Designing of Pseudo Random Number Generator Using LFSR

By
Darshil Patel (11BCE070)
Reshma Sanghariyat (11BCE127)

NIRMA
UNIVERSITY
INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Ahmedabad – 382481

May 2015

Designing of Pseudo Random Number Generator Using LFSR

Major Project
Submitted in partial fulfilment of the requirements
For the course of
Major Project

By
Darshil Patel (11BCE070)
Reshma Sanghariyat (11BCE127)

Guide
Prof. Darshana Upadhyay

NIRMA UNIVERSITY
INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering
Ahmedabad – 382481

May 2015

CERTIFICATE

This is to certify that the Major Project entitled **Designing of Psedo Random Number Generator using LFSR** submitted by Darshil Patel (11BCE070) and Reshma Sanghariyat (11BCE127) towards the partial fulfilment of the requirements for the completion of the **Major Project in Institute of Technology, Nirma University, Ahmedabad** is the record of work carried out by them under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this project, to the best of my knowledge, haven’t been submitted to any other university or institution.

Prof. Darshana Upadhyay	Dr. Sanjay Garg
Assistant Professor,	HOD,
Institute of Technology,	Institute of Technology,
Nirma University,	Nirma University,
Ahmedabad	Ahmedabad

ACKNOWLEDGEMENT

We would like to sincerely thank our guide, Prof. Darshana Upadhyay for her continuous guidance and encouragement throughout the course.

We are thankful to Dr. Sanjay Garg, Head of Department, Department Of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad and to Dr. Ketan Kotecha, Director, Institute of Technology for providing such a course in which we were able to get the insights to some booming topics in the field.

We are also thankful to our friends and classmates who have helped us in solving some of the designing problems when we were stuck. Finally, we would like to express our deep sense of gratitude to our family members who have supported and enriched us by sharing their ideas and have given us emotional and mental support.

ABSTRACT

The project focusses on creating a Pseudo Random Number Generator using the Linear Feedback Shift Registers. The project is majorly divided into two parts: Design and Implementation of a PRNG using VHDL and Implementation of Bluetooth Cipher using CUDA. The PRNG is a newly designed PRNG which can be highly useful in the fields of Information Security as well as Network Security. This PRNG is implemented using VHDL with an aim to dump it on an IC for implementing it on the hardware level. The new design has been developed keeping in mind that it is highly secure and most of the attacks can be prevented on it increasing the level of security.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

SECTION 1 Design of PRNG using LFSR

CHAPTER: 1 INTRODUCTION

1.1 The System

1.1.1 Definition of System

The primary aim of the project is to develop a Pseudo Random Number Generator using LFSRs and develop its corresponding VHDL code using the Xilinx ISE Design Suite 12.4. The PRNG will be able to generate pseudo random numbers such that it will increase the level of security for its utilities. The PRNG can be used with the existing encryption algorithms for the purpose of key generation and thus the randomly obtained keys can aid in securing the data.

1.1.2 Purpose and Objectives

The purpose of the project is to provide the users with a seemingly random list of numbers which can be used in various network protocols and can also act as a basis for information security. Also in the network security domain, pseudo random numbers are used in key generation, re-keying, authentication etc. The more is the randomness in the key, the better it is and the tougher it is to break it. Thus the random numbers generated by this PRNG aims at a high level of randomness providing the higher security.

1.1.3 About Present System

In this era of internet and fast pace, information travels at an equally fast rate. Many of the network based applications require some crucial information like credit card pins, personal details and account details through the internet. As such crucial information is supposed to be kept away from the intruders, it becomes essential to secure this data while being transferred through the network. Presently, there are a certain number of encryption and key generation algorithms providing the required security but most of them are presently breakable. GSM based telecommunication systems uses A5/1 algorithm for providing over-the-air privacy to the GSM users around the globe. As a result of the security drawbacks in all these algorithms, it is important to build a secure system having a blend of confidentiality, integrity and availability.

The present cryptography systems are designed as either stream ciphers or as block ciphers. But most of them are based on the stream ciphers where the encryption is done on the basis on bit by bit instead of encrypting a block of data at a time. The proposed system also is a stream cipher wherein bit by bit encryption is considered. This is because it is faster, easier and a more efficient. Moreover most technologies already use stream based ciphers and hence upgrading them also becomes easy. Also in case of communication technologies, the amount of data to be encrypted is not fixed as more of the data is real time; and hence, block ciphers becomes a bit illogical leading to the selection of stream ciphers.

Till date, many stream ciphers have been created which are used for the generation of pseudo random numbers. Very few of the ciphers have been designed for both hardware and software based applications. Even if done on the software base most of them are sequential, leading to a higher execution time.

The present systems are vulnerable to attacks like the linear approximation attack, Correlation attack and the Berlekamp – Massey Algorithm. The linear approximation attack is based on the known ciphers. It is approximated that whether the produced sequence of pseudo random numbers is based on an existing cipher. The correlation attack is done on the basis of correlation between two or more LFSRs. Having a higher level of correlation between the output state of an LFSR and the output state of a function combining the output states of all the LFSRs and a partial knowledge of the keystream can allow the attacker to easily determine the key using brute force. The Berlekamp – Massey algorithm is an efficient algorithm for finding the shortest LFSR for a given keystream. Thus most of the presently working systems for securing the network data are in fact not actually secure.

1.1.4 Proposed System

The proposed system is designed keeping in mind that it is resistant towards the above mentioned attacks to which the present systems are not. The proposed design of the cipher is based on the LFSRs. The architecture of the cipher uses a large number of different LFSRs and it also uses different feedback polynomials for each LFSR at different times. Such an architecture aids in making the design more secure from the attackers. In the used architecture, an LFSR pool is also used from which the selection of an LFSR will be done on the basis of a mathematical function. The feedback polynomial for each LFSR is constant for a particular cycle but it changes for the upcoming cycles. Also the selection of the LFSR from the pool is changed every cycle. The proposed architecture is shown in the figure.

The architecture is basically divided into four different modules namely, Counter, Generator, LFSR pool and a Non Linear Function. Each of these modules has an independent task to be done and as a result provides a pseudo random number as an output keeping in mind the basic attacks done on the present PRNGs. Each modules are explained in depth in the following chapter.

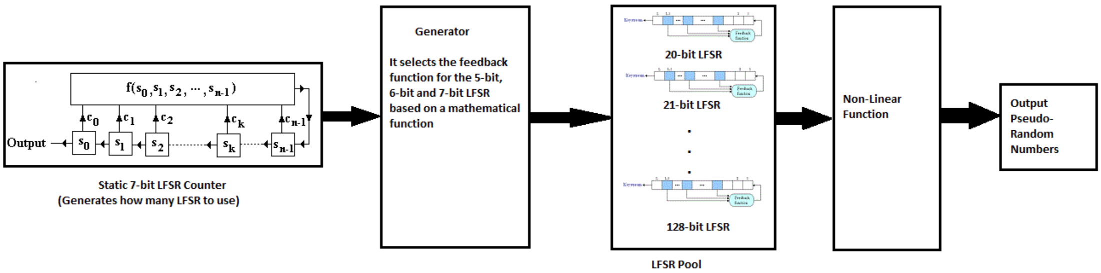
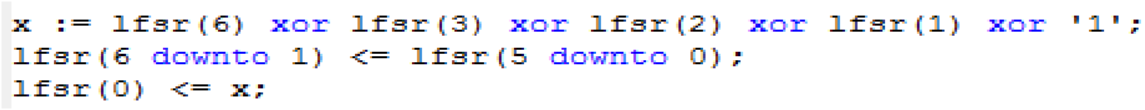

f(s0, s1, s2, ···, sn-1)
c0
c1
c2
ck
cn-1
Output
s0
s1
s2
sk
sn-1
Static 7-bit LFSR Counter
(Generates how many LFSR to use)
Generator
It selects the feedback function for the 5-bit, 6-bit and 7-bit LFSR based on a mathematical function
20-bit LFSR
21-bit LFSR
128-bit LFSR
LFSR Pool
Non-Linear Function
Output Pseudo-Random Numbers

FIGURE 1: SYSTEM ARCHITECTURE

1.2.4.1 Static Counter

The counter is a static 7-bit LFSR with an initial predefined seed value. This static counter determines the number of LFSRs to be selected from the LFSR pool. As the LFSR pool contains 119 LFSRs, this particular counter is kept as a static 7-bit counter. This particular counter uses the feedback polynomial $x^7 + x^4 + x^3 + x^2 + 1$. The implementation of an LFSR using VHDL has been done as follows:



```
x := lfsr(6) xor lfsr(3) xor lfsr(2) xor lfsr(1) xor '1';
lfsr(6 downto 1) <= lfsr(5 downto 0);
lfsr(0) <= x;
```

FIGURE 2: LFSR IMPLEMENTATION IN VHDL

In one clock pulse, the counter will generate one value from the possible 2^7-1 combinations. The decimal number equivalent to the binary number obtained in a particular clock pulse of this counter is the number of LFSRs to be selected from the LFSR pool. Also, the decimal equivalence of the binary number obtained in the LFSR will be used as an input to the Generator which will determine the upcoming feedback polynomials to be used.

1.2.4.2 Generator

The second module of the architecture is the generator, which based on the output of the counter determines the feedback polynomial to be used based on a mathematical function. Considering x as the decimal equivalent to the output obtained from the static counter, below are the mathematical functions used to determine the feedback polynomials to be used for the 5-bit, 6-bit and 7-bit LFSRs.

$$\left(\frac{x-3}{25}\right)\%5+1$$

5-bit LFSR

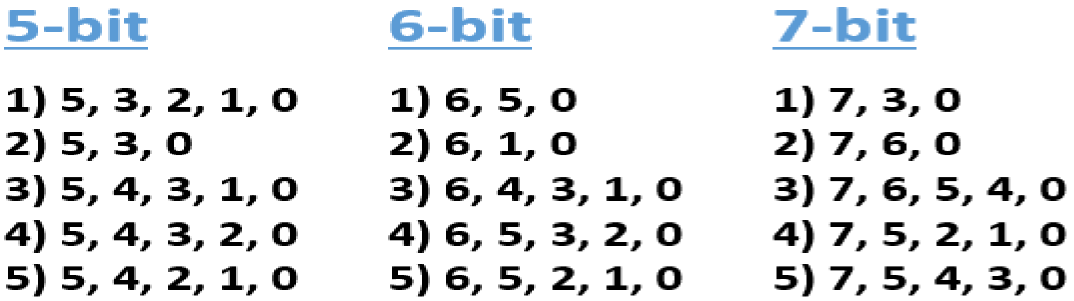
$$\left(\frac{x-3}{5}\right)\%5+1$$

6-bit LFSR

$$(x-3)\%5+1$$

7-bit LFSR

As all these mathematical functions are modulo-5 operations, the answer to each of these functions would lie between 0 to 4, but as 1 is added to each of the functions, the output of these functions have the range of 1 to 5 (both including). Depending on the output for each LFSR, the feedback polynomial is selected from the following list of polynomials.



5-bit	6-bit	7-bit
1) 5, 3, 2, 1, 0	1) 6, 5, 0	1) 7, 3, 0
2) 5, 3, 0	2) 6, 1, 0	2) 7, 6, 0
3) 5, 4, 3, 1, 0	3) 6, 4, 3, 1, 0	3) 7, 6, 5, 4, 0
4) 5, 4, 3, 2, 0	4) 6, 5, 3, 2, 0	4) 7, 5, 2, 1, 0
5) 5, 4, 2, 1, 0	5) 6, 5, 2, 1, 0	5) 7, 5, 4, 3, 0

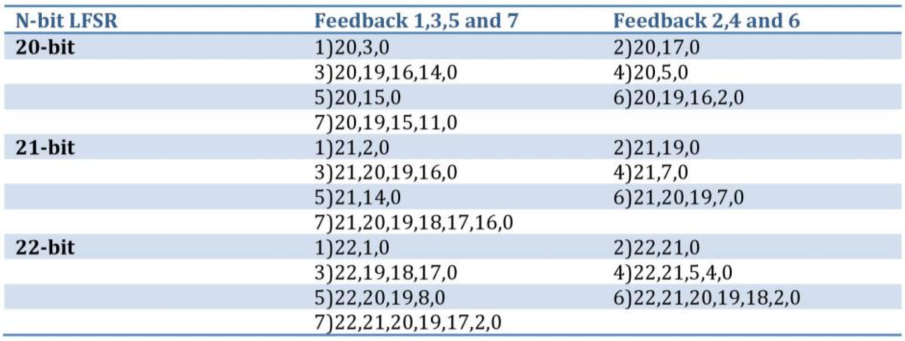
FIGURE 3: FEEDBACK POLYNOMIAL SELECTION

Based on the selected feedback polynomials for each of these LFSRs, three tables are created using each of the 5-bit, 6-bit and 7-bit LFSR with the selected feedback polynomials and a predefined seed value. The table created from the 5-bit LFSR is a 6*5 table in which each cell consists of an output generated by the 5-bit LFSR using the selected feedback polynomial. As it is a 5-bit LFSR is will generate 31 different value and the table has 30 cells, thus the last value is discarded. In a similar manner, 7*9 and 7*18 tables are created using the 6-bit LFSR and 7-bit LFSR respectively. The 6-bit LFSR can generate 63 different values, all of which can be accommodated in the 7*9 table whereas the 7-bit LFSR can generate 127 different values but the 7*18 table can accommodate 126 different values and hence here too the last value is discarded. As a result of this, we have 3 tables filled up with values and none of the value in a single table is repeating as these feedback polynomials are primitive feedback polynomials.

1.2.4.3 LFSR Pool

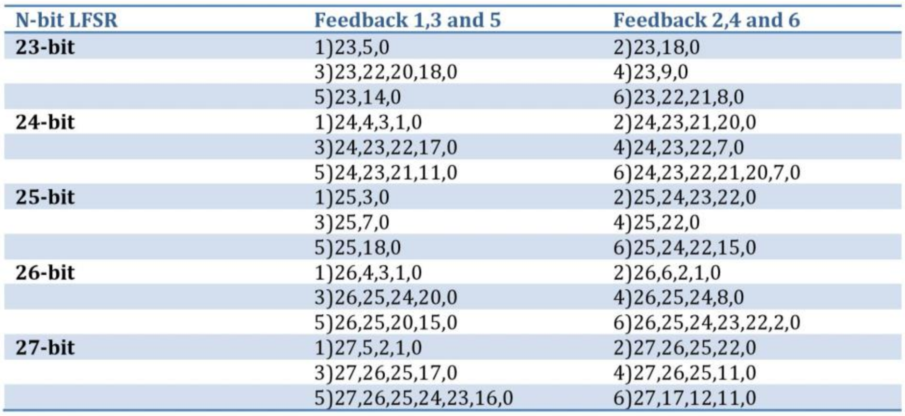
The next module is the LFSR Pool which consists of 119 different LFSRs ranging from a 20-bit LFSR to 128-bit LFSR. Here the smaller LFSRs are not used because the LFSRs till 19-bit can be broken by some means. Now based on the number generated by static counter in the first module, the number is divided equally into three parts, each for 5-bit, 6-bit and 7-bit LFSRs of second module. This division is prioritized in the decreasing order keeping in mind the exhaustion of each table. For example, if 101 is obtained from the counter, then 30 LFSRs from the pool will be selected based on the table created by 5-bit LFSR in the second module (this is because of the exhaustion of the table created by 5-bit LFSR), 36 LFSRs using the table created by 6-bit LFSR (this is because the remaining 71 are divided with giving a higher priority to 6-bit LFSR) and 35-LFSRs using the table created by 7-bit LFSR. Similarly, if the number obtained from the counter is 28, then the division would be 10, 9, and 9. Now, the selection of LFSRs is done based on the numbers obtained in the middle of each row of the corresponding tables, prioritizing the left side in case of even number of elements in a row. Also an element once selected will be discarded for the next round. For example, in case of the numbers obtained as 10, 9 and 9 respectively for the tables generated by 5-bit, 6-bit and 7-bit LFSRs, 10 numbers are selected from the 5*6 table generated using the 5-bit LFSR. This selection is taking into consideration the middle number of each rows. In case, the number of elements in the row are even, the number to the left is selected from the two numbers in the middle. Thus the 3rd element of the first row is selected as the first number out of the required 10 from the table created by the 5-bit LFSR. Hence the first 5 numbers to be selected would be the 3rd element of each of the 5 rows of the 5*6 table. Keeping in mind that each number once selected is discarded, now each row consists of 5 element from which again the selection is made in the same way. Similarly the selection is made from the tables generated by 6-bit and 7-bit LFSR. The numbers thus obtained will be used to select the LFSRs from the pool of 20-bit to 128-bit LFSR. Thus if the number obtained is 56, then a 56-bit LFSR is selected. The major problem here is that the numbers less than 20 are also present in the tables created earlier. Hence in case when such a number which is less than 20 is obtained, 20 is added to that number and corresponding selection of the LFSR is done from the pool. Thus, if the number obtained from the table is 5, the 25-bit LFSR is selected. If an LFSR is already selected, then the selection of the last unused LFSR from the pool is done. Thus if the numbers obtained are 30, 128 and 10 then 30-bit and 128-bit LFSRs are selected normally. Now again a 30-bit LFSR is to be selected as 10 is less than 20. But a 30-bit LFSR is already selected and hence the last unused LFSR from the pool is to be selected now. The last LFSR of the pool is 128-bit LFSR which is also used selecting the 127-bit LFSR which is unused yet.

Another problem arises when the number obtained from the counter is greater than 109. This is because of the fact that we have only 109 LFSRs in the pool. In this case, once all 109 LFSRs are selected based on the last unused LFSR method, the selection of the LFSRs start again and the count for the upcoming selected LFSRs is increased by 1 in the same manner. Each of these 109 LFSRs too have a multiple feedback polynomials if a particular LFSR is used multiple times. The feedback polynomials for each n-bit LFSR is as shown in the following figures.



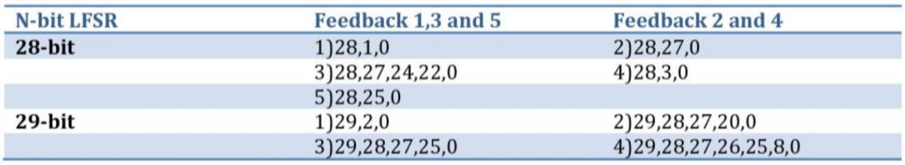
N-bit LFSR	Feedback 1,3,5 and 7	Feedback 2,4 and 6
20-bit	1)20,3,0	2)20,17,0
	3)20,19,16,14,0	4)20,5,0
	5)20,15,0	6)20,19,16,2,0
	7)20,19,15,11,0	
21-bit	1)21,2,0	2)21,19,0
	3)21,20,19,16,0	4)21,7,0
	5)21,14,0	6)21,20,19,7,0
	7)21,20,19,18,17,16,0	
22-bit	1)22,1,0	2)22,21,0
	3)22,19,18,17,0	4)22,21,5,4,0
	5)22,20,19,8,0	6)22,21,20,19,18,2,0
	7)22,21,20,19,17,2,0	

FIGURE 4: FEEDBACK POLYNOMIALS (20-BIT TO 22-BIT LFSR)



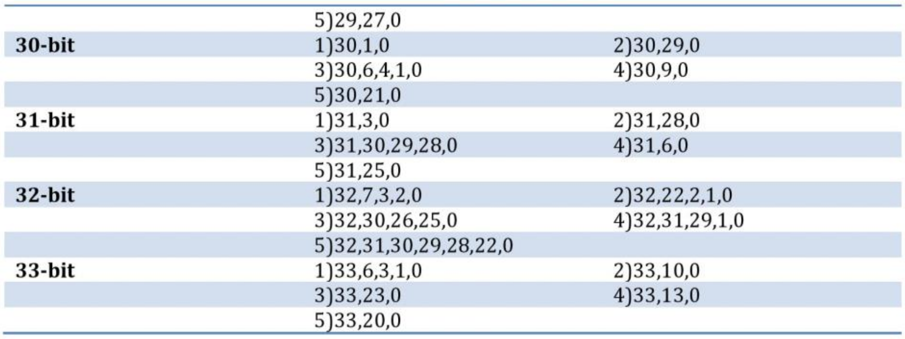
N-bit LFSR	Feedback 1,3 and 5	Feedback 2,4 and 6
23-bit	1)23,5,0	2)23,18,0
	3)23,22,20,18,0	4)23,9,0
	5)23,14,0	6)23,22,21,8,0
24-bit	1)24,4,3,1,0	2)24,23,21,20,0
	3)24,23,22,17,0	4)24,23,22,7,0
	5)24,23,21,11,0	6)24,23,22,21,20,7,0
25-bit	1)25,3,0	2)25,24,23,22,0
	3)25,7,0	4)25,22,0
	5)25,18,0	6)25,24,22,15,0
26-bit	1)26,4,3,1,0	2)26,6,2,1,0
	3)26,25,24,20,0	4)26,25,24,8,0
	5)26,25,20,15,0	6)26,25,24,23,22,2,0
27-bit	1)27,5,2,1,0	2)27,26,25,22,0
	3)27,26,25,17,0	4)27,26,25,11,0
	5)27,26,25,24,23,16,0	6)27,17,12,11,0

FIGURE 5: FEEDBACK POLYNOMIALS (23-BIT TO 27-BIT LFSR)



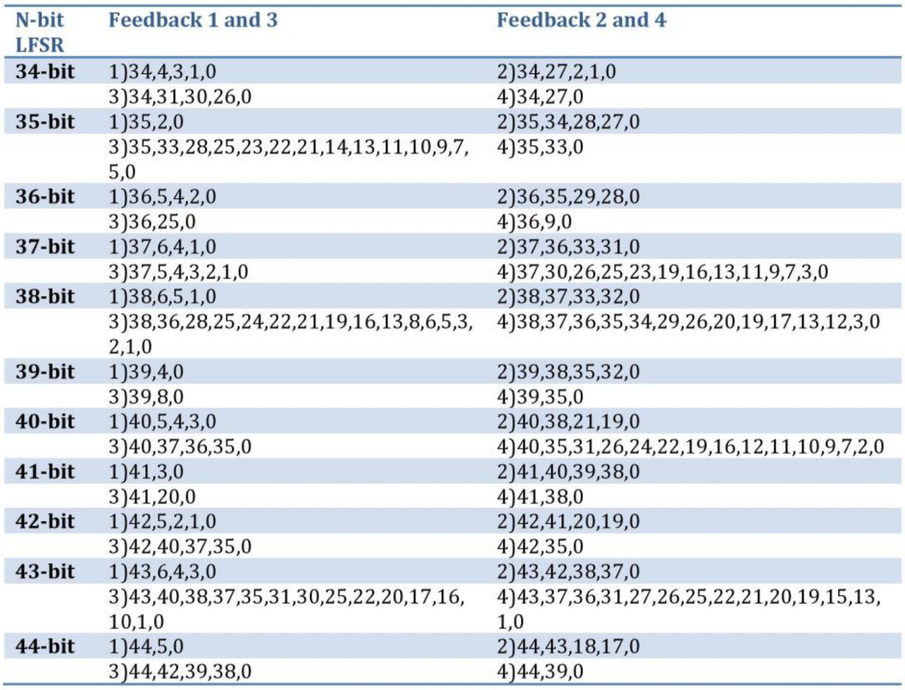
N-bit LFSR	Feedback 1,3 and 5	Feedback 2 and 4
28-bit	1)28,1,0	2)28,27,0
	3)28,27,24,22,0	4)28,3,0
	5)28,25,0	
29-bit	1)29,2,0	2)29,28,27,20,0
	3)29,28,27,25,0	4)29,28,27,26,25,8,0

