LINEAR FEEDBACK SHIFT REGISTERS IN WIRELESS COMMUNICATION SYSTEMS

Thesis submitted towards the partial fulfillment of requirements for the award of the degree of

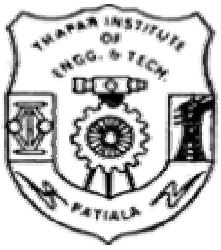
Master of Technology (VLSI Design & CAD)

Submitted by

(AMANDEEP KAUR)
Roll No 6040401

Under the Guidance of

(Mr.SANJAY SHARMA)
Asst.Professor



Department Of Electronics and CommunicationEngineering
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY,
(Deemed University), PATIALA – 147004, INDIA

JUNE 2006

ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an untrodden path towards an unexplored destination is an arduous adventure unless one gets a true torchbearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. The enlightening guidance, I found in my revered guide Mr. Sanjay Sharma, Assistant Professor, ECED, Thapar Institute of Engineering & Technology (Deemed University), Patiala, without whose patronization it was never possible to give final shape to this thesis. I wish to express my deep gratitude towards him for providing individual guidance and support throughout the Thesis work.

I shall be failing in my duties if I do not express my deep sense of gratitude towards Dr. R. S. Kaler, Professor & Head of the Department, Electronics & Communication Engineering Department and Dr. A. K. Chatterjee, P.G. Coordinator, Electronics and Communication Engineering Department.

I would also like to thank all the staff members and my co-students who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of my thesis. I am also thankful to the authors whose works I have consulted and quoted in this work.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Amandeep Kaur

DECLARATION

I hereby certify that the work which is being presented in the thesis entitled, **"Linear Feedback Shift Register in Wireless Communication Systems"** in partial fulfillment of the requirements for the award of degree of M. Tech. (VLSI Design and CAD) at Electronics and Communication Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Mr. Sanjay Sharma, Assistant Professor, ECED.

The matter presented in this thesis has not been submitted in any other University/ Institute for the award of any degree.

Amandeep Kaur

Date: ---------- **Roll. No. 6040401**

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Mr. Sanjay Sharma

(Asst. Professor)

Supervisor

Counter Signed By:

Head
Electronics and Communication
Engineering Department,
TIET, Patiala

Dean of Academic Affairs
TIET, Patiala

ABSTRACT

The field of wireless communications is currently growing at an unprecedented rate. This growth, prompted in part by the demand for more high-speed, broadband communication systems, has led to the need for high-resolution, broadband wireless measurement equipment. The mobile communication system is one of the most important phenomenon in the history of telecommunication which has enriched human civilization and mankind by bringing business and community together.. In Spread Spectrum CDMA (SS-CDMA) system each user is assigned a pseudo noise (PN) sequence for the purpose of spreading as well as dispreading. Thus PN-sequence generation is considered to be the heart of SS-CDMA system. The maximal length PN-sequence (m-sequence) is the best-known best-described PN-sequence whose length is equal to its period. Various PN-codes can be generated using Linear Feedback Shift Register (LFSR).The generator polynomial provides the necessary feedback taps for the LFSR circuit. The implementation of the LFSR circuit with VLSI technology makes it useful in low-power communication system design.

LFSR is basically, a shift register configuration that propagates the stored patterns from left to right. The modification that provides the PRBS generation is due to the XOR feedback of the selected flip-flop outputs, named *taps*. When the taps are chosen properly, the LFSR will traverse through all possible states except for the all 0s state and will produce a maximum length PRBS sequence named *M-sequence*. In order for the desired operation, the LFSR should be first initialized to a well-known stage, which is usually referred to as *seed*. For an n stage LFSR, there are 2^n-1 states, and the M-sequence is 2^n-1 bits long. Hence, the M-sequence is periodic, and after the 2^n-1 distinct values, it repeats itself in the next samples.

Thus the linear feedback shift register (LFSR) is a shift register which, using feedback, modifies itself on each rising edge of the clock. The feedback causes the value in the shift register to cycle through a set of unique values. The choice of LFSR length, gate type, LFSR type, maximum length logic, and tap positions allows the user to control

the implementation and feedback of the LFSR, which, in turn, controls the sequence of repeating values the LFSR will iterate through.

Linear Feedback Shift Registers (LFSRs) are a fundamental function in applications such as pseudo-random noise (PN) generators, RS Code Generators and BIST(Build in Self Test). PN generators are at the heart of every spread spectrum system, and are a good example for demonstrating how you can dramatically reduce FPGA utilization by exploiting the Virtex SRL. In a CDMA system, many PN generators are needed to distinguish channels, base stations, and handsets. You can achieve extremely efficient LFSR implementations by using the Virtex Shift Register LUT (SRL). Though the mathematics behind a PN code can be extremely complicated, the LFSR implementation can be relatively simple.

