov and Agrawal 1989). Signature Analyzer (SA) consists of an LFSR and a modulo-2 adder, M-2, as shown in Figure 2.5.

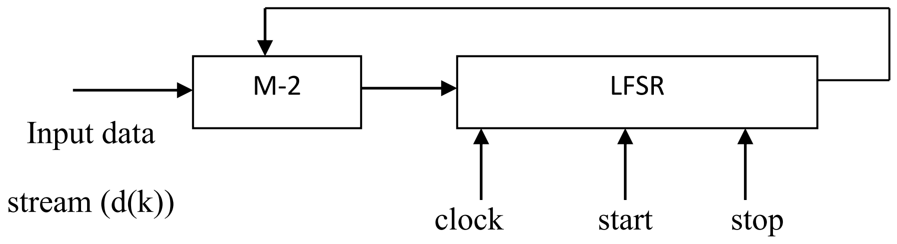


Fig. 2.5: Block diagram of a signature analyzer

The „start" and the „stop"signal establish the time window within which the SA performs data compression. The start signal initializes all the memory elements to zero and after that, the SA starts shifting the input data stream to be compressed bit by bit with the edge of every clock cycle. On each clock cycle, the 1st memory position of the SA is loaded by the incoming data and shifted towards right.

Let"s assume an input data stream of length l is applied at the input of the SA. When l clock cycles elapse, the binary data left in the memory of the SA is the signature of the input data sequence. In the signature analysis approach, cyclic redundancy checking (CRC) technique is used for data compression (Bardell et al. 1987). In this technique, the input data sequences are divided by the feedback polynomial of the SA and the remainder from the division is considered the signature of the input data sequences. Let"s the polynomial representation of an m-bit input data stream ($d_j \in \{0,1\}, j = 0,1,.......,m-1$) and the feedback connection of a r-stage SA ($h_i \in \{0,1\}, i = 0,1,............,r-1$) are as follows respectively:

$$d(x) = d_0 + d_1x^1 + d_2x^2 + ...+ d_{m-1}x^{m-1} \quad (2.9)$$

$$h(x) = h_0 + h_1x^1 + h_2x^2 + ...+ h_rx^r \quad (2.10)$$

where m > r.

Polynomial division of d(x) by h(x) is as follows:

$$\frac{d(x)}{h(x)} = Q(x) + \frac{R(x)}{h(x)} \tag{2.11}$$

$$\text{i.e. } R(x) = d(x) + Q(x).h(x) \tag{2.12}$$

In the Equation (2.12), Q(x) and R(x) indicate quotient and remainder respectively.

Figure 2.6 shows an example of a 5-stage single input SA with feedback polynomial $1+x+x^3+x^5$. Binary representation of the feedback polynomial of the SA is 101011.

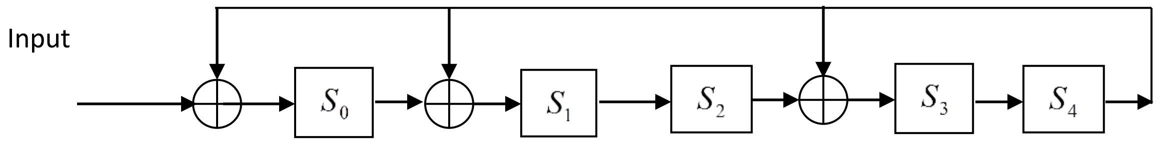


Fig. 2.6: Block diagram of a 5-stage single input signature analyzer

Let the data stream to be compressed is 10010101. The division of 10010101 by 101011 is as follows:

```
          101 = Q          Q(x) = x^2 + 1
        ____________
101011 | 10010101
         101011
         --------
          00111001          R(x) = x^4 + x
```

In the above division, the remainder i.e. signature is 10010. Table 2.4 shows the timing chart of the states of SA‟s memory element. The SA is initially set to zero. The data to be compressed is serially fed to the SA‟s input with the edge of every clock cycle, high order bit first, the content of the SA after the last input data bit is the remainder (signature) of the input data.

TABLE 2.4: STATES OF THE SIGNATURE ANALYZER DURING DIVISION OF INPUT DATA=10010101

clock	input	s_0	s_1	s_2	s_3	s_4	
0		0	0	0	0	0	$\leftarrow$ Initial state
1	1	1	0	0	0	0	
2	0	0	1	0	0	0	
3	0	0	0	1	0	0	
4	1	1	0	0	1	0	
5	0	0	1	0	0	1	
6	1	0	1	1	1	0	
7	0	0	0	1	1	1	
8	1	0	1	0	0	1	$\leftarrow$ Remainder

When multiple parallel data stream needs to be compressed, multiple input signature register (MISR) is preferred rather than using single input SA for each input data stream. Figure 2.7 shows 5-stage MISR with feedback polynomial $1+x+x^3+x^5$ where the modulo-2 adders are placed between two stages of MISR and inputs are inserted with the modulo-2 adder.

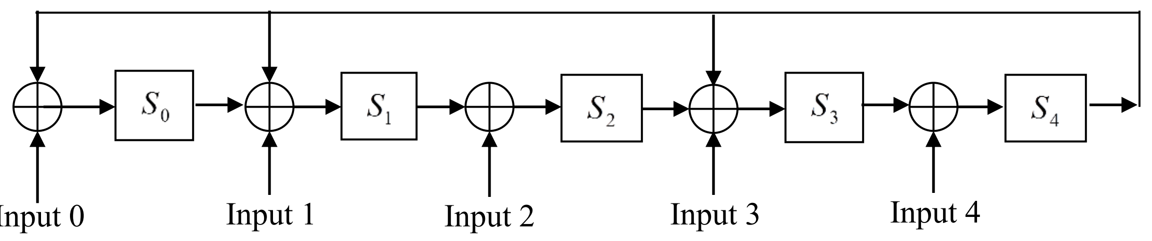


Figure 2.7: Block diagram of a 5-stage multiple input signature analyzer

Signature analysis is not a fault-free process. In the SA, an erroneous sequence from a faulty circuit in some cases is compressed into the same signature as that of the fault-free circuit of the same type. This phenomenon is known as „aliasing‟ or „masking‟ since the effect of fault is masked by the compression process in the SA (Bardell et al. 1987; Abramavoci et al. 1990). Masking is a loss of information caused by the compression of the output sequence of the CUT. Let‟s assume an n-bit output data and r-bit signature SA. For an n-bit output, the possible combination of the output sequence is 2^n. Since only one output sequence is correct out of these sequences, total numbers of possible erroneous sequence are $2^n - 1$. Again for r-bit signature register, possible number of signature is 2^r. If it is assumed that all signatures are equally likely, the number of bit sequences that generates a particular signature is given by (Abramavoci et al. 1990):

$$\frac{2^n}{2^r} = 2^{n-r} \tag{2.13}$$

For the signature of a particular fault-free response, there are $2^{n-r}-1$ erroneous bit sequences that produce the same signature. Since total number of the possible error sequences is 2^n-1, the probability of aliasing or masking is given by:

$$P_{al} = \frac{2^{n-r}-1}{2^n-1} \tag{2.14}$$

$$\text{If } n\rangle\rangle r,\ P_{al} \cong \frac{1}{2^r} \tag{2.15}$$

where P_{al} indicates the probability that incorrect response will go undetected, hence the probability of no masking:

$$1\text{-}P_{al} = 1-\frac{1}{2^r} = 1-2^{-r} \tag{2.16}$$

Hence aliasing error is possible to be made negligible by sufficiently increasing the value of r.

2.8 Overview of FPGA

Field Programmable Gate Array (FPGA) is a semiconductor device containing programmable logic components and interconnects. It contains up to thousands of gates. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple math functions. In most FPGA, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip-flops or more complete blocks of memories. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer (hence the “field programmable” i.e. programmable in the field) so that the FPGA can perform whatever logical function is needed. There are various vendor manufacturers for different types of FPGA chip such as Altera, Xilinx, Lattice Semiconductor, Actel, Quick Logic, Cypress Semiconductor, Atmel, Achromix Semiconductor etc. Among them Altera and Xilinx are the most famous FPGA companies since both of the companies have lot of varieties of FPGA device from small number of gate counts to higher number of gate counts. However Altera devices offer the general benefits of PLDs as innovative architectures, advanced process technologies, state-of- the-art development tools and a wide selection of mega function. The common advantages of Altera devices include: High performance, High density logic integration, Cost effectiveness, Short development cycles with the Quartus II software, Mega core functions, Benefits of in-system programming. In this project the used FPGA device is Altera provided EP2C5F256C7 from Cyclone II family.

2.8.1 FPGA Cyclone II Device

This subsection presents some basic information about this device which will help for development of the proposed ASIC with this Cyclone II device [32-33].

Cyclone II FPGAs are manufactured on 300-mm wafers using TSMC's 90-nm low-k dielectric process to ensure rapid availability and low cost. By minimizing silicon area, Cyclone II devices can support complex digital systems on a single chip at a cost that rivals that of ASICs. Unlike other FPGA vendors who compromise power consumption and performance for low-cost, Altera‟s latest generation of low-cost FPGAs—Cyclone II FPGAs, offer 60% higher performance and half the power consumption of competing 90-nm FPGAs. The low cost and optimized feature set of Cyclone II FPGAs make them ideal solutions for a wide array of automotive, consumer, communications, video processing, test and measurement, and other end-market solutions.

The Cyclone II device family offers the following features:

■ High-density architecture with 4,608 to 68,416 LEs

● M4K embedded memory blocks

● Up to 1.1 Mbits of RAM available without reducing available
logic

● 4,096 memory bits per block (4,608 bits per block including 512
parity bits)

● Variable port configurations of ×1, ×2, ×4, ×8, ×9, ×16, ×18, ×32,
and ×36

● True dual-port (one read and one write, two reads, or two
writes) operation for ×1, ×2, ×4, ×8, ×9, ×16, and ×18 modes

● Byte enables for data input masking during writes

● Up to 260-MHz operation

■ Embedded multipliers

● Up to 150 18- × 18-bit multipliers are each configurable as two
independent 9- × 9-bit multipliers with up to 250-MHz
performance

● Optional input and output registers

■ Advanced I/O support

● High-speed differential I/O standard support, including LVDS, RSDS, mini-LVDS, LVPECL, differential HSTL, and differential SSTL

● Single-ended I/O standard support, including 2.5-V and 1.8-V, SSTL class I and II, 1.8-V and 1.5-V HSTL class I and II, 3.3-V PCI and PCI-X 1.0, 3.3-, 2.5-, 1.8-, and 1.5-V LVCMOS, and 3.3-, 2.5-, and 1.8-V LVTTL

● Peripheral Component Interconnect Special Interest Group (PCI SIG) *PCI Local Bus Specification, Revision 3.0* compliance for 3.3-V operation at 33 or 66 MHz for 32- or 64-bit interfaces

● PCI Express with an external TI PHY and an Altera PCI Express ×1 Megacore® function

● 133-MHz PCI-X 1.0 specification compatibility

● High-speed external memory support, including DDR, DDR2, and SDR SDRAM, and QDRII SRAM supported by drop in Altera IP MegaCore functions for ease of use

● Three dedicated registers per I/O element (IOE): one input register, one output register, and one output-enable register

● Programmable bus-hold feature

● Programmable output drive strength feature

● Programmable delays from the pin to the IOE or logic array

● I/O bank grouping for unique VCCIO and/or VREF bank settings

● MultiVolt™ I/O standard support for 1.5-, 1.8-, 2.5-, and 3.3-interfaces

● Hot-socketing operation support

● Tri-state with weak pull-up on I/O pins before and during configuration

● Programmable open-drain outputs

● Series on-chip termination support

■ Flexible clock management circuitry

● Hierarchical clock network for up to 402.5-MHz performance

● Up to four PLLs per device provide clock multiplication and division, phase shifting, programmable duty cycle, and external clock outputs, allowing system-level clock management and skew control

● Up to 16 global clock lines in the global clock network that drive throughout the entire device

■ Device configuration

● Fast serial configuration allows configuration times less than 100 ms

● Decompression feature allows for smaller programming file storage and faster configuration times

● Supports multiple configuration modes: active serial, passive serial, and JTAG-based configuration

● Supports configuration through low-cost serial configuration devices

● Device configuration supports multiple voltages (either 3.3, 2.5, or 1.8 V)

■ Intellectual property

● Altera megafunction and Altera MegaCore function support, and Altera Megafunctions Partners Program (AMPPSM) megafunction support, for a wide range of embedded processors, on-chip and off-chip interfaces, peripheral functions, DSP functions, and communications functions and protocols.

● Nios II Embedded Processor support

2.9 Development Tool Quartus II

The proposed Hummingbird Crypto ASIC implementing BIST technique is designed using Quartus II EDA tool (provided by Altera Company). Quartus II enables analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation. Quartus II software provides a simple, automated mechanism to allow designers to obtain the best performance for their designs. This software provides the way to design the solution through Verilog HDL and compile the design to ensure the workability and efficiency logically.

Compiling mode: The Quartus II compiler consists of a set of independent modules that check the design for errors, synthesize the logic, fit the design into an Altera Device, and generate output files for simulation, timing analysis, software building and device programming. The basic compiler consists of the Analysis & Synthesis, Partition Merge, Fitter, Assembler and Classic Timing Analyzer modules. Each of the compiler modules can be run individually or together from the Quartus II user Interface. Alternatively, these modules can be run independently with the appropriate command line executable.

Compile the Design: The compiler automatically locate and uses all non- design files associated with the design, such as include files (.inc) containing AHDL, Function Prototype statements; Memory initialization files (.mif) or Hexadecimal intel format files (.hex) containing the initial content of the memories; as well as Quartus II Project Files (.qpf) and Quartus II Settings Files (.qsf) containing project and setting information. During compilation the Compiler generates information, warning and error messages that appear automatically in the Message window.

Simulation mode: Simulation allows testing a design thoroughly to ensure that it responds correctly in every possible situation before configuring a device. Depending on the type of information need, functional or timing simulation can be performed with the simulator. Functional simulation tests only the logical operation of a design by simulating the behavior of flattened netlist extracted from the design files, while timing simulation uses a fully compiled netlist containing information to test both the logical operation and the worst case timing for the design in the target device. Before running a simulation input vectors need to specify as the stimuli for the Quartus II simulator. The simulator uses these input vectors to simulate the output signals that a programmed device would produce under the same condition. The simulator supports input vector stimuli in the form a Vector Waveform File (.vwf), Vector Table File (.tbf), Power Input File (.pwf), or a Quartus II generated vector File (.vec) or Simulator Channel File (.scf).

Program an Altera Device: When the design is ready to program a device, it needs to open the Programmer and create a Chain Description File (.cdf) that stores the device name, device order, programming and hardware setup information. CDFscan can be used to program or configure one or more devices in a JTAG chain or a Passive Serial chain.

Chapter 3

Implementation of Hummingbird Crypto ASIC with BIST

3.1 Introduction

This chapter will discuss about the FPGA implementation of the proposed design and also intend procedure of the proposed ASIC using Verilog HDL and FPGA implementation will be described.

3.2 Architecture of the Design

Fig 3.1 shows the main module of the design and its internal connections and relations.

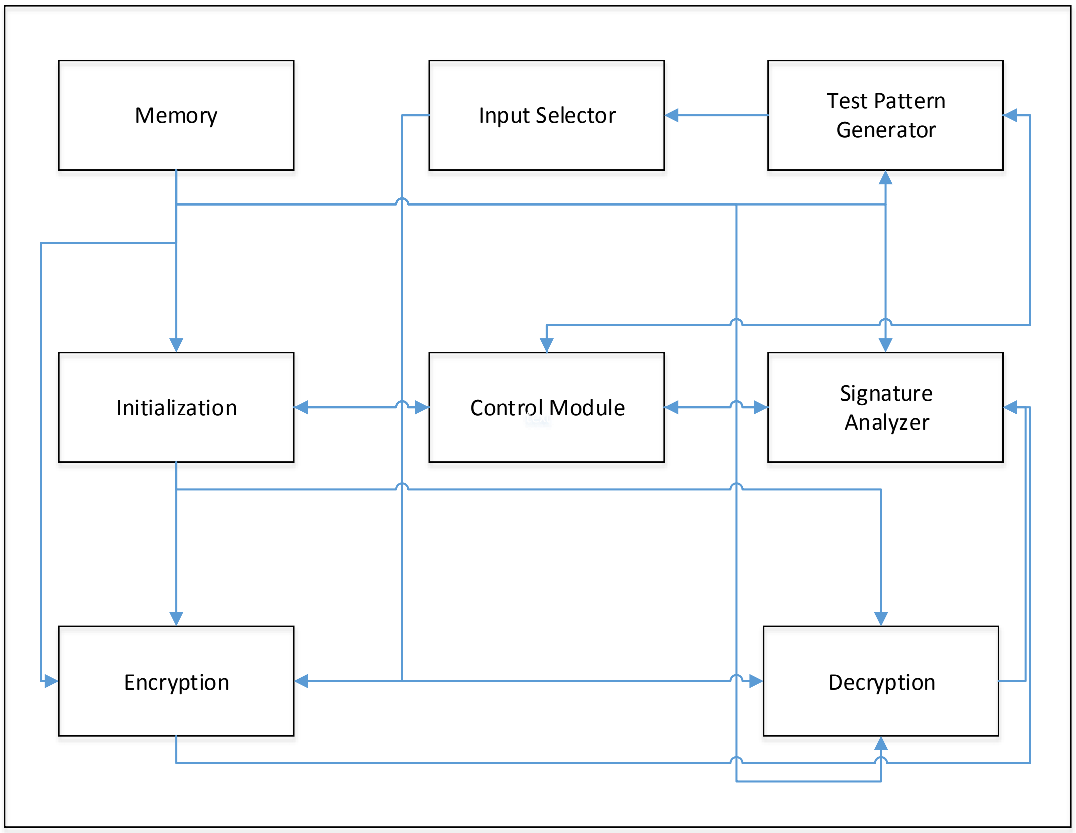


Fig. 3.1: Functional Blocks of Hummingbird Crypto ASIC with BIST

It is a standard practice to partition a complex design into different modules based on their specific functionality and features.