FIGURE 6: FEEDBACK POLYNOMIALS (28-BIT TO 29-BIT LFSR)



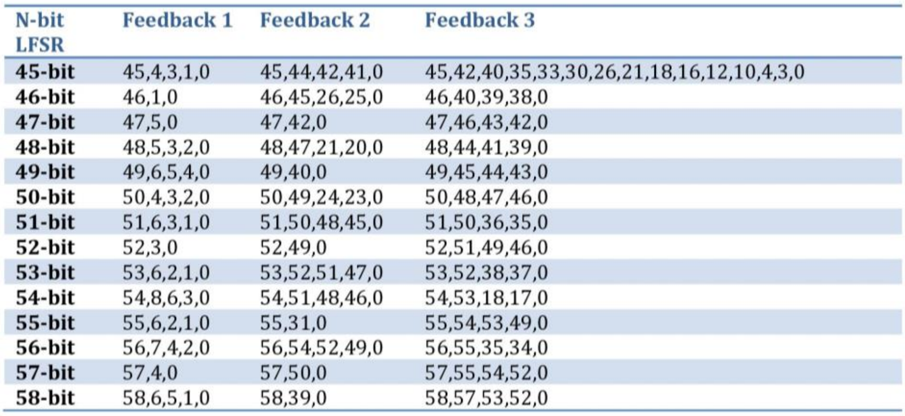
	5)29,27,0	
30-bit	1)30,1,0	2)30,29,0
	3)30,6,4,1,0	4)30,9,0
	5)30,21,0	
31-bit	1)31,3,0	2)31,28,0
	3)31,30,29,28,0	4)31,6,0
	5)31,25,0	
32-bit	1)32,7,3,2,0	2)32,22,2,1,0
	3)32,30,26,25,0	4)32,31,29,1,0
	5)32,31,30,29,28,22,0	
33-bit	1)33,6,3,1,0	2)33,10,0
	3)33,23,0	4)33,13,0
	5)33,20,0	

FIGURE 7: FEEDBACK POLYNOMIALS (30-BIT TO 33-BIT LFSR)



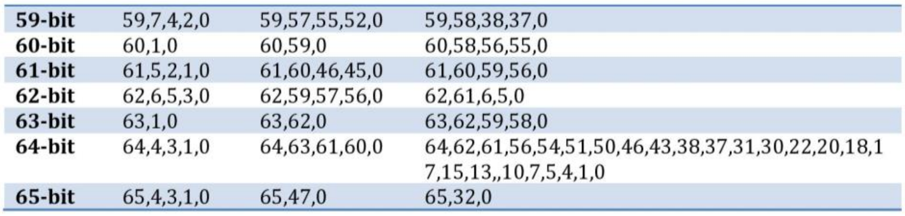
N-bit LFSR	Feedback 1 and 3	Feedback 2 and 4
34-bit	1)34,4,3,1,0	2)34,27,2,1,0
	3)34,31,30,26,0	4)34,27,0
35-bit	1)35,2,0	2)35,34,28,27,0
	3)35,33,28,25,23,22,21,14,13,11,10,9,7,5,0	4)35,33,0
36-bit	1)36,5,4,2,0	2)36,35,29,28,0
	3)36,25,0	4)36,9,0
37-bit	1)37,6,4,1,0	2)37,36,33,31,0
	3)37,5,4,3,2,1,0	4)37,30,26,25,23,19,16,13,11,9,7,3,0
38-bit	1)38,6,5,1,0	2)38,37,33,32,0
	3)38,36,28,25,24,22,21,19,16,13,8,6,5,3,2,1,0	4)38,37,36,35,34,29,26,20,19,17,13,12,3,0
39-bit	1)39,4,0	2)39,38,35,32,0
	3)39,8,0	4)39,35,0
40-bit	1)40,5,4,3,0	2)40,38,21,19,0
	3)40,37,36,35,0	4)40,35,31,26,24,22,19,16,12,11,10,9,7,2,0
41-bit	1)41,3,0	2)41,40,39,38,0
	3)41,20,0	4)41,38,0
42-bit	1)42,5,2,1,0	2)42,41,20,19,0
	3)42,40,37,35,0	4)42,35,0
43-bit	1)43,6,4,3,0	2)43,42,38,37,0
	3)43,40,38,37,35,31,30,25,22,20,17,16,10,1,0	4)43,37,36,31,27,26,25,22,21,20,19,15,13,1,0
44-bit	1)44,5,0	2)44,43,18,17,0
	3)44,42,39,38,0	4)44,39,0

FIGURE 8: FEEDBACK POLYNOMIALS (34-BIT TO 44-BIT LFSR)



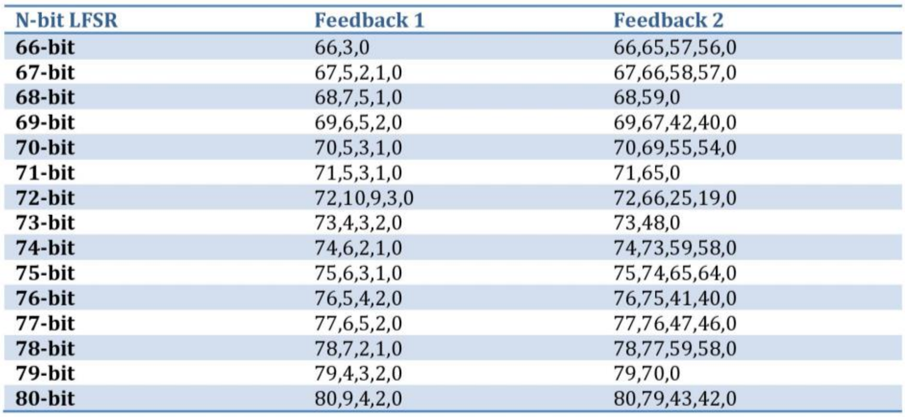
N-bit LFSR	Feedback 1	Feedback 2	Feedback 3
45-bit	45,4,3,1,0	45,44,42,41,0	45,42,40,35,33,30,26,21,18,16,12,10,4,3,0
46-bit	46,1,0	46,45,26,25,0	46,40,39,38,0
47-bit	47,5,0	47,42,0	47,46,43,42,0
48-bit	48,5,3,2,0	48,47,21,20,0	48,44,41,39,0
49-bit	49,6,5,4,0	49,40,0	49,45,44,43,0
50-bit	50,4,3,2,0	50,49,24,23,0	50,48,47,46,0
51-bit	51,6,3,1,0	51,50,48,45,0	51,50,36,35,0
52-bit	52,3,0	52,49,0	52,51,49,46,0
53-bit	53,6,2,1,0	53,52,51,47,0	53,52,38,37,0
54-bit	54,8,6,3,0	54,51,48,46,0	54,53,18,17,0
55-bit	55,6,2,1,0	55,31,0	55,54,53,49,0
56-bit	56,7,4,2,0	56,54,52,49,0	56,55,35,34,0
57-bit	57,4,0	57,50,0	57,55,54,52,0
58-bit	58,6,5,1,0	58,39,0	58,57,53,52,0

FIGURE 9: FEEDBACK POLYNOMIALS (45-BIT TO 58-BIT LFSR)



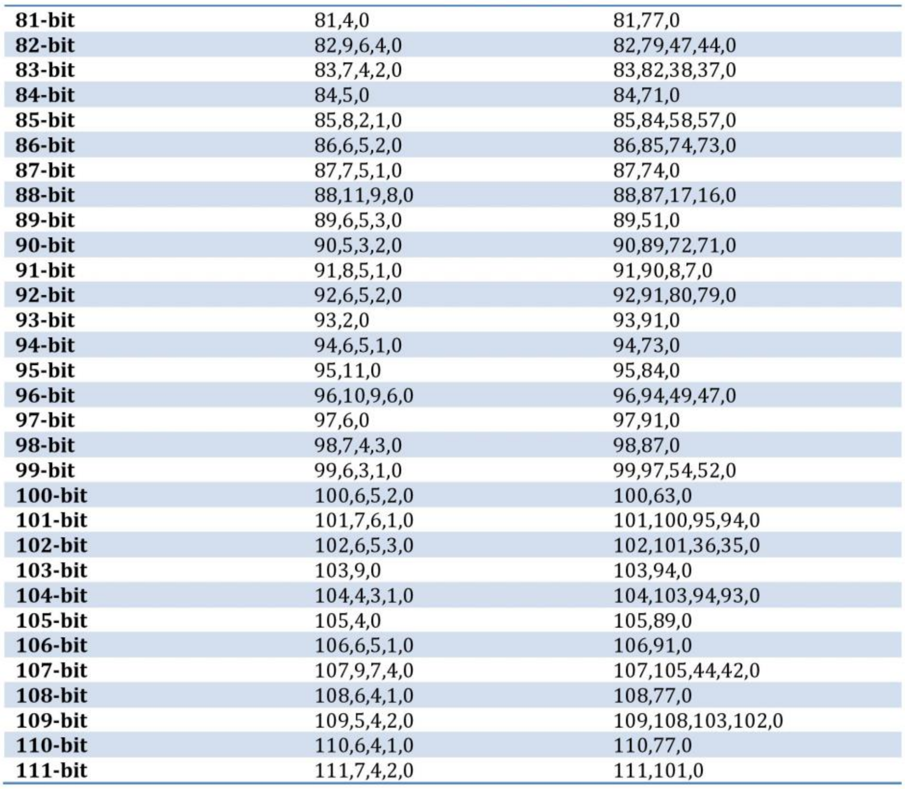
59-bit	59,7,4,2,0	59,57,55,52,0	59,58,38,37,0
60-bit	60,1,0	60,59,0	60,58,56,55,0
61-bit	61,5,2,1,0	61,60,46,45,0	61,60,59,56,0
62-bit	62,6,5,3,0	62,59,57,56,0	62,61,6,5,0
63-bit	63,1,0	63,62,0	63,62,59,58,0
64-bit	64,4,3,1,0	64,63,61,60,0	64,62,61,56,54,51,50,46,43,38,37,31,30,22,20,18,17,15,13,,10,7,5,4,1,0
65-bit	65,4,3,1,0	65,47,0	65,32,0

FIGURE 10: FEEDBACK POLYNOMIALS (59-BIT TO 65-BIT LFSR)



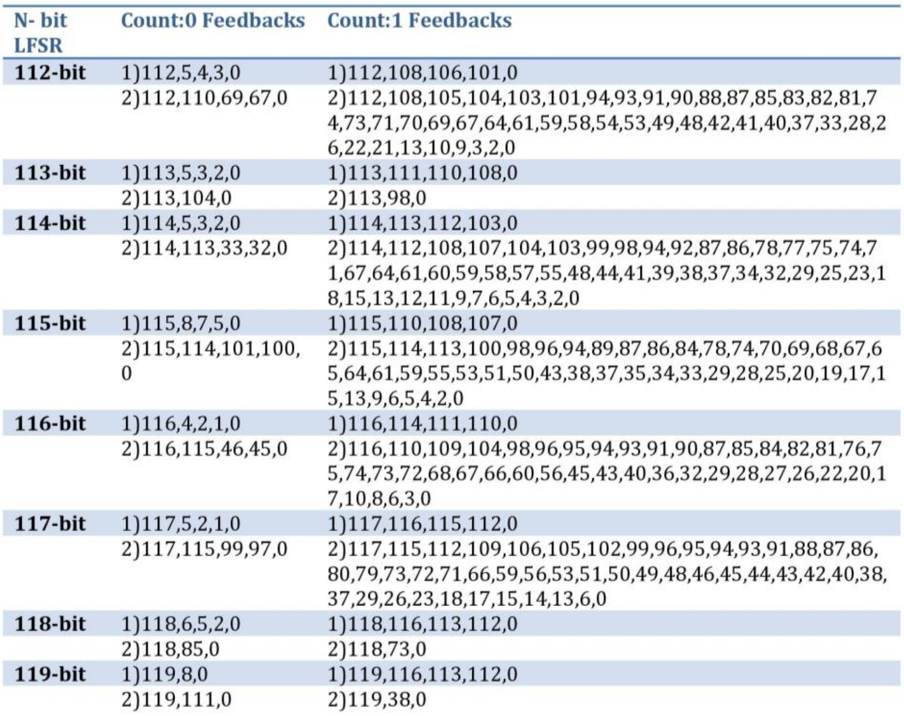
N-bit LFSR	Feedback 1	Feedback 2
66-bit	66,3,0	66,65,57,56,0
67-bit	67,5,2,1,0	67,66,58,57,0
68-bit	68,7,5,1,0	68,59,0
69-bit	69,6,5,2,0	69,67,42,40,0
70-bit	70,5,3,1,0	70,69,55,54,0
71-bit	71,5,3,1,0	71,65,0
72-bit	72,10,9,3,0	72,66,25,19,0
73-bit	73,4,3,2,0	73,48,0
74-bit	74,6,2,1,0	74,73,59,58,0
75-bit	75,6,3,1,0	75,74,65,64,0
76-bit	76,5,4,2,0	76,75,41,40,0
77-bit	77,6,5,2,0	77,76,47,46,0
78-bit	78,7,2,1,0	78,77,59,58,0
79-bit	79,4,3,2,0	79,70,0
80-bit	80,9,4,2,0	80,79,43,42,0

FIGURE 11: FEEDBACK POLYNOMIALS (66-BIT TO 80-BIT LFSR)



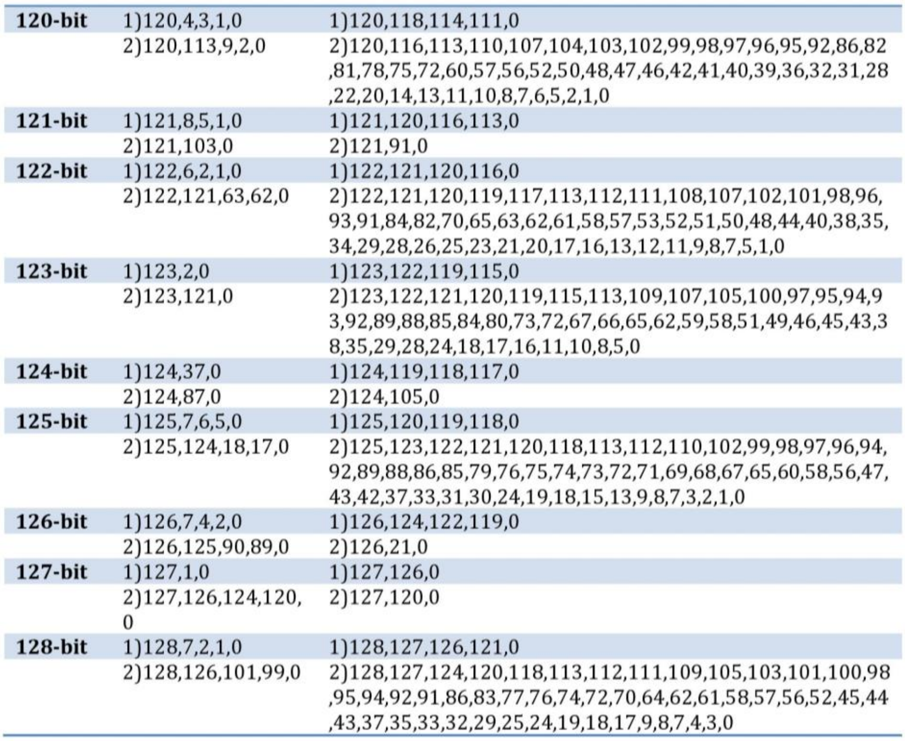
81-bit	81,4,0	81,77,0
82-bit	82,9,6,4,0	82,79,47,44,0
83-bit	83,7,4,2,0	83,82,38,37,0
84-bit	84,5,0	84,71,0
85-bit	85,8,2,1,0	85,84,58,57,0
86-bit	86,6,5,2,0	86,85,74,73,0
87-bit	87,7,5,1,0	87,74,0
88-bit	88,11,9,8,0	88,87,17,16,0
89-bit	89,6,5,3,0	89,51,0
90-bit	90,5,3,2,0	90,89,72,71,0
91-bit	91,8,5,1,0	91,90,8,7,0
92-bit	92,6,5,2,0	92,91,80,79,0
93-bit	93,2,0	93,91,0
94-bit	94,6,5,1,0	94,73,0
95-bit	95,11,0	95,84,0
96-bit	96,10,9,6,0	96,94,49,47,0
97-bit	97,6,0	97,91,0
98-bit	98,7,4,3,0	98,87,0
99-bit	99,6,3,1,0	99,97,54,52,0
100-bit	100,6,5,2,0	100,63,0
101-bit	101,7,6,1,0	101,100,95,94,0
102-bit	102,6,5,3,0	102,101,36,35,0
103-bit	103,9,0	103,94,0
104-bit	104,4,3,1,0	104,103,94,93,0
105-bit	105,4,0	105,89,0
106-bit	106,6,5,1,0	106,91,0
107-bit	107,9,7,4,0	107,105,44,42,0
108-bit	108,6,4,1,0	108,77,0
109-bit	109,5,4,2,0	109,108,103,102,0
110-bit	110,6,4,1,0	110,77,0
111-bit	111,7,4,2,0	111,101,0

FIGURE 12: FEEDBACK POLYNOMIALS (81-BIT TO 111-BIT LFSR)



N- bit LFSR	Count:0 Feedbacks	Count:1 Feedbacks
112-bit	1)112,5,4,3,0	1)112,108,106,101,0
	2)112,110,69,67,0	2)112,108,105,104,103,101,94,93,91,90,88,87,85,83,82,81,74,73,71,70,69,67,64,61,59,58,54,53,49,48,42,41,40,37,33,28,26,22,21,13,10,9,3,2,0
113-bit	1)113,5,3,2,0	1)113,111,110,108,0
	2)113,104,0	2)113,98,0
114-bit	1)114,5,3,2,0	1)114,113,112,103,0
	2)114,113,33,32,0	2)114,112,108,107,104,103,99,98,94,92,87,86,78,77,75,74,71,67,64,61,60,59,58,57,55,48,44,41,39,38,37,34,32,29,25,23,18,15,13,12,11,9,7,6,5,4,3,2,0
115-bit	1)115,8,7,5,0	1)115,110,108,107,0
	2)115,114,101,100,0	2)115,114,113,100,98,96,94,89,87,86,84,78,74,70,69,68,67,65,64,61,59,55,53,51,50,43,38,37,35,34,33,29,28,25,20,19,17,15,13,9,6,5,4,2,0
116-bit	1)116,4,2,1,0	1)116,114,111,110,0
	2)116,115,46,45,0	2)116,110,109,104,98,96,95,94,93,91,90,87,85,84,82,81,76,75,74,73,72,68,67,66,60,56,45,43,40,36,32,29,28,27,26,22,20,17,10,8,6,3,0
117-bit	1)117,5,2,1,0	1)117,116,115,112,0
	2)117,115,99,97,0	2)117,115,112,109,106,105,102,99,96,95,94,93,91,88,87,86,80,79,73,72,71,66,59,56,53,51,50,49,48,46,45,44,43,42,40,38,37,29,26,23,18,17,15,14,13,6,0
118-bit	1)118,6,5,2,0	1)118,116,113,112,0
	2)118,85,0	2)118,73,0
119-bit	1)119,8,0	1)119,116,113,112,0
	2)119,111,0	2)119,38,0

FIGURE 13: FEEDBACK POLYNOMIALS (112-BIT TO 119-BIT LFSR)



120-bit	1)120,4,3,1,0	1)120,118,114,111,0
	2)120,113,9,2,0	2)120,116,113,110,107,104,103,102,99,98,97,96,95,92,86,82,81,78,75,72,60,57,56,52,50,48,47,46,42,41,40,39,36,32,31,28,22,20,14,13,11,10,8,7,6,5,2,1,0
121-bit	1)121,8,5,1,0	1)121,120,116,113,0
	2)121,103,0	2)121,91,0
122-bit	1)122,6,2,1,0	1)122,121,120,116,0
	2)122,121,63,62,0	2)122,121,120,119,117,113,112,111,108,107,102,101,98,96,93,91,84,82,70,65,63,62,61,58,57,53,52,51,50,48,44,40,38,35,34,29,28,26,25,23,21,20,17,16,13,12,11,9,8,7,5,1,0
123-bit	1)123,2,0	1)123,122,119,115,0
	2)123,121,0	2)123,122,121,120,119,115,113,109,107,105,100,97,95,94,93,92,89,88,85,84,80,73,72,67,66,65,62,59,58,51,49,46,45,43,38,35,29,28,24,18,17,16,11,10,8,5,0
124-bit	1)124,37,0	1)124,119,118,117,0
	2)124,87,0	2)124,105,0
125-bit	1)125,7,6,5,0	1)125,120,119,118,0
	2)125,124,18,17,0	2)125,123,122,121,120,118,113,112,110,102,99,98,97,96,94,92,89,88,86,85,79,76,75,74,73,72,71,69,68,67,65,60,58,56,47,43,42,37,33,31,30,24,19,18,15,13,9,8,7,3,2,1,0
126-bit	1)126,7,4,2,0	1)126,124,122,119,0
	2)126,125,90,89,0	2)126,21,0
127-bit	1)127,1,0	1)127,126,0
	2)127,126,124,120,0	2)127,120,0
128-bit	1)128,7,2,1,0	1)128,127,126,121,0
	2)128,126,101,99,0	2)128,127,124,120,118,113,112,111,109,105,103,101,100,98,95,94,92,91,86,83,77,76,74,72,70,64,62,61,58,57,56,52,45,44,43,37,35,33,32,29,25,24,19,18,17,9,8,7,4,3,0

FIGURE 14: FEEDBACK POLYNOMIALS (120-BIT TO 128-BIT LFSR)

1.2 Project Profile

1.2.1 Project Title

The project is entitled as Designing of Pseudo Random Numbers using LFSR

1.2.2 Scope of Project

Pseudo random numbers are used in most of the network and information security applications. Such random numbers are generated by the pseudo random number generators. Thus an erroneous or a predictable output from a PRNG will result in a totally unsecure network and will fail to provide the needed security. The project undertaken is using a very large number of LFSRs for the creation of pseudo random numbers. The PRNG is designed in such a way that it is resistant to most of the attacks to which other PRNGs are vulnerable to. A try has been made to implement the design on hardware using VHDL as the basic language.

