

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

A pseudo-random number generator (PRNG) is a program written for, and used in, probability and statistics applications when large quantities of random digits are needed. Most of these programs produce endless strings of single-digit numbers, usually in base 10, known as the decimal system. When large samples of pseudo-random numbers are taken, each of the 10 digits in the set $\{0,1,2,3,4,5,6,7,8,9\}$ occurs with equal frequency, even though they are not evenly distributed in the sequence.

Many algorithms have been developed in an attempt to produce truly random sequences of numbers, endless strings of digits in which it is theoretically impossible to predict the next digit in the sequence based on the digits up to a given point. But the very existence of the algorithm, no matter how sophisticated, means that the next digit can be predicted. This has given rise to the term pseudo-random for such machine-generated strings of digits. They are equivalent to random-number sequences for most applications, but they are not truly random according to the rigorous definition.

The digits in the decimal expansions of irrational numbers such as π (the ratio of a circle's circumference to its diameter in a Euclidean plane), e (the natural-logarithm base), or the square roots of numbers that are not perfect squares (such as $2^{1/2}$ or $10^{1/2}$) are believed by some mathematicians to be truly random. But computers can be programmed to expand such numbers to thousands, millions, billions, or trillions of decimal places; sequences can be selected that begin with digits far to the right of the decimal (radix) point, or that use every second, third, fourth, or n th digit.

A pseudo random number generator (PRNG) refers to an algorithm that uses mathematical formulas to produce sequences of random numbers. PRNGs generate a sequence of numbers approximating the properties of random numbers. This is determined by a small group of initial values. PRNGs are fundamental to the use of cryptographic mechanisms and key generation as they ensure message uniqueness. This term is also known as deterministic random number generator.

Widely used PRNG algorithms include: linear congruential generators, lagged Fibonacci generators, linear feedback shift registers, Blum Blum Shub, Fortuna and Mersenne Twister. A pseudo random number generator starts from an arbitrary starting state using a seed state. Many numbers are generated in a short time and can also be reproduced later, if the starting point in the sequence is known. Hence, the numbers are deterministic and efficient.

PRNGs used in cryptographic purposes are called cryptographically secure PRNGs (CSPRNGs). This includes stream ciphers and block ciphers. The essential requirement for a CSPRNG is that it should pass all statistical tests restricted to polynomial time in the size of the seed.

1.2. PROBLEM STATEMENT

To design a Pseudo Random Binary Sequence Generator which is used to generate random bit sequences using a linear shift register.

1.3 OBJECTIVE

To implement a 8-bit Pseudo Random Binary Sequence (PRBS) generator using VHDL programming and executing the designed PRBS in a Xilinx FPGA trainer kit.

1.4 ORGANIZATION OF THE REPORT

The report is organized as follows. Chapter 1 provides brief introduction to the PRNG followed by the problem statement and the objective of the project.

The detailed implementation of an eight bit PRBS is described in Chapter 2.

In Chapter 3, the simulation of the PRBS in VDHL and its hardware implementation is shown.

In Chapter 4, the conclusion to the project is provided.

CHAPTER 2

IMPLEMENTATION OF AN EIGHT BIT PRBS

2.1. INTRODUCTION

In analog work, the standard test message is the sine wave, followed by the two toned signal for more rigorous tests. The property being optimized in analog messages is generally the signal-to-noise ratio (SNR). However it is very difficult to interpret the signal in a definite mathematical equation or formula.

In order to counter this problem, a digital sequence, with a known pattern of '1's and '0's is tried. It is observed that it is much easier to measure the bit error rates (BER) than SNR.

A common digital sequence algorithm that can be used to measure the bit error rate is the pseudo random binary sequence.

PRBS or Pseudo Random Binary Sequence is essentially a random sequence of binary numbers. It is random in a sense that the value of an element of the sequence is independent of the values of any of the other elements. It is 'pseudo' because it is deterministic and after N elements it starts to repeat itself, unlike real random sequences. Examples of random sequences are radioactive decay and white noise.