In our design, we have used the following blocks/modules:

1. Memory module
2. Initialization module
3. Input Selector
4. Test Pattern generator
5. Encryption
6. Decryption
7. Signal Analyzer
8. Control Module

Brief description of each module is given below:

Memory module:

We have used a memory (32X16 RAM) to store data which are used in different phases of the design. Initially the memory is loaded (write enable, when we signal is High) through data_in port with 4 NONCE values which are used in initialization module, 5 golden signatures to test encryption module and 5 golden signatures to test decryption module, 256 bit key subdivided into 16 16-bit and one 16 bit seed which is used to initialize LFSR and MISR. The stored values are read out from Memory when write enable, we signal is low.

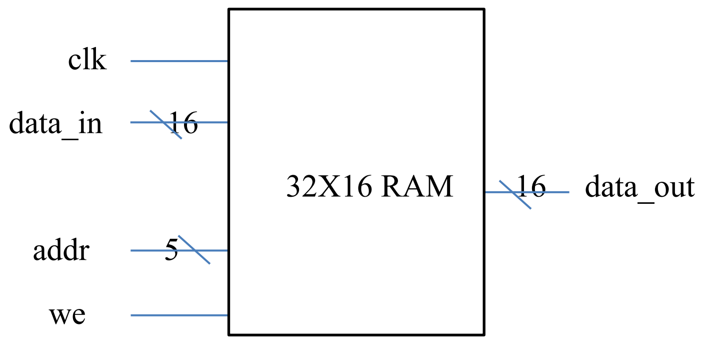


Fig. 3.2: Block diagram of Memory Module

Initialization Module:

When ‘init’ signal is high, initialization started. After getting ‘init’ signal is high, control module generates high ‘en_initialization’ which is wired with ‘en’ port of initialization module.

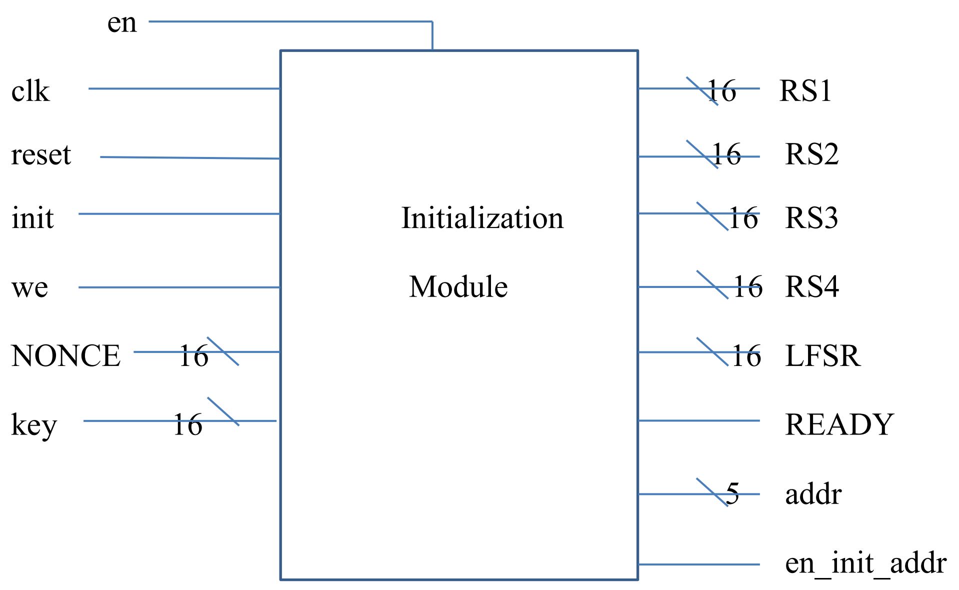


Fig.3.3: Block diagram of Initialization Module

256 bit key subdivided into 16 16-bit subkey (K11, K12, K13, K14, K21 ….K44) and RS1, RS2, RS3 & RS4 are first initialized by 4 16-bit NONCE values stored in RAM. After loading data from memory for key and initial value of Registers ciphering started using four block ciphers. After 16 cycles, READY signal goes high and we get 4 initialized registers (RS1, RS2, RS3, RS4) and LFSR which are used in encryption and decryption modules.

Input Selector:

After initialization completed, Input Selector Module choses input to be used for encryption and decryption modules based on the modes determined by user through ‘tst’ signal.

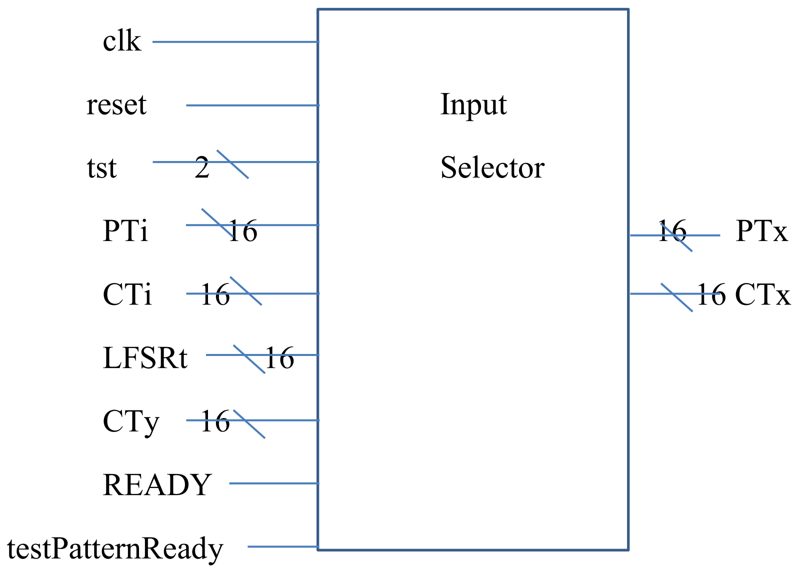


Fig. 3.4: Block Diagram of Input Selector Module

In the below table, depending on the mode of operation value of ‘tst’ and selected inputs are shown. For the test cases, output (LFSRt) of pseudo random test pattern generator is used as input to encryption/decryption modules

Table 3.1: Mode of Operation

Mode of Operation	**Value of tst**	**Selected Input**
Normal encryption/decryption	00	PTi, CTi
Test mode (Encryption Module)	01	LFSRt
Test mode (Decryption Module)	10	LFSRt
Decryption followed by encryption of the same input	11	PTi, CTy

Test Pattern Generator:

An LFSR is used as test pattern generator which generates pseudorandom test pattern to be used as input of encryption and decryption modules based on 'testCount' signal. In this project, we have used 100/200/300/400/500 as test count number.

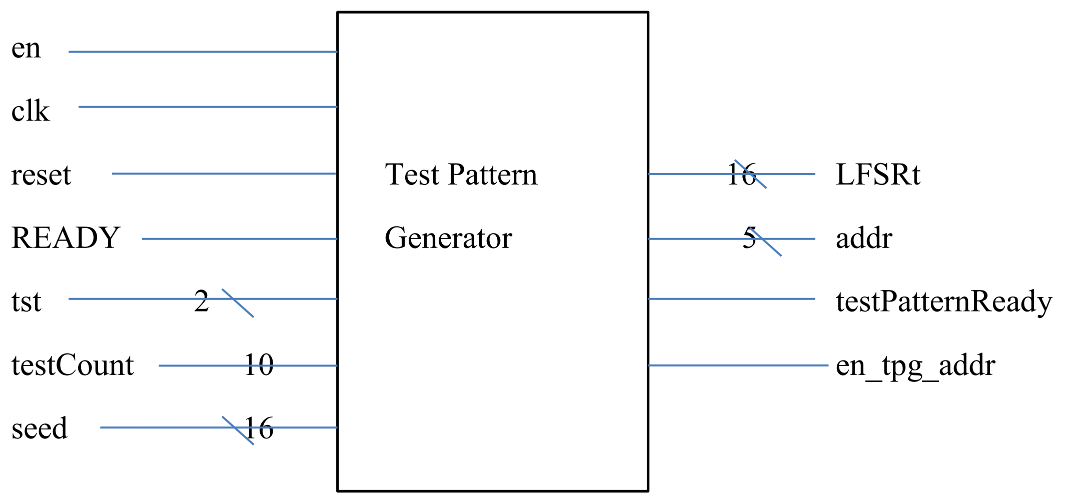


Fig. 3.5: Block Diagram of Test Pattern Generator

Encryption Module:

Encryption module operates in two modes based on the 'tst' signal. When the value of 'tst' is '00' it operates as normal mode and takes external plain text PT as its input and when the value of 'tst' is '01' it enters in test mode and takes test generator produced output LFSRt as its input.

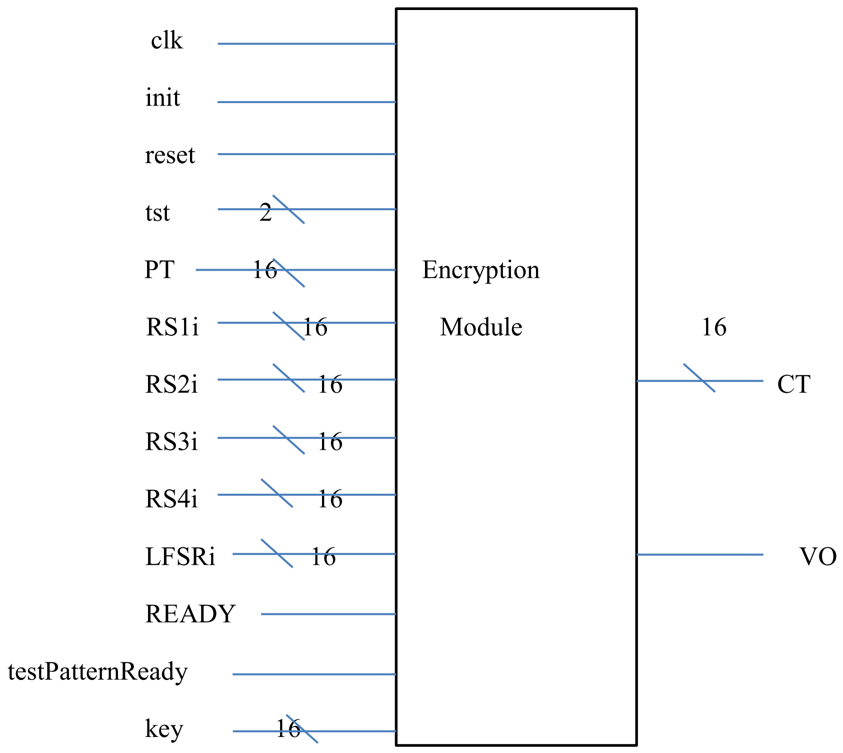


Fig. 3.6: Block Diagram of Encryption Module

When the 'init' signal is high 256 bit key is subdivided into 16 16-bit subkey are loaded into 16 registers K11,K12,K13,K14,K21, …. K44 of the module. After completion of the initialization 'READY' signal goes high and outputs of initialization module are used to initialize 4 registers (RS1, RS2, RS3, RS4) and LFSR of the encryption module. After each 4 cycles it provides ciphered text (CT) of the signal PT and updates status of its internal 4 registers RS1, RS2, RS3, RS4 and LFSR.

Decryption Module:

Decryption module also operates in two modes based on the ‘tst’ signal. When the value of ‘tst’ is ‘00’ it operates as normal mode and takes external cipher text CT as its input and when the value of ‘tst’ is ‘10’ it enters in test mode and takes test generator produced output LFSRt as its input.

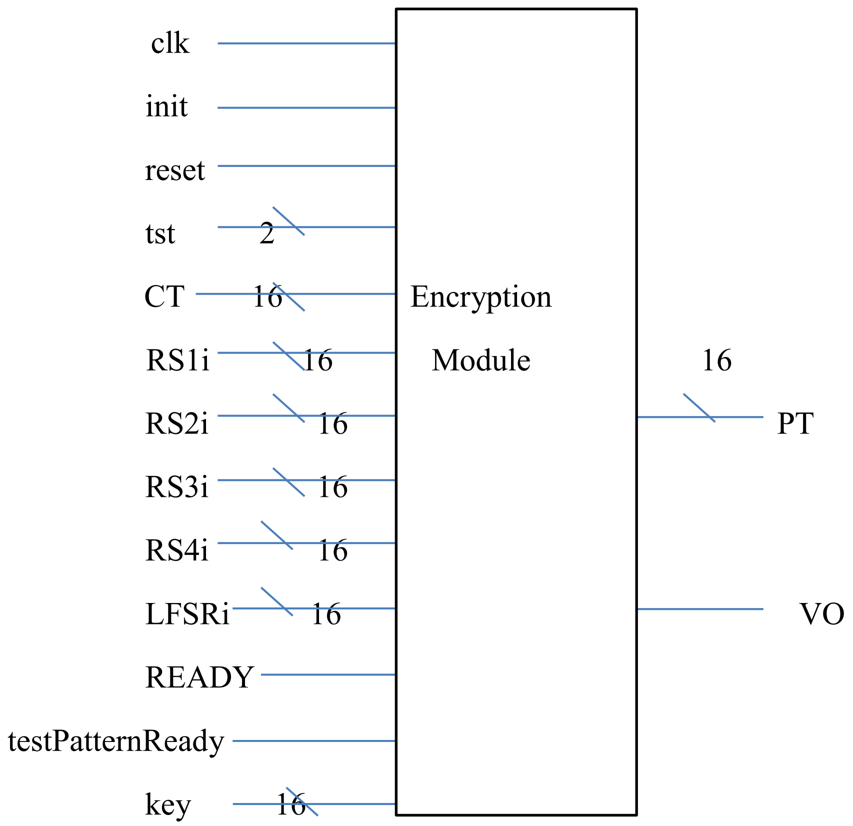


Fig. 3.7: Block Diagram of Decryption Module

Its keys are also loaded from memory module and internal registers are initialized same as encryption module. It provides plain text output after every 4 clock cycles and updates status of its internal registers and LFSR.

Signature Analyzer:

This module tests functionalities of encryption and decryption modules separately. When ‘tst’ signal is ‘01’ functionality of encryption module is tested. Depending on the value of ‘testCount’ 100/200/300/400/500, number of test count varied. Pseudorandom pattern generated by the test pattern generator is input to the encryption module. Cipher text output from encryption module is fed to the ‘CTy’ of Signature Analyzer in which MISR generates sign output, after the desired test count final sign is compared with the stored golden signature. ‘OK’ signal goes high if final sign matches with golden signature, otherwise it remains low. In this way module is tested for other test counts. For decryption module test, pseudorandom test pattern is input to the decryption module and plain text output of decryption module is fed to the ‘PTy’ of Signature Analyzer.

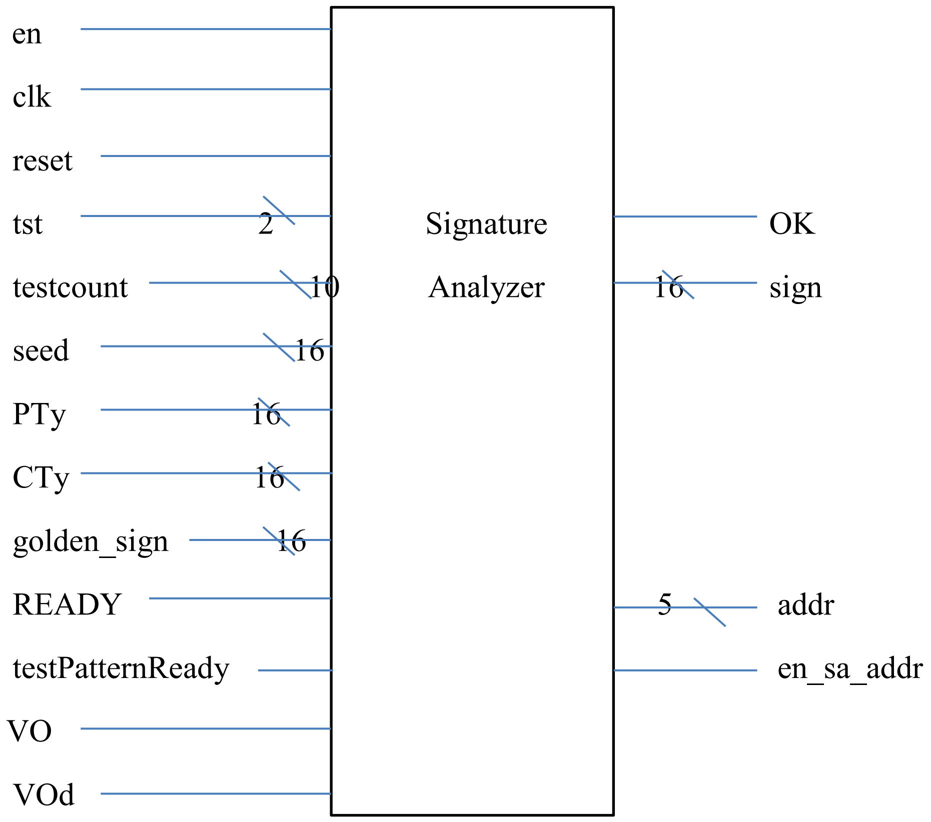


Fig. 3.8: Block Diagram of Signature Analyzer Module

Control Module:

One control module is used to control the sequence which module will be in action in which time.