CHAPTER 2 SYSTEM ANALYSIS

2.1 Feasibility Study

Feasibility study is the process of analysing and evaluating a project to decide whether it is the project undertaken is technically, financially feasible or not.

2.1.1 Operational Feasibility

The operational feasibility deals with the operation of the system. It analyses how much does the system satisfies the requirements identified. The given project has partially met the requirements which were identified. The hardware level coding which was supposed to be done using VHDL as the basic language has been partially completed.

2.1.2 Technical Feasibility

The technical feasibility is based on understanding of the present softwares and hardware used in the project. The project at hand uses VHDL (VHSIC Hardware Description Language). The coding is to be done using Xilinx ISE Design Suite 12.4. AS the project aims at hardware implementation of the PRNG, the coding consists only of conditional statements and is devoid of control structures. This is because while implementing on the hardware, the time synchronization of the clock pulses with the one iteration of a control structure is very difficult and inaccurate. The Xilinx Design Suite produces a series of reports with a couple of hardware equivalent views which helps the developer to understand the feasibility in more detail.

2.1.3 Financial and Economic Feasibility

Financial and Economic Feasibility is the analysis of the project based on the finance it may require and the gain it will reproduce on completion. The designed PRNG is financially feasible as it doesn’t require any beforehand finance and once in use, it can attract many customers as it may provide a hardware level random number generation. Moreover with the requirement of pseudo random numbers in many applications, it can be useful for such application developers.

2.2 Context Diagram

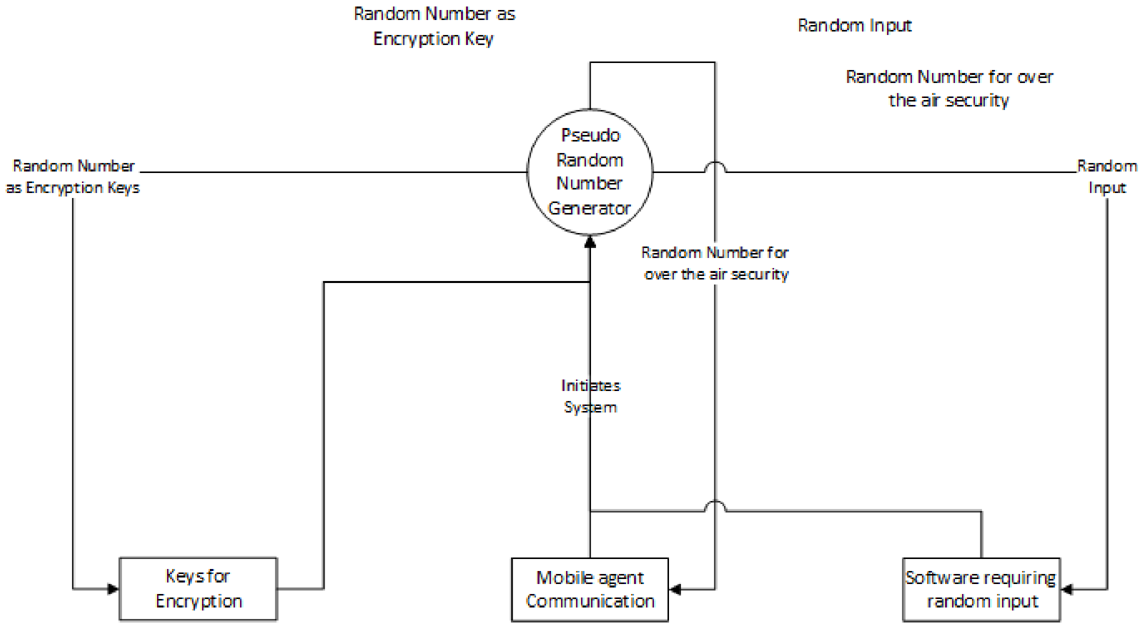

Random Number as Encryption Key
Random Input
Random Number for over the air security
Random Number as Encryption Keys
Pseudo Random Number Generator
Random Input
Random Number for over the air security
Initiates System
Keys for Encryption
Mobile agent Communication
Software requiring random input

FIGURE 15: CONTEXT DIAGRAM

The system context diagram shows the system as a high-level process and then shows its relationships with external entities. Here the relationship of Pseudo random number generator with external entities like the key generation, mobile agents and external softwares is shown. Many of the encryption algorithms uses random keys as an input to the algorithm. The system can provide an encryption key to such algorithms in the form of random numbers. In mobile agent communication, data travels through the air and in such cases over the air security becomes a concern. Here also algorithms like A5/1 which need an input pseudorandom number can make use of the system to get a random number. Many of the applications or softwares require a random input to be provided by the user. Such softwares can also interact with the system and take an input from the system to get the desired random input.

2.3 Data Flow Diagrams

2.3.1 First Level DFD

The first level Data Flow Diagram shows the main processes within the system. The system as a whole is divided into the main modules providing a modularized view. It shows the first level of details into the system. Though it doesn’t provide the in depth information about the data flow, it can help to understand the flow into the main processes of the system. Here, the initial 7-bit static counter provides a number which is used by the generator as well as the LFSR pool. This number is used by the generator to decide the feedback functions for the 5-bit, 6-bit and 7-bit LFSRs. The same number generated by the counter is used by the pool to select that many LFSRs. The feedback functions decided by the generator are then passed on to the LFSR pool which selects which LFSRs are to be selected based on the mathematical function used. The random bit stream generated by the selection of the LFSRs by the LFSR pool is a deterministic random number and hence it passes through a nonlinear function to give a pseudo random number which is used by the external entities such as shown in the context diagram.

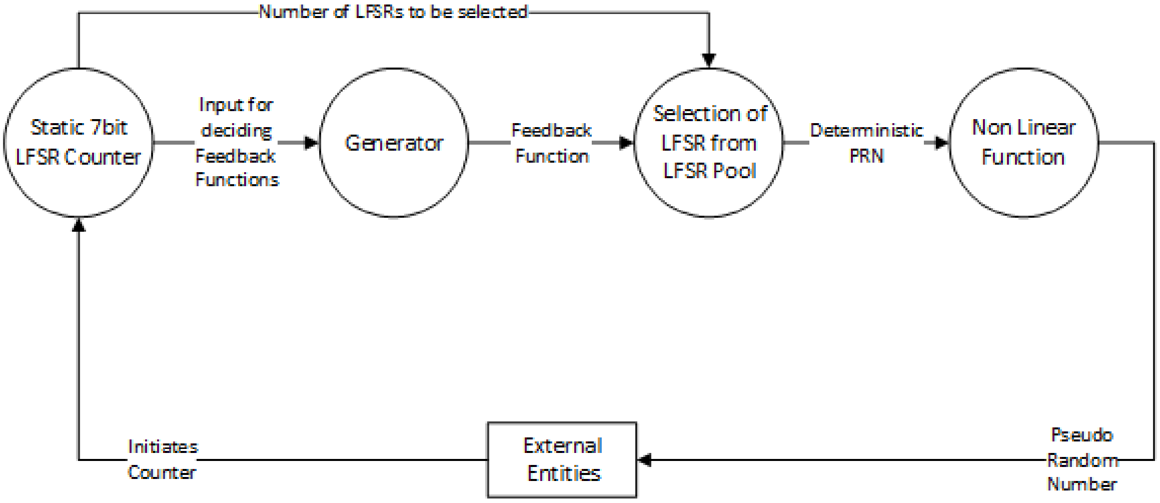


FIGURE 16: FIRST LEVEL DFD

2.3.2 Second Level DFD

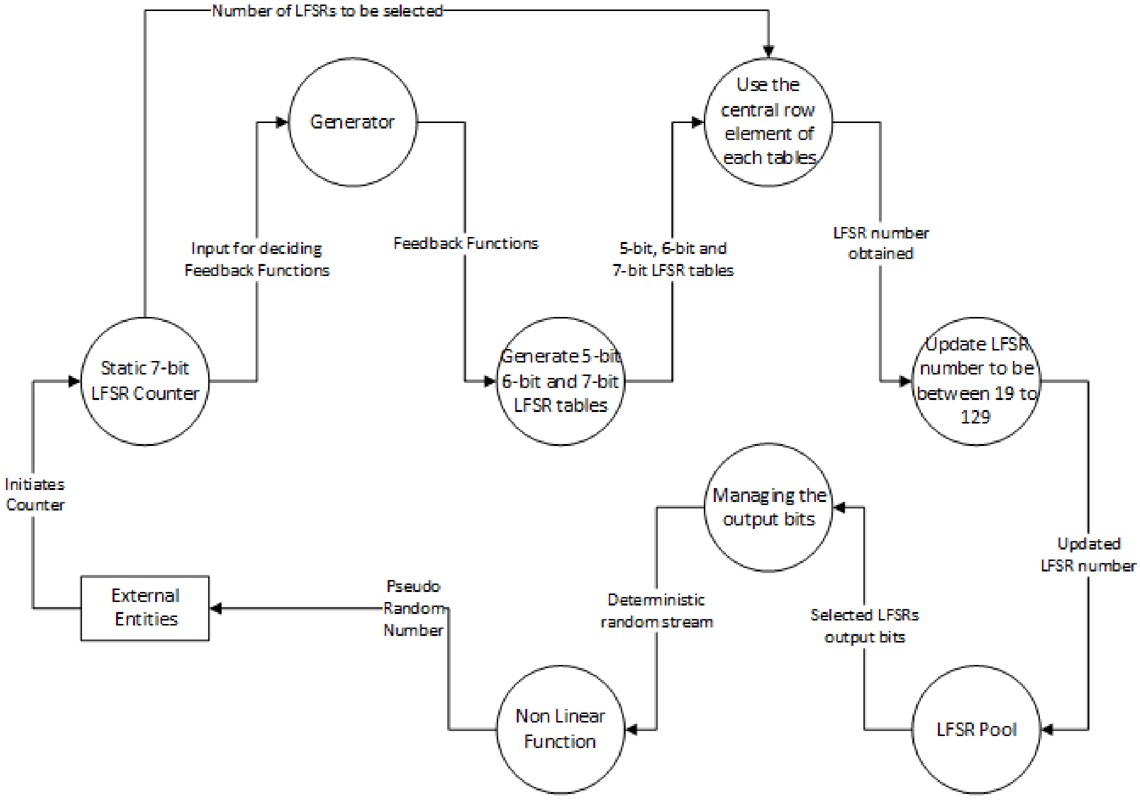

Number of LFSRs to be selected
Generator
Use the central row element of each tables
Input for deciding Feedback Functions
Feedback Functions
5-bit, 6-bit and 7-bit LFSR tables
LFSR number obtained
Static 7-bit LFSR Counter
Generate 5-bit 6-bit and 7-bit LFSR tables
Update LFSR number to be between 19 to 129
Initiates Counter
Managing the output bits
Updated LFSR number
External Entities
Pseudo Random Number
Deterministic random stream
Selected LFSRs output bits
Non Linear Function
LFSR Pool

FIGURE 17: SECOND LEVEL DFD

The second level Data Flow Diagram shows all the processes within the system. It gives an in depth flow of the system. Each and every process, whether a main process or a secondary process is to be displayed in detail. As shown in the figure, a static 7-bit counter provides a number to the generator which based on this number decides the feedback function for the 5-bit, 6-bit and 7-bit LFSRs. These functions are used for the process of generation of 6*5, 7*9 and 7*18 tables. The initial number generated by the counter is also now used in addition to these tables to select the central row elements of each row for all the tables. The number of such selections is same as the number provided by the counter. All the LFSR numbers obtained from the table are then updated to bring them between 19 and 129 if the number is less than 20. This updating of the LFSR number is done by adding 20 to the number obtained from the table, if it is less than 20, or else the number is used as it is. Next the LFSRs are selected based on these updated LFSR numbers. The output bits obtained from all the selected LFSRs is made into a deterministic stream of random numbers which passes through a nonlinear function to provide the pseudo random number. These pseudo random numbers are used by the external entities.

2.4 Future Enhancements

The project design can be modified in future to improve its level of security if any threat poses above the present cipher design. Moreover, the design can be used as a basis for newer and better designs. Due to the partial fulfilment of the implementation of the system, the design can be successfully completed so that it can be dumped onto an IC for commercial use.

CHAPTER 3 USER MANUALS

3.1 Menu Screens along with Description

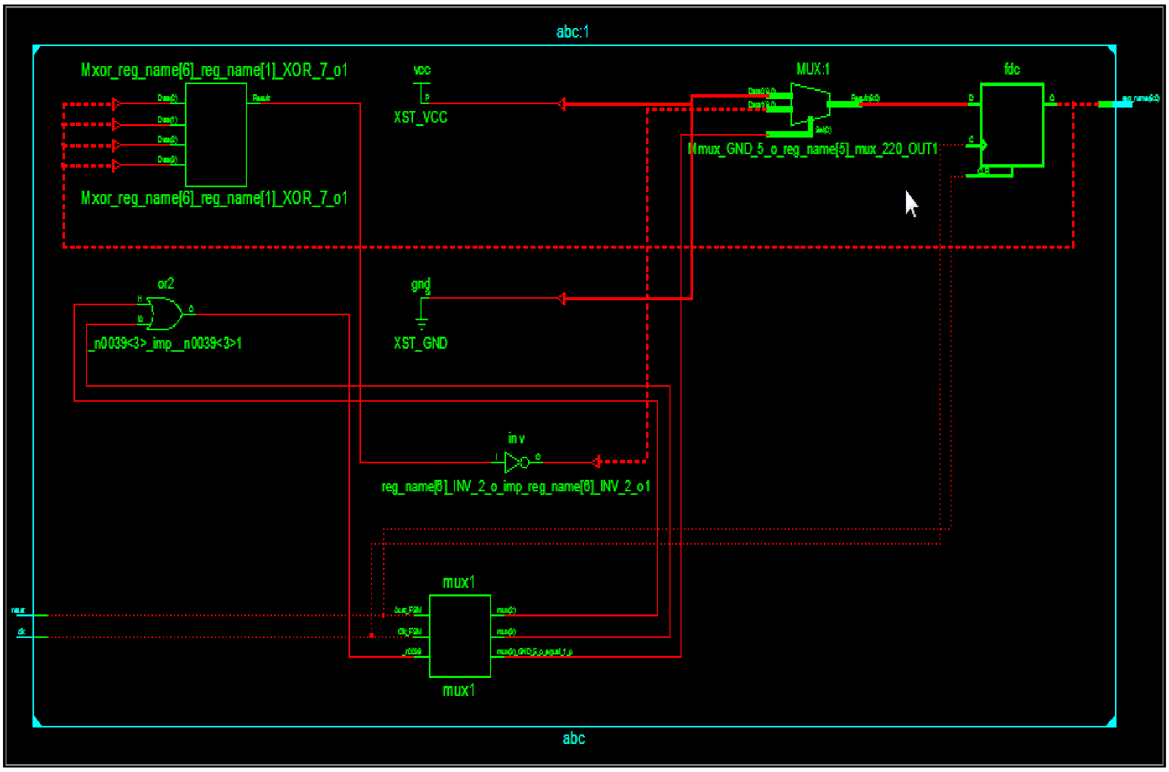

abc:1
Mxor_reg_name[6]_reg_name[1]_XOR_7_o1
vcc
XST_VCC
MUX:1
fdc
Mmux_GND_5_o_reg_name[5]_mux_220_OUT1
Mxor_reg_name[6]_reg_name[1]_XOR_7_o1
or2
_n0039<3>_imp__n0039<3>1
gnd
XST_GND
inv
reg_name[6]_INV_2_o_imp_reg_name[6]_INV_2_o1
mux1
mux1
abc

FIGURE 18: RTL SCHEMATIC

RTL View is a Register Transfer Level graphical representation of the design. It is generated at earlier stages of the synthesis process when technology mapping is still pending. The goal of this view is to be as close as possible to the original HDL code. The RTL design is independent of the targeted device and contains generic symbols, such as adders, multipliers, counters and basic gates.

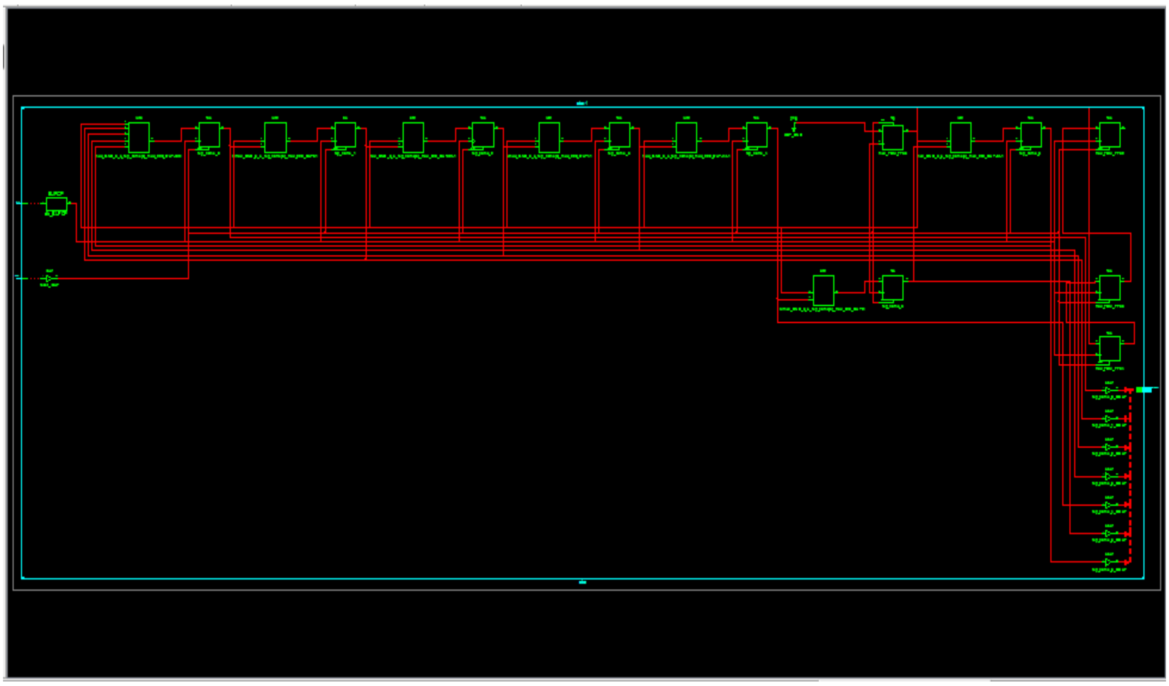


FIGURE 19: TECHNOLOGY SCHEMATIC

Technology Schematic or the Post-Synthesis Netlist is generated after the optimization and technology targeting phase of the synthesis process. It shows a representation of the design in terms of logic elements optimized to the target Xilinx device in terms of LUTs, I/O buffers, carry logics etc. The target Xilinx device used here is Spartan 6 XC6SLX45T in our case. Viewing this schematic allows us to see a technology-level representation of the HDL optimized for a specified Xilinx architecture.