The implementation of PRBS generator is based on the linear feedback shift register, which consists of 'n' master slave flip-flops. The PRBS generator produces a predefined sequence of 1's and 0's, with 1 and 0 occurring with the same probability. In order to determine the bit rate, or the number of bits per second, the frequency of the external clock is calculated, which is used to drive the generator. For each clock period a single bit is emitted from the generator; either at the '1' or '0' level, and of a width equal to the clock period. For this reason the external clock is referred to as a bit clock.

For a long sequence the 1's and 0's are distributed in a (pseudo) random manner. The sequence pattern repeats after a defined number of clock periods. In a typical generator the length of the sequence may be set to 2^n clock periods, where n is an integer. PRBS is implemented using LFSR or Linear Feedback Shift Register.

LFSR is an n-bit shift register which pseudo-randomly scrolls between $2^n - 1$ values, but does it very quickly because there is minimal combinational logic involved. Once it reaches its final state, it will traverse the sequence exactly as before.

2.2. SHIFT REGISTERS

One of the two main parts of an LFSR is the shift register (the other being the feedback function). A shift register is a device whose identifying function is to shift its contents into adjacent positions within the register or, in the case of the position on the end, out of the register. The position on the other end is left empty unless some new content is shifted into the register.

The contents of a shift register are usually thought of as being binary, that is, ones and zeroes. If a shift register contains the bit pattern 1101, a shift (to the right in this case) would result in the contents being 0110; another shift yields 0011. After two more shifts, things tend to get boring since the shift register will never contain anything other than zeroes.

Two uses for a shift register are:

- 1) Convert between parallel and serial data
- 2) Delay a serial bit stream.

The conversion function can go either way – fill the shift register positions all at once (parallel) and then shift them out (serial) or shift the contents into the register bit by bit (serial) and then read the contents after the register is full (parallel). The delay function simply shifts the bits from one end of the shift register to the other, providing a delay equal to the length of the shift register.

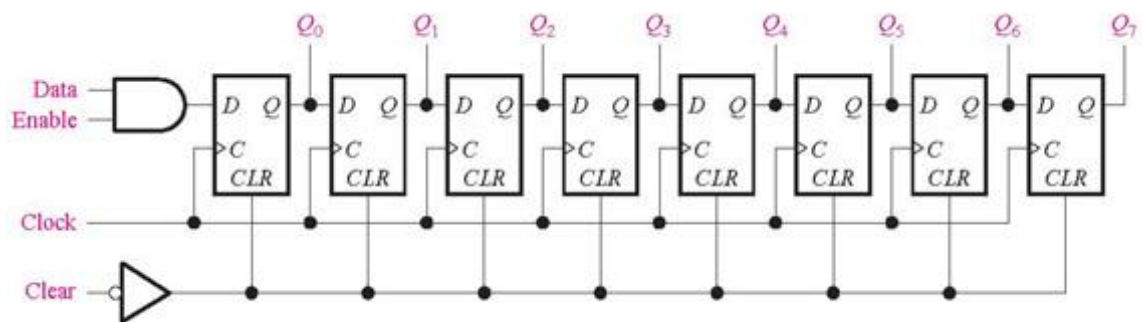


Fig 2.1 An eight bit shift register.

Nomenclature:

CLOCKING: One of the inputs to a shift register is the clock; a shift occurs in the register when this clock input changes state from one to zero (or from zero to one, depending on the implementation). From this, the term “clocking” has arisen to mean activating a shift of the register. Sometimes the register is said to be “strobed” to cause the shift.

SHIFT DIRECTION: A shift register can shift its contents in either direction depending on how the device is designed. (Some registers have extra inputs that dictate the direction of the shift.) For the purposes of this discussion, the shift direction will always be from left to right.

OUTPUT: During a shift, the bit on the far right end of the shift register is moved out of the register. This end bit position is often referred to as the output bit. To confuse matters a bit, the bits that are shifted out of the register are also often referred to as output bits. To really muddy the waters, every bit in the shift register is considered to be output during a serial to parallel conversion. Happily, the context in which the term “output” is used generally clears things up.