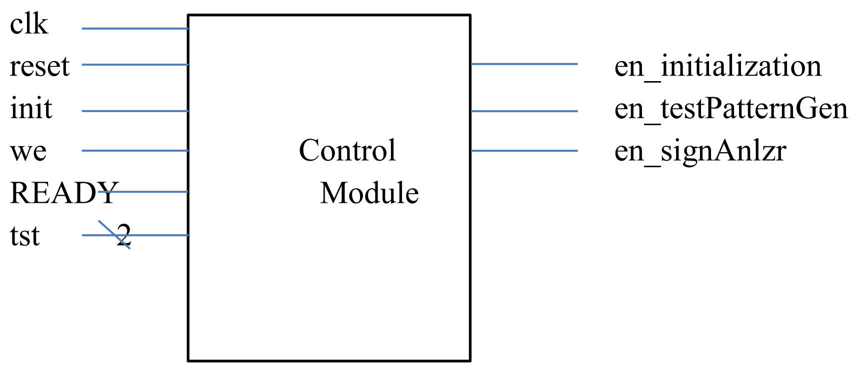


Fig. 3.9: Block Diagram of Control Module

3.3 Flow Chart of the Design

Fig 3.10 shows the Flowchart of the Hummingbird Crypto ASIC with BIST

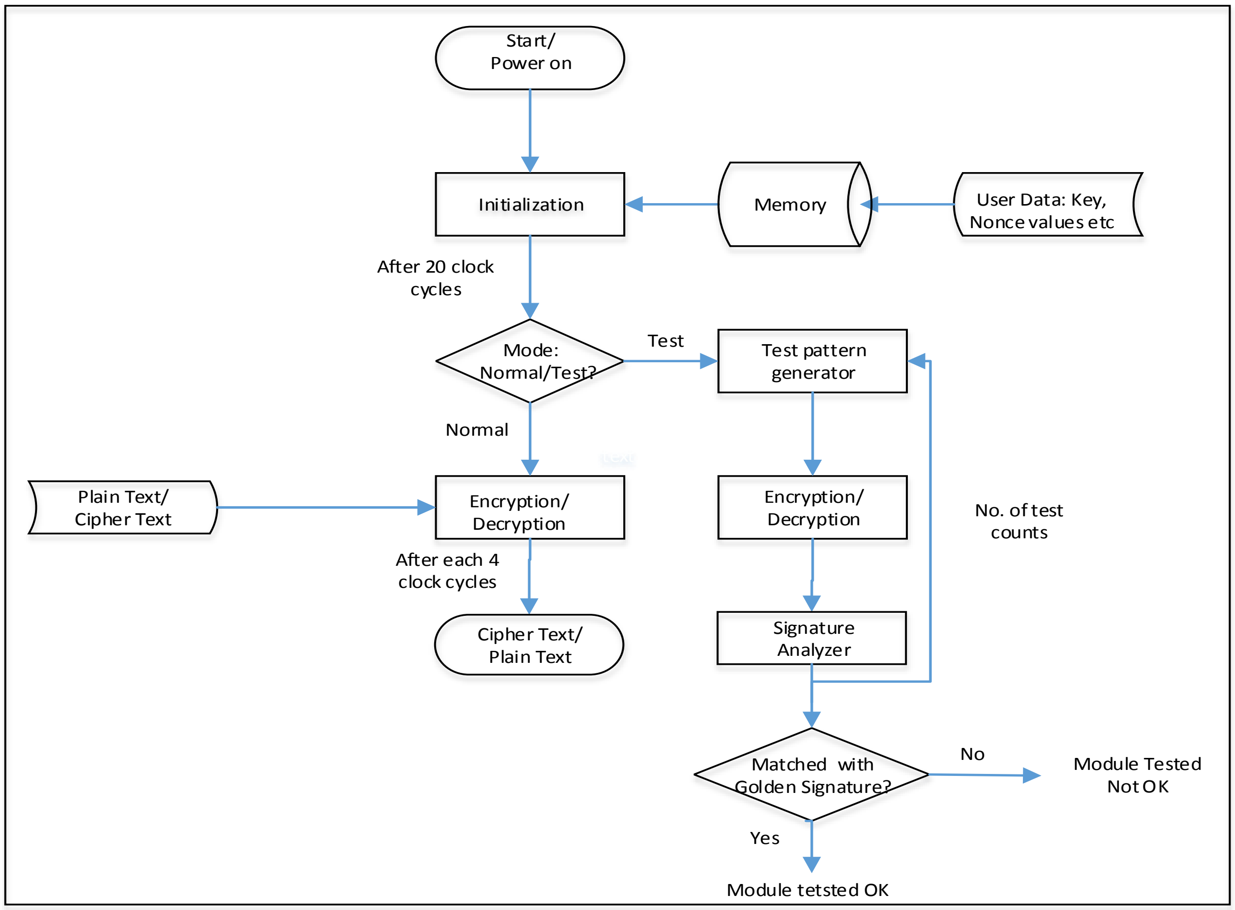


Fig. 3.10: Flowchart of the Hummingbird Crypto ASIC with BIST

The Hummingbird Algorithm operates on a block of 16 bits of input data and generates 16 bits of output. The length of the key used to encrypt/decrypt is 256 bits and there are 80 bits of internal states.

At the start, our desired Crypto ASIC is initialized. In this phase, internal registers and LFSR are initialized using NONCE and key values stored in the Memory which are preloaded. These initialized internal registers and LFSR are used in later phases in encryption/decryption modules. Depending on the mode of operation, the proposed ASIC can operate in Normal/Test mode. In Normal mode, both encryption and decryption modules works in parallel and provide output cipher text or plain text respectively for input of plain text or cipher text. In Test mode the proposed ASIC performs test for either encryption/decryption module, i.e. one at a time. In test mode, pseudorandom pattern generated by the test pattern generator is input to the desired test module (encryption/decryption) and then output is fed to the Signature Analyzer which produces a signature. This process is continued for the desired number of test count and the final signature is matched with the golden signature stored in the Memory. If it matches it indicates the functionality of the module under test is OK.

3.4 Device Used

The proposed Hummingbird crypto ASIC is implemented in FPGA device of family Altera, Cyclone II, EP2C5F256C7 device. This device is chosen considering below:

- Has required numbers of Logic Elements, Pins for the design
- Small device hence low power consumption
- Low cost

The device specifications are given below:

Total Logic Elements	: 4608
Total Combinational Functions	: 4608
Dedicated Logic Registers	: 1615
Total Registers	: 1615
Total Pins	: 158
Total Virtual Pins	: 0
Total Memory Bits	: 119,808
Embedded Multiplier 9-bits Elements	: 26
Total PLLs	: 2

3.6 Tools Used

The proposed Hummingbird Crypto ASIC implementing BIST technique is designed using Quartus II EDA tool (provided by Altera Company). Full compilation of the design using the Quartus II simulation software includes the following modules:

- Analysis & Synthesis
- Partition Merge
- I/O Assignment Analysis
- Fitter
- Assembler
- Classic Timing Analyzer
- EDA Netlist Writer

Then symbol files are created for the design files.

Then vector waveform simulation is performed using Simulator tool. Signal Activity File is also generated during simulation which is used later for power analysis.

PowerPlay Power Analyzer Tool is used for power analysis.

Chapter 4

Experimental Results and Discussions

4.1 Introduction

The proposed Hummingbird crypto ASIC is implemented in FPGA device of family Altera, Cyclone II, EP2C5F256C7 device. The simulation is performed in Quartus II simulation software. The device specifications are given below:

Total Logic Elements	: 4608
Total Combinational Functions	: 4608
Dedicated Logic Registers	: 1615
Total Registers	: 1615
Total Pins	: 158
Total Virtual Pins	: 0
Total Memory Bits	: 119,808
Embedded Multiplier 9-bits Elements	: 26
Total PLLs	: 2

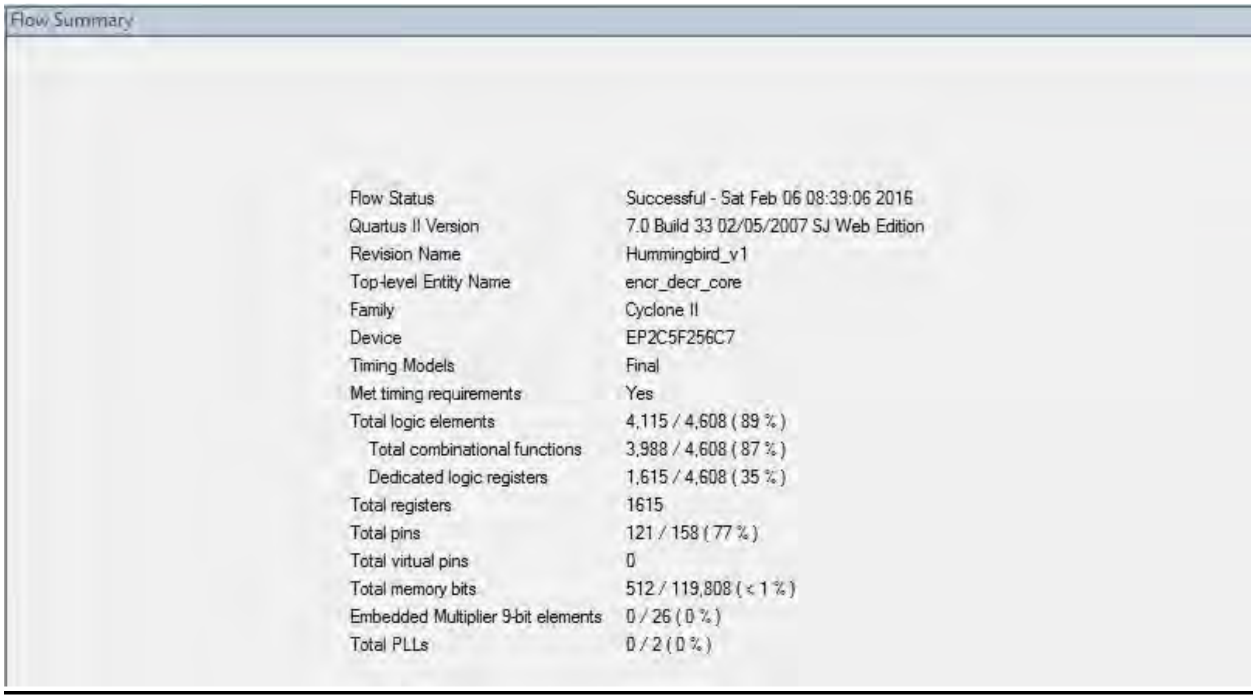
4.2 Resources Used

The compilation result of the design shows the resources used:

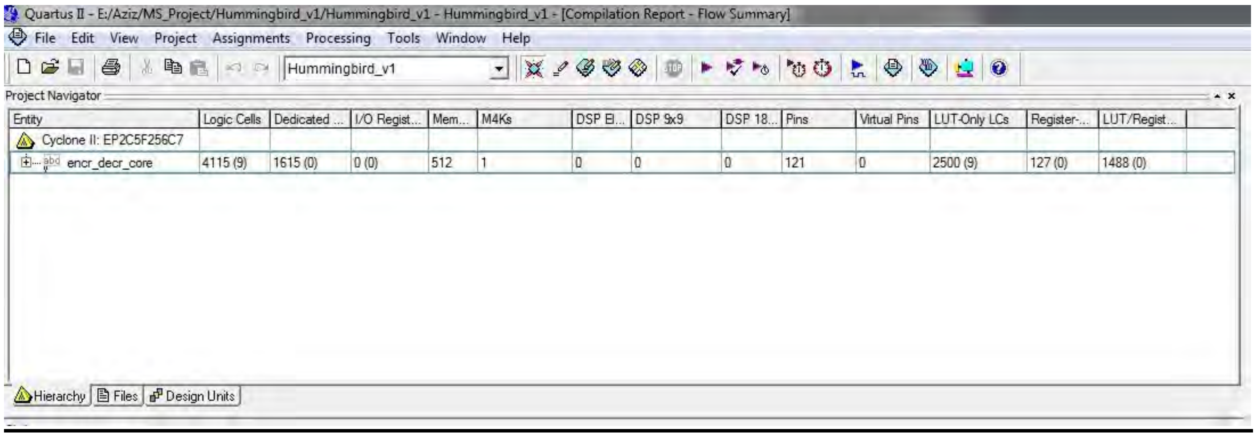
Total Logic Elements	: 4122
Total Combinational Functions	: 3988
Dedicated Logic Registers	: 1615
Total Registers	: 1615

Total Pins	: 121
Total Virtual Pins	: 0
Total Memory Bits	: 512
Embedded Multiplier 9-bits Elements	: 0
Total PLLs	: 0

Below is the snapshot of the Compilation Report – Flow Summary


Flow Summary

```
Flow Status                         Successful - Sat Feb 06 08:39:06 2016
Quartus II Version                  7.0 Build 33 02/05/2007 SJ Web Edition
Revision Name                       Hummingbird_v1
Top-level Entity Name               encr_decr_core
Family                              Cyclone II
Device                              EP2C5F256C7
Timing Models                       Final
Met timing requirements             Yes
Total logic elements                4,115 / 4,608 ( 89 % )
   Total combinational functions    3,988 / 4,608 ( 87 % )
   Dedicated logic registers        1,615 / 4,608 ( 35 % )
Total registers                     1615
Total pins                          121 / 158 ( 77 % )
Total virtual pins                  0
Total memory bits                   512 / 119,808 ( < 1 % )
Embedded Multiplier 9-bit elements  0 / 26 ( 0 % )
Total PLLs                          0 / 2 ( 0 % )
```



Quartus II - E:/Aziz/MS_Project/Hummingbird_v1/Hummingbird_v1 - Hummingbird_v1 - [Compilation Report - Flow Summary]
File Edit View Project Assignments Processing Tools Window Help
Hummingbird_v1
Project Navigator
Hierarchy
Files
Design Units

Entity	Logic Cells	Dedicated ...	I/O Regist...	Mem...	M4Ks	DSP El...	DSP 9x9	DSP 18...	Pins	Virtual Pins	LUT-Only LCs	Register-...	LUT/Regist...
Cyclone II: EP2C5F256C7													
encr_decr_core	4115 (9)	1615 (0)	0 (0)	512	1	0	0	0	121	0	2500 (9)	127 (0)	1488 (0)

Fig. 4.1: Snapshot of the Compilation Report – Flow Summary

4.3 Simulation Results

This project only considers the implementation with BIST to get Hummingbird Crypto ASIC. During the simulation, the top module is the core module which is simulated using Quartus II simulator. And also each block (Memory, Initialization, input selector, test pattern generator, encryption, decryption, and signature analyzer) is simulated independently. The simulation results of each block are provided in the following subsections.

4.3.1 Core Module:

In simulation, we used 4 16-bit NONCE 39F1, 268A, 19A4, 59AE; 256 bit key 11223810AB52EC9F11223810AB52EC9F11223810AB52EC9F11223810AB52EC9F, and 100F as seed. Besides these 5 golden signatures 4F58,737D,92CC,4C88,04BE are used for testing of encryption module for 5 different test counts (100,200,300,400,500) and 5 golden signatures 460E,6262,8F9D,E15D,1E52 for testing of decryption module. All these values are first loaded into RAM when 'we' signal is high. Later during initialization 4 NONCE values are used to initialize 4 registers RS1, RS2, RS3 and RS4 of initialization module. In this phase 256 bit key also loaded as 16 16-bit subkeys in Initialization, Encryption and Decryption modules.