Another application of LFSR includes RS-Code Generator. RS code generators are used extensively in Code Division Multiple Access (CDMA) systems to generate code sequences with good correlation properties. The RS code generators use efficiently implemented Linear Feedback Shift Registers (LFSRs) in both the Virtex/Virtex-II series and Spartan-II family using the SRL16 macro.

LFSR's are also used as a important building block in Build in Self Test (BIST) Generation. Built-in self-test (BIST) techniques enable an integrated circuit (IC) to test itself. BIST reduces test and maintenance costs for an IC by eliminating the need for expensive test equipment and by allowing fast location of failed ICs in a system. BIST also allows an IC to be tested at its normal operating speed which is very important for detecting timing faults. Despite all of these advantages, BIST has seen limited use in industry because of area and performance overhead and increased design time. This dissertation presents automated techniques for implementing BIST in a way that minimizes area and performance overhead.

Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1
INTRODUCTION

1.1 INTRODUCTION

With the advancement of VLSI technology spread spectrum CDMA system has now come up as a highly emerging digital technology for mobile systems. In this system the basic hardware involves maximal length PN sequence or m-sequence generated by a Linear Feedback Shift Register (LFSR) provided this LFSR is represented by a primitive polynomial. The mobile communication system is one of the most important phenomenons in the history of telecommunication which has enriched human civilization and mankind by bringing business and community together. To accommodate more traffic and emerging new services within a limited amount of radio spectrum, a new digital wireless technology called CDMA was developed for 2G and 3G mobile systems. In Spread Spectrum CDMA (SS-CDMA) system each user is assigned a pseudo noise (PN) sequence for the purpose of spreading as well as dispreading. Thus PN-sequence generation is considered to be the heart of SS-CDMA system. The maximal length PN-sequence (m-sequence) is the best-known best-described PN-sequence whose length is equal to its period. Various PN-codes can be generated using Linear Feedback Shift Register (LFSR).The generator polynomial provides the necessary feedback taps for the LFSR circuit. The implementation of the LFSR circuit with VLSI technology makes it useful in low-power communication system design [1].

1.2 CODE GENERATION IN WIRELESS COMMUNICATION

A wide variety of wireless applications, including data encryption and circuit testing, require random numbers. As the cost of the hardware become cheaper, it is feasible and frequently necessary to implement the random number generator directly in hardware itself. Ideally, the generated random number should be uncorrelated. A generator can be either "truly random" or "pseudo random." The former exhibits true randomness and the value of next number is unpredictable. The later only appears to be random. The sequence is based on specific mathematical algorithms, and thus the pattern is repetitive and predictable. However, if the cycle period is very large, the sequence appears to be non-repetitive and random. Ideally the spreading codes used in direct sequence spread spectrum systems would be truly random binary sequences, as might be produced by

consecutive tosses of an unbiased, memoryless coin. This is not practical. Both transmitter and receiver must generate the same sequence, time-aligned, in order to communicate with one another. Receivers thus must perform synchronization searches by changing their time offset hypothesis until the transmitter timing is located. Achieving high capacity in the CDMA environment also requires that the spreading rate be high: 1.2288 MHz in IS-95A CDMA. If truly random sequences were to be used then they would have to be pre-generated and pre-stored in all transmitters, with a matching copy in all receivers. Deterministic methods of generating the pseudo-random sequences are preferable. The CDMA applications use linear feedback shift register (LFSR) generators for this purpose. Code Division Multiple Access (CDMA) systems offer high spectrum efficiency, thanks to their capability to allocate all available bandwidth to each user [1, 2].

Spreading is accomplished by multiplying the information symbols with a high rate pseudo-random sequence (so called pseudo-noise, PN) known to the receiver. The resulting signal is wideband and can be demodulated again by multiplying it with a synchronized replica of the PN sequence used by the transmitter. Spreading codes have good correlation properties so that each spread spectrum signal is uncorrelated with every other signal sharing the same bandwidth. The PN sequence is private to each user, thus allowing bandwidth sharing without any loss of information [3].

Several design techniques have been examined for hardware random number generators and this feasibility for FPGA devices. LFSR is the most effective method for single bit random number generator. When multiple bits are required, LFSR can be extended by utilizing extra circuitry. For a small number of bits, leap-forward LFSR method is ideal because it balances the combinational circuitry and register, and balances the FPGA resource. We can extend this technique to a greater number of bits where one can get even better uncorrelated sequences for random number generation. The maximal-length binary sequences produced by linear feedback shift registers are widely used for direct sequence spectrum spreading. There are several reasons for this:

1. LFSR sequences are easily generated by very simple binary logic circuits.
2. Very high speed generators are possible because of the simple logic.