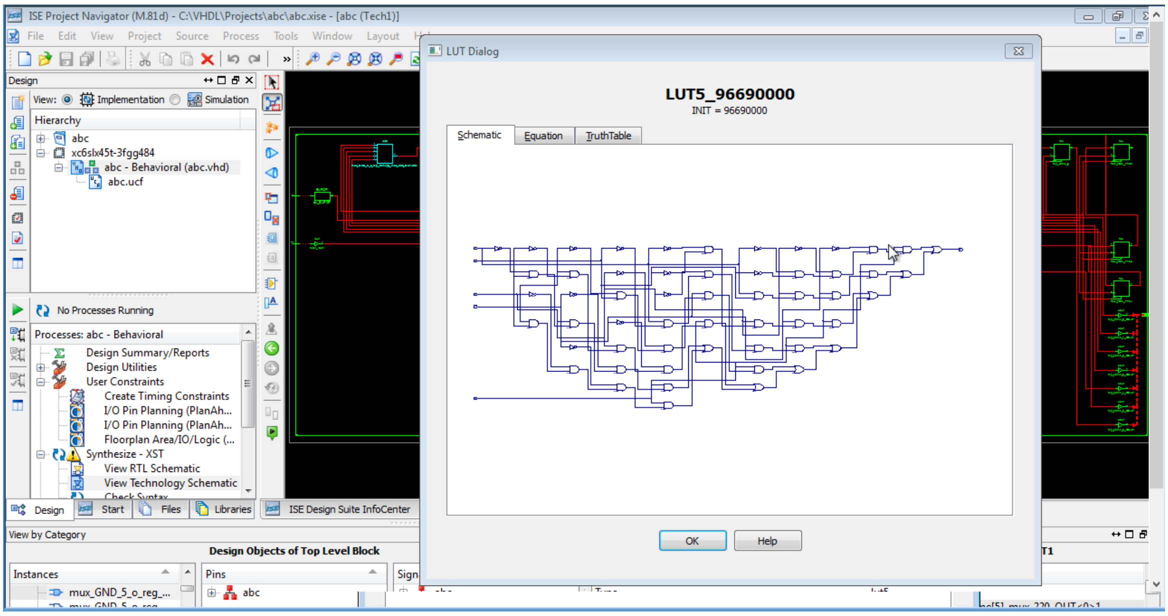
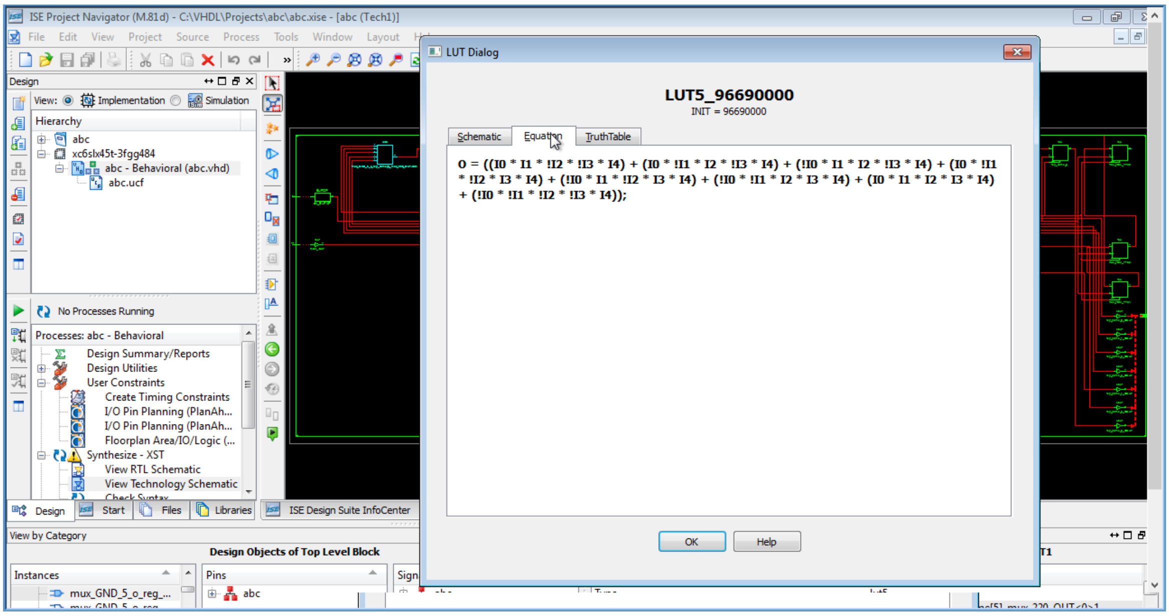

ISE Project Navigator (M.81d) - C:\VHDL\Projects\abc\abc.xise - [abc (Tech1)]
File Edit View Project Source Process Tools Window Layout
Design
View: Implementation Simulation
Hierarchy
abc
xc6slx45t-3fgg484
abc - Behavioral (abc.vhd)
abc.ucf
No Processes Running
Processes: abc - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAh...
I/O Pin Planning (PlanAh...
Floorplan Area/IO/Logic (...
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Design
Start
Files
Libraries
ISE Design Suite InfoCenter
View by Category
Design Objects of Top Level Block
Instances
mux_GND_5_o_reg_...
Pins
abc
LUT Dialog
LUT5_96690000
INIT = 96690000
Schematic
Equation
TruthTable
OK
Help

FIGURE 20: LUT SCHEMATIC



LUT Dialog

LUT5_96690000

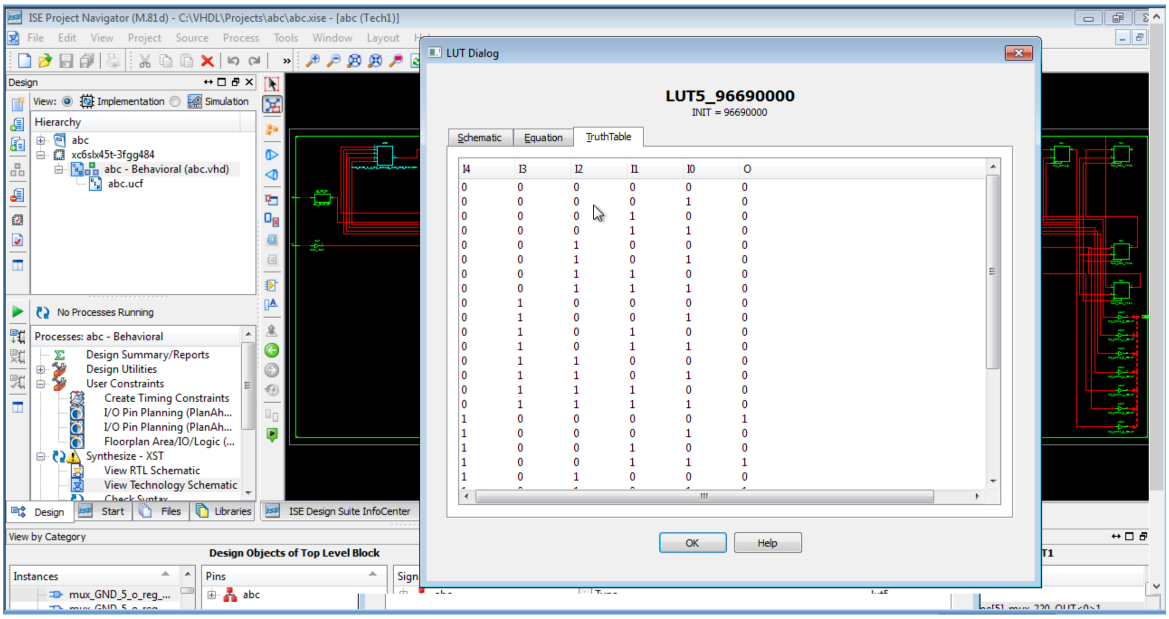
INIT = 96690000

Schematic | Equation | TruthTable

```
O = ((I0 * I1 * !I2 * !I3 * I4) + (I0 * !I1 * I2 * !I3 * I4) + (!I0 * I1 * I2 * !I3 * I4) + (I0 * !I1
* !I2 * I3 * I4) + (!I0 * I1 * !I2 * I3 * I4) + (!I0 * !I1 * I2 * I3 * I4) + (I0 * I1 * I2 * I3 * I4)
+ (!I0 * !I1 * !I2 * !I3 * I4));
```

ISE Project Navigator (M.81d) - C:\VHDL\Projects\abc\abc.xise - [abc (Tech1)]
File Edit View Project Source Process Tools Window Layout
Design
View: Implementation Simulation
Hierarchy
abc
xc6slx45t-3fgg484
abc - Behavioral (abc.vhd)
abc.ucf
No Processes Running
Processes: abc - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAh...
I/O Pin Planning (PlanAh...
Floorplan Area/IO/Logic (...
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Design Start Files Libraries
ISE Design Suite InfoCenter
View by Category
Design Objects of Top Level Block
Instances
mux_GND_5_o_reg_...
Pins
abc
OK
Help

FIGURE 21: LUT EQUATION



LUT Dialog

LUT5_96690000

INIT = 96690000

Schematic | Equation | TruthTable

I4	I3	I2	I1	I0	O
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0

ISE Project Navigator (M.81d) - C:\VHDL\Projects\abc\abc.xise - [abc (Tech1)]
File Edit View Project Source Process Tools Window Layout
Design
View: Implementation Simulation
Hierarchy
abc
xc6slx45t-3fgg484
abc - Behavioral (abc.vhd)
abc.ucf
No Processes Running
Processes: abc - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAh...
I/O Pin Planning (PlanAh...
Floorplan Area/IO/Logic (...
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Design Start Files Libraries
ISE Design Suite InfoCenter
View by Category
Design Objects of Top Level Block
Instances
mux_GND_5_o_reg_...
Pins
abc
OK
Help

FIGURE 22: LUT TRUTH TABLE

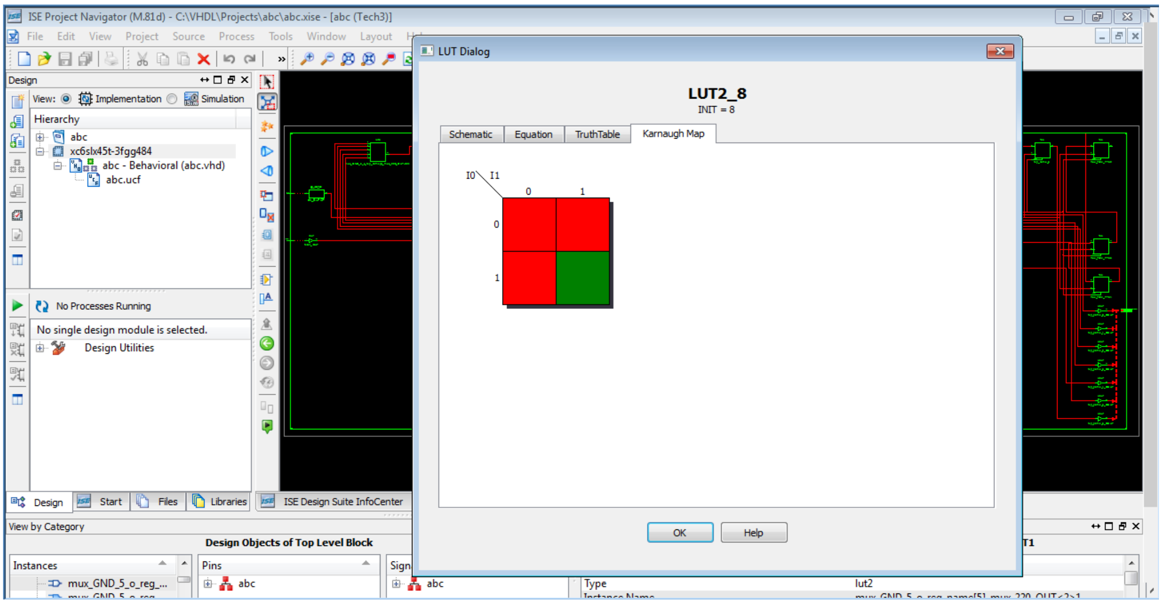
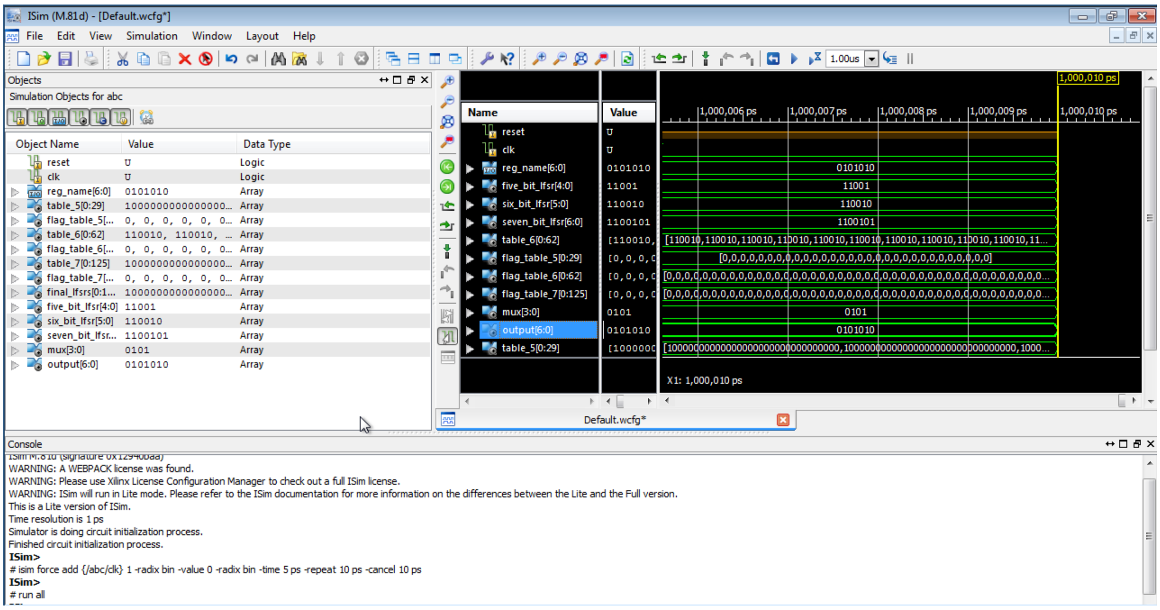

ISE Project Navigator (M.81d) - C:\VHDL\Projects\abc\abc.xise - [abc (Tech3)]
LUT Dialog
LUT2_8
INIT = 8
Schematic
Equation
TruthTable
Karnaugh Map
I0
I1
0
1
0
1
OK
Help
Design
View: Implementation Simulation
Hierarchy
abc
xc6slx45t-3fgg484
abc - Behavioral (abc.vhd)
abc.ucf
No Processes Running
No single design module is selected.
Design Utilities
Design
Start
Files
Libraries
ISE Design Suite InfoCenter
View by Category
Design Objects of Top Level Block
Instances
Pins
Type
lut2

FIGURE 23: LUT KMAP

On double clicking an LUT in the Technology Schematic gives an LUT Dialog where the Schematic Tab (Figure 3) displays the under the hood logical schematic of the LUT. It shows the LUT design using the AND gate, OR gate and NOT gate. The Equation Tab (Figure 4) displays the equation corresponding to the LUT in the Sum-of-Products form. The Truth Table Tab (Figure 5) shows the corresponding Truth Table for the given LUT whereas the Karnaugh Map Tab (Figure 6) displays the K–Map for the LUT provided the number of inputs to the LUT is less and it is easily possible to generate a K–Map.



Object Name	Value	Data Type
reset	U	Logic
clk	U	Logic
reg_name[6:0]	0101010	Array
table_5[0:29]	1000000000000000...	Array
flag_table_5[...	0, 0, 0, 0, 0, 0...	Array
table_6[0:62]	110010, 110010, ...	Array
flag_table_6[...	0, 0, 0, 0, 0, 0...	Array
table_7[0:125]	1000000000000000...	Array
flag_table_7[...	0, 0, 0, 0, 0, 0...	Array
final_lfsrs[0:1...	1000000000000000...	Array
five_bit_lfsr[4:0]	11001	Array
six_bit_lfsr[5:0]	110010	Array
seven_bit_lfsr...	1100101	Array
mux[3:0]	0101	Array
output[6:0]	0101010	Array

Console

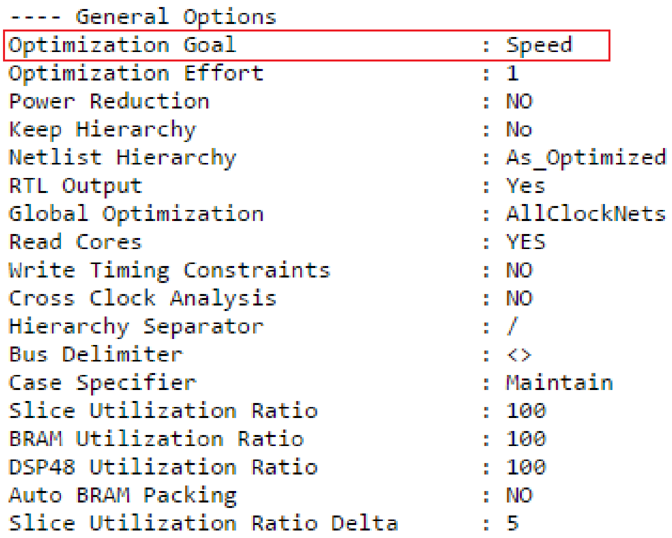
```
ISim M.81d (signature 0x12940baa)
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# isim force add {/abc/clk} 1 -radix bin -value 0 -radix bin -time 5 ps -repeat 10 ps -cancel 10 ps
ISim>
# run all
```

FIGURE 24: LSIM SIMULATION

The above figure shows the snapshot of an LSIM Simulator. The Objects Selection Screen on the top–left allows us to select the objects to be viewed during the simulation. The Console at the bottom uses the commands to perform the tasks. The required tasks can be done using the GUI or by providing appropriate commands to the console. The screen on the top–right shows the simulation of the selected objects.

3.2 Report

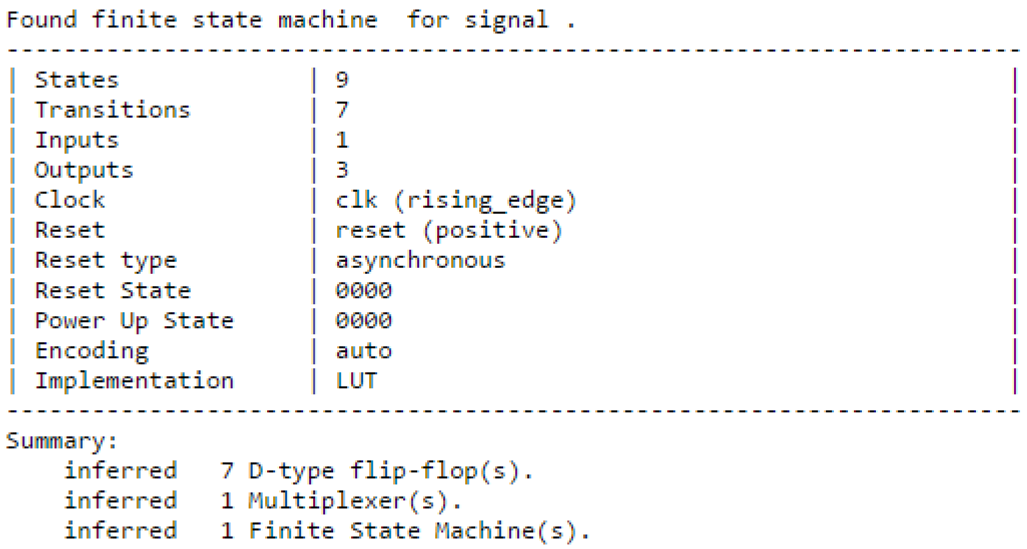
The Xilinx Design Suite 12.4 also helps by creating a number of reports for the created project. It creates the report for Synthesis Phase, Translation Phase, timing statistics, power statistics, device utilization and many more. All these reports helps in understanding the whereabouts of the project, if actually dumped onto the hardware. It also helps to reduce any of the criteria according to the requirements. The Synthesis tool also does the required optimization based on the user's requirement of power minimization or time minimization. Below mentioned are the reports of the partial project done using VHDL on the Xilinx Design Suite 12.4 which are taken on May 10, 2015.



```
---- General Options
Optimization Goal                  : Speed
Optimization Effort                : 1
Power Reduction                    : NO
Keep Hierarchy                     : No
Netlist Hierarchy                  : As_Optimized
RTL Output                         : Yes
Global Optimization                : AllClockNets
Read Cores                         : YES
Write Timing Constraints           : NO
Cross Clock Analysis               : NO
Hierarchy Separator                : /
Bus Delimiter                      : <>
Case Specifier                     : Maintain
Slice Utilization Ratio            : 100
BRAM Utilization Ratio             : 100
DSP48 Utilization Ratio            : 100
Auto BRAM Packing                  : NO
Slice Utilization Ratio Delta      : 5
```

FIGURE 25: GENERAL OPTIONS

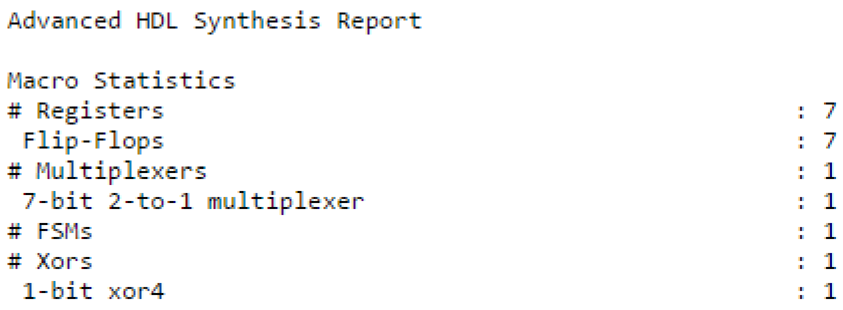
The above figure shows a part of the synthesis report where the general options for the given project are mentioned. This particular project aims at a higher speed and thus the optimization goal is speed. As the RTL Output field is set to YES, the corresponding RTL view is created.



```
Found finite state machine  for signal .
-----------------------------------------------------------------------
| States               | 9                                            |
| Transitions          | 7                                            |
| Inputs               | 1                                            |
| Outputs              | 3                                            |
| Clock                | clk (rising_edge)                            |
| Reset                | reset (positive)                             |
| Reset type           | asynchronous                                 |
| Reset State          | 0000                                         |
| Power Up State       | 0000                                         |
| Encoding             | auto                                         |
| Implementation       | LUT                                          |
-----------------------------------------------------------------------
Summary:
    inferred   7 D-type flip-flop(s).
    inferred   1 Multiplexer(s).
    inferred   1 Finite State Machine(s).
```

FIGURE 26: HDL SYNTHESIS

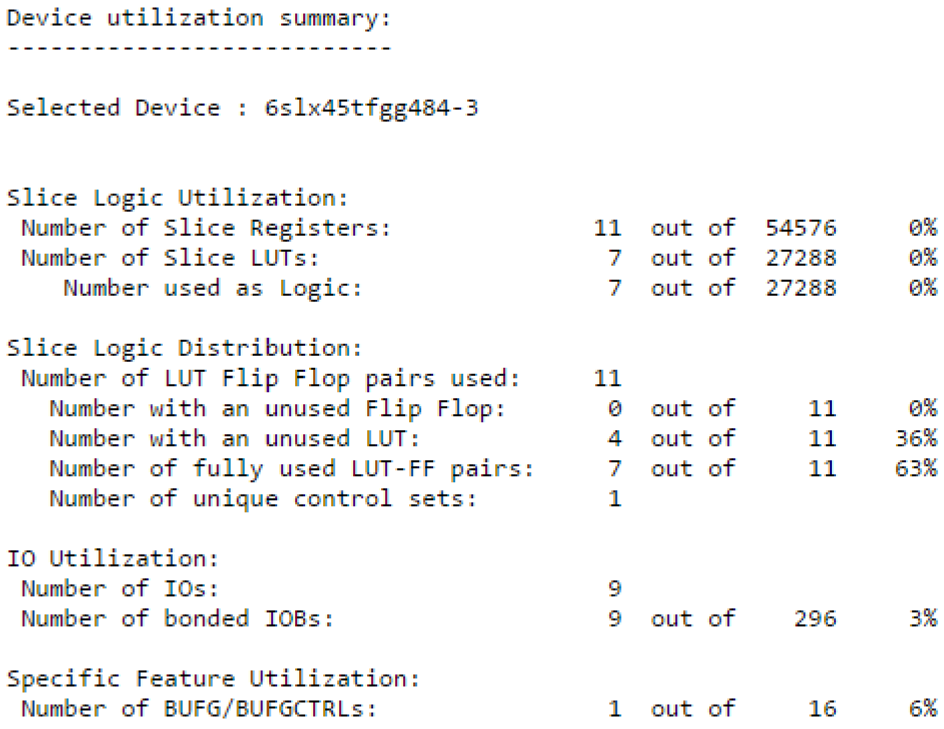
The above figure is also a part of the Synthesis Report and it shows the HDL synthesis which states the number of states, transitions, inputs and outputs in the code and infers the corresponding required number of devices for carrying out the process.



```
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                            : 7
 Flip-Flops                                            : 7
# Multiplexers                                         : 1
 7-bit 2-to-1 multiplexer                              : 1
# FSMs                                                 : 1
# Xors                                                 : 1
 1-bit xor4                                            : 1
```