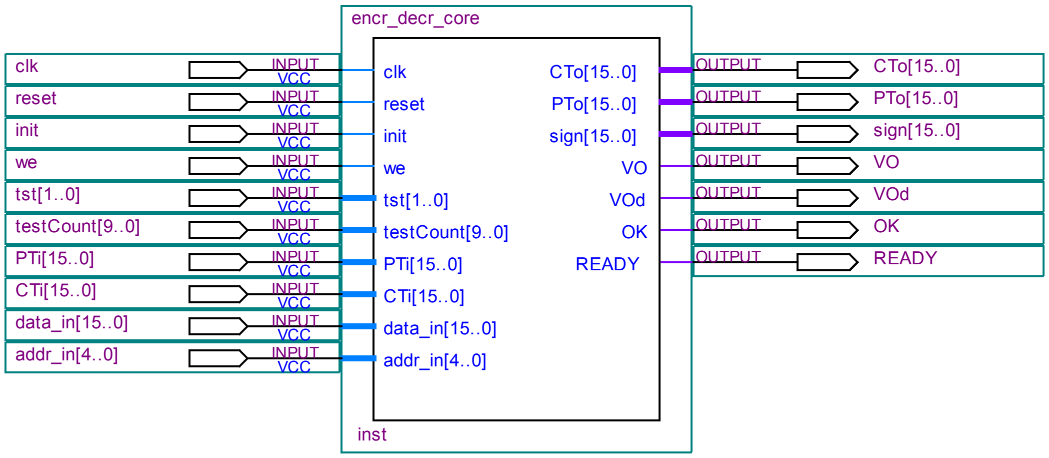


Fig. 4.2: Schematic Diagram for Core Module

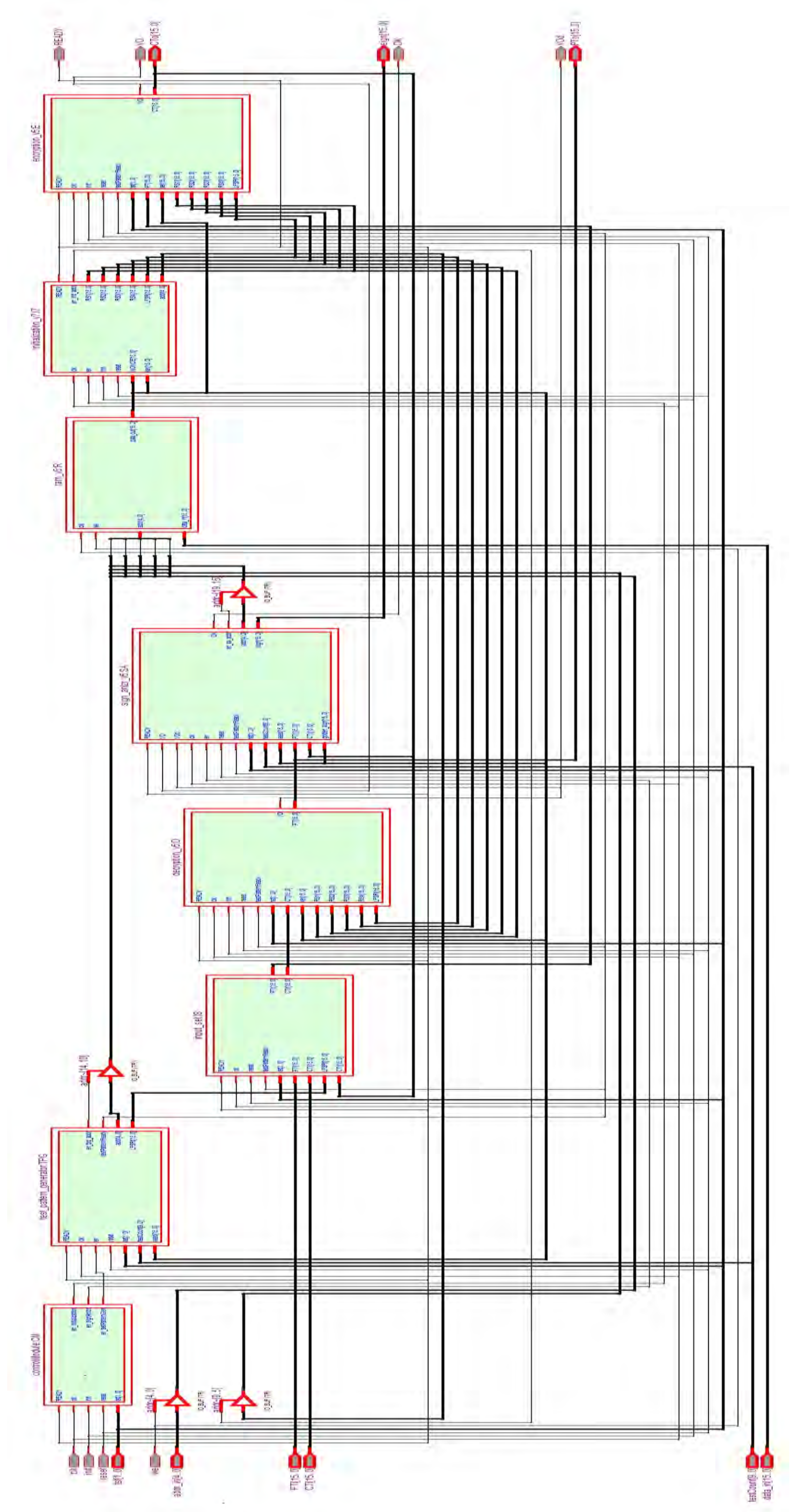


Fig. 4.3: RTL Viewer of Hummingbird Crypto ASIC with BIST

Normal Mode:

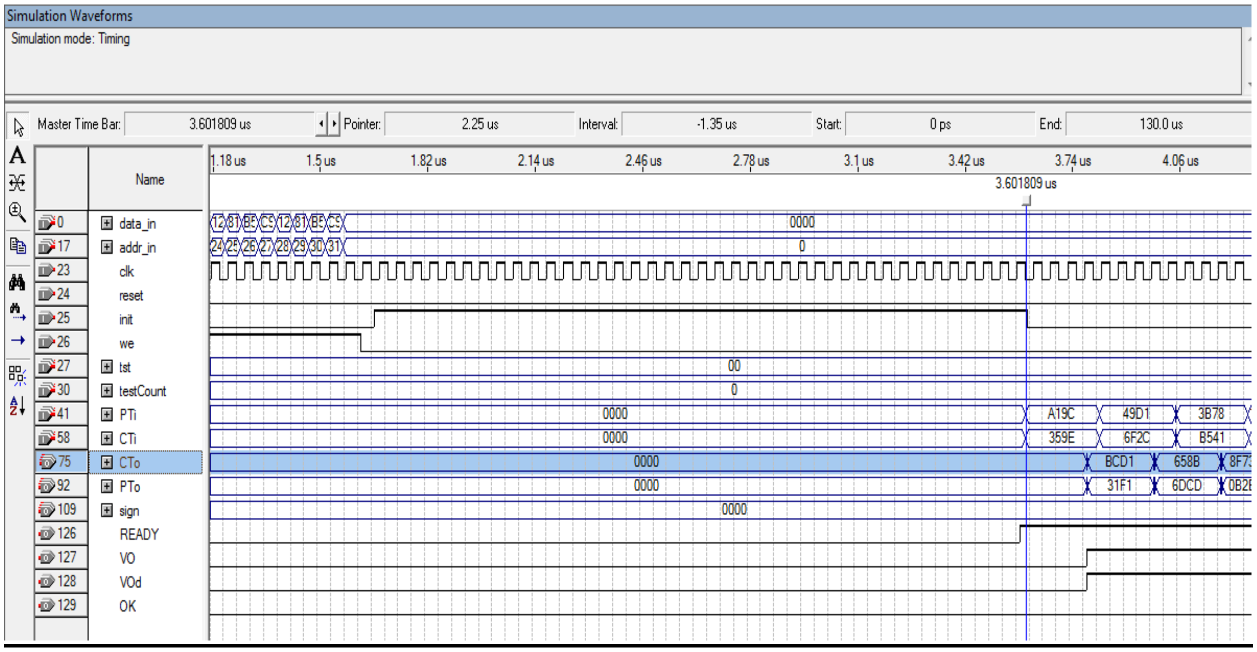


Fig. 4.4: Simulation result of Core module in normal operation mode (‘tst’=’00’)

After completion of the initialization ‘READY’ signal goes high and after each 4 cycle encryption module generates cipher text output (CTo) of the input plain text (PTi) whereas decryption module generates plain text output (PTo) of the input cipher text (CTi).

Test mode:

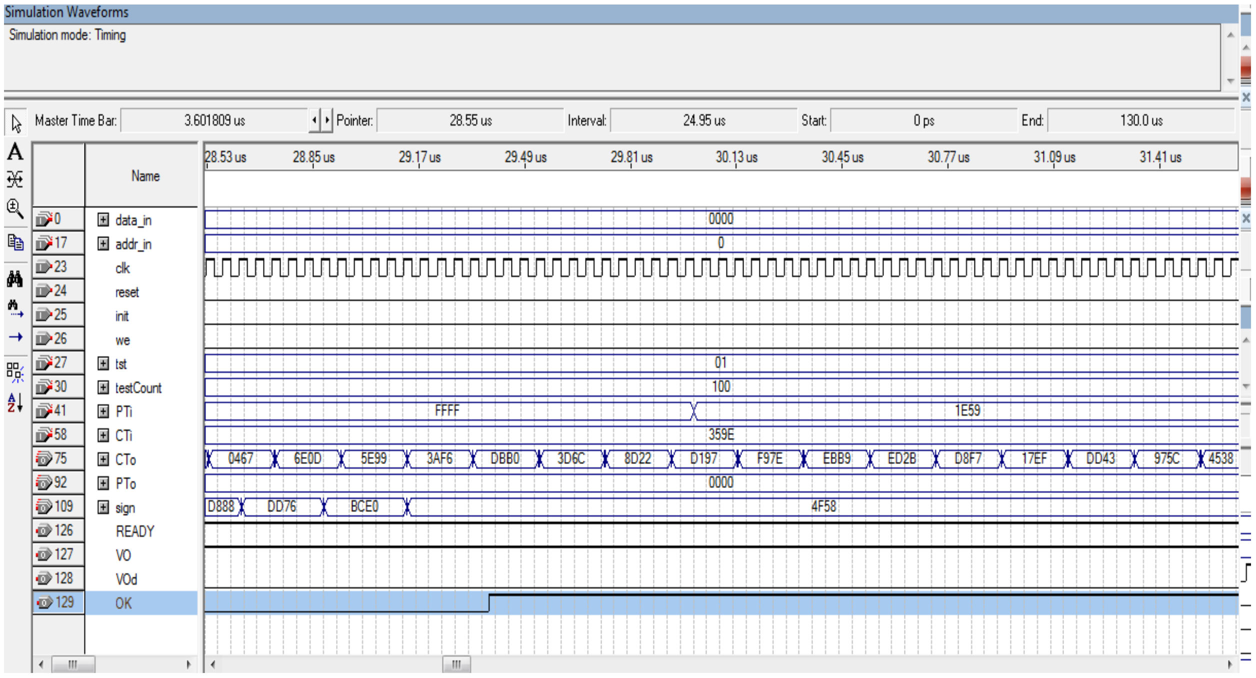


Fig. 4.5: Simulation result of Core module in test mode (test of encryption block with 100 test count)

For this mode, ‘tst’ is set as ‘01’ and test pattern generator produces 100 (‘testCount’) pseudorandom test pattern (LFSRt) which are input to encryption module. For each test input, output of encryption module fed to signature analyzer which generates corresponding sign. After 100 run of encryption module, final sign 4F58 matches with golden signature and ‘OK’ signal goes high meaning encryption module is functioning as expected. In this way, the module can be tested for test count 200,300,400 and 500.

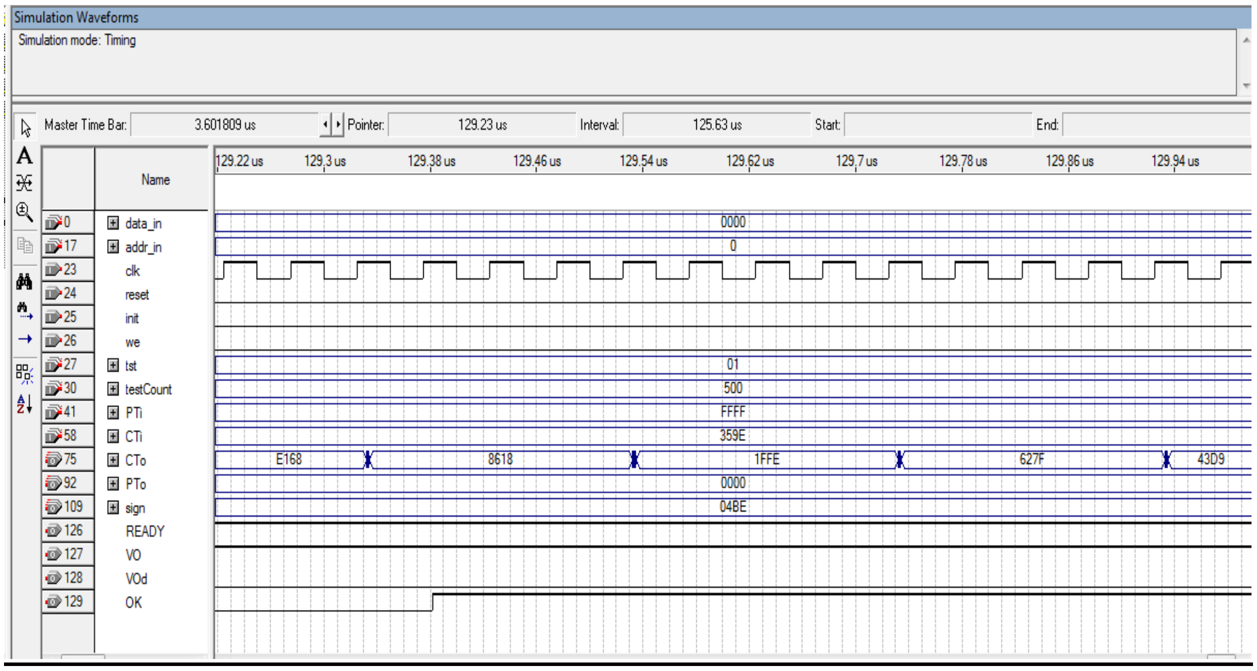


Fig. 4.6: Simulation result of Core module in test mode (test of encryption block with 500 test count)

In the same way, decryption block can be tested. For this, ‘tst’ is set to ‘10’.

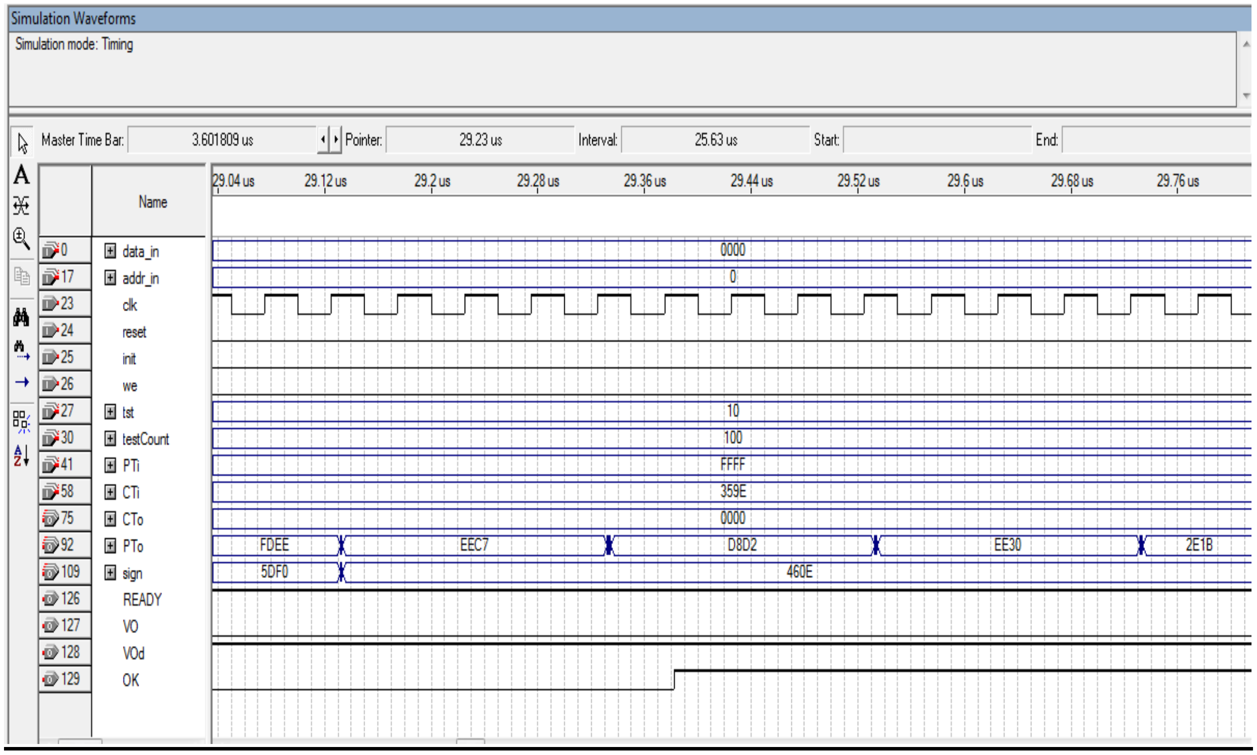


Fig. 4.7: Simulation result of Core module in test mode (test of decryption block with 100 test count)