INPUT: After a shift, the bit on the left end of the shift register is left empty unless a new bit (one not contained in the original contents) is put into it. This bit is sometimes referred to as the input bit. As with the output bit, there are several different references to input that are clarified by context.

2.3. FEEDBACK ACTION

In an LFSR, the bits contained in selected positions in the shift register are combined in some sort of function and the result is fed back into the register’s input bit. By definition, the selected bit values are collected before the register is clocked and the result of the feedback function is inserted into the shift register during the shift, filling the position that is emptied as a result of the shift.

Feedback around an LFSR’s shift register comes from a selection of points (taps) in the register chain and constitutes XORing these taps to provide tap(s) back into the register.

Register bits that do not need an input tap, operate as a standard shift register. It is this feedback that causes the register to loop through repetitive sequences of pseudo-random value. The choice of taps determines how many values there are in a given sequence before the sequence repeats. The implemented LFSR uses a one-to-many structure, rather than a many-to-one structure, since this structure always has the shortest clock-to-clock delay path.

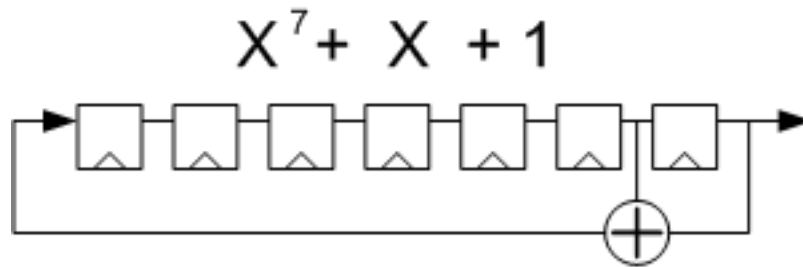


Fig 2.2 Feedback Action using LFSR.

The feedback is done so as to make the system more stable and free from errors. Specific taps are taken from the tapping points and then by using the XOR operation on them they are fed back into the registers. The table for XOR is given below for various inputs:

Table 2.1: 2-Input XOR Gate.

A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

The bit positions selected for use in the feedback function are called "taps". The list of the taps is known as the "tap sequence". By convention, the output bit of an LFSR that is n bits long is the n th bit; the input bit of an LFSR is bit 1.

2.4. TAPPING ACTION

An LFSR is one of a class of devices known as state machines. The contents of the register, the bits tapped for the feedback function, and the output of the feedback function together describe the state of the LFSR. With each shift, the LFSR moves to a new state. (There is one exception to this -- when the contents of the register are all zeroes, the LFSR will never change state.) For any given state, there can be only one succeeding state. The reverse is also true: any given state can have only one preceding state. For the rest of this discussion, only the contents of the register will be used to describe the state of the LFSR.

A state space of an LFSR is the list of all the states the LFSR can be in for a particular tap sequence and a particular starting value. Any tap sequence will yield at least two state spaces for an LFSR. (One of these spaces will be the one that contains only one state -- the all zero one.) Tap sequences that yield only two state spaces are referred to as maximal length tap sequences.

The state of an LFSR that is n bits long can be any one of 2^n different values. The largest state space possible for such an LFSR will be $2^n - 1$ (all possible values minus the zero state). Because each state can have only one succeeding state, an LFSR with a maximal length tap sequence will pass through every non-zero state once and only once before repeating a state.

One corollary to this behavior is the output bit stream. The period of an LFSR is defined as the length of the stream before it repeats. The period, like the state space, is tied to the tap sequence and the starting value. As a matter of fact, the period is equal to the size of the state space. The longest period possible corresponds to the largest possible state space, which is produced by a maximal length tap sequence. (Hence "maximal length")

2.5. TRUTH TABLE

Table 2.2: Truth table for 8 bit PRBS.