FIGURE 27: ADVANCED HDL SYNTHESIS

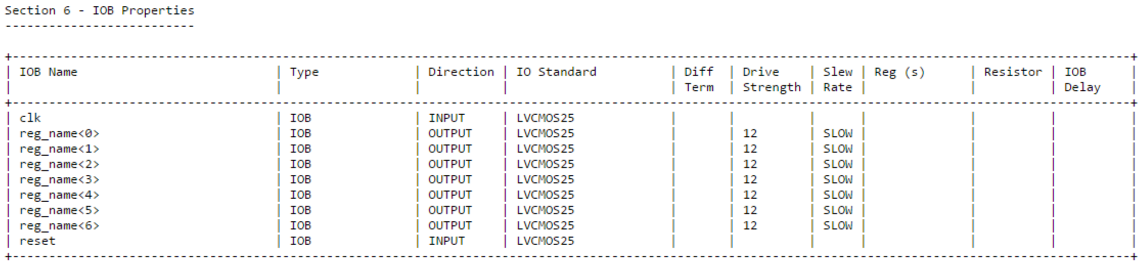
Here the macro level for the device utilization is specified. The Advanced HDL Synthesis too is a part of the Synthesis Report. It shows the number of flip-flops, multiplexers, finite state machines and XORs used.



```
Device utilization summary:
---------------------------

Selected Device : 6slx45tfgg484-3


Slice Logic Utilization:
 Number of Slice Registers:              11  out of  54576     0%
 Number of Slice LUTs:                    7  out of  27288     0%
    Number used as Logic:                 7  out of  27288     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     11
   Number with an unused Flip Flop:       0  out of     11     0%
   Number with an unused LUT:             4  out of     11    36%
   Number of fully used LUT-FF pairs:     7  out of     11    63%
   Number of unique control sets:         1

IO Utilization:
 Number of IOs:                           9
 Number of bonded IOBs:                   9  out of    296     3%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                1  out of     16     6%
```

FIGURE 28: DEVICE UTILIZATION

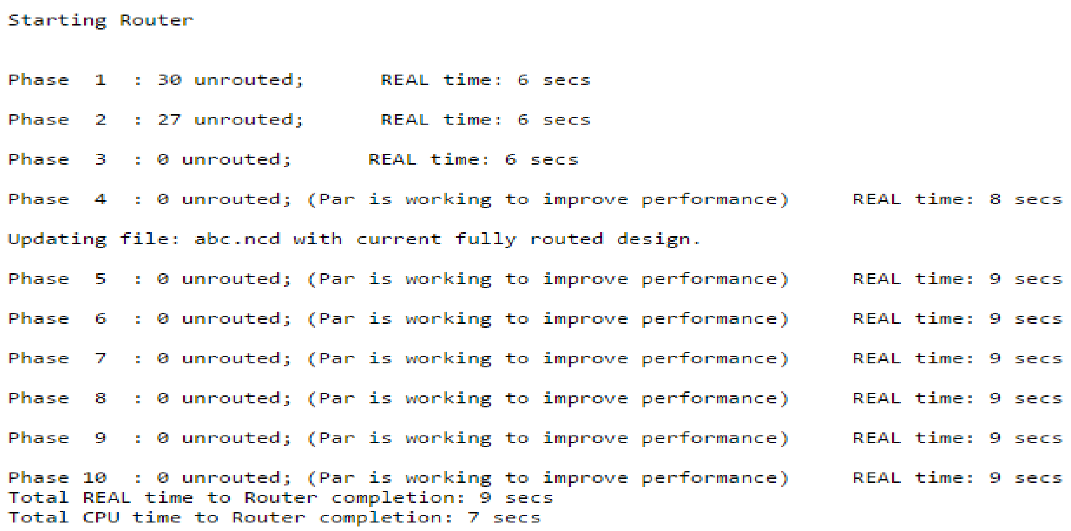
The device utilization summary of the synthesis report, as the name suggests, shows the utilization of the devices. The number of registers, LUTs, inputs, outputs and buffers. It also mentions the total number of available devices and the percentage of devices used.



```
Section 6 - IOB Properties
--------------------------

+---------------------------------------------------------------------------------------------------------------------------+
| IOB Name                         | Type             | Direction | IO Standard          | Diff  | Drive    | Slew | Reg (s)      | Resistor | IOB      |
|                                  |                  |           |                      | Term  | Strength | Rate |              |          | Delay    |
+---------------------------------------------------------------------------------------------------------------------------+
| clk                              | IOB              | INPUT     | LVCMOS25             |       |          |      |              |          |          |
| reg_name<0>                      | IOB              | OUTPUT    | LVCMOS25             |       | 12       | SLOW |              |          |          |
| reg_name<1>                      | IOB              | OUTPUT    | LVCMOS25             |       | 12       | SLOW |              |          |          |
| reg_name<2>                      | IOB              | OUTPUT    | LVCMOS25             |       | 12       | SLOW |              |          |          |
| reg_name<3>                      | IOB              | OUTPUT    | LVCMOS25             |       | 12       | SLOW |              |          |          |
| reg_name<4>                      | IOB              | OUTPUT    | LVCMOS25             |       | 12       | SLOW |              |          |          |
| reg_name<5>                      | IOB              | OUTPUT    | LVCMOS25             |       | 12       | SLOW |              |          |          |
| reg_name<6>                      | IOB              | OUTPUT    | LVCMOS25             |       | 12       | SLOW |              |          |          |
| reset                            | IOB              | INPUT     | LVCMOS25             |       |          |      |              |          |          |
+---------------------------------------------------------------------------------------------------------------------------+
```

FIGURE 29: IOB PROPERTIES

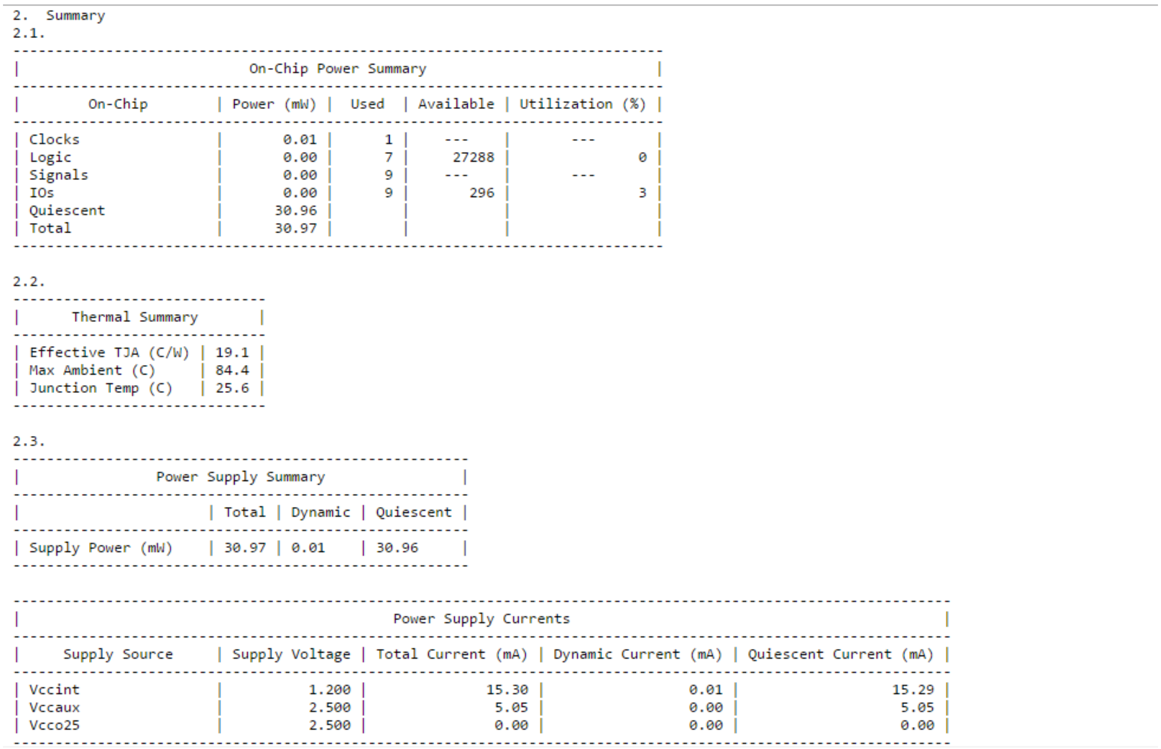
The IOB Properties is a part of the Map Report. The Synthesis Phase is followed by the implementation phase where four major tasks are done. Translate – which merges the incoming source files and the constraints into a Xilinx design file; Map – which fits the design file into the available resources on the target device; Place and Route – which places and routes the design according to the timing constraints and Programming File Generation – which generates a bitstream file which can be downloaded on the device. The slew rate shown in the figure is the maximum rate of change of output voltage per unit time.



```
Starting Router

Phase  1  : 30 unrouted;       REAL time: 6 secs

Phase  2  : 27 unrouted;       REAL time: 6 secs

Phase  3  : 0 unrouted;       REAL time: 6 secs

Phase  4  : 0 unrouted; (Par is working to improve performance)     REAL time: 8 secs

Updating file: abc.ncd with current fully routed design.

Phase  5  : 0 unrouted; (Par is working to improve performance)     REAL time: 9 secs

Phase  6  : 0 unrouted; (Par is working to improve performance)     REAL time: 9 secs

Phase  7  : 0 unrouted; (Par is working to improve performance)     REAL time: 9 secs

Phase  8  : 0 unrouted; (Par is working to improve performance)     REAL time: 9 secs

Phase  9  : 0 unrouted; (Par is working to improve performance)     REAL time: 9 secs

Phase 10  : 0 unrouted; (Par is working to improve performance)     REAL time: 9 secs
Total REAL time to Router completion: 9 secs
Total CPU time to Router completion: 7 secs
```

FIGURE 30: PLACE AND ROUTE

The Place and Route phase executes multiple phases of router. The router performs a process to find a solution that routes the design to completion and meets the timing constraints. The above figure shows the phases of the router wherein the design is routed in the first two phases but it still undergoes all the phases to improve the performance.



```
2.  Summary
2.1.
-----------------------------------------------------------------------------
|                           On-Chip Power Summary                           |
-----------------------------------------------------------------------------
|        On-Chip        | Power (mW) |  Used  | Available | Utilization (%) |
-----------------------------------------------------------------------------
| Clocks                |       0.01 |      1 |    ---    |       ---       |
| Logic                 |       0.00 |      7 |     27288 |               0 |
| Signals               |       0.00 |      9 |    ---    |       ---       |
| IOs                   |       0.00 |      9 |       296 |               3 |
| Quiescent             |      30.96 |        |           |                 |
| Total                 |      30.97 |        |           |                 |
-----------------------------------------------------------------------------

2.2.
-----------------------------
|      Thermal Summary      |
-----------------------------
| Effective TJA (C/W) | 19.1 |
| Max Ambient (C)     | 84.4 |
| Junction Temp (C)   | 25.6 |
-----------------------------

2.3.
-----------------------------------------------------
|                Power Supply Summary                |
-----------------------------------------------------
|                      | Total | Dynamic | Quiescent |
-----------------------------------------------------
| Supply Power (mW)    | 30.97 | 0.01    | 30.96     |
-----------------------------------------------------

-------------------------------------------------------------------------------------------------------------
|                                          Power Supply Currents                                          |
-------------------------------------------------------------------------------------------------------------
|     Supply Source     | Supply Voltage | Total Current (mA) | Dynamic Current (mA) | Quiescent Current (mA) |
-------------------------------------------------------------------------------------------------------------
| Vccint                |          1.200 |              15.30 |                 0.01 |                  15.29 |
| Vccaux                |          2.500 |               5.05 |                 0.00 |                   5.05 |
| Vcco25                |          2.500 |               0.00 |                 0.00 |                   0.00 |
-------------------------------------------------------------------------------------------------------------
```

FIGURE 31: POWER REPORT

The above figure shows the power consumption for the design. The supply voltage required, energy utilization by each device, total current and all the other factors are displayed.

SECTION 2 E0 CIPHER

CHAPTER 4 INTRODUCTION TO E0 CIPHER

4.1 Definition of System

The primary aim of this project is to develop a Bluetooth Cipher (E0 Cipher) using CUDA 5.5 (Compute Unified Device Architecture). Bluetooth is standard for sort range communication devices. Bluetooth standard use in very large set of devices likes wired and wireless, devices like mobile phone, mobile PC's, cars, PDA's, digital cameras and other lots of devices. Bluetooth propose methods for generating keys, Authenticating users for communication, encryption and decryption of data. Security is major concern for point to point communication of devices. For security purpose some encryption mechanism used in Bluetooth technology which is E0 cipher.

4.2 Purpose and Objective

The purpose of this project is to get less execution time compare to hardware working time and compare our parallel code with sequential code. At last, if we get less time compare to hardware so we can say that software is replaced by hardware. So main objective of our system is get best throughput as much as possible.

4.3 About Present System

Mostly every one of us has used Bluetooth functionality of mobile phones. We all are aware about Bluetooth pairing process which authenticating process, after that data will transfer, but this data is in encrypted form. This encryption process uses E0 Cipher or extended version of E0 cipher, which is implemented on hardware.

CHAPTER 5 DESIGN OF E0 CIPHER

5.1 Encryption Process

In Bluetooth standard security is one of the important or baseband layer and which control by one of the upper layer. Security layer doing tasks like user authentication, device authentication, generation of keys, management of keys, encryption of data, decryption of data. Encryption engine used within Bluetooth technology is the E0 stream cipher.

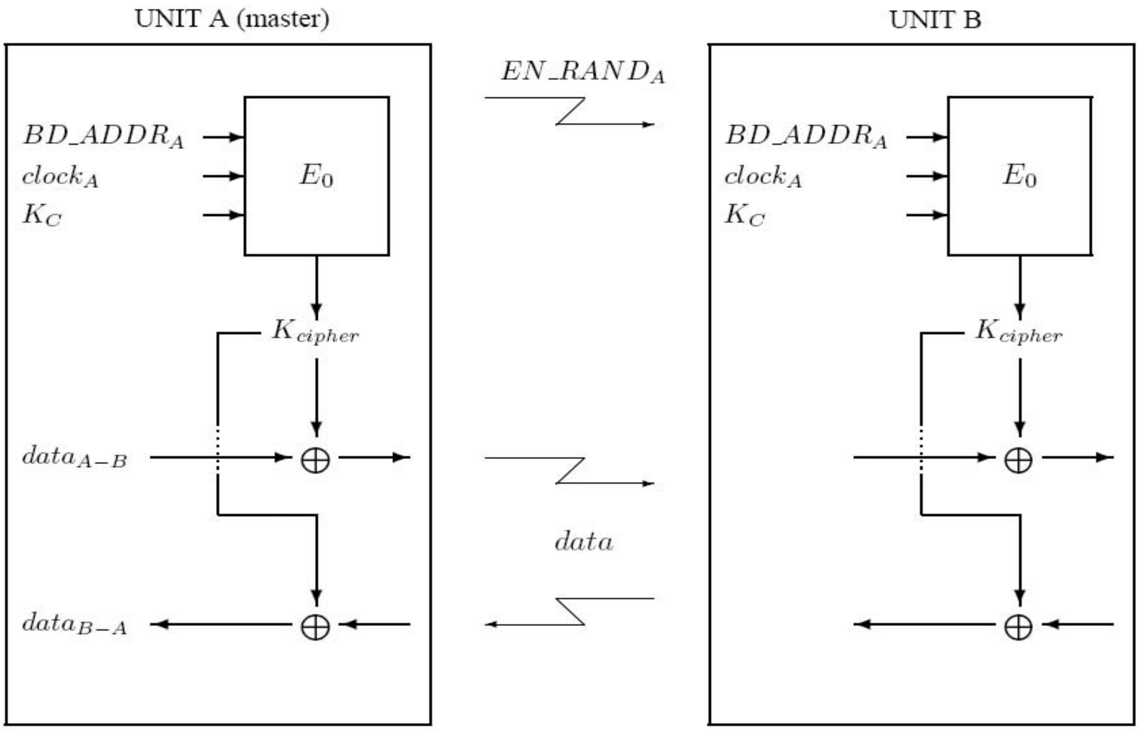


FIGURE 32: ENCRYPTION MECHANISM OF BLUETOOTH

If two Bluetooth devices wants to communicate with each other , First they need to pass through key exchange protocol and user/device authentication that that finishes with every unit concurring on an shared key (the connection key), which is utilized to create the encryption key (KC). KC is gotten from the present connection key, a ciphering(figuring) offset number (COF), which can be derive from the verification technique done preceding the encryption stage, and an public EN RAND (known random number). To Encrypt a data, this private key (KC) is changed into another key meant as K_C. At that point K_C is utilized as a part of a direct way, alongside the public known values, the expert devices Bluetooth location, and a clock, which is different for every packet, to make the introductory state for a two level stream generator (see Figure 1).The encryption calculation E0 creates a parallel Kcipher, keystream, which is XORed with the plain content. The cipher is symmetric; decryption process is performed in precisely the same way utilizing the same key as utilized for encryption.

5.2 Design

The E0 keystream generator comprises of 4 LFSRs with an aggregate length of 128 bits, and a 4 bit limited state machine (Blend and Summation Combiner Logic). At every clock tick all LFSRs are timed once, and their most significant bits are XORed together with one bit of the finite state machine to generate the keystream bit. At that point, the all LFSR's most significant bits are added together. After that, the two most significant bits of this 3 bit entirety are utilized to upgrade the condition of the finite state machine. Table 1 depicts the four input polynomials of the LFSRs and figure 2 shows design of E0 cipher.

LFSR Number	LFSR Length	LFSR Feedback Polynomial
1	25	$q^{25} + q^{20} + q^{12} + q^{8} +1$
2	31	$q^{31} + q^{24} + q^{16} + q^{12} +1$
3	33	$q^{33} + q^{28} + q^{24} + q^{4} +1$
4	39	$q^{39} + q^{36} + q^{28} + q^{4} +1$

TABLE 1: FEEDBACK POLYNOMIALS

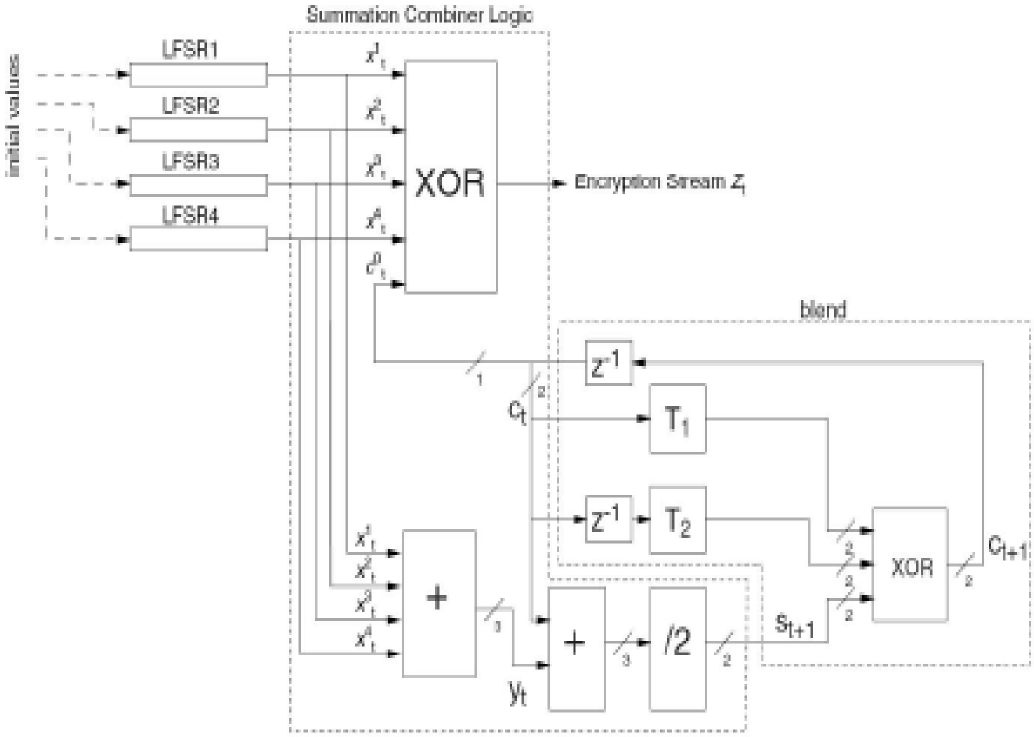


FIGURE 33: E0 CIPHER DESIGN

Let X^i_t be tth symbol of ith LFSR. From four LFSR's output bits X^1_t, X^2_t, X^3_t, X^4_t we derives value Y_t as follow:

$$y_t = \sum_{i=1}^{4} x_t^i \ ,$$

Where $Y_t \in \{0, 1, 2, 3, 4\}$

EQUATION 1

Where Y_t is sum of four integers and value of Y_t must be 0, 1, 2, 3, 4. Because summation of four 1's should not be more than 4. Summation Combiner logic contain 4 state bits as C^1_t, C^0_t, $C^1_{(t-1)}$, $C^0_{(t-1)}$.Now output of Summation Combiner logic is denoted as Z_t, which is XOR of four LFSR's output bit and one state bit(see eq. 2).

$$z_t = x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0$$

EQUATION 2

And intermediate value of SCLB is St+1, which is describe in eq. 3

$$s_{t+1} = \begin{bmatrix} s_{t+1}^0 \\ s_{t+1}^1 \end{bmatrix} = \left\lfloor \frac{y_t + c_t}{2} \right\rfloor \in \{0, 1, 2, 3\}$$

EQUATION 3

is the value of (C0t, C1t) viewed as an integer in [0, 3].

$$c_{t+1} = \begin{bmatrix} c_{t+1}^0 \\ c_{t+1}^1 \end{bmatrix} = s_{t+1} \oplus T_1[c_t] \oplus T_2[c_{t-1}]$$

EQUATION 4

Where T1[] and T2[] is linear bijections over Galois Field(4) .Functionality of T1[] and T2[] described in Table 2.