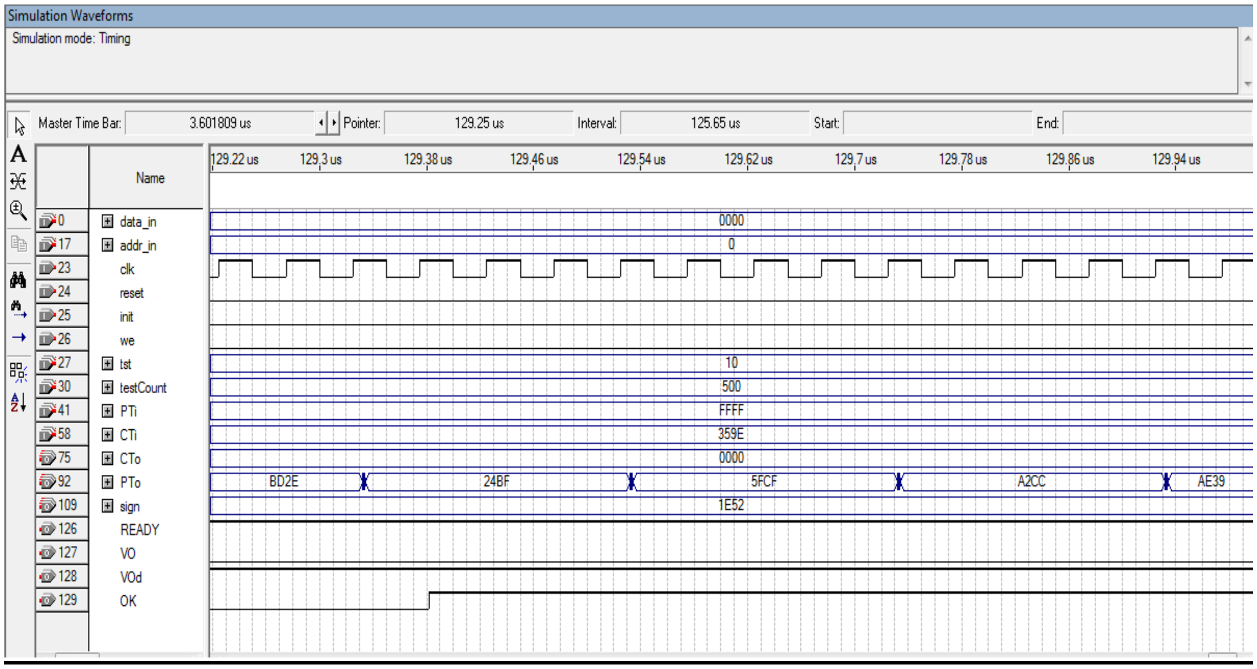


Fig. 4.8: Simulation result of Core module in test mode (test of decryption block with 500 test count)

4.3.2 Memory Module:

In our design, we used a bit memory module to store 256 bit Key, 4 NONCE values, 10 golden signatures for 5 different test counts of encryption/decryption block.

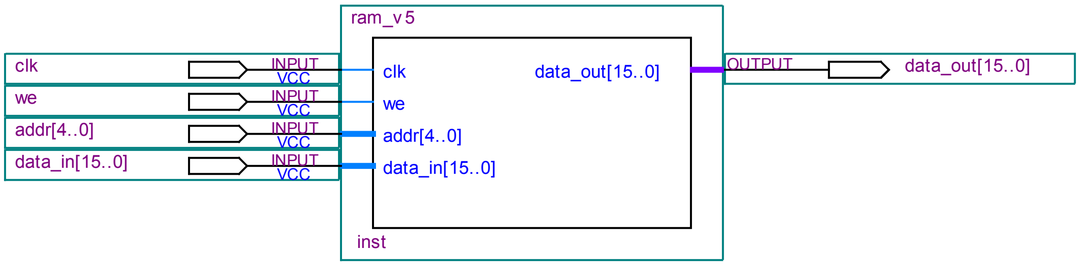
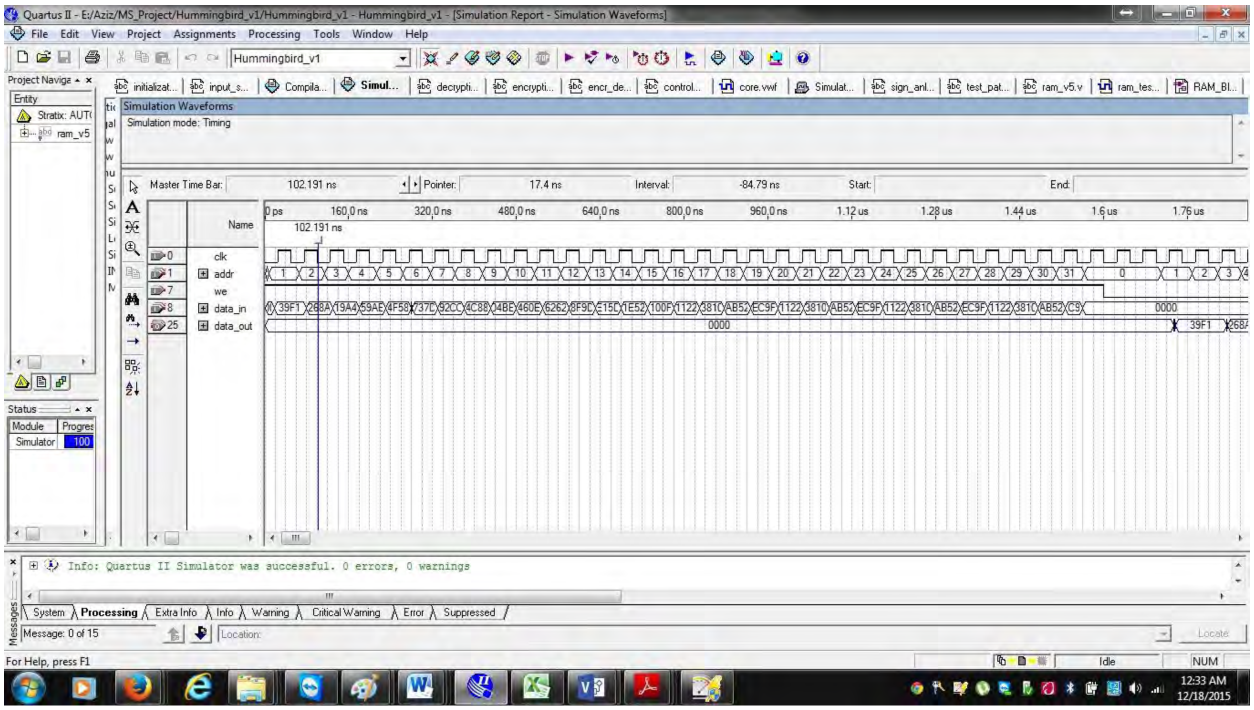


Fig.4.9: Schematic Diagram for RAM

When 'we' signal is high, the required values are written into different addresses of RAM through 'data_in' ports. These values are read by other blocks on demand basis through 'data_out' port by passing appropriate address.


Quartus II - E:/Aziz/MS_Project/Hummingbird_v1/Hummingbird_v1 - Hummingbird_v1 - [Simulation Report - Simulation Waveforms]
File Edit View Project Assignments Processing Tools Window Help
Simulation Waveforms
Simulation mode: Timing
Master Time Bar: 102.191 ns
Pointer: 17.4 ns
Interval: -84.79 ns
clk
addr
we
data_in
data_out

```
Info: Quartus II Simulator was successful. 0 errors, 0 warnings
```

Fig. 4.10: Simulation Result of Memory Module

4.3.3 Initialization Module

4 NONCE values are read from Memory and loaded into 4 Registers in 4 clock cycles after getting high ‘init’ signal. Then after another 16 cycles we got initialized 4 internal Registers RS1, RS2, RS3, RS4 and LFSR which are used in encryption and decryption blocks. At the end of the initialization ‘READY’ signal goes high marking that it’s ready for encryption and decryption.

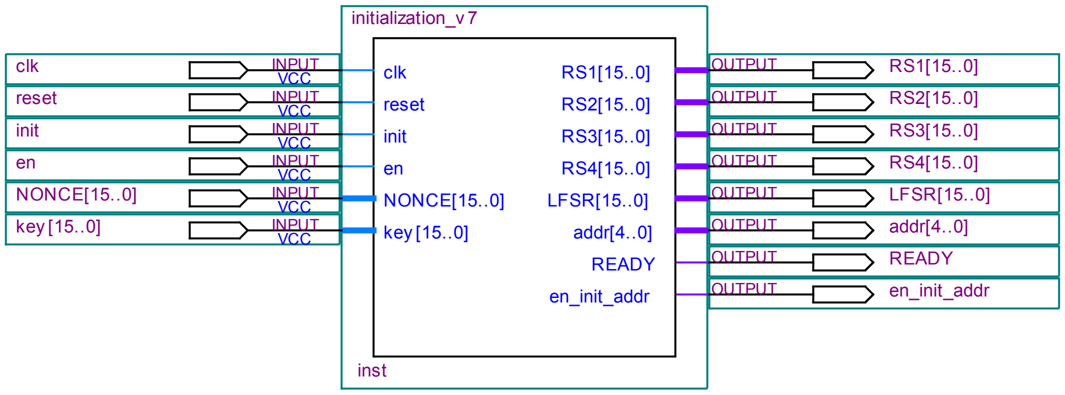
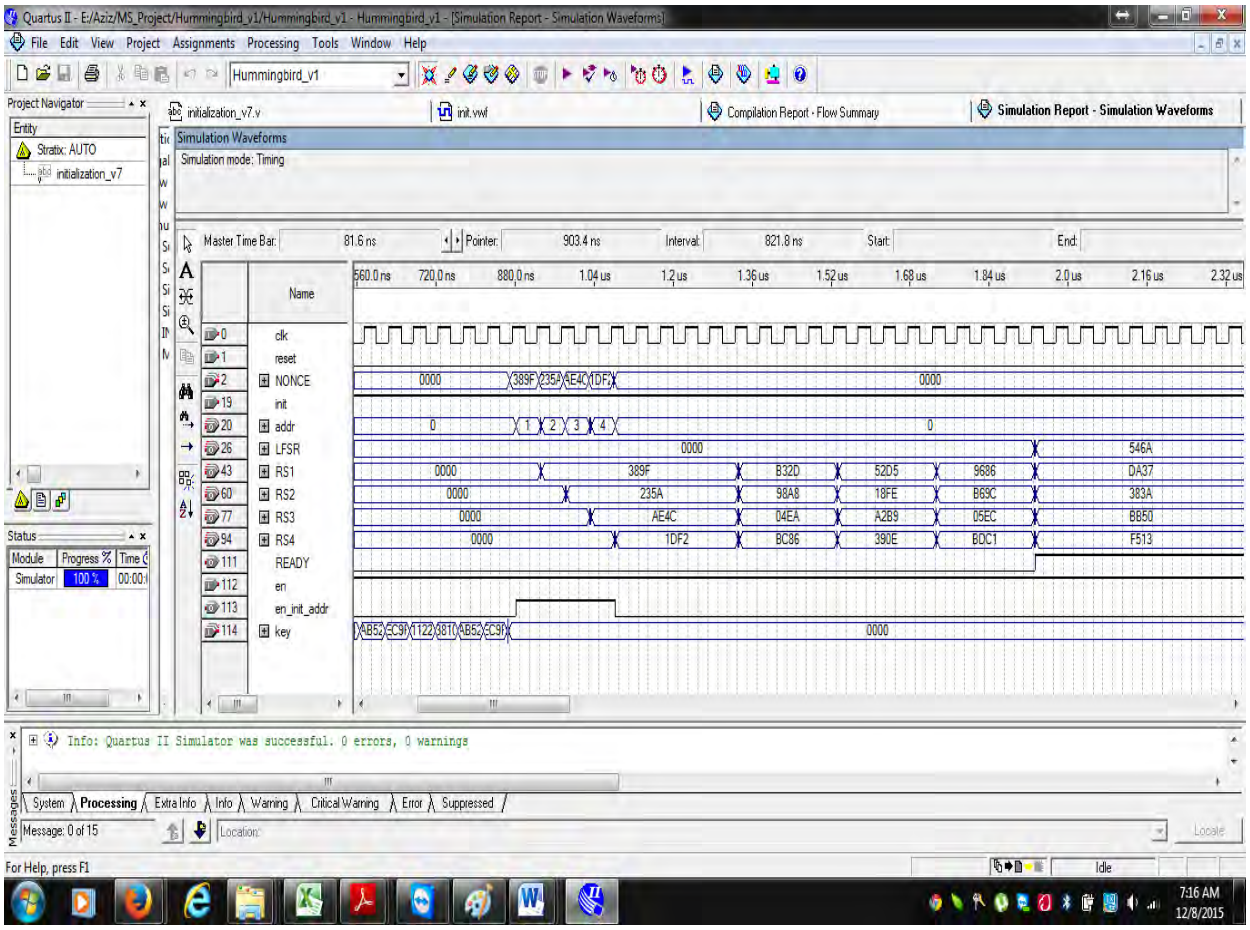


Fig. 4.11: Schematic Diagram of Initialization Module



```
Info: Quartus II Simulator was successful. 0 errors, 0 warnings
```

Fig. 4.12: Simulation Result of Initialization Module

4.3.4 Input Selector

This module decides whether external data or test pattern generated by the test pattern generator will be used as input for encryption or decryption blocks depending on the mode of operation.

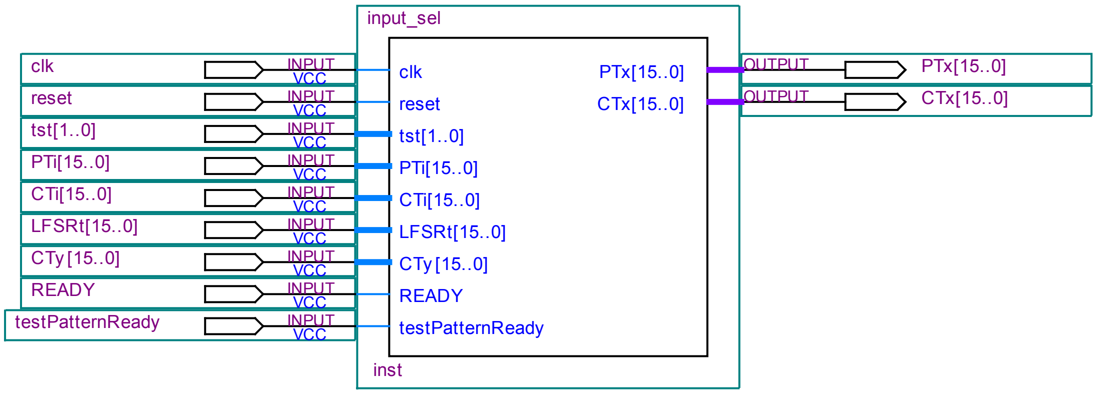


Fig. 4.13: Schematic Diagram of Input_Sel Module

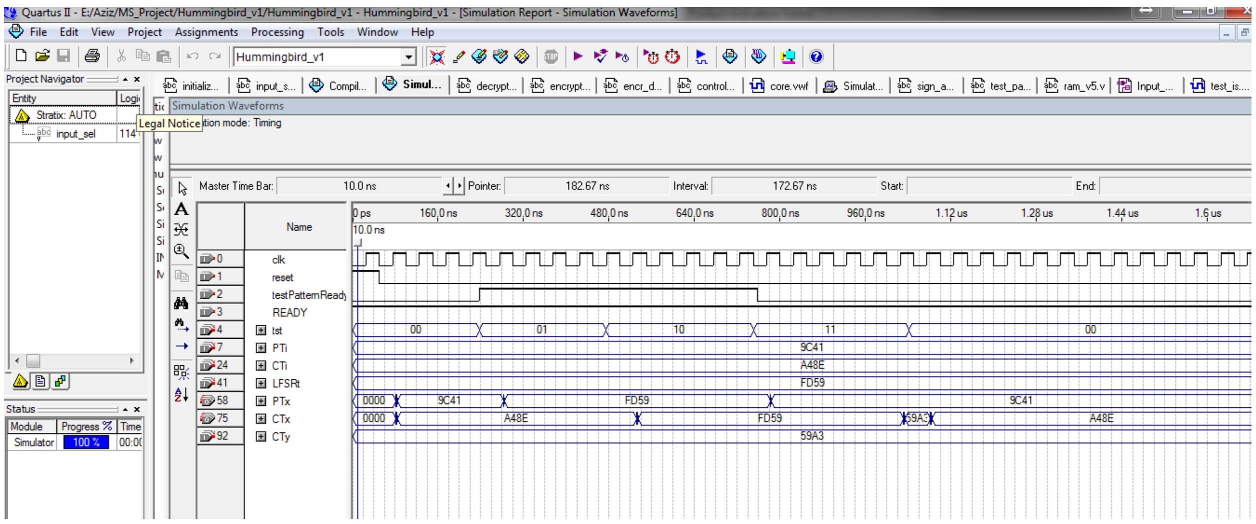


Fig.4.14: Simulation Result of Input Selector Module

4.3.5 Test Pattern Generator

This module generates required number of test patterns depending on the number of test counts. ‘testPatternReady’ signal goes high when a test pattern is ready.