<u>CYCLE</u>	Q[7]	Q[6]	Q[5]	Q[4]	Q[3]	Q[2]	Q[1]	Q[0]
0	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1
3	0	0	0	1	1	1	1	1
4	0	0	0	0	1	1	1	1
5	0	0	0	0	0	1	1	1
6	0	0	0	0	0	0	1	1
7	0	0	0	0	0	0	0	1
8	1	0	0	0	0	0	0	0
9	0	1	0	0	0	0	0	0
10	0	0	1	0	0	0	0	0
11	0	0	0	1	0	0	0	0
12	0	0	0	0	1	0	0	0
13	0	0	0	0	0	1	0	0
14	0	0	0	0	0	0	1	0
15	1	0	0	0	0	0	0	1
16	1	1	0	0	0	0	0	0
17	0	1	1	0	0	0	0	0
18	0	0	1	1	0	0	0	0
19	0	0	0	1	1	0	0	0
20	0	0	0	0	1	1	0	0
21	0	0	0	0	0	1	1	0
22	1	0	0	0	0	0	1	1
23	0	1	0	0	0	0	0	1
24	1	0	1	0	0	0	0	0
25	0	1	0	1	0	0	0	0

26	0	0	1	0	1	0	0	0
27	0	0	0	1	0	1	0	0
28	0	0	0	0	1	0	1	0
29	1	0	0	0	0	1	0	1
30	1	1	0	0	0	0	1	0
31	1	1	1	0	0	0	0	1
32	1	1	1	1	0	0	0	0
33	0	1	1	1	1	0	0	0
34	0	0	1	1	1	1	0	0
35	0	0	0	1	1	1	1	0
36	1	0	0	0	1	1	1	1
37	0	1	0	0	0	1	1	1
38	0	0	1	0	0	0	1	1
39	0	0	0	1	0	0	0	1
40	1	0	0	0	1	0	0	0
41	0	1	0	0	0	1	0	0
42	0	0	1	0	0	0	1	0
43	1	0	0	1	0	0	0	1
44	1	1	0	0	1	0	0	0
45	0	1	1	0	0	1	0	0
46	0	0	1	1	0	0	1	0
47	1	0	0	1	1	0	0	1
48	1	1	0	0	1	1	0	0
49	0	1	1	0	0	1	1	0
50	1	0	1	1	0	0	1	1
51	0	1	0	1	1	0	0	1
52	1	0	1	0	1	1	0	0
53	0	1	0	1	0	1	1	0
54	1	0	1	0	1	0	1	1
55	0	1	0	1	0	1	0	1

56	1	0	1	0	1	0	1	0
57	1	1	0	1	0	1	0	1
58	1	1	1	0	1	0	1	0
59	1	1	1	1	0	1	0	1
60	1	1	1	1	1	0	1	0
61	1	1	1	1	1	1	0	1
62	1	1	1	1	1	1	1	0
63	1	1	1	1	1	1	1	1

2.6. MAXIMAL LENGTH TAP SEQUENCES:

LFSR's can have multiple maximal length tap sequences. A maximal length tap sequence also describes the exponents in what is known as a primitive polynomial mod 2.

Example, A tap sequence of 4, 1 describes the primitive polynomial $x^4 + x^1 + 1$. Finding a primitive polynomial mod 2 of degree n (the largest exponent in the polynomial) will yield a maximal length tap sequence for an LFSR that is n bits long. There is no quick way to determine if a tap sequence is maximal length. However, there are some ways to tell if one is not maximal length:

- 1) Maximal length tap sequences always have an even number of taps.
- 2) The tap values in a maximal length tap sequence are all relatively prime.

A tap sequence like 12, 9, 6, 3 will not be maximal length because the tap values are all divisible by 3. Discovering one maximal length tap sequence leads automatically to another. If a maximal length tap sequence is described by [n, A, B, C], another maximal length tap sequence will be described by [n, n-C, n-B, n-A]. Thus, if [32, 3, 2, 1] is a maximal length tap sequence, [32, 31, 30, 29] will also be a maximal length tap sequence. An interesting behavior of two such tap sequences is that the output bit streams are mirror images in time.

2.7. CHARACTERISTICS OF OUTPUT STREAM:

By definition, the period of an LFSR is the length of the output stream before it repeats. Besides being non-repetitive, a period of a maximal length stream has other features that are characteristic of random streams.