X	T1[X]	T2[X]
00	00	00
01	01	11
10	10	01
11	11	10

TABLE 2: FUNCTIONALITY OF T1[] AND T2[]

The mappings of T1[] and T2[] are linear, and therefore we used XOR Gates for mapping; i.e.

$$\begin{aligned} T_1 : \quad & (x_1, x_0) \mapsto (x_1, x_0), \\ T_2 : \quad & (x_1, x_0) \mapsto (x_0, x_1 \oplus x_0), \end{aligned}$$

$$s_{t+1} = \begin{bmatrix} s_{t+1}^0 \\ s_{t+1}^1 \end{bmatrix} = \left\lfloor \frac{y_t + c_t}{2} \right\rfloor \in \{0, 1, 2, 3\}$$

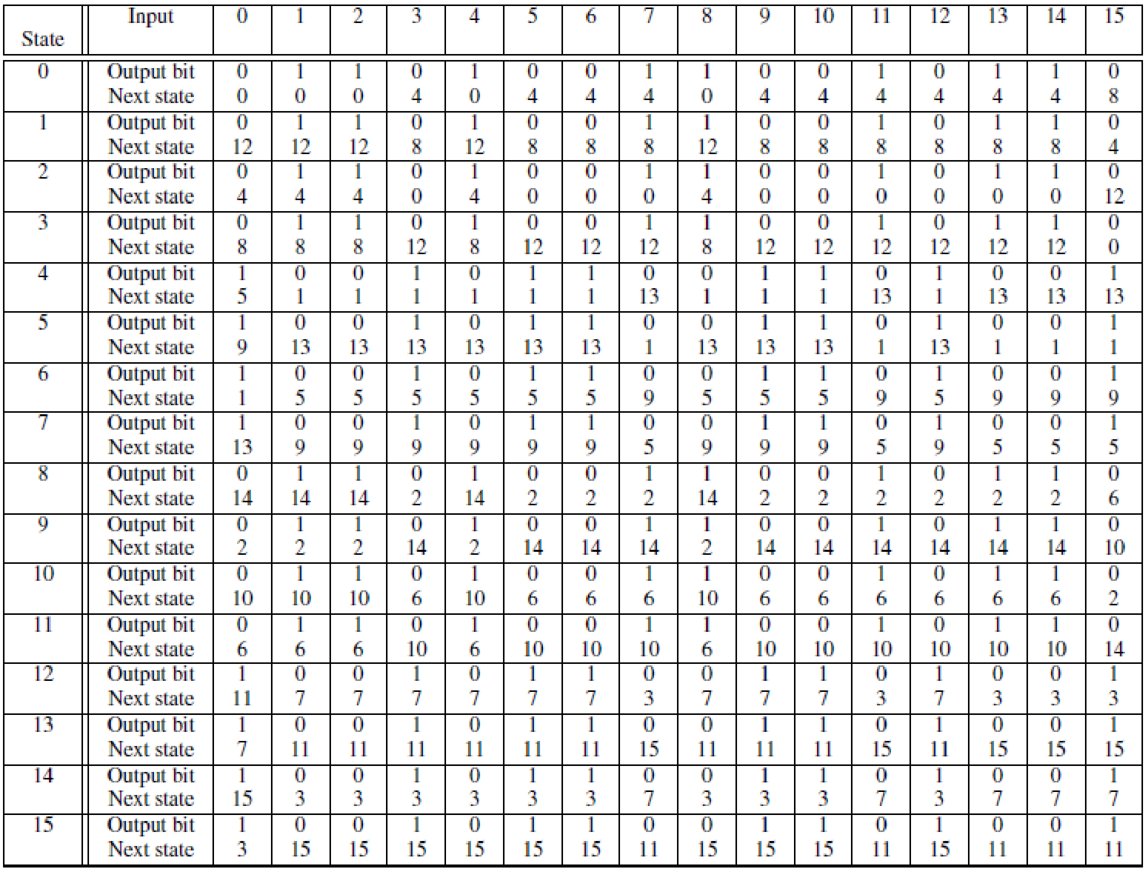
EQUATION 5

The keystream generator needs to be initialized with an initial value for the all LFSRs (128 bits altogether) and the 4 bits that determine the estimations of Ct and Ct−1. Amid the initialization stage these 4 bits are focused. The 128-bit beginning estimation of the LFSRs is gotten from four inputs by utilizing the keystream generator itself. The information parameters are the secret key, a 128-bit arbitrary number (RAND), a 48-bit Bluetooth address, and the 26 expert clock bits. At the point when the encryption key has been made, the LFSRs re stacked with their starting values.

At that point, 200 stream cipher bits are made by working the generator. Of these bits, the last 128 are encouraged back in parallel into the keystream generator as a starting estimation of the four LFSRs. The estimation of Ct and Ct−1 are kept. Starting here on, when clocked, the generator produces the encryption arrangement, which is XORed with the transmitted (or got) payload data.

5.3 Analysis of SCLB and Finite State Machine

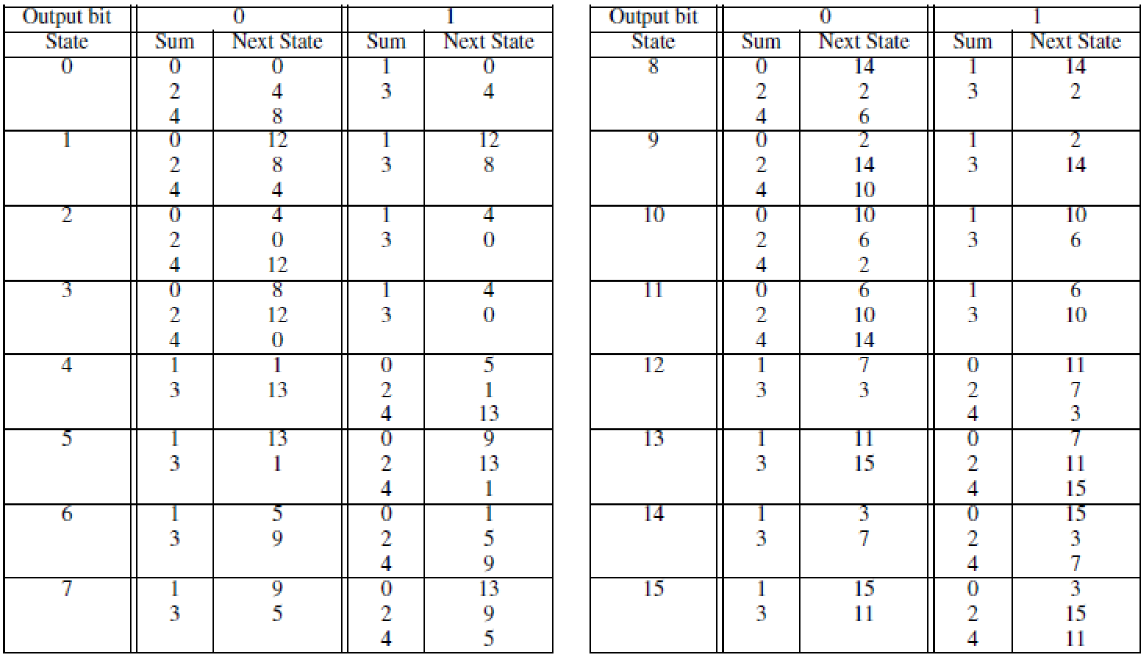
In this section mainly deal with four output bits of LFSR's, four state bits and one key bit (Z_t). We have 4 input bits that means we have total 16 combinations. Now, same thing happens with 4 state bits, we have total 16 combinations. We can compute 16*16 table for input and state bits and get key bit (Z_t), which is shown in table 3.



State	Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	0	0	0	4	0	4	4	4	0	4	4	4	4	4	4	8
1	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	12	12	12	8	12	8	8	8	12	8	8	8	8	8	8	4
2	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	4	4	4	0	4	0	0	0	4	0	0	0	0	0	0	12
3	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	8	8	8	12	8	12	12	12	8	12	12	12	12	12	12	0
4	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	5	1	1	1	1	1	1	13	1	1	1	13	1	13	13	13
5	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	9	13	13	13	13	13	13	1	13	13	13	1	13	1	1	1
6	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	1	5	5	5	5	5	5	9	5	5	5	9	5	9	9	9
7	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	13	9	9	9	9	9	9	5	9	9	9	5	9	5	5	5
8	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	14	14	14	2	14	2	2	2	14	2	2	2	2	2	2	6
9	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	2	2	2	14	2	14	14	14	2	14	14	14	14	14	14	10
10	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	10	10	10	6	10	6	6	6	10	6	6	6	6	6	6	2
11	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	6	6	6	10	6	10	10	10	6	10	10	10	10	10	10	14
12	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	11	7	7	7	7	7	7	3	7	7	7	3	7	3	3	3
13	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	7	11	11	11	11	11	11	15	11	11	11	15	11	15	15	15
14	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	15	3	3	3	3	3	3	7	3	3	3	7	3	7	7	7
15	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	3	15	15	15	15	15	15	11	15	15	15	11	15	11	11	11

TABLE 3: 16*16 STATAE MACHINE TABLE

We can future optimize 16*16 table into 16*5, see below optimize table.



Output bit	0		1	
State	Sum	Next State	Sum	Next State
0	0 2 4	0 4 8	1 3	0 4
1	0 2 4	12 8 4	1 3	12 8
2	0 2 4	4 0 12	1 3	4 0
3	0 2 4	8 12 0	1 3	4 0
4	1 3	1 13	0 2 4	5 1 13
5	1 3	13 1	0 2 4	9 13 1
6	1 3	5 9	0 2 4	1 5 9
7	1 3	9 5	0 2 4	13 9 5
8	0 2 4	14 2 6	1 3	14 2
9	0 2 4	2 14 10	1 3	2 14
10	0 2 4	10 6 2	1 3	10 6
11	0 2 4	6 10 14	1 3	6 10
12	1 3	7 3	0 2 4	11 7 3
13	1 3	11 15	0 2 4	7 11 15
14	1 3	3 7	0 2 4	15 3 7
15	1 3	15 11	0 2 4	3 15 11

TABLE 4: REARRANGED TABLE

CHAPTER 6 ATTACKS ON E0 CIPHER

As usual in cryptanalysis we mainly focus on known plain text attack. In known plain text attack, we assume a situation in which attacker knows some part of decrypted text one or other way. Using this know text part he or she will able to find remaining unknown text. Some other attack also possible on E0 cipher, which explain in below.

6.1 Correlation Attack

The principal technique comprises in separately recovering parts of the initial state of the generator by misusing particular connection (correlation) properties of the finite state machine. A critical drawback of this methodology, then again, is that it can't be connected to the genuine E0 algorithm, as it expect successions of successive key stream bits which are impressively more than the most extreme packet size.

6.2 Guess and Determine Attacks

In the second approach, a few sections of the starting state are first guessed and the watched key succession is utilized to infer the remaining parts in a deterministic way a short time later. In an easy route attack taking into account the perception that the substance of a solitary LFSR can specifically be gotten from the substance of the three different LFSRs, the 4 bits of the FSM, and some known key stream. By speculating the beginning condition of the FSM and the substance of the three shortest LFSRs (4 + 25 + 31 + 33 = 93 bits), the assault succeeds in recouping the whole interior state in 292 stages.

Scott R. Fluhrer recommends an attack in which just the contents of the two shortest LFSRs are guessed together with the beginning condition of the FSM. This time, the known key stream bits don't permit the aggressor to focus the bits of both remaining LFSRs straightforwardly. Rather, an arrangement of direct comparisons is gathered and checked for irregularities. When one is discovered, the comparing supposition is rejected. Fluhrer's calculation lessens the unpredictability of the assault to 285 and requires the same measure of information as the past assault (132 key stream bits). Also, he proposes a streamlining sparing processor time to the detriment of obliging significantly more key stream bits. Dissimilar to connection assaults, how- ever, his calculation permits the known key stream bits do be spread over various information packets.

CHAPTER 7 IMPLEMENTATION OF E0 CIPHER

7.1 Simulation and Design (Logisim)

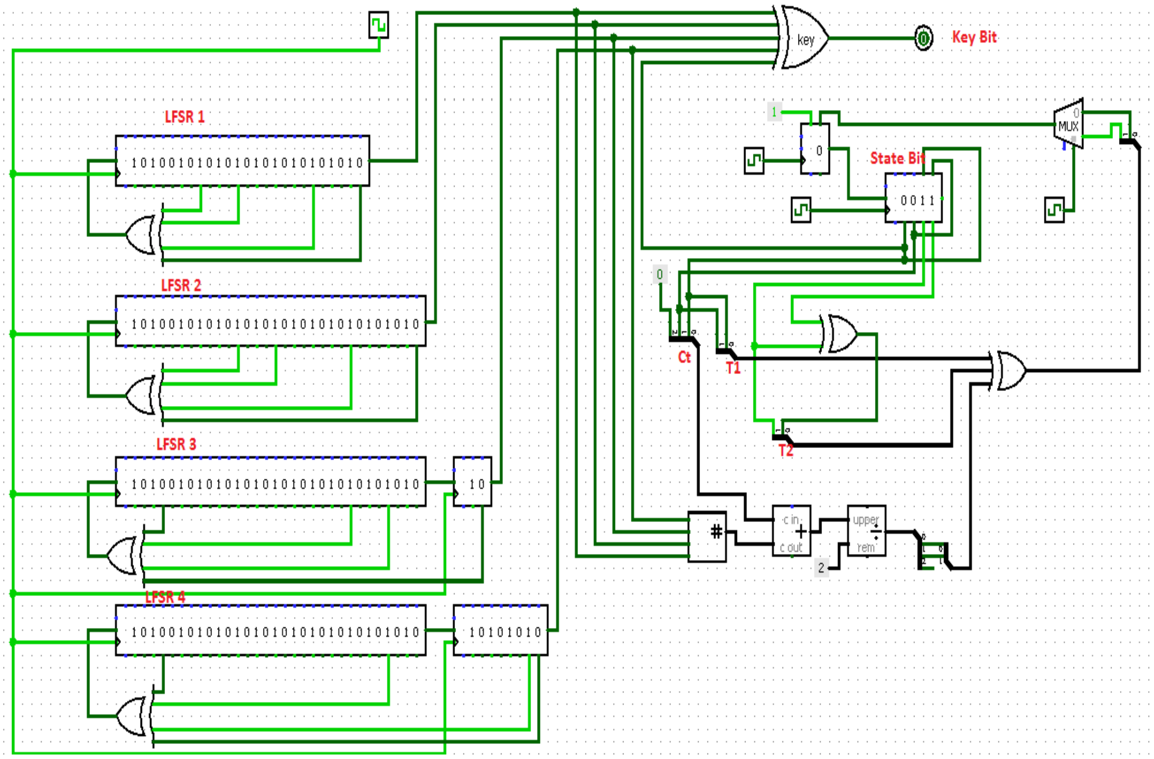
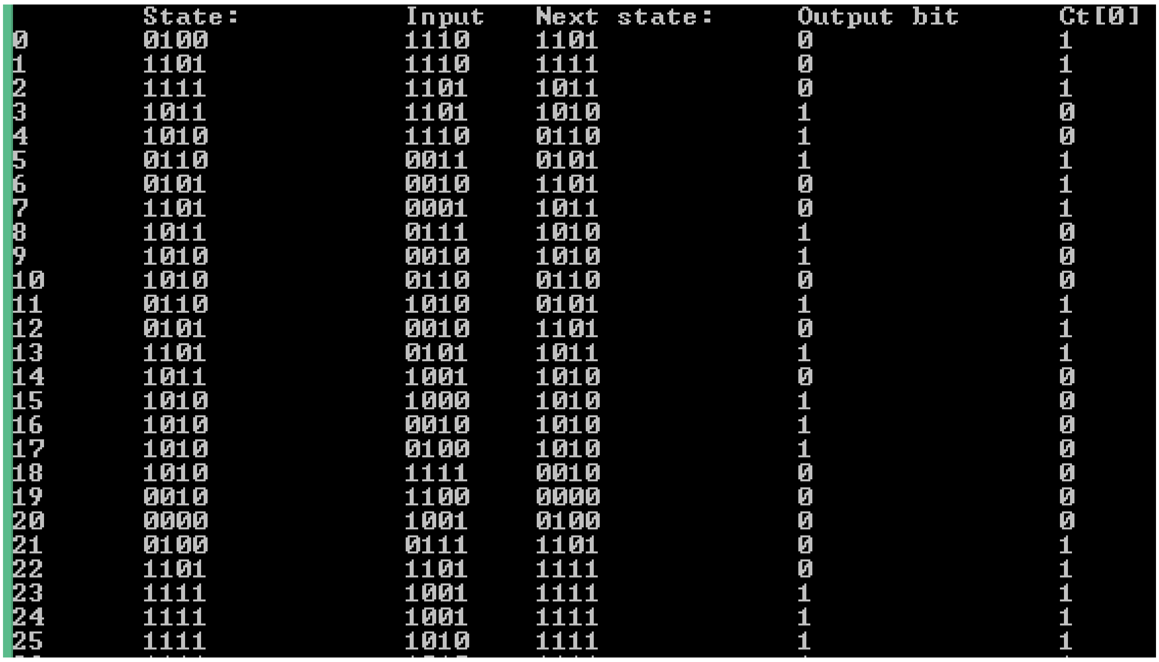


FIGURE 34: SIMULATION DESIGN FOR E0 CIPHER

7.2 Parallel and Sequential Implementation Results

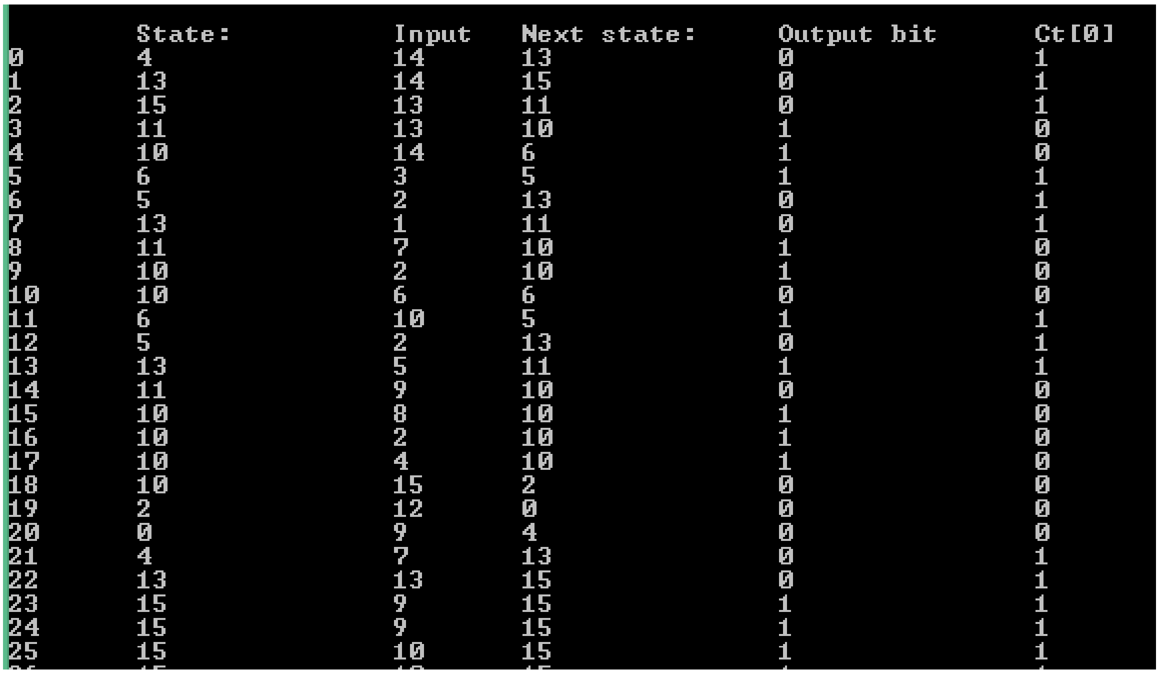
Throughput of hardware implementation is 0.7 to 2.1 Mbits/sec. That means it generate 0.7 to 2.1 Mbits in one second. So, now we just compare our software time with hardware time.