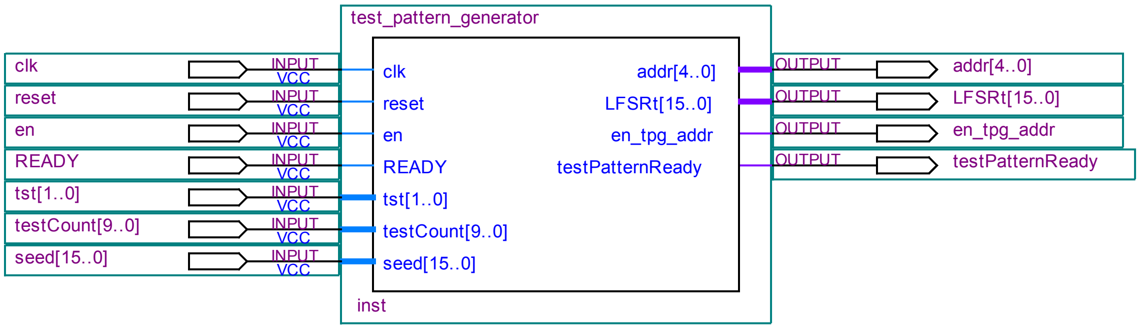


Fig.4.15: Schematic Diagram of Test Pattern Generator Module

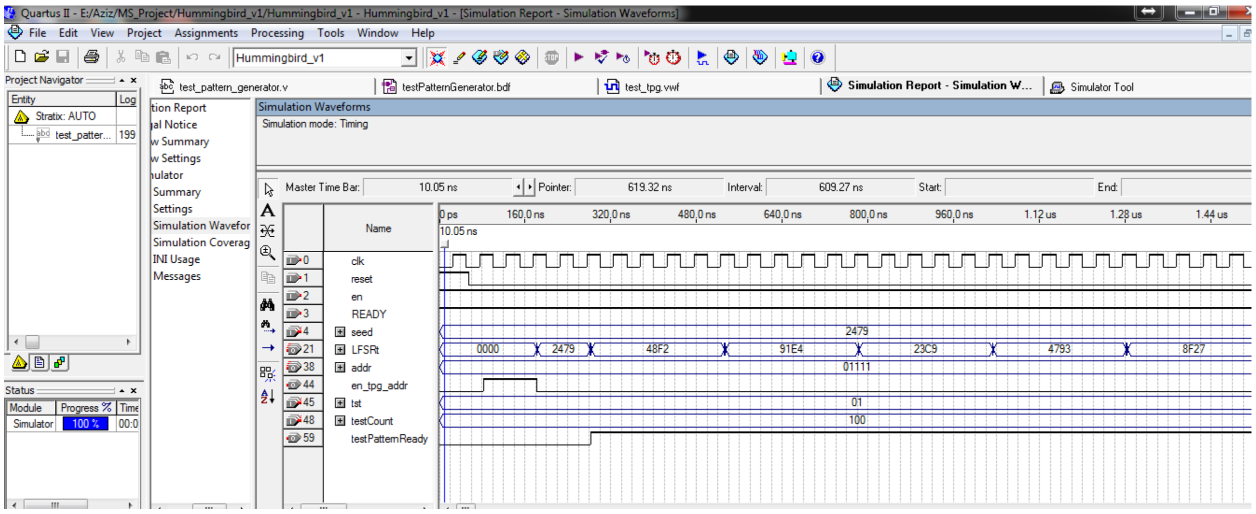


Fig.4.16: Simulation Result of Test Pattern Generator Module

4.3.6 Encryption

This module provides encrypted output for the plain text input of its ‘PT’ port in 4 clock cycles. It started with 4 initialized internal registers RS1, RS2, RS3, RS4 and LFSR. These represent 80 bits internal states which change their state in each cycle of encryption/decryption.

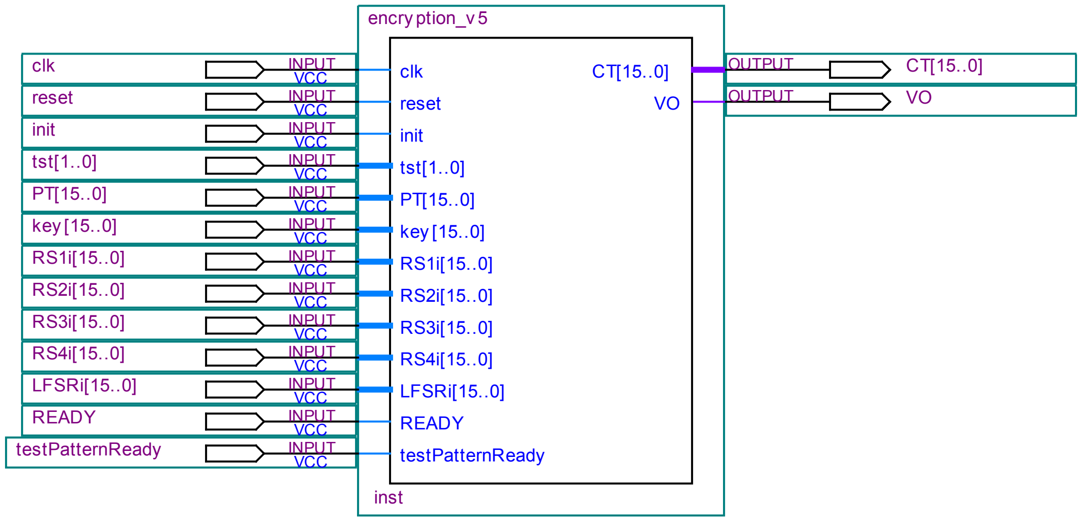


Fig. 4.17: Schematic Diagram for Encryption Module

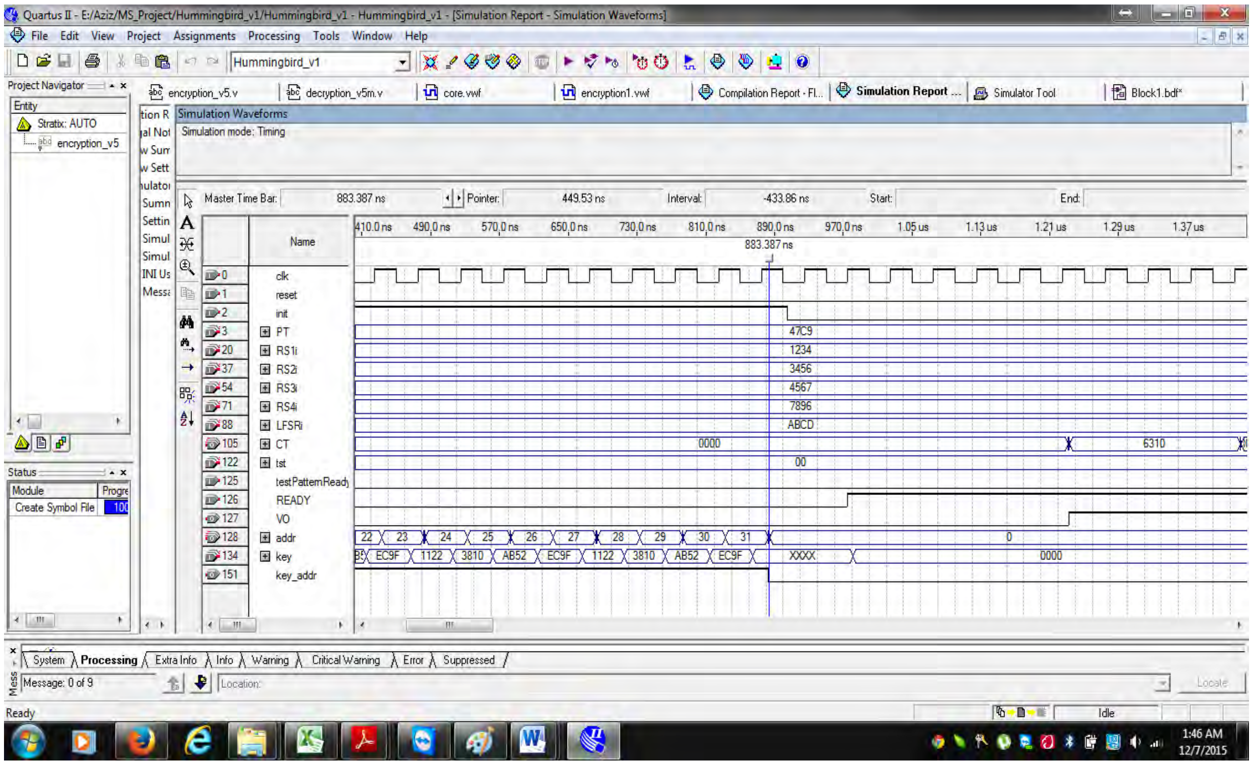

Quartus II - E:/Aziz/MS_Project/Hummingbird_v1/Hummingbird_v1 - Hummingbird_v1 - [Simulation Report - Simulation Waveforms]
File Edit View Project Assignments Processing Tools Window Help
Simulation Waveforms
Simulation mode: Timing
Master Time Bar: 883.387 ns
Pointer: 449.53 ns
Interval: -433.86 ns
clk
reset
init
PT 47C9
RS1i 1234
RS2i 3456
RS3i 4567
RS4i 7896
LFSRi ABCD
CT 0000 6310
tst 00
testPatternReady
READY
VO
addr 22 23 24 25 26 27 28 29 30 31 0
key EC9F 1122 3810 AB52 EC9F 1122 3810 AB52 EC9F XXXX 0000
key_addr
Message: 0 of 9
Ready

Fig.4.18: Simulation Result of Encryption Module

4.3.7 Decryption

This module provides deciphered output for the cipher text input of its ‘CT’ port in 4 clock Cycles. Like encryption block it also started with 4 initialized internal registers RS1, RS2, RS3, RS4 and LFSR which change their state in each cycle of decryption.

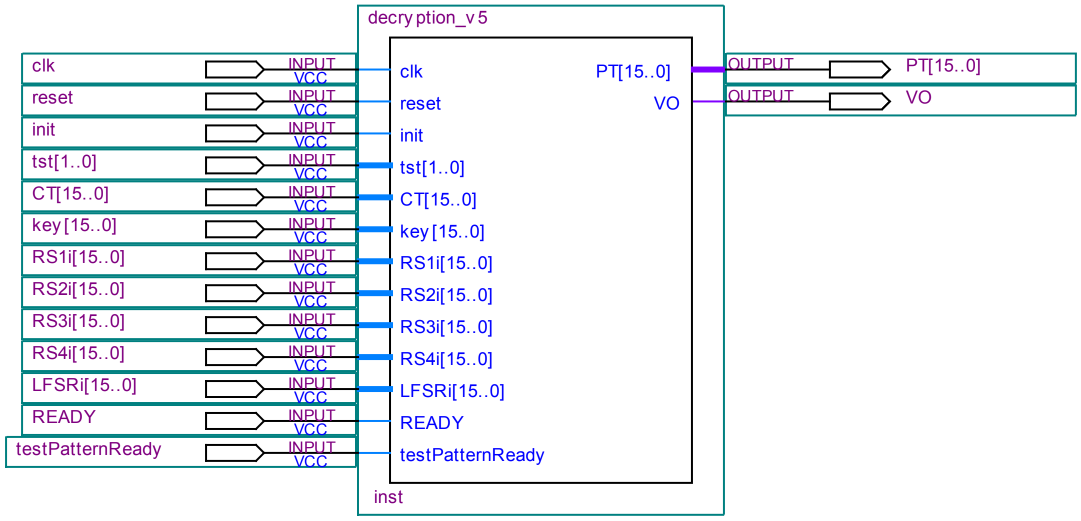


Fig. 4.19: Schematic Diagram for Decryption Module

4.3.8 Signature Analyzer

Below figure shows the block diagram of Signature Analyzer module. Output of the module, encryption/decryption to be tested is fed to the CTy/PTy ports. The input drives a Multiple Input Shift Register (MISR). This is continued upto the number determined by the 'testCount' signal. Final output of the MISR is the signature which is compared with the golden signature.

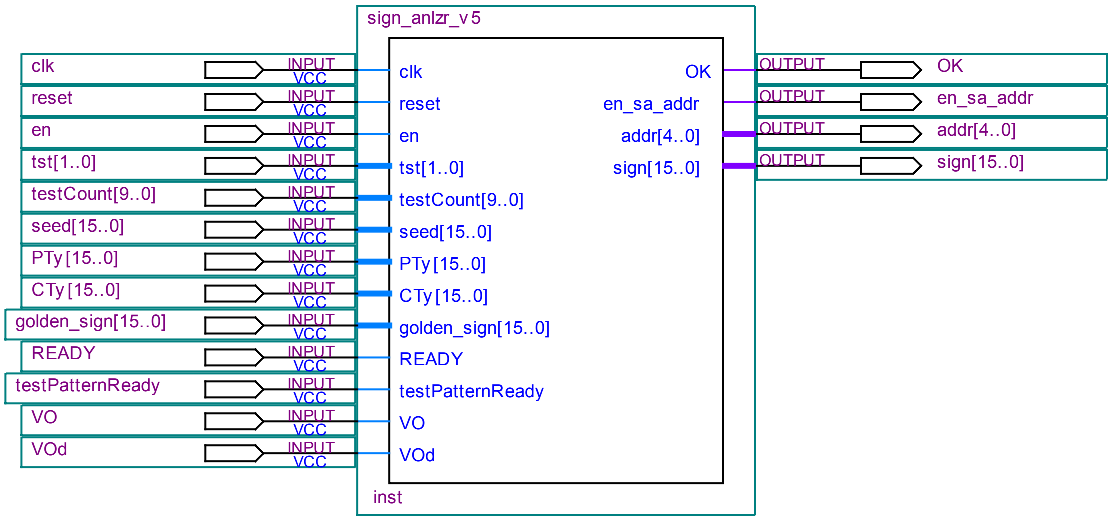


Fig. 4.20: Schematic Diagram for Signature Analyzer Module

4.3.9 Control Module

This module determines the sequence of initialization, test pattern generator and signature analyzer module operations as these modules use RAM, we need to maintain a proper sequence.

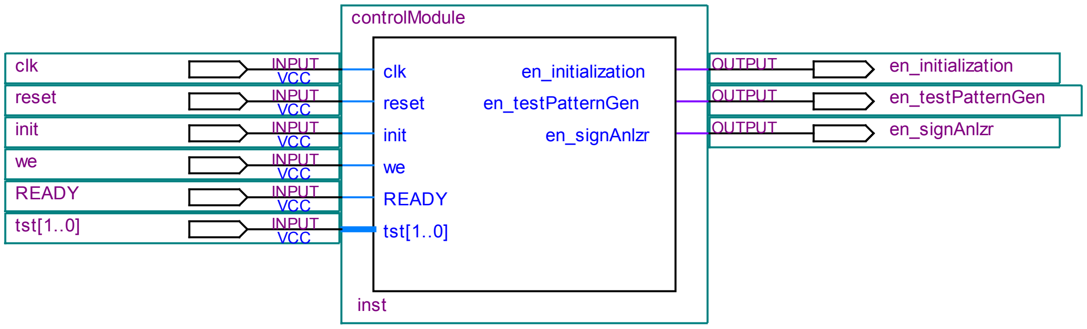


Fig. 4.21: Schematic Diagram for Control Module

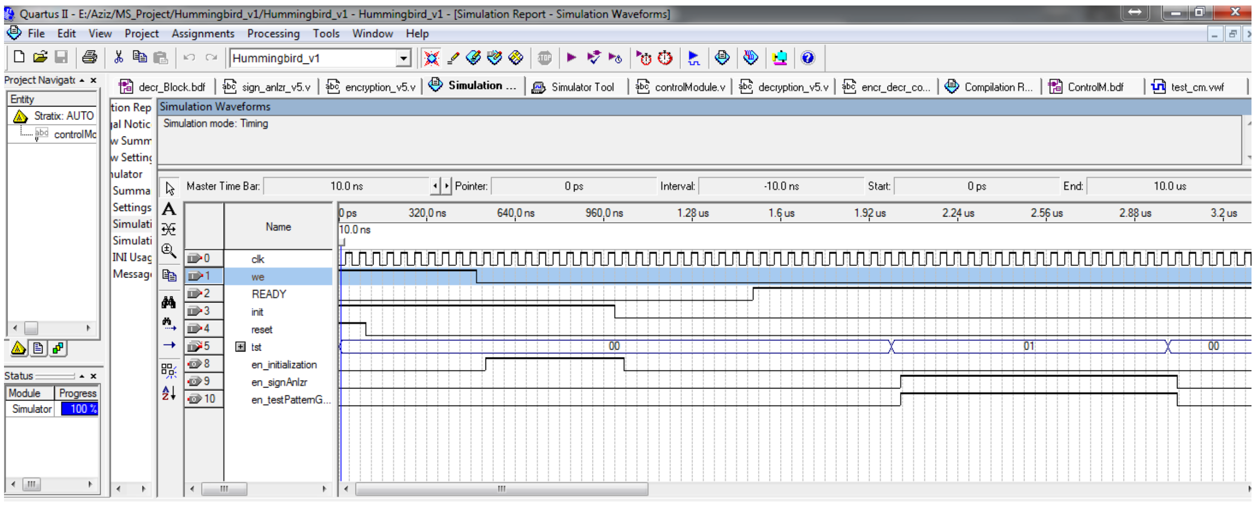


Fig.4.22: Simulation Result of Control Module

4.4 Test Result

Encryption/Decryption Functionality of the proposed ASIC is tested by using the output of the encryption block as the input to the decryption block. In this case we should get back the original plain text. In our design we have this arrangement to test the encryption followed by decryption.