1) Sums of ones and zeroes.

In one period of a maximal length stream, the sum of all ones will be one greater than the sum of all zeroes. In a random stream, the difference between the two sums will tend to grow progressively smaller in proportion to the length of the stream as the stream gets longer. In an infinite random stream, the sums will be equal.

2) Runs of ones and zeroes.

A run is a pattern of equal values in the bit stream. A bit stream like 10110100 has six runs of the following lengths in order: 1, 1, 2, 1, 1, 2. One period of an n -bit LFSR with a maximal length tap sequence will have $2^{(n-1)}$ runs (e.g., a 5 bit device yields 16 runs in one period). $1/2$ the runs will be one bit long, $1/4$ the runs will be 2 bits long, $1/8$ the runs will be 3 bits long, etc., up to a single run of zeroes that is $n-1$ bits long and a single run of ones that is n bits long. A random stream of sufficient length shows similar behavior statistically.

3) Shifted stream.

Take the stream of bits in one period of an LFSR with a maximal length tap sequence and circularly shift it any number of bits less than the total length. Do a bitwise XOR with the original stream. A random stream also shows this behavior. One characteristic of the LFSR output not shared with a random stream is that the LFSR stream is deterministic. Given knowledge of the present state of the LFSR, the next state can always be predicted.

CHAPTER 3

SIMULATION IN VERILOG HDL

3.1. INTRODUCTION

The code for implementing the required PRBS is realized by writing VHDL program. In the program the logic implemented is very simple. An 8-bit PRBS is realized by shifting the input through the D-flip flops and feed backing the outputs of some registers known as taps again into the first register after passing them through a XOR gate.

3.2. FEATURES

The process of realizing LFSR is carried out by first developing the VHDL code for a D-flip flop. The same D- flip flop code is then called 8 times in the main program code to realize the required LFSR. In the code for the PRBS tapings are taken so as to get the maximum range of the binary numbers generated. In the developed code tapings are taken from 6th and 7th taps so as to obtain the maximum length of binary digits produced.

Initially when the reset is kept at zero the outputs of each of the registers is uninitialized and hence the output is uninitialized as well. However as soon as the reset is made high the output of all the registers start coming out. A dead lock condition arises in the case when the initial input into the first register as output of the XOR gate are all 0's. Under this condition the output of all the register of the PRBS Generator remains as 0 at all instants of time. Therefore it is necessary that the initial input to the PRBS Generator be equal to 1, the output of the XOR gate.