7.2.1 Output



```
         State:            Input    Next state:       Output bit       Ct[0]
0        0100              1110     1101              0                1
1        1101              1110     1111              0                1
2        1111              1101     1011              0                1
3        1011              1101     1010              1                0
4        1010              1110     0110              1                0
5        0110              0011     0101              1                1
6        0101              0010     1101              0                1
7        1101              0001     1011              0                1
8        1011              0111     1010              1                0
9        1010              0010     1010              1                0
10       1010              0110     0110              0                0
11       0110              1010     0101              1                1
12       0101              0010     1101              0                1
13       1101              0101     1011              1                1
14       1011              1001     1010              0                0
15       1010              1000     1010              1                0
16       1010              0010     1010              1                0
17       1010              0100     1010              1                0
18       1010              1111     0010              0                0
19       0010              1100     0000              0                0
20       0000              1001     0100              0                0
21       0100              0111     1101              0                1
22       1101              1101     1111              0                1
23       1111              1001     1111              1                1
24       1111              1001     1111              1                1
25       1111              1010     1111              1                1
```

FIGURE 35: OUTPUT IN BINARY FORM

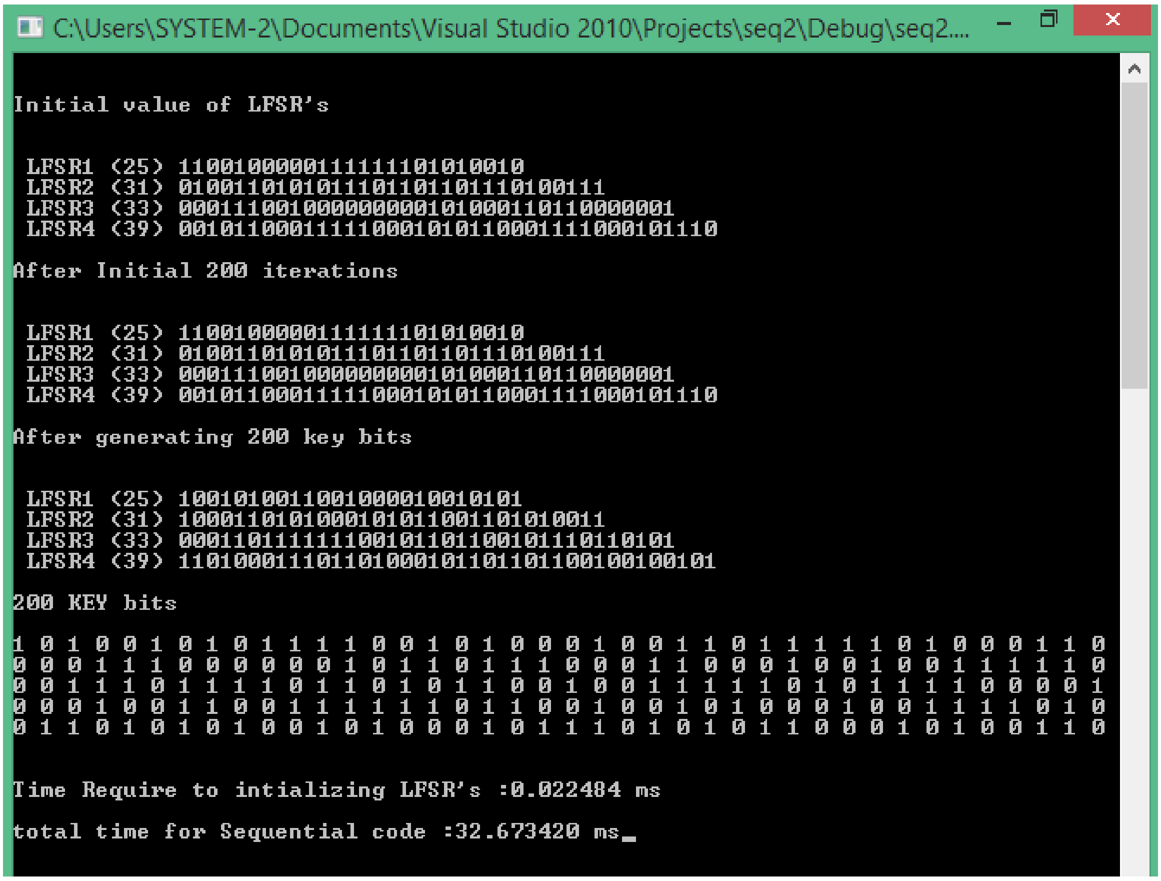


```
         State:            Input    Next state:      Output bit       Ct[0]
0        4                 14       13               0                1
1        13                14       15               0                1
2        15                13       11               0                1
3        11                13       10               1                0
4        10                14       6                1                0
5        6                 3        5                1                1
6        5                 2        13               0                1
7        13                1        11               0                1
8        11                7        10               1                0
9        10                2        10               1                0
10       10                6        6                0                0
11       6                 10       5                1                1
12       5                 2        13               0                1
13       13                5        11               1                1
14       11                9        10               0                0
15       10                8        10               1                0
16       10                2        10               1                0
17       10                4        10               1                0
18       10                15       2                0                0
19       2                 12       0                0                0
20       0                 9        4                0                0
21       4                 7        13               0                1
22       13                13       15               0                1
23       15                9        15               1                1
24       15                9        15               1                1
25       15                10       15               1                1
```

FIGURE 36: OUTPUT IN DECIMAL FORM

7.2.2 Sequential Code Timing and Results

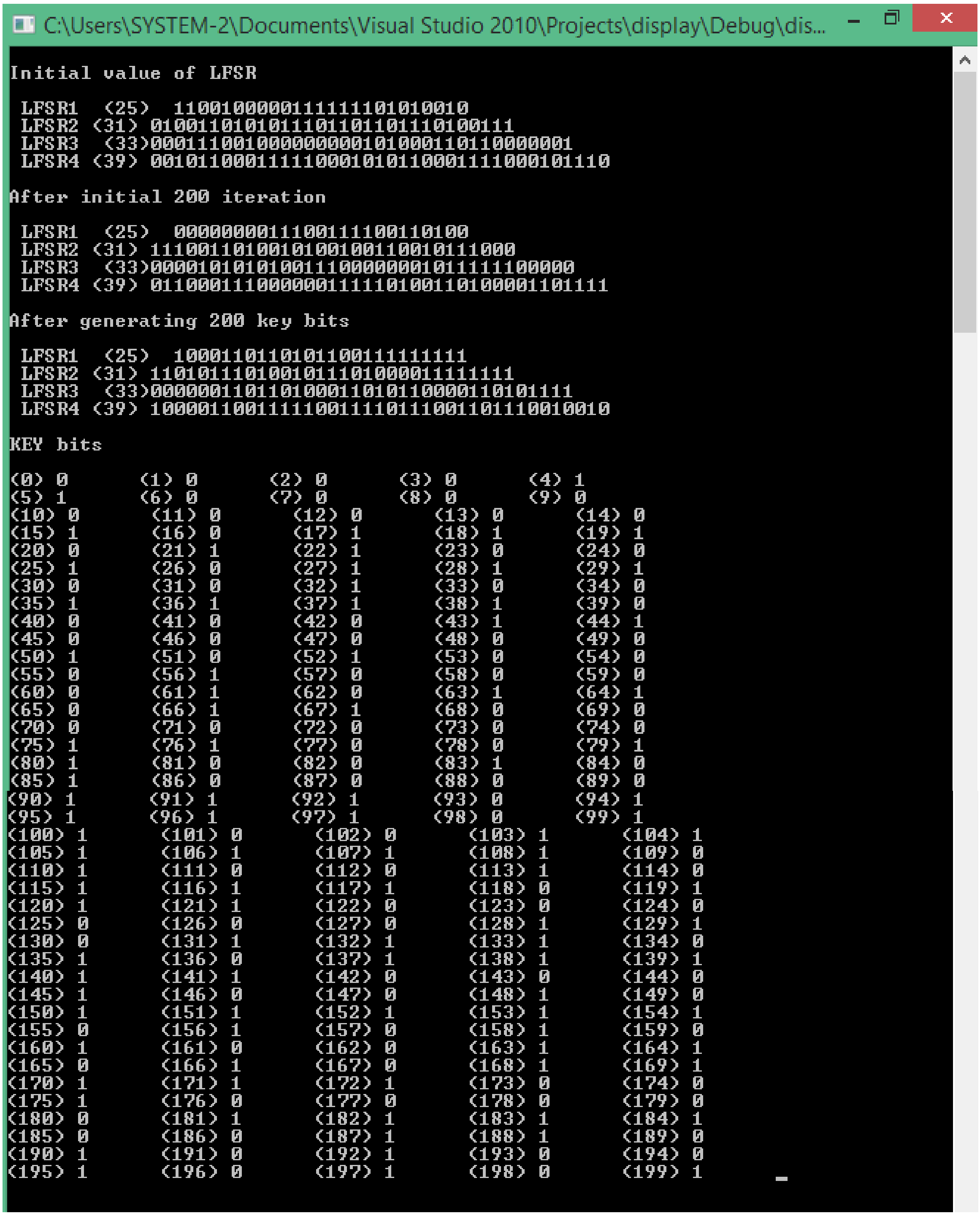
Time require for software sequential code is very high compare to hardware time.


C:\Users\SYSTEM-2\Documents\Visual Studio 2010\Projects\seq2\Debug\seq2....

```
Initial value of LFSR's


 LFSR1 (25) 1100100000111111101010010
 LFSR2 (31) 0100110101011101101101110100111
 LFSR3 (33) 000111001000000000101000110110000001
 LFSR4 (39) 001011000111110001010110001111000101110

After Initial 200 iterations


 LFSR1 (25) 1100100000111111101010010
 LFSR2 (31) 0100110101011101101101110100111
 LFSR3 (33) 000111001000000000101000110110000001
 LFSR4 (39) 001011000111110001010110001111000101110

After generating 200 key bits


 LFSR1 (25) 1001010011001000010010101
 LFSR2 (31) 1000110101000101011001101010011
 LFSR3 (33) 000110111111100101101100101110110101
 LFSR4 (39) 110100011101101000101101101100100100101

200 KEY bits

1 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 1 0 1 0 0 0 1 1 0
0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 1 1 1 0
0 0 1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 1
0 0 0 1 0 0 1 1 0 0 1 1 1 1 1 1 0 1 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 1 1 1 1 0 1 0
0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 0


Time Require to intializing LFSR's :0.022484 ms

total time for Sequential code :32.673420 ms_
```

FIGURE 37: STATUS OF LFSRS AFTER EACH MODULE, 200 KEY BITS AND TIME REQUIRED FOR EXECUTING SEQUENTIAL CODE

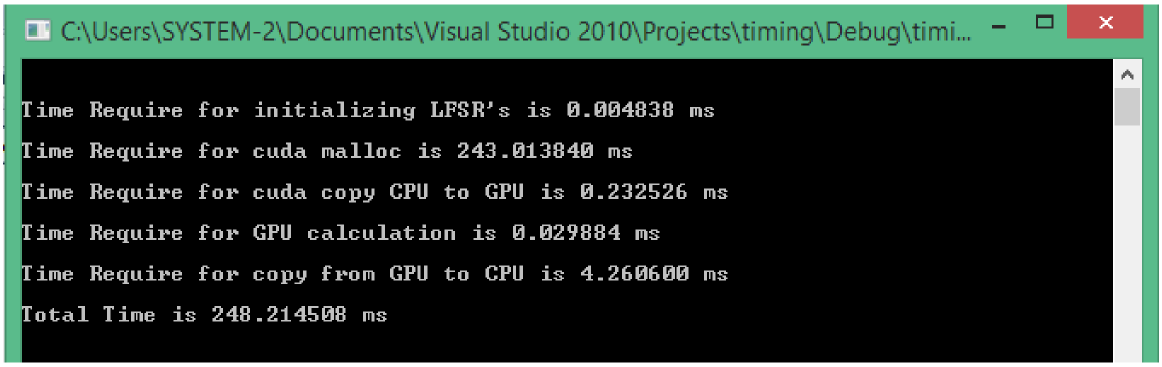
7.2.3 CUDA Code Timing and Results



C:\Users\SYSTEM-2\Documents\Visual Studio 2010\Projects\display\Debug\dis...

```
Initial value of LFSR

 LFSR1  (25)  11001000000111111101010010
 LFSR2 (31) 0100110101011101101101110100111
 LFSR3  (33)0001110010000000000101000110110000001
 LFSR4 (39) 001011000111110001010110001111000101110

After initial 200 iteration

 LFSR1  (25)  0000000011100111100110100
 LFSR2 (31) 1110011010010100100110010111000
 LFSR3  (33)0000101010100111000000001011111100000
 LFSR4 (39) 011000111000000111110100110100001101111

After generating 200 key bits

 LFSR1  (25)  1000110110101100111111111
 LFSR2 (31) 1101011101001011101000011111111
 LFSR3  (33)0000001101101000110101100000110101111
 LFSR4 (39) 100001100111110011110111001101110010010

KEY bits

(0) 0      (1) 0      (2) 0      (3) 0      (4) 1
(5) 1      (6) 0      (7) 0      (8) 0      (9) 0
(10) 0      (11) 0      (12) 0      (13) 0      (14) 0
(15) 1      (16) 0      (17) 1      (18) 1      (19) 1
(20) 0      (21) 1      (22) 1      (23) 0      (24) 0
(25) 1      (26) 0      (27) 1      (28) 1      (29) 1
(30) 0      (31) 0      (32) 1      (33) 0      (34) 0
(35) 1      (36) 1      (37) 1      (38) 1      (39) 0
(40) 0      (41) 0      (42) 0      (43) 1      (44) 1
(45) 0      (46) 0      (47) 0      (48) 0      (49) 0
(50) 1      (51) 0      (52) 1      (53) 0      (54) 0
(55) 0      (56) 1      (57) 0      (58) 0      (59) 0
(60) 0      (61) 1      (62) 0      (63) 1      (64) 1
(65) 0      (66) 1      (67) 1      (68) 0      (69) 0
(70) 0      (71) 0      (72) 0      (73) 0      (74) 0
(75) 1      (76) 1      (77) 0      (78) 0      (79) 1
(80) 1      (81) 0      (82) 0      (83) 1      (84) 0
(85) 1      (86) 0      (87) 0      (88) 0      (89) 0
(90) 1      (91) 1      (92) 1      (93) 0      (94) 1
(95) 1      (96) 1      (97) 1      (98) 0      (99) 1
(100) 1      (101) 0      (102) 0      (103) 1      (104) 1
(105) 1      (106) 1      (107) 1      (108) 1      (109) 0
(110) 1      (111) 0      (112) 0      (113) 1      (114) 0
(115) 1      (116) 1      (117) 1      (118) 0      (119) 1
(120) 1      (121) 1      (122) 0      (123) 0      (124) 0
(125) 0      (126) 0      (127) 0      (128) 1      (129) 1
(130) 0      (131) 1      (132) 1      (133) 1      (134) 0
(135) 1      (136) 0      (137) 1      (138) 1      (139) 1
(140) 1      (141) 1      (142) 0      (143) 0      (144) 0
(145) 1      (146) 0      (147) 0      (148) 1      (149) 0
(150) 1      (151) 1      (152) 1      (153) 1      (154) 1
(155) 0      (156) 1      (157) 0      (158) 1      (159) 0
(160) 1      (161) 0      (162) 0      (163) 1      (164) 1
(165) 0      (166) 1      (167) 0      (168) 1      (169) 1
(170) 1      (171) 1      (172) 1      (173) 0      (174) 0
(175) 1      (176) 0      (177) 0      (178) 0      (179) 0
(180) 0      (181) 1      (182) 1      (183) 1      (184) 1
(185) 0      (186) 0      (187) 1      (188) 1      (189) 0
(190) 1      (191) 0      (192) 1      (193) 0      (194) 0
(195) 1      (196) 0      (197) 1      (198) 0      (199) 1
```

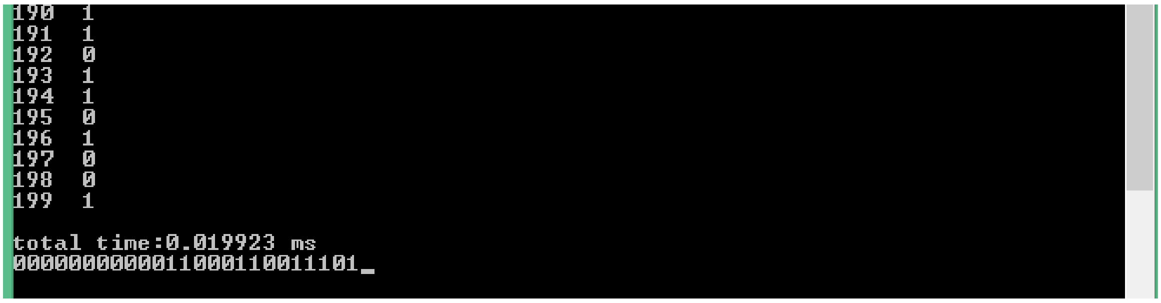
FIGURE 38: CONTENT OF LFSRS AFTER EACH MODULE AND 200 KEY BITS



C:\Users\SYSTEM-2\Documents\Visual Studio 2010\Projects\timing\Debug\timi...

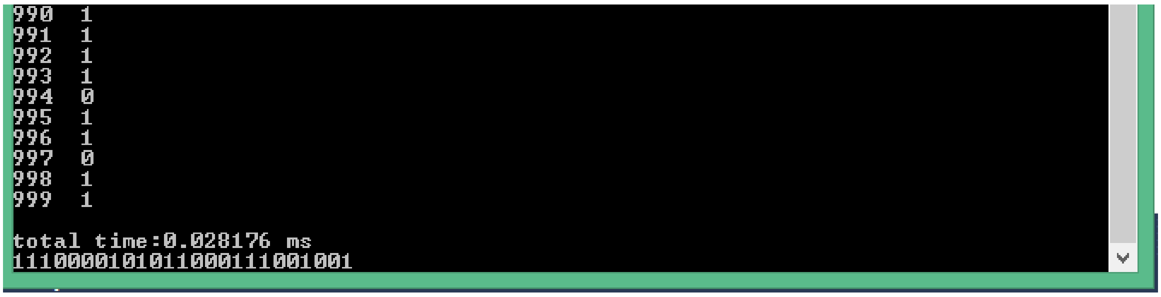
```
Time Require for initializing LFSR's is 0.004838 ms

Time Require for cuda malloc is 243.013840 ms

Time Require for cuda copy CPU to GPU is 0.232526 ms

Time Require for GPU calculation is 0.029884 ms

Time Require for copy from GPU to CPU is 4.260600 ms

Total Time is 248.214508 ms
```

FIGURE 39: EXECUTION TIME OF DIFFERENT MODULES AND TOTAL EXECUTION TIME



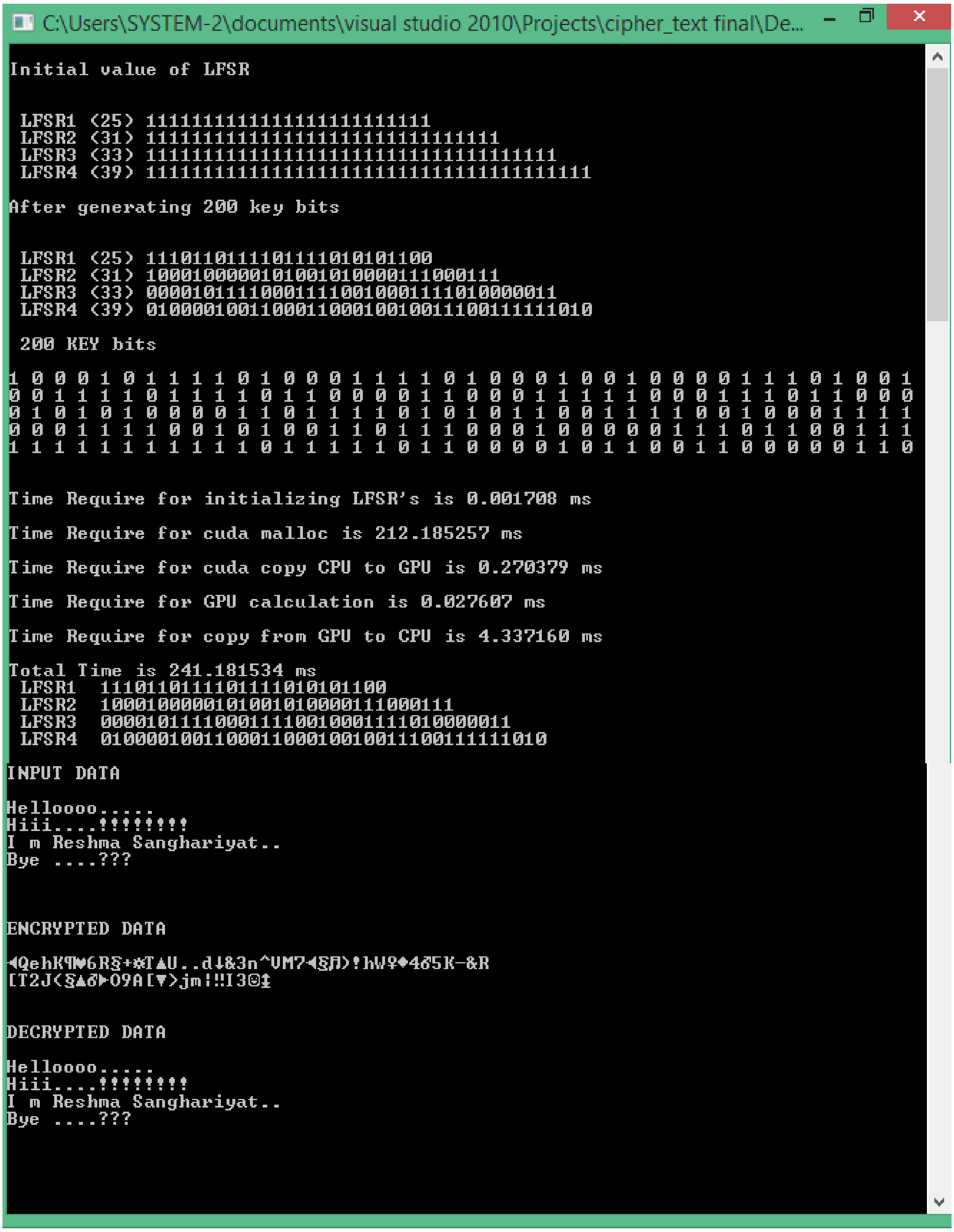
```
190  1
191  1
192  0
193  1
194  1
195  0
196  1
197  0
198  0
199  1

total time:0.019923 ms
0000000000011000110011101_
```

FIGURE 40: TIME ON GPU FOR GENERATING 200 BITS



```
990  1
991  1
992  1
993  1
994  0
995  1
996  1
997  0
998  1
999  1

total time:0.028176 ms
1110000101011000111001001
```

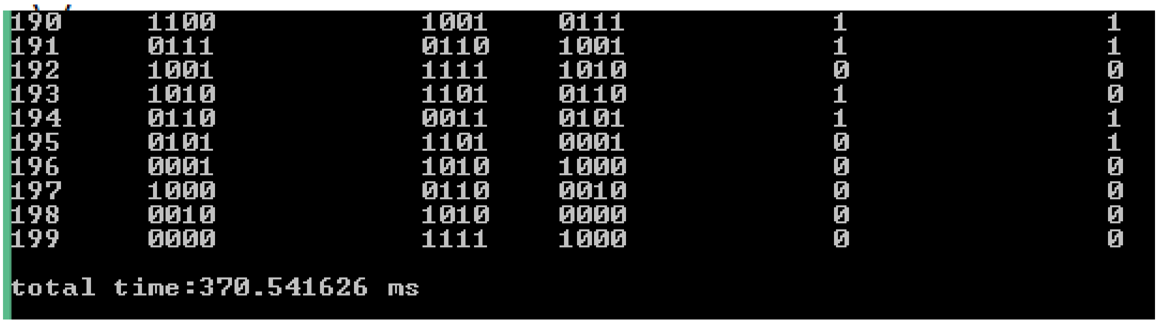
FIGURE 41: TIME ON GPU FOR GENERATING 1000 BITS


C:\Users\SYSTEM-2\documents\visual studio 2010\Projects\cipher_text final\De...