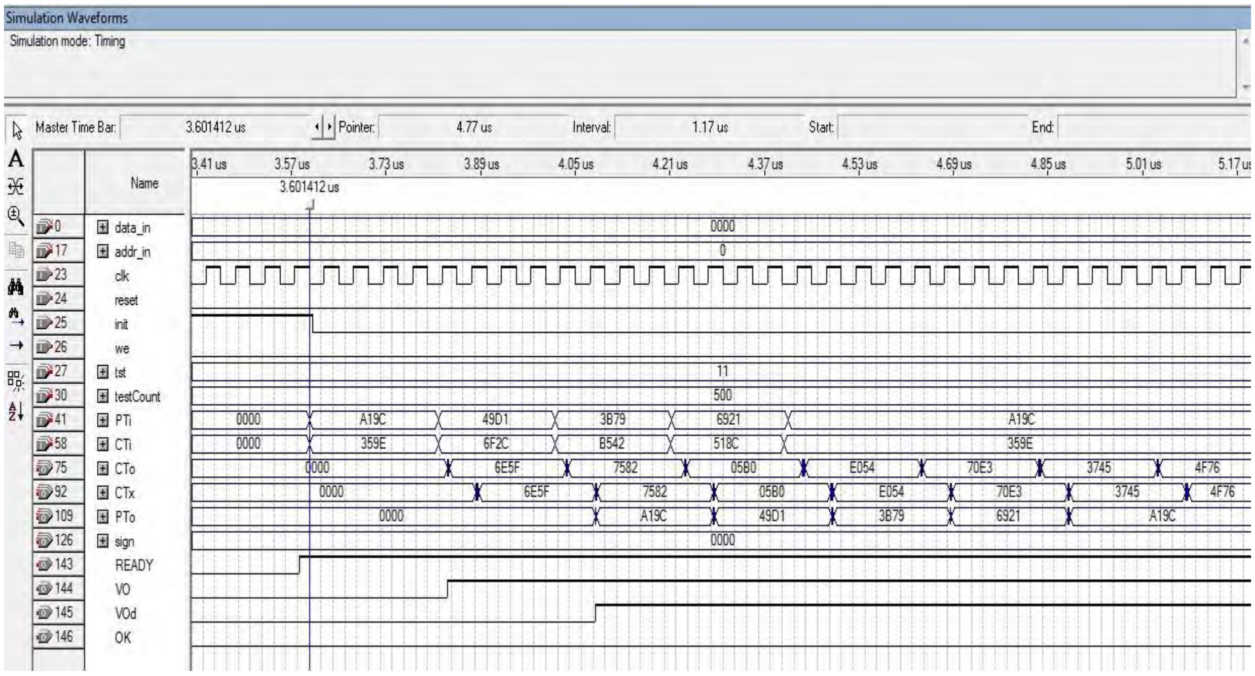


Fig.4.23: Simulation Result of Core Module (tst='11')

In this case, test mode value tst ='11' is used.

Table 4.1 Input and Output in Test Result

Encryption Block input (PTi)	Encryption Block output (CTo)	Decryption Block input (CTx)	Decryption Block output (PTo)
A19C	6E5F	6E5F	A19C
49D1	7582	7582	49D1
3B79	05B0	05B0	3B79
6921	E054	E054	6921

In the simulation result show above we have used plain text (PTi) input stream as A19C, 49D1, 3B79, 6921 for Encryption module. After 4 clock cycles we got the encrypted output (CTo) stream as 6E5F, 7582, 05B0, E054. In the next clock these cipher text used as input (CTx) to the Decryption module. After another 4 clock cycles, we got the plain text (PTo) output stream as A19C, 49D1, 3B79, 6921 which is found same as Plain Text (PTi) input to the encryption module.

4.5 HW Implementation

We have used Altera DE2 board for HW implementation. Below are the snapshots of HW implementation with different test mode.

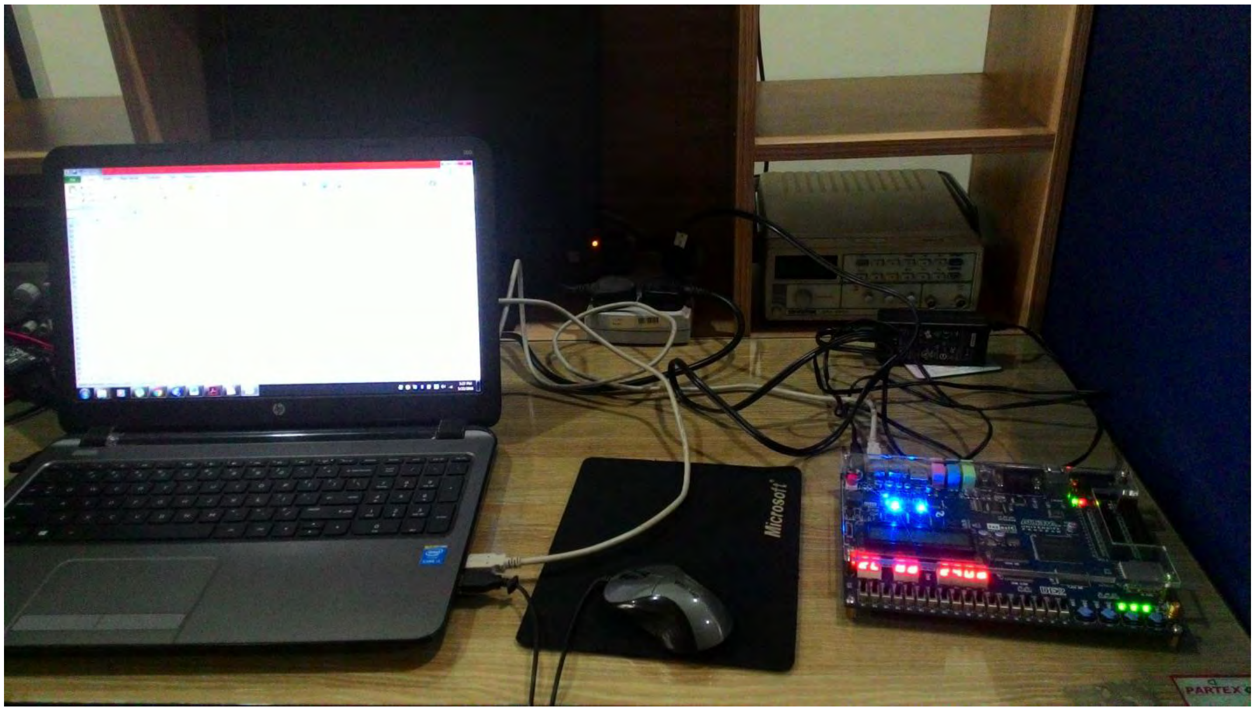


Fig.4.24: Snapshot of HW Implementation

Table 4.2 Pin Assignment

Signal Name Description in DE2	No.	Pin	Signal Name in Design
CLOCK_50		PIN_N2	clk
HEX0	6	PIN_V13	Cto
HEX0	5	PIN_V14	Cto
HEX0	4	PIN_AE11	Cto
HEX0	3	PIN_AD11	Cto
HEX0	2	PIN_AC12	Cto
HEX0	1	PIN_AB12	Cto
HEX0	0	PIN_AF10	Cto
HEX1	6	PIN_AB24	Cto
HEX1	5	PIN_AA23	Cto
HEX1	4	PIN_AA24	Cto
HEX1	3	PIN_Y22	Cto
HEX1	2	PIN_W21	Cto
HEX1	1	PIN_V21	Cto
HEX1	0	PIN_V20	Cto
HEX2	6	PIN_Y24	Cto
HEX2	5	PIN_AB25	Cto
HEX2	4	PIN_AB26	Cto
HEX2	3	PIN_AC26	Cto
HEX2	2	PIN_AC25	Cto
HEX2	1	PIN_V22	Cto
HEX2	0	PIN_AB23	Cto
HEX3	6	PIN_W24	Cto
HEX3	5	PIN_U22	Cto
HEX3	4	PIN_Y25	Cto
HEX3	3	PIN_Y26	Cto
HEX3	2	PIN_AA26	Cto

Signal Name Description in DE2	No.	Pin	Signal Name in Design
HEX3	1	PIN_AA25	Cto
HEX3	0	PIN_Y23	Cto
HEX4	6	PIN_T3	Pto
HEX4	5	PIN_R6	Pto
HEX4	4	PIN_R7	Pto
HEX4	3	PIN_T4	Pto
HEX4	2	PIN_U2	Pto
HEX4	1	PIN_U1	Pto
HEX4	0	PIN_U9	Pto
HEX5	6	PIN_R3	Pto
HEX5	5	PIN_R4	Pto
HEX5	4	PIN_R5	Pto
HEX5	3	PIN_T9	Pto
HEX5	2	PIN_P7	Pto
HEX5	1	PIN_P6	Pto
HEX5	0	PIN_T2	Pto
HEX6	6	PIN_M4	Pto
HEX6	5	PIN_M5	Pto
HEX6	4	PIN_M3	Pto
HEX6	3	PIN_M2	Pto
HEX6	2	PIN_P3	Pto
HEX6	1	PIN_P4	Pto
HEX6	0	PIN_R2	Pto
HEX7	6	PIN_N9	Pto
HEX7	5	PIN_P9	Pto
HEX7	4	PIN_L7	Pto
HEX7	3	PIN_L6	Pto
HEX7	2	PIN_L9	Pto

Signal Name Description in DE2	**No.**	**Pin**	**Signal Name in Design**
HEX7	1	PIN_L2	Pto
HEX7	0	PIN_L3	Pto
SW[17]		PIN_V2	tst
SW[16]		PIN_V1	tst
SW[15]		PIN_U4	Pti
SW[14]		PIN_U3	Pti
SW[13]		PIN_T7	Pti
SW[12]		PIN_P2	Pti
SW[11]		PIN_P1	Pti
SW[10]		PIN_N1	Pti
SW[9]		PIN_A13	Pti
SW[8]		PIN_B13	Pti
SW[7]		PIN_C13	Pti
SW[6]		PIN_AC13	Pti
SW[5]		PIN_AD13	Pti
SW[4]		PIN_AF14	Pti
SW[3]		PIN_AE14	Pti
SW[2]		PIN_P25	Pti
SW[1]		PIN_N26	Pti
SW[0]		PIN_N25	Pti
KEY[3]		PIN_W26	
KEY[2]		PIN_P23	
KEY[1]		PIN_N23	init
KEY[0]		PIN_G26	reset
LEDG[8]		PIN_Y12	
LEDG[7]		PIN_Y18	
LEDG[6]		PIN_AA20	
LEDG[5]		PIN_U17	

Signal Name Description in DE2	**No.**	**Pin**	**Signal Name in Design**
LEDG[4]		PIN_U18	
LEDG[3]		PIN_V18	VO
LEDG[2]		PIN_W19	VOd
LEDG[1]		PIN_AF22	READY
LEDG[0]		PIN_AE22	OK
LEDR[17]		PIN_AD12	
LEDR[16]		PIN_AE12	
LEDR[15]		PIN_AE13	Sign
LEDR[14]		PIN_AF13	Sign
LEDR[13]		PIN_AE15	Sign
LEDR[12]		PIN_AD15	Sign
LEDR[11]		PIN_AC14	Sign
LEDR[10]		PIN_AA13	Sign
LEDR[9]		PIN_Y13	Sign
LEDR[8]		PIN_AA14	Sign
LEDR[7]		PIN_AC21	Sign
LEDR[6]		PIN_AD21	Sign
LEDR[5]		PIN_AD23	Sign
LEDR[4]		PIN_AD22	Sign
LEDR[3]		PIN_AC22	Sign
LEDR[2]		PIN_AB21	Sign
LEDR[1]		PIN_AF23	Sign
LEDR[0]		PIN_AE23	Sign

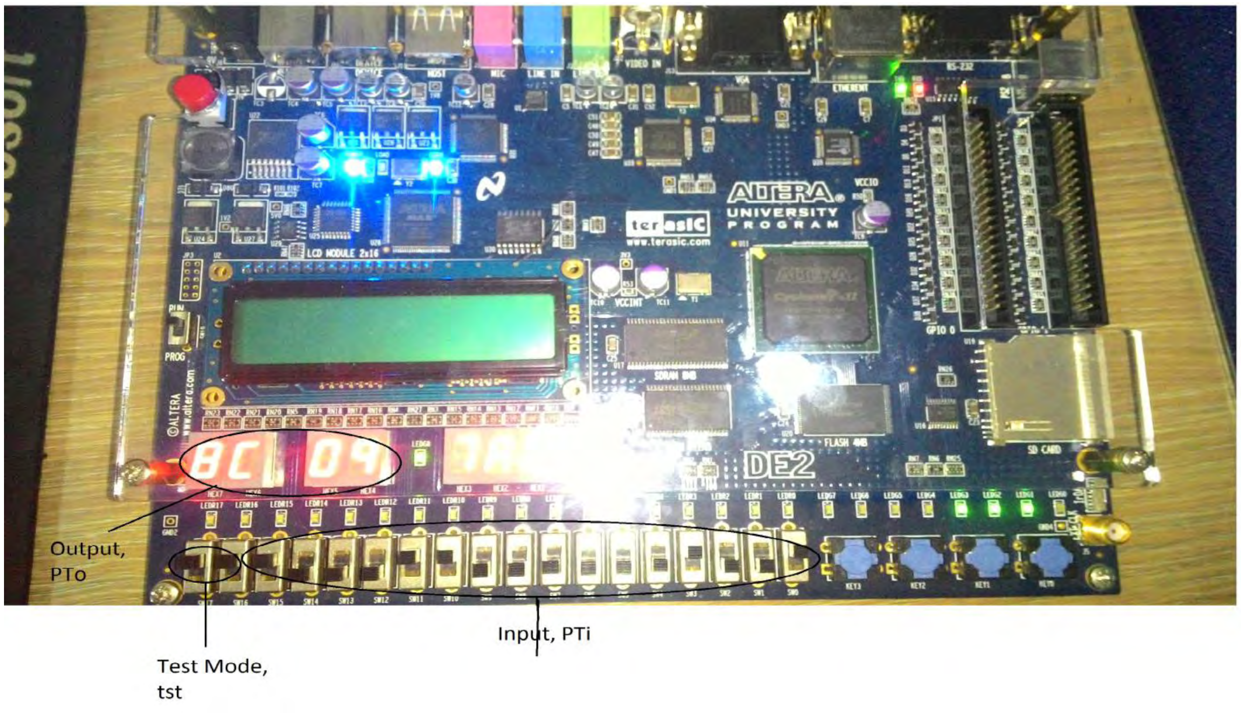


Fig.4.25: Snapshot of HW Implementation (tst='11')

In the above setup, tst is set to 11 which indicate the module is in self-test. Here SW0 to SW15 is used as input PTi and HEX4 to HEX7 is used as output. Here PTo showed the same value as PTi and thus proved the proper functionality in this mode.

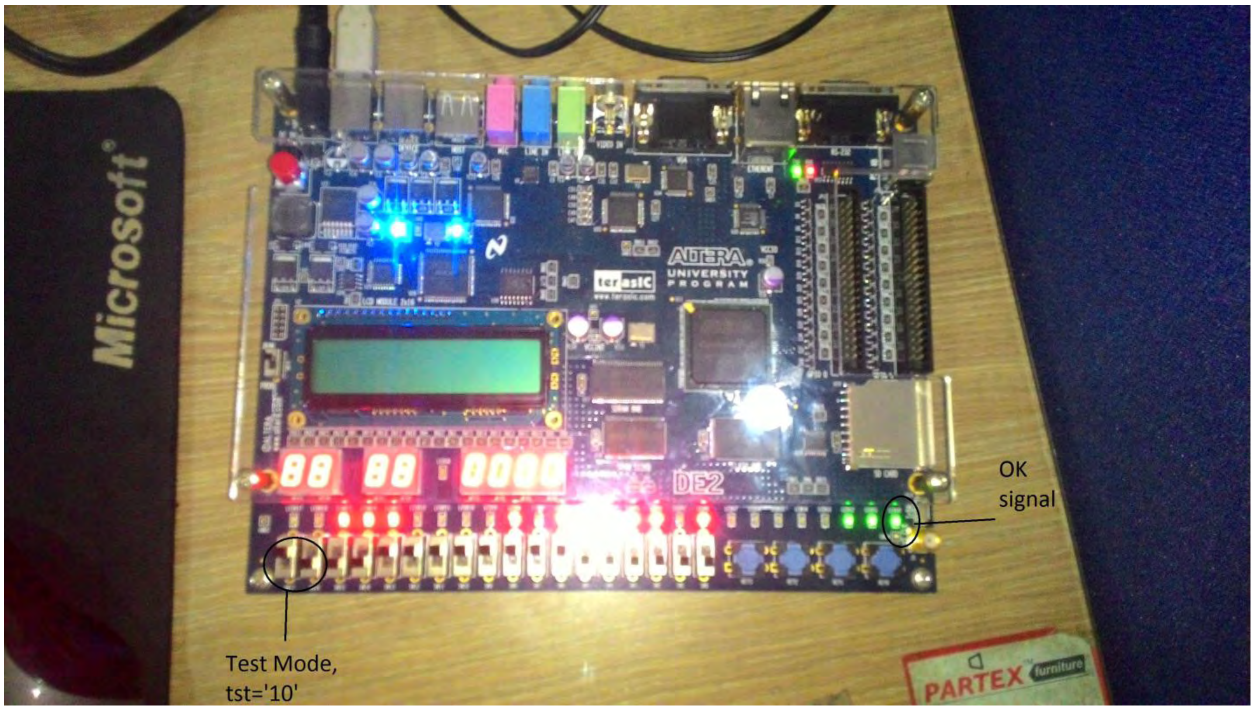


Fig.4.26: Snapshot of HW Implementation (tst='10')

In the above setup, test mode is set at 10 by SW17 and SW16, LEDR0 to LEDR15 showed the final sign and LEDG0 assigned as OK sign. High 'OK' sign showed final sign matched with the golden signature.