3.3. SIMULATION OUTPUT:

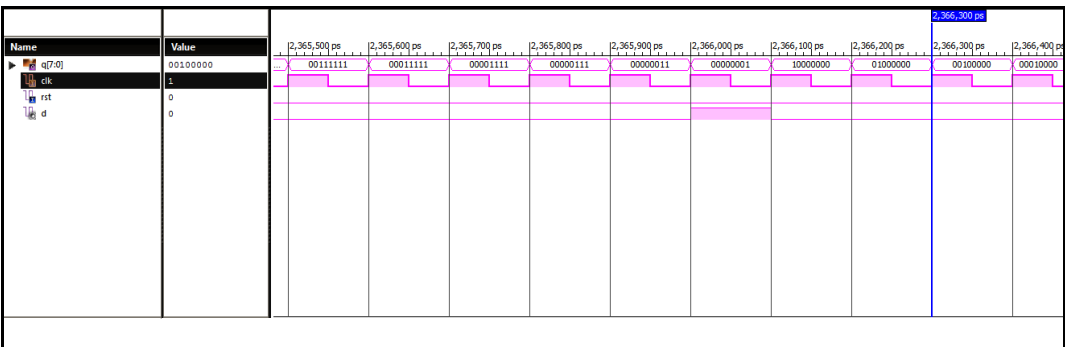
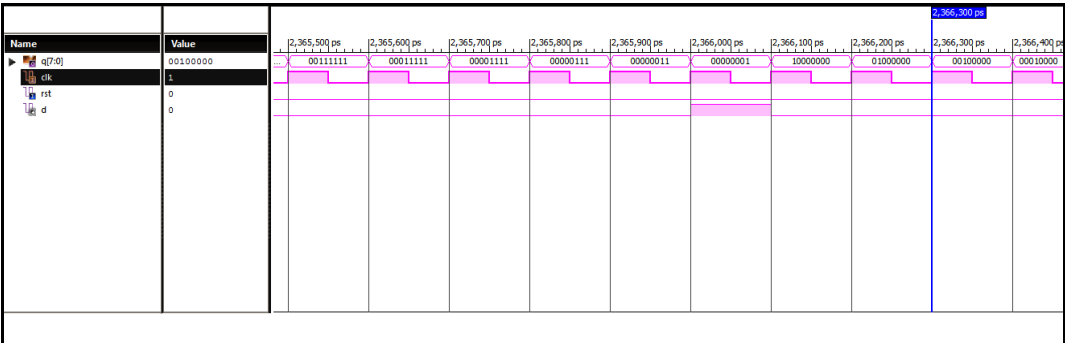
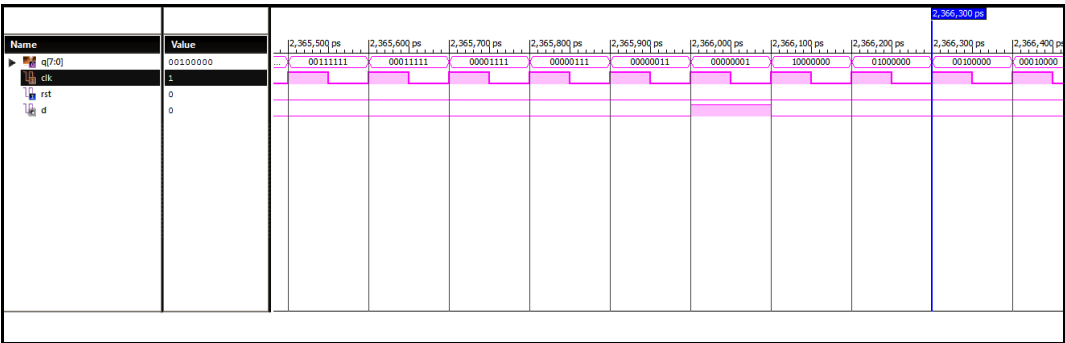
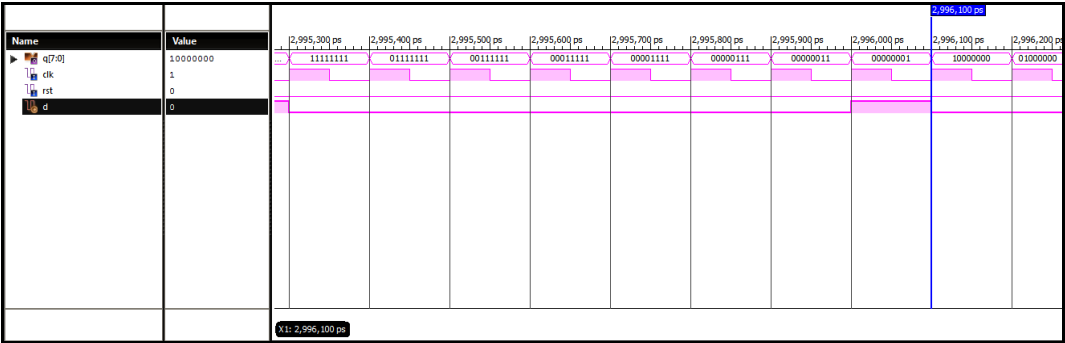




Fig 3.1 Simulation output for 8 bit PRBS.

Table 3.1: Project Summary in XILINX.