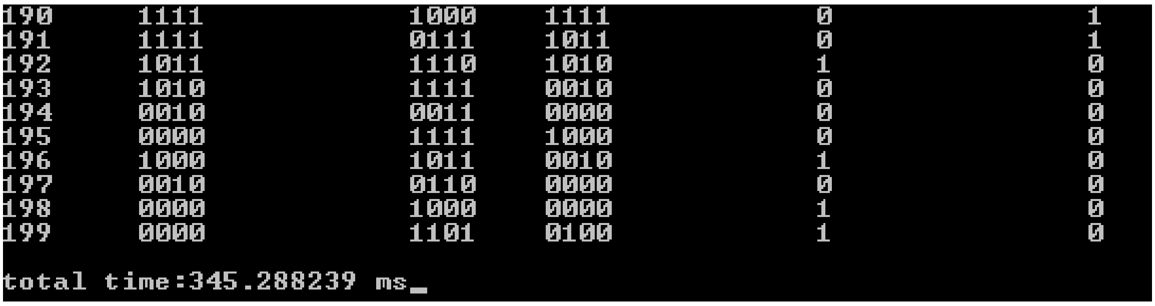
```
Initial value of LFSR


 LFSR1 (25) 1111111111111111111111111
 LFSR2 (31) 1111111111111111111111111111111
 LFSR3 (33) 111111111111111111111111111111111
 LFSR4 (39) 111111111111111111111111111111111111111

After generating 200 key bits


 LFSR1 (25) 1110110111101111010101100
 LFSR2 (31) 1000100000010100101000011100 0111
 LFSR3 (33) 000010111100011110010001111010000011
 LFSR4 (39) 010000100110001100010010011100111111010

 200 KEY bits

1 0 0 0 1 0 1 1 1 1 0 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 1 1 1 0 1 0 0 1
0 0 1 1 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 1 1 0 0 0
0 1 0 1 0 1 0 0 0 0 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0 0 1 0 0 0 1 1 1 1
0 0 0 1 1 1 1 0 0 1 0 1 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 0 1 1 1 0 1 1 0 0 1 1 1
1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 1 0


Time Require for initializing LFSR's is 0.001708 ms

Time Require for cuda malloc is 212.185257 ms

Time Require for cuda copy CPU to GPU is 0.270379 ms

Time Require for GPU calculation is 0.027607 ms

Time Require for copy from GPU to CPU is 4.337160 ms

Total Time is 241.181534 ms
 LFSR1  1110110111101111010101100
 LFSR2  1000100000010100101000011100 0111
 LFSR3  000010111100011110010001111010000011
 LFSR4  010000100110001100010010011100111111010
INPUT DATA

Helloooo.....
Hiii....!!!!!!!!
I m Reshma Sanghariyat..
Bye ....???



ENCRYPTED DATA

◄QehK¶♥6R§+☼T▲U..d↓&3n^VM7◄§♫)!hW♀♦4δ5K-&R
[T2J<§▲δ►O9A[▼>jm¦‼I3☺↨


DECRYPTED DATA

Helloooo.....
Hiii....!!!!!!!!
I m Reshma Sanghariyat..
Bye ....???
```

FIGURE 42: ENCRYPTION AND DECRYPTION PROCESS OUTPUT

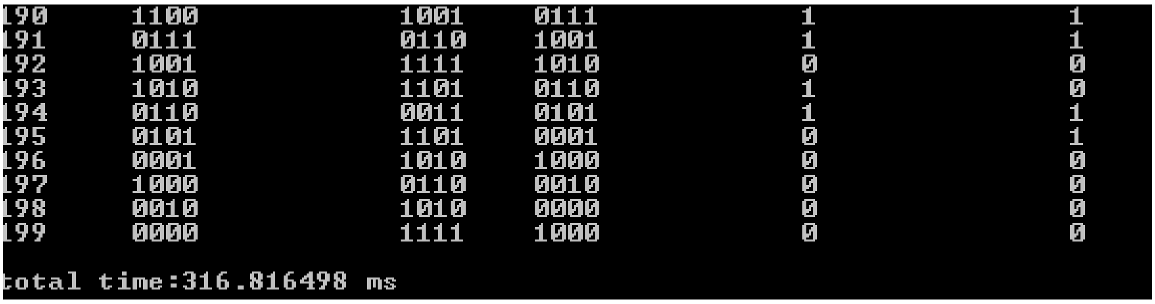
7.2.4 Different Timing while Optimizing



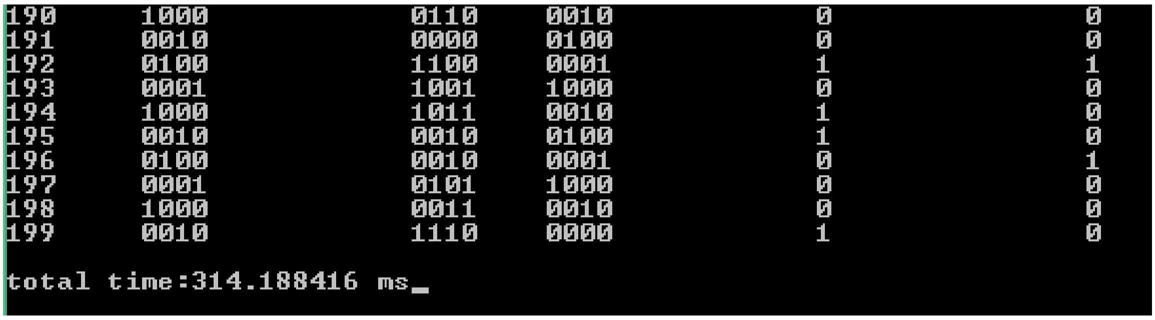
```
190     1100            1001    0111            1               1
191     0111            0110    1001            1               1
192     1001            1111    1010            0               0
193     1010            1101    0110            1               0
194     0110            0011    0101            1               1
195     0101            1101    0001            0               1
196     0001            1010    1000            0               0
197     1000            0110    0010            0               0
198     0010            1010    0000            0               0
199     0000            1111    1000            0               0

total time:370.541626 ms
```



```
190     1111            1000    1111            0               1
191     1111            0111    1011            0               1
192     1011            1110    1010            1               0
193     1010            1111    0010            0               0
194     0010            0011    0000            0               0
195     0000            1111    1000            0               0
196     1000            1011    0010            1               0
197     0010            0110    0000            0               0
198     0000            1000    0000            1               0
199     0000            1101    0100            1               0

total time:345.288239 ms_
```



```
190     1100            1001    0111            1               1
191     0111            0110    1001            1               1
192     1001            1111    1010            0               0
193     1010            1101    0110            1               0
194     0110            0011    0101            1               1
195     0101            1101    0001            0               1
196     0001            1010    1000            0               0
197     1000            0110    0010            0               0
198     0010            1010    0000            0               0
199     0000            1111    1000            0               0

total time:316.816498 ms
```



```
190     1000            0110    0010            0               0
191     0010            0000    0100            0               0
192     0100            1100    0001            1               1
193     0001            1001    1000            0               0
194     1000            1011    0010            1               0
195     0010            0010    0100            1               0
196     0100            0010    0001            0               1
197     0001            0101    1000            0               0
198     1000            0011    0010            0               0
199     0010            1110    0000            1               0

total time:314.188416 ms_
```

FIGURE 43: DIFFERENT TIMING WHILE OPTIMIZING PROCESS

7.3 Analysis and Results

7.3.1 LFSR Seed Value

LFSR 1	1111111111111111111111111
LFSR 2	1111111111111111111111111111111
LFSR 3	111111111111111111111111111111111
LFSR 4	111111111111111111111111111111111111111

State Bit	Out 1 Bit	Out 2 Bit	Out 3 Bit	Out 4 Bit	Ct[0] Bit	Key Bit
0000	1	1	1	1	0	0
1000	1	1	1	1	1	1
0110	1	1	1	1	0	0
1001	1	1	1	1	1	1
1010	1	1	1	1	1	1
0010	1	1	1	1	0	0
1100	1	1	1	1	1	1
0011	1	1	1	1	0	0
0000	1	1	1	1	0	0
1000	1	1	1	1	1	1
0110	1	1	1	1	0	0

7.3.2 LFSR Seed Value (mod of 2)

LFSR 1	0101010101010101010101010
LFSR 2	0101010101010101010101010101010
LFSR 3	010101010101010101010101010101010
LFSR 4	010101010101010101010101010101010101010

State Bit	Out 1 Bit	Out 2 Bit	Out 3 Bit	Out 4 Bit	Ct[0] Bit	Key Bit
0000	0	0	0	0	0	0
0000	1	1	1	1	0	0
1000	0	0	0	0	1	1
1110	1	1	1	1	1	1
0111	0	0	0	0	0	0
1101	1	1	1	1	1	1
1111	0	0	0	0	1	1
0011	1	1	1	1	0	0
0000	0	0	0	0	0	0
0000	1	1	1	1	0	0
1000	0	0	0	0	1	1

7.3.3 LFSR Seed Value (mod of 3)

LFSR 1	1001001001001001001001001					
LFSR 2	1001001001001001001001001001001					
LFSR 3	001001001001001001001001001001001					
LFSR 4	001001001001001001001001001001001001001					
State Bit	Out 1 Bit	Out 2 Bit	Out 3 Bit	Out 4 Bit	Ct[0] Bit	Key Bit
0000		1	1	1	**0**	**0**
1000	0	0	0	0	**1**	**1**
1110	0	0	0	0	**1**	**1**
1111	1	1	1	1	**1**	**1**
1011	0	0	0	0	**1**	**1**
0110	0	0	0	0	**0**	**0**
0001	1	1	1	1	**0**	**0**
0100	0	0	0	0	**0**	**0**
0101	0	0	0	0	**0**	**0**
1001	1	1	1	1	**1**	**1**
1010	0	0	0	0	**0**	**0**

7.3.4 Random Seed Value of LFSR

LFSR 1	0101 0000 0111 1010 0000 1101 1					
LFSR 2	0000 0011 0111 1110 1110 1111 0101 010					
LFSR 3	0100 0010 1101 1100 0001 1100 0010 0111 0					
LFSR 4	1101 1111 0011 0010 0001 0111 0101 1110 1011 100					
State Bit	Out 1 Bit	Out 2 Bit	Out 3 Bit	Out 4 Bit	Ct[0] Bit	Key Bit
0000	1	0	0	0	0	1
0000	1	1	1	0	0	1
0100	0	0	1	1	0	0
0001	1	1	1	1	0	0
0100	1	0	0	1	0	0
0001	0	0	0	0	0	1
1100	0	0	1	1	1	1
0111	0	1	0	0	0	1
1001	0	1	0	1	1	1
1110	0	1	0	1	1	1
0011	1	1	0	1	0	1

Result:

- If LFSRs seed value LFSRs multiple of 2's, multiple of 3, multiple of 5, multiple of 7 positions have 1's and other positions have 0's or vice versa. In this cases key bit is same as first state bit (ct[0], 0th bit of state bit at t time).
- If all 1's in seed value of LFSRs the key bits in form of repetition of patterns.

7.4 Literature Survey

Title	Author	Publication Year	Method
A Uniform Framework for Cryptanalysis of the Bluetooth *E*0 Cipher	Ophir Levy and Avishai Wool	2005	**E0 Algorithm** for key generation using 4 LFSR, Summation Combiner Logic and Blend. Short-key cryptanalysis of the E0 cipher
Cryptanalysis of the Bluetooth *E*0 Cipher using OBDD's	Yaniv Shaked and Avishai Wool	international conference on Information Security 2006	This is based on **Binary Decision Diagrams** which is implementation for an attack against E0.
A Hybrid Encryption Technique to Secure Bluetooth Communication	P S Patheja, Akhilesh A. Waoo and Sudhir Nagwanshi	International Journal of Computer Application 2011	**Triple DES** for encryption of the key for which we use **Tiger algorithm**
Improving the Performance of the SYND Stream Cipher	Mohammed Meziani , Gerhard Ho_mann and Pierre-Louis Cayrel	Proceeding of the 5th International conference on Cryptography 2007	Use **Extended SYND** for generating pseudo random number.
Performance Evaluation of Crypt Analytical Approaches in Bluetooth Networks	Mrs. Sandhya S1 and Dr. Sumithra Devi K A2	**IJAIEM** 2013	Use AES-blake Algorithm and comparison of AES-blake and DES-Tiger Algorithm.
Linear Cryptanalysis of Bluetooth Stream Cipher	Jovan Dj. Goli´c, Vittorio Bagini, and Guglielmo Morgari	Theory and Applications of Cryptographic Techniques Conference 2002	Linear cryptanalysis method shows that correlation attacks may also be applicable to stream ciphers producing very short keystream sequences which are reinitialized frequently by using a cryptographically strong initialization scheme.

8. Comparison and Conclusion

8.1 Execution time of each module in C as in CUDA

Module	Time for sequential code	Time for CUDA code
Initializing SEED value	0.022484 ms	0.004838 ms
Blend and Summation Combiner Logic	0.054360 ms	0.017646 ms
Key Generation	32.673420 ms	0.664239 ms

TABLE 5: COMPARISON OF EXECUTION TIME OF CUDA AND C CODE

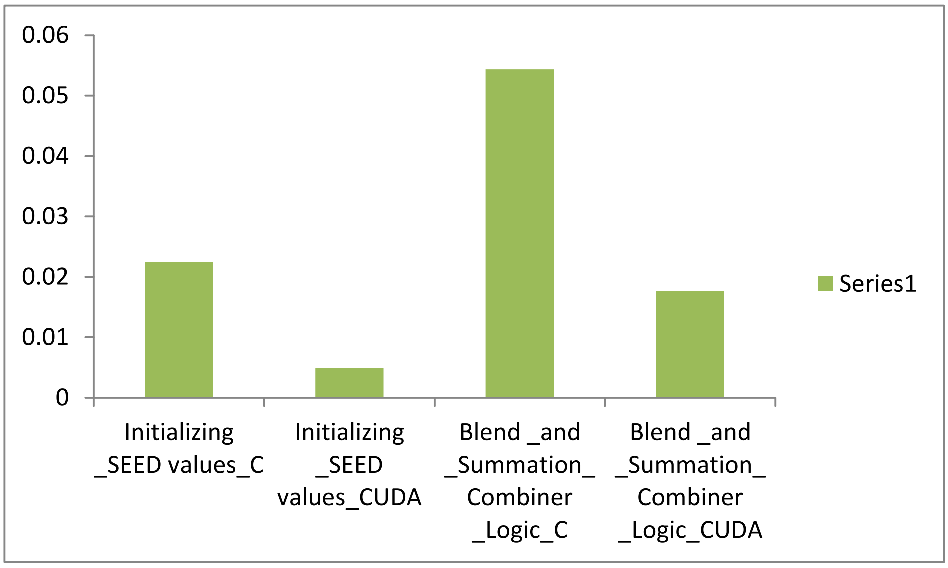


FIGURE 44: GRAPH OF COMPARING DIFFERENT MODULE EXECUTION TIME

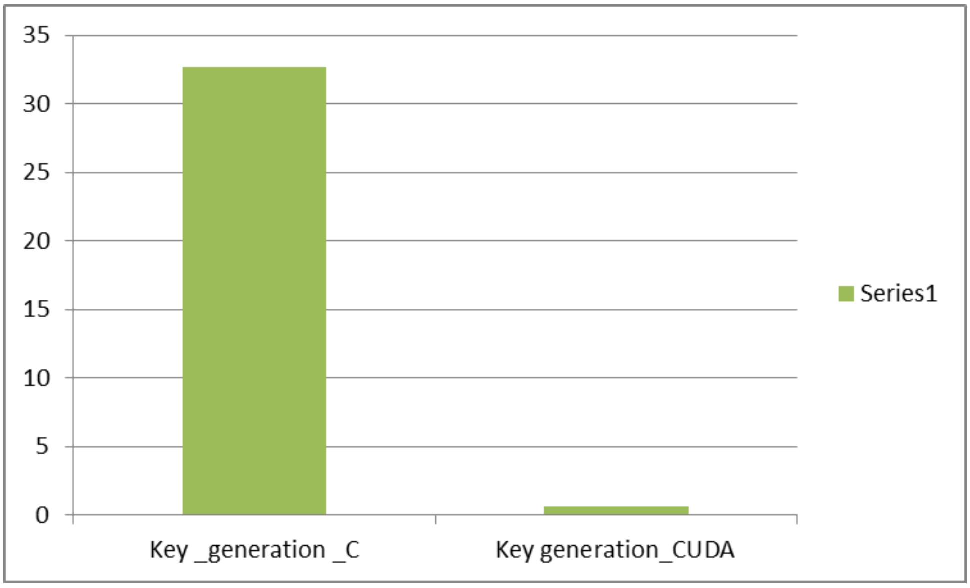


FIGURE 45: GRAPH FOR COMPARING EXECUTION TIME OF CUDA CODE AND C CODE

8.2 Analysis of execution time in CUDA

Initializing LFSR's	0.004838 ms
CUDA malloc	243.013840 ms
CPU to GPU copy	0.006840 ms
GPU calculation	0.029884 ms
GPU to CPU copy	4.260600 ms
Total Time for key generation	248.214508 ms

TABLE 6: EXECUTION TIME OF DIFFERENT MODULE OF CUDA CODE

VHDL(FPGA) :Execution time for 200 bits	**CUDA: Execution time for 200 bits**	**C : Execution time for 200 bits**
	0.664 ms	32 ms

TABLE 7: TIME FOR GENERATING 200 BITS

8.3 Conclusion

In real time application of Bluetooth Communication, communication speed is 0.7 to 2.1 Mbps. Whereas in CUDA we will get communication speed is **0.33 Mbps** which is very less compare to actual Bluetooth communication. Here, we conclude that we can replace lower version of Bluetooth communication hardware with our software code(CUDA code).

CHAPTER 9 APPENDIX

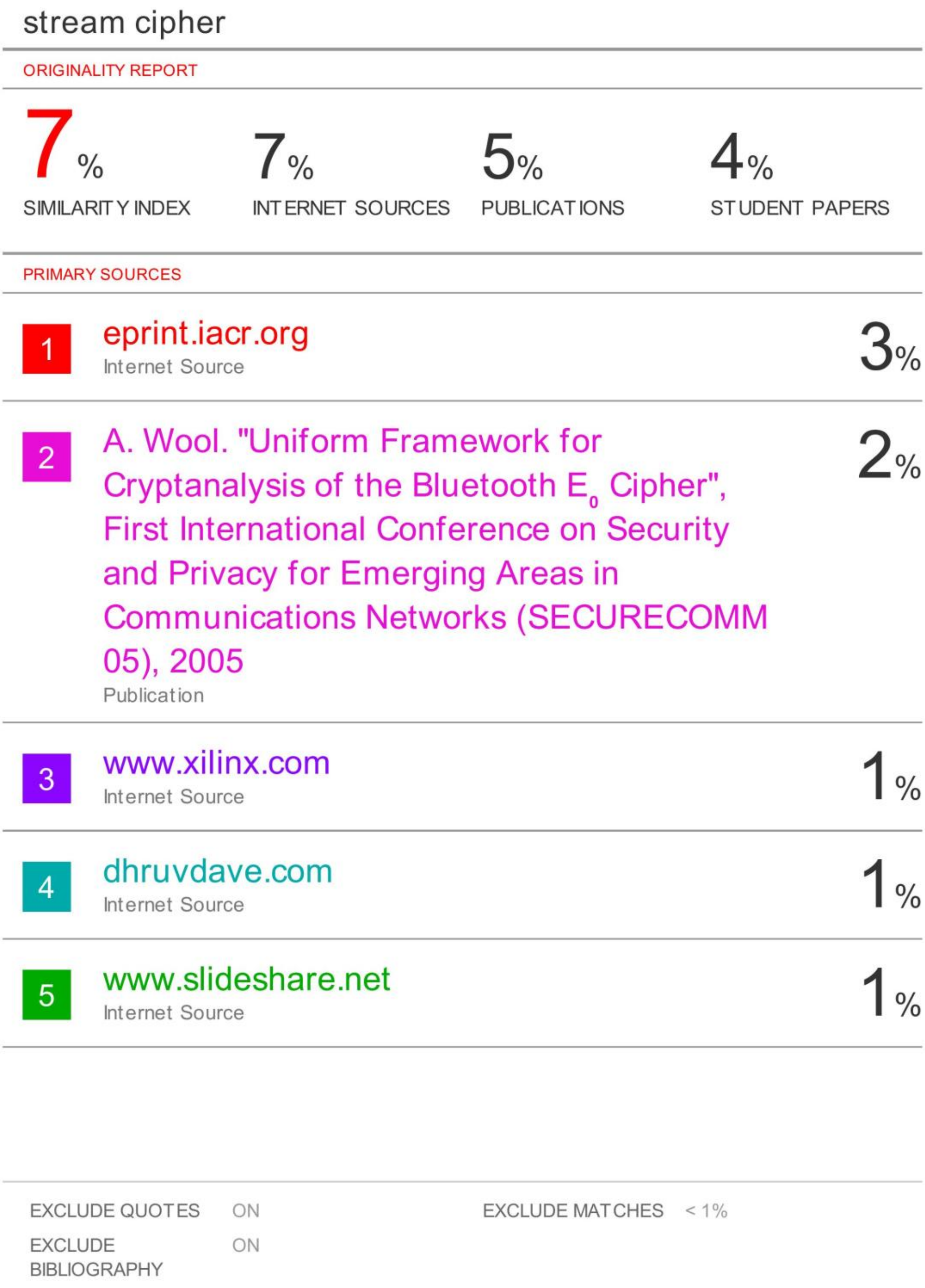
9.1. Acronyms Used

1. LFSR – Linear Feedback Shift Register
2. VHDL – VHSIC Hardware Description Language
3. VHSIC – Very High Speed Integrated Circuit
4. PRNG – Pseudo Random Number Generator
5. CUDA – Compute Unified Device Architecture
6. IC – Integrated Circuit
7. ISE – Integrated Synthesis Environment
8. GSM – Global System for Mobile communications
9. RTL – Register Transfer Level
10. HDL – Hardware Description Language
11. I/O – Input/output
12. LUT – Look Up Table
13. K–Map – Karnaugh Map
14. GUI – Graphical User Interface
15. IOB – Input/output Block
16. PC – Personal Computer
17. PDA – Personal Digital Assistant

CHAPTER 10 REFERENCES

1 http://crypto.stackexchange.com/
2 https://www.random.org/randomness/
3 http://fpgablog.com/
4 http://insidehpc.com/
5 https://groups.google.com
6 http://www.csee.umbc.edu/
7 https://books.google.ru
8 http://www.xilinx.com/support/documentation
9 Virtex-6 Libraries Guide for Schematic Designs by Xilinx
10 http://en.wikipedia.org
11 A Uniform Framework for Cryptanalysis of the Bluetooth *E*0 Cipher by Ophir Levy and Avishai Wool
12 Cryptanalysis of the Bluetooth E0 Cipher using OBDD's by Yaniv Shaked and Avishai Wool
13 A Hybrid Encryption Technique to Secure Bluetooth Communication by P S Patheja, Akhilesh A. Waoo and Sudhir Nagwanshi
14 Improving the Performance of the SYND Stream Cipher by Mohammed Meziani , Gerhard Ho_mann and Pierre-Louis Cayrel
15 Performance Evaluation of Crypt Analytical Approaches in Bluetooth Networks by Mrs. Sandhya S1 and Dr. Sumithra Devi K A2
16 Linear Cryptanalysis of Bluetooth Stream Cipher by Jovan Dj. Goli´c, Vittorio Bagini, and Guglielmo Morgari

PLAGIARISM REPORT



stream cipher

ORIGINALITY REPORT

7%	7%	5%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eprint.iacr.org Internet Source	3%
2	A. Wool. "Uniform Framework for Cryptanalysis of the Bluetooth E_0 Cipher", First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM 05), 2005 Publication	2%
3	www.xilinx.com Internet Source	1%
4	dhruvdave.com Internet Source	1%
5	www.slideshare.net Internet Source	1%

EXCLUDE QUOTES ON

EXCLUDE BIBLIOGRAPHY ON

EXCLUDE MATCHES < 1%

FIGURE 46: PLAGIRAISM REPORT