3. Maximal length sequence generators are easily designed using finite (Galois) field mathematics.
4. The full period autocorrelation functions of maximal-length LFSR sequences are binary-valued, facilitating synchronization searching [4].

CDMA is a form of *spread-spectrum*, a family of digital communication techniques that have been used in military applications for many years. ***Code Division Multiple Access (CDMA)*** is a radically new concept in wireless communications. It has gained widespread international acceptance by cellular radio system operators as an upgrade that will dramatically increase both their system capacity and the service quality. It has likewise been chosen for deployment by the majority of the winners of the United States Personal Communications System spectrum auctions. It may seem, however, mysterious for those who aren't familiar with it [5].

1.3 SPREAD SPECTRUM CODING

PN sequences are commonly used in a variety of situations such as ranging and error checking. The codes used in spread spectrum systems are inherently much longer than those found in other systems as they are intended for bandwidth spreading rather than information transfer. A spread spectrum system is largely categorized by its coding scheme. The type of code employed, its length, and its chip-rate all dense the overall system parameters. In order to alter the system's spreading capability it is necessary to alter the coding arrangement [6].

All spread spectrum communication systems employ a pseudo-noise (PN) code sequence to spread the data modulated carrier at the transmitter and despread the desired carrier at the receiver. This experiment investigates hardware necessary to generate such PN codes and observe them in both time and frequency domains. One of the simplest techniques for generating a pseudo random sequence of ones and zeros is a linear feedback shift register with a feedback loop. Autocorrelation, cross correlation, and power spectrum of PN codes are important functions to know when evaluating the performance of spread spectrum communication systems they are used in [6, 7].

1.4 PSEUDO RANDOM BINARY SEQUENCES

Pseudo random binary sequences (PRBSs) are widely used for testing hardware for digital communication. Testing of hardware for digital communication requires transmission and reception of a signal that subjects the transmission channel to the characteristics of random digital signal. A PRBSs is a random bit sequence that repeats it self, thus not truly random, as the name implies. A truly random sequence never repeats it self, but truly random sequences are difficult to generate, and would have very little use in practical systems. However PRBSs with long sequence lengths (several billion bits) show close resemblance to truly random signals, and are sufficient for test purposes. PRBSs have well known properties, and the generation and acquisition of them are simple. Knowing how a PRBS is generated makes it possible to predict the sequence. This is a very desirable feature when testing hardware for digital communication, as it allows you to predict how an incoming sequence is supposed to look. This makes it possible to register and count any errors that might occur in the sequence. PRBSs can be generated by shifting bits through a number (N) of cascaded registers, where some of the register outputs (referred to as tap sets) are added modulo-2 and fed back to the input of the first register. The maximal length of the sequence is determined by the number of possible states that the shift register can assume, and the properties of the sequence is determined by which tap sets that are modulo-2 added and feed back to the first register. This type of PRBS generator is called a linear feedback shift

Register (LFSR) [8, 9].

CHAPTER 2
LINEAR FEEDBACK SHIFT REGISTER

2.1 INTRODUCTION

A **linear feedback shift register** (LFSR) is a shift register whose input bit is a linear function of its previous state.The only linear functions of single bits are xor and inverse-xor; thus it is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the overall shift register value.The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the sequence of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, a LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle.An LFSR is a shift register that, when clocked, advances the signal through the register from one bit to the next most-significant bit (see Figure 2.1). Some of the outputs are combined in exclusive-OR configuration to form a feedback mechanism. A linear feedback shift register can be formed by performing exclusive-OR on the outputs of two or more of the flip-flops together and feeding those outputs back into the input of one of the flip-flops as shown in (Figure 2.2) [10].

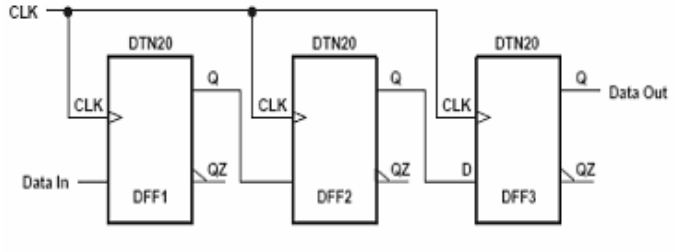


Figure 2.1 a 3-Bit Shift Register

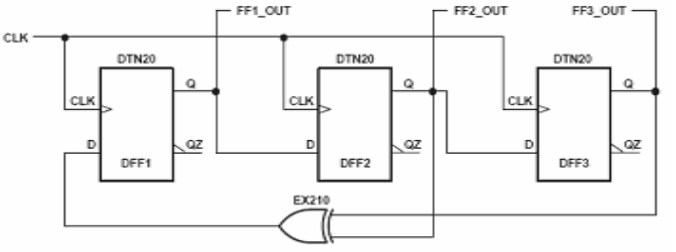


Figure 2.2 Linear Feedback Shift Register

2.2 PSEUDORANDOM PATTERN GENERATION

Linear feedback shift registers make extremely good pseudorandom pattern generators. When the outputs of the flip-flops are loaded with a seed value (anything except all 0s, which would cause the LFSR to produce all 0 patterns) and when the LFSR is clocked, it will generate a pseudorandom pattern of 1s and 0s. Note that the only signal necessary to generate the test patterns is the clock [11].