PRBS Project Status (09/14/2015 - 12:00:20)			
Project File:	PRBS.xise	Parser Errors:	No Errors
Module Name:	PRBS	Implementation State:	Programming File Generated
Target Device:	xc3s250e-4pq208	• Errors:	No Errors
Product Version:	ISE 12.1	• Warnings:	1 Warning (0 new)
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

Performance Summary			
Final Timing Score:	0 (Setup: 0, Hold: 0)	Pinout Data:	Pinout Report
Routing Results:	All Signals Completely Routed	Clock Data:	Clock Report

Detailed Reports					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Mon Sep 14 11:55:18 2015	0	0	0
Translation Report	Current	Mon Sep 14 11:56:53 2015	0	0	0
Map Report	Current	Mon Sep 14 11:57:09 2015	0	0	2 Infos (0 new)
Place and Route Report	Current	Mon Sep 14 11:57:30 2015	0	1 Warning (0 new)	4 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Mon Sep 14 11:57:32 2015	0	0	5 Infos (0 new)
Bitgen Report	Current	Mon Sep 14 12:00:16 2015	0	0	0
TiminConstraints:		All Constraints Met			

Secondary Reports			
Report Name	Status	Generated	
ISIM Simulator Log	Out of Date	Fri Sep 11 14:21:25 2015	
Post-Synthesis Simulation Model	Out of	Fri Sep 11 14:24:25 2015	

<u>Report</u>	Date	
<u>WebTalk Report</u>	Current	Mon Sep 14 12:00:16 2015
<u>WebTalk Log File</u>	Current	Mon Sep 14 12:00:19 2015