4.6 Power Analysis and Measurement of Power Consumption

In this section the total power consumption of the proposed ASIC is determined. We used PowerPlay Power Analyzer Tool of Quartus II for this analysis.

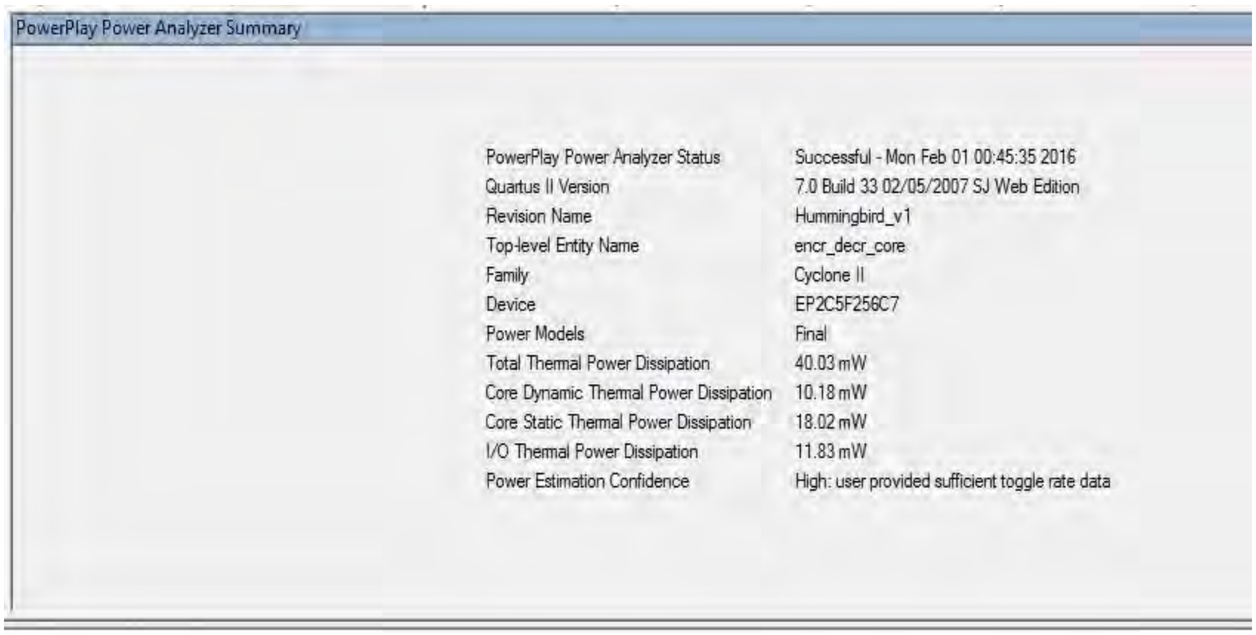
The total power consumed by a device, output loading, and external termination networks (if present) is generally comprised of the following major power components:

- Standby (Static)
- Dynamic
- I/O

It can be shown as below:

Total thermal power dissipation= Core Static Thermal Power Dissipation + Core Dynamic Thermal Power Dissipation + I/O Thermal Power Dissipation

Static Power is consumed regardless of the activity in a chip; it depends on the processor temperature. Static power does not depend on the switching activity and toggle count. So in this project we don't focus on the static power. Dynamic power is consumed from internal switching within the device (charging and discharging capacitance on internal nodes). It requires knowledge of switching activity of a node (toggle count of a node). I/O power is from external switching (charging and discharging external load capacitance connected to device pins), I/O drivers, and external termination network. Below snapshot of PowerPlay Power Analyzer report shows the component of power consumption


PowerPlay Power Analyzer Summary

```
PowerPlay Power Analyzer Status          Successful - Mon Feb 01 00:45:35 2016
Quartus II Version                       7.0 Build 33 02/05/2007 SJ Web Edition
Revision Name                            Hummingbird_v1
Top-level Entity Name                    encr_decr_core
Family                                   Cyclone II
Device                                   EP2C5F256C7
Power Models                             Final
Total Thermal Power Dissipation          40.03 mW
Core Dynamic Thermal Power Dissipation   10.18 mW
Core Static Thermal Power Dissipation    18.02 mW
I/O Thermal Power Dissipation            11.83 mW
Power Estimation Confidence              High: user provided sufficient toggle rate data
```

Fig.4.27: Power Consumption of the Proposed ASIC

4.7 Comparison with other Research Works

In this project we have implemented Hummingbird Crypto Core with BIST. Here we have used Cyclone II device of Altera family and simulation is performed in Quartus II simulation software. From simulation result we found it took 282 LAB and F_{Max} is 59.3 MHz. Maximum Throughput is calculated as 238.4 Mbps and Efficiency (Mbps/# of Slices) is calculated as 0.85. Comparing this with other Hummingbird implementations as shown in below Table 4.3 we found the performance is quite satisfactory over other works of the table in terms of Throughput and Efficiency. Above all the proposed design has the unique feature of BIST which we didn't find in other Hummingbird implementations.

Table 4.3 Comparison with other research works

Design/Cipher	Device	Total Occupied Slices/LAB	F_{Max} (MHz)	Throughput (Mbps)	Efficiency (Mbps/# of Slices)
Hummingbird[31]	Spartan-3 XC3S200-5	273	40.1	160.4	0.59
Hummingbird[34]	Spartan-3 XC3S200-5	40	260.8	55.64	1.38
Hummingbird-2[35]	Spartan-3 XC 35200	273	40.1	160.4	0.59
Hummingbird[36]	Virtex-5 XC5V1X20T-2-FF-323	4242	152.905	NA	NA
This work	Cyclone II	282	59.3	238.4	0.85

Chapter 5

Conclusion

5.1 Conclusion

The widespread deployment of various wireless networks such as mobile ad-hoc networks, sensor networks, mesh networks, personal area networks and RFID systems is making possible a world of pervasive computing a reality. While the wireless communication technology and devices under development are enabling our march toward the era of pervasive computing, the security and privacy concerns in pervasive computing remains a serious impediment to widespread adoption of emerging technologies. Employing cryptographic primitives to perform strong authentication and encryption and provide other security functionalities is a promising solution to overcome those concerns. Classical cryptographic primitives designed for full-fledged computers might not be suited for resource-constrained pervasive devices and it is often desirable to have cryptographic primitives as small as possible. As a response to the aforementioned issue, lightweight cryptography, which focuses on designing new cryptographic primitives with small footprint in hardware and low average and peak power consumption, has received a lot of attention from both academia and industry in recent years. Hummingbird is a recently proposed ultra-lightweight cryptographic algorithm targeted for low-cost smart devices. Again testability of an IC is of prime concern now a days. Built-In-Self-Test (BIST) is a norms of this day because external testing using ATE is not cost effective in this case. This project presented design of Hummingbird Crypto ASIC implementing BIST. The design is implemented in FPGA device of Family Altera, Cyclone II. Simulation result of the design ensures that the design is functioning properly. The ciphered output from the design is verified by using it as input to the decryption and revert back the original plain text.

5.2 Future Works

The work performed in this project also exposes several new areas that can be explored. In future, rigorous performance analysis and optimization in terms of power, speed and hardware resources can be performed and also to improve fault coverage different approaches of BIST like recursive pseudo exhaustive two pattern generator can be explored.

References:

[1] Stallings, W., “Cryptography and Network Security: Principles and Practices,” Pearson Education, Inc. 2010.

[2] Coppersmith, D., “The Data Encryption Standard (DES) and its strength against attacks,” IBM Journal of Research and Development, Vol. 38 (3), May 1994.

[3] Smid, M. E., and Branstad, D. K., “Data Encryption Standard: past and future,” published in proceedings of the IEEE, Vol. 76(5), May 1988.

[4] Standaert, F. -X., Rouvroy, G., and Quiswater, J. -J., “FPGA Implementations of the DES and Triple-DES Masked against Power Analysis Attacks,” published in International conference on Field Programmable Logic and Applications, 2006. FPL ’06.

[5] “Advanced encryption standard (AES),” Federal Information Processing Standards Publication (FIPS PUB) 197, National Institute of Standards and Technology (NIST), November, 2001.

[6] Fan, X., Gong, G., Lauffenburger, K., and Hicks, T., “FPGA Implementations of the Hummingbird Cryptographic Algorithm”, IEEE International Symposium on Hardware -Oriented Security and Trust (HOST), 13-14 June, 2010.

[7] Engels, D., Fan, X., Gong, G., Hu, H. and Smith, E.M., “Hummingbird: Ultra-Lightweight Cryptography for Resource- Constrained Devices”, 14th International Conference on Financial Cryptography and Data Security - FC 2010, Berlin, Germany, Springer-Verlag, 2010.

[8] Fan, X., Hu, H., Gong, G., Smith, E.M., and Engels, D., “Lightweight Implementation of Hummingbird Cryptographic Algorithm on 4-Bit Microcontrollers”, 1st International Workshop on RFID Security and Cryptography 2009 (RISC’09), pp. 838-844, 2009.

[9] K. Nivita and A. Titus, “A BIST Circuit for Fault Detection Using Recursive Pseudo Exhaustive Two Pattern Generator,” International Journal of Modern Engineering Research (IJMER), vol. 2, Issue.3, pp. 676-681, May-June 2012.

[10] Parag, K. Lala, “An introduction to logic circuit testing,” Morgan&Claypool publishers, 2008.

[11] Karaklaji , D., Kne evi , M., and Verbauwhede, I., “Low Cost Built In Self Test for Public Key Crypto Cores,” published in Workshop on Fault Diagnosis and Tolerance in Cryptography, 2010.

[12] Natale, G.D., Doulcier, M., Flottes, L., and Rouzeyre, B., “Low–Cost Self-Test of Crypto Devices,” 2nd Workshop on Dependable and Secure Nano computing, Anchorage, Canada, United States. Pp.41-46, Jun 2008.

[13] Yang, B., Wu, K., and Karri, R., “Secure Scan: A Design-for-Test Architecture for Crypto Chips,” published in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 25(10), August, 2006.

[14] http://en.wikipedia.org/wiki/Built-in_self-test; last modified on 25 October 2014 at 19:54.

[15] Poschmann, A., “Lightweight Cryptography - Cryptographic Engineering for a Pervasive World,” Ph.D. Thesis, Department of Electrical Engineering and Information Sciences, Ruhr-University Bochum, Bochum, Germany, 2009.

[16] Feldhofer, M., Dominikus, S., and Wolkerstorfer, J., “Strong Authentication for RFID Systems Using the AES Algorithm,” The 6th International Workshop on Cryptographic Hardware and Embedded Systems-CHES 2004, LNCS 3156, M. Joye and J.-J. Quisquater (eds.), Berlin, Germany: Springer-Verlag, pp. 357-370, 2004.

[17] Feldhofer, M., Wolkerstorfer, J. and Rijmen, V., “AES Implementation on a Grain of Sand”, IEE Proceedings Information Security, vol. 15, no. 1, pp. 13-20, 2005.

[18] Hämäläinen, P., Alho, T., Hännikäinen, M. and Hämäläinen, T. D., “Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core”, The 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools - DSD 2006, pp. 577-583, IEEE Computer Society, 2006.

[19] Liu, D., Yang, Y., Wang, J. and Min, H., “A Mutual Authentication Protocol for RFID Using IDEA,” Auto-ID Labs White Paper, WP-HARDWARE-048, March 2009, available at http://www.autoidlabs.org/uploads/media/AUTOIDLABS-WP-HARDWARE-048.pdf.

[20] Kaps, J.-P. “Chai-Tea, Cryptographic Hardware Implementations of xTEA,” The 9th International Conference on Cryptology in India-INDOCRYPT 2008, LNCS 5356, D.R. Chowdhury, V. Rijmen, and A. Das (eds.), Berlin, Germany: Springer-Verlag, pp. 363-375, 2008.

[21] Leander, G., Paar, C., Poschmann, A. and Schramm, K. “New Lightweight DES Variants,” The 14th Annual Fast Software Encryption Workshop-FSE 2007, LNCS 4593, A. Biryukov (ed.), Berlin, Germany: Springer-Verlag, pp. 196-210, 2007.

[22] Hong, D. , Sung, J., Hong, S. , Lim, J., Lee, S., Koo, B. S. , Lee, C. , Chang, D. , Lee, J., Jeong, Kim, K. H., and Chee, S., "HIGHT: A New Block Cipher Suitable for Low-Resource Device," The 8th International Workshop on Cryptographic Hardware and Embedded Systems-CHES 2006, LNCS 4249, L. Goubin and M. Matsui (eds.), Berlin, Germany: Springer-Verlag, pp. 46-59, 2006.

[23] Lim , C. and Korkishko, T., "mCrypton - A Lightweight Block Cipher for Security of Low-cost RFID Tags and Sensors," Workshop on Information Security Applications-WISA 2005, LNCS 3786, Song, J.T. and Yung (eds.), M. Berlin, Germany: Springer-Verlag, pp. 243-258, 2005.

[24] Standaert, F.-X. , Piret, G. , Gershenfeld, N. and Quisquater, J.-J., "SEA: A Scalable Encryption Algorithm for Small Embedded Applications," The 7th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications-CARDIS 2006, LNCS 3928, Domingo-Ferrer, J., Posegga, J. and Schreckling (eds.), D. Berlin, Germany: Springer-Verlag, pp. 222-236, 2006.

[25] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y. and Vikkelsoe, C., "PRESENT: An Ultra-Lightweight Block Cipher," The 9th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2007, LNCS 4727, Paillier, P. and Verbauwhede (eds.), I. Berlin, Germany: Springer-Verlag, pp. 450-466, 2007.

[26] De Cannière, C., Dunkelman, O. and Kne evi , M., "KATAN and KTANTAN – A Family of Small and Efficient Hardware-Oriented Block Ciphers," The 11th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2009, LNCS 5747, Clavier , C. and Gaj (eds.), K. Berlin, Germany: Springer-Verlag, pp. 272-288, 2009.

[27] Hell, M., Johansson, T. and Meier, W., "Grain: A Stream Cipher for Constrained Environments," International Journal of Wireless and Mobile Computing, vol. 2, no. 1, pp. 86-93, 2007.

[28] De Cannière, C. and Preneel, B. "Trivium – A Stream Cipher Construction Inspired by Block Cipher Design Principles," ECRYPT Stream Cipher, Available at http://www.ecrypt.eu.org/stream/papersdir/2006/021.pdf, 2005.

[29] Babbage, S. and Dodd, M. “The Stream Cipher MICKEY 2.0,” ECRYPT Stream Cipher, Available at http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf, 2006.

[30] Eisenbarth,T., Kumar,S., Paar, C., Poschmann, A. and Uhsadel, L., “A Survey of Lightweight-Cryptography Implementations,” IEEE Design & Test of Computers, vol. 24, no. 6, pp. 522-533, 2007.

[31] Fan, X., Gong, G., Lauffenburger, K., and Hicks, T., “Design Space Exploration of Hummingbird Implementations on FPGAs”. (2010) The citeseerx website. [Online] Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.233.1601.

[32] “Stratix Device Handbook, Volume 1,” on Altera website. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stx/stratix_handbook.pdf

[33] (2016) Altera website. [Online]. Available: https://www.altera.com/products/fpga/overview.html

[34] San, I., and At, N., "Compact Hardware Architecture for Hummingbird Cryptographic Algorithm," 21st International Conference on Field Programmable Logic and Applications, Chania, 2011, pp. 376-381, 2011.

[35] John, J., “Performance Analysis of New Light Weight Cryptographic Algorithms,” IOSR Journal of Computer Engineering (IOSRJCE) ISSN: 2278-0661, ISBN: 2278-8727, vol. 5, Issue 5, pp. 01-04, Sep-Oct. 2012.

[36] Arora, N., and Gigras, Y., “FPGA Implementation of Low Power and High Speed Hummingbird Cryptographic Algorithm,” International Journal of Computer Applications (0975-8887), vol. 92- No.16, April 2014.