2.3 WORKING OF LFSR

Linear Feedback Shift Registers sequence through (2^n-1) states, where *n* is the number of registers in the LFSR. At each clock edge, the contents of the registers are shifted right by one position. There is feedback from predefined registers or taps to the left most register through an exclusive-NOR (XNOR) or an exclusive-OR (XOR) gate. A value of all "1"s is illegal in the case of a XNOR feedback. A count of all "0"s is illegal for an XOR feedback. This state is illegal because the counter would remain locked-up in this state. The LFSR shown below is implemented with XNOR feedback. A 4-bit LFSR sequences through (24 - 1) = 15 states (the state 1111 is in the lock-up/illegal state). From (Table 1) the feedback taps are 4, 3. On the other hand, a 4-bit binary up-counter would sequence through 24 = 16 states with no illegal states. LFSR counters are very fast since they use no carry signals. However, the dedicated carry in Virtex devices is rarely a speed limiting factor because it is intrinsically fast. LFSRs can replace conventional binary counters in performance critical applications where the count sequence is not important (e.g., FIFO). LFSRs are also used as pseudo-random bit stream generators. They are important building blocks in the implementation of encryption and decryption algorithms.The list of the bits positions that affect the next state is called the tap sequence. In the diagram below, the sequence is (16,14,13,11)

- The outputs that influence the input are called *taps*.
- A maximal LFSR produces an n-sequence(i.e. cycles through all possible states within the shift register), unless it contains all zeros, in which case it will never change.

The sequence of numbers generated by a LFSR can be considered a binary numeral system just as valid as Gray code or the natural binary code.The tap sequence of an LFSR

can be represented as a polynomial mod 2. This means that the coefficients of the polynomial must be 1's or 0's. This is called the feedback polynomial or characteristic polynomial. For example, if the taps are at the 16th, 14th, 13th and 11th bits (as below), the resulting LFSR polynomial is

$$x^{11} + x^{13} + x^{14} + x^{16} + 1$$

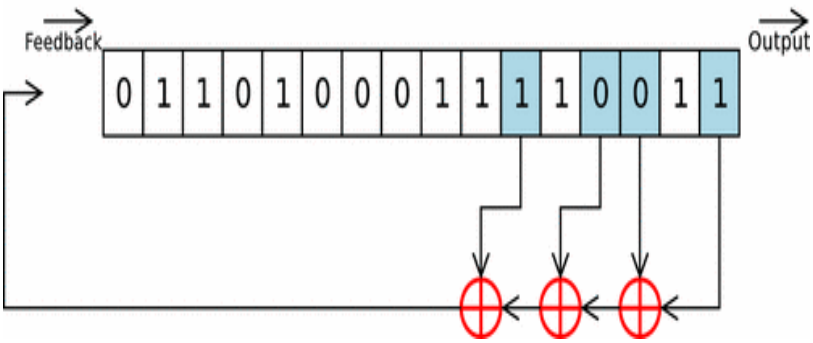


Figure 2.3 LFSR With 16,14,13,11 Bits As Tap

The 'one' in the polynomial does not correspond to a tap. The powers of the terms represent the tapped bits, counting from the left.

- If (and only if) this polynomial is a primitive, then the LFSR is maximal
- The LFSR will only be maximal if the number of taps is even.
- The tap values in a maximal LFSR will be relatively prime.
- There can be more than one maximal tap sequence for a given LFSR length [11,12].

CHAPTER-3
LINEAR FEEDBACK SHIFT REGISTER IN WIRELESS COMMUNICATION

3.1 INTRODUCTION

Linear Feedback Shift Registers (LFSRs) are commonly used in wireless applications where pseudorandom bit streams are required. LFSRs are the functional building blocks of circuits like the pseudo-random noise (PN) code generator and Gold code generators commonly used in Code Division Multiple Access (CDMA) systems. LFSRs can be used for performance-critical binary counters used to generate sequences of random numbers. LFSRs will often satisfy this requirement, although the generated sequence is pseudo-random in nature. Pseudo-random patterns repeat over time; the longer the LFSR, however, the longer the sequence of random numbers before pattern repetition occurs. When longer sequences are desired, the