Date Generated: 09/14/2015 - 12:00:20

3.4. HARDWARE IMPLEMENTATION:

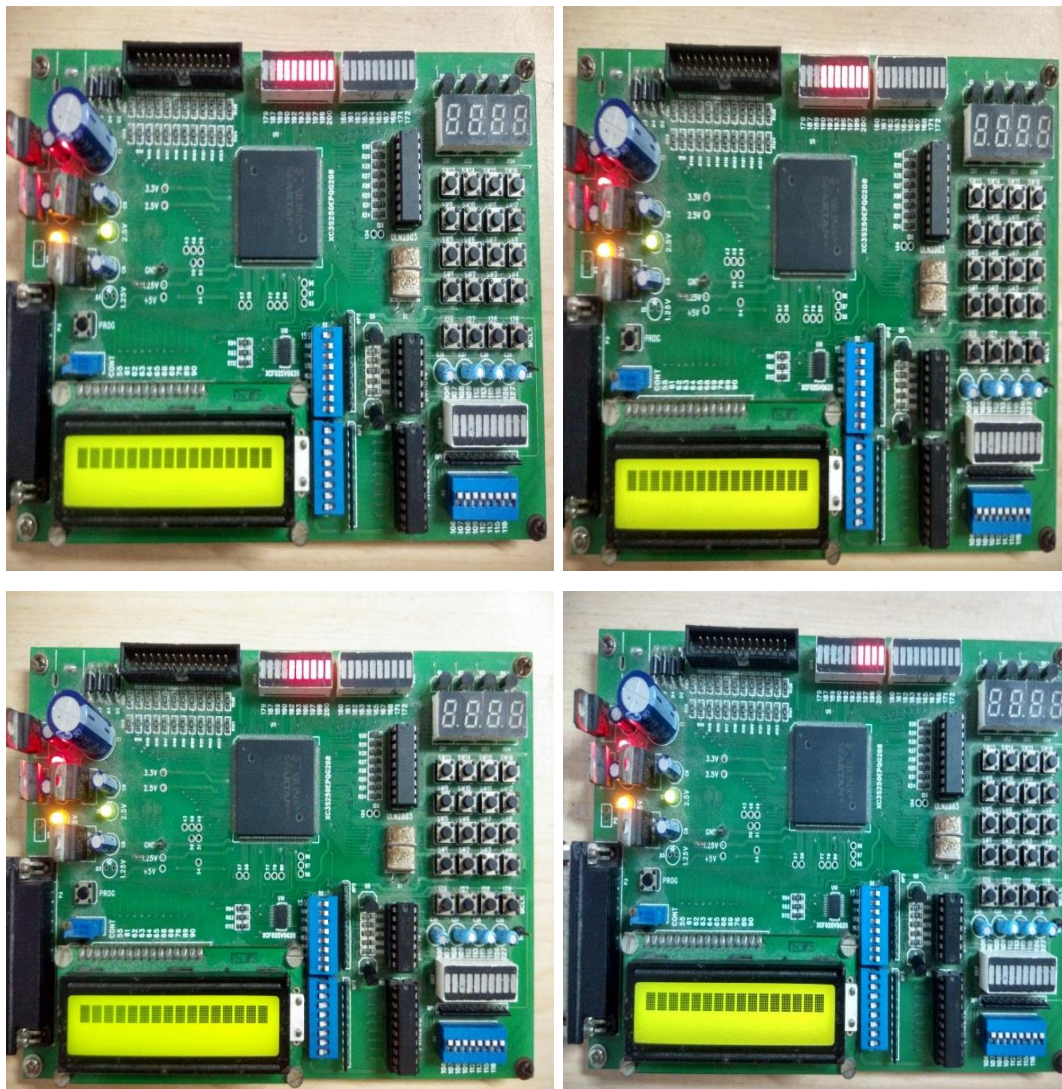


Fig 3.2 Hardware implementation using Xilinx FPGA trainer kit.

CHAPTER 4

CONCLUSION

The code for implementing the required PRBS is realized by writing VHDL program. In the program the logic implemented is very simple. An 8-bit PRBS is realized by shifting the input through the D-flip flops and feed backing the outputs of some registers known as taps again into the first register after passing them through a XOR gate. The process of realizing LFSR is carried out by first developing the VHDL code for a D-flip flop. The same D- flip flop code is then called 8 times in the main program code to realize the required LFSR. In the code for the PRBS tapings are taken so as to get the maximum range of the binary numbers generated. In the developed code tapings are taken from 6th and 7th taps so as to obtain the maximum length of binary digits produced. Initially when the reset is kept at zero the outputs of each of the registers is uninitialized and hence the output is uninitialized as well. However as soon as the reset is made high the output of all the registers start coming out. A dead lock condition arises in the case when the initial input into the first register as output of the XOR gate are all 0's. Under this condition the output of all the register of the PRBS Generator remains as 0 at all instants of time. Therefore it is necessary that the initial input to the PRBS Generator be equal to 1, the output of the XOR gate. The code for implementing the above circuit was written and hence the simulation results were generated and tested.

REFERENCES

- [1] Horowitz and Hill, *The Art of Electronics*, 2nd Edition, pp. 665-667, 1989.
- [2] P.Alfke, "Efficient Shift Registers, LFSR, Counters and Long Pseudo-Random Sequence Generators," *XAPP 052*, Version 1.1, July 7, 1996.
- [3] Douglas J. Smith, "HDL Chip Design", *Doone Publications*, pp. 423-430, 1996.
- [4] Woody Johnson, "Linear Feedback Registers", *Freecore*, pp. 322-325, 1997.
- [5] Joshua Barona, Yuval Ishaib, Rafail Ostrovsky, "On linear-size pseudorandom generators and hardcore functions", *Theoretical Computer Science*, Vol. 554, pp. 50–63, October 2014.
- [6] Hongsong Shi, Shaoquan Jiang, Zhiguang Qin, "More efficient DDH pseudorandom generators", *Designs, Codes and Cryptography*, Vol. 55, Issue 1, pp. 45-64, April 2010.
- [7] Jimenez-Horas, San Millan, Lopez-Ongil, Portela-Garcia, "Pseudo-random number generation applied to robust modern cryptography: A new technique for block ciphers", *On-Line Testing Symposium-15th IEEE International*, 24-26 June 2009, pp. 203-205.
- [8] L. Blum, M. Blum, M. Shub, "Comparison of two pseudo-random number generators", *Proc. CRYPTO'82*, 1983, pp. 61-78.
- [9] William Aiello, S.Raj Rajagopalana, Ramarathnam Venkatesan, "Design of Practical and Provably Good Random Number Generators", *Journal of Algorithms*, Vol. 29, Issue 2, pp. 358–389, November 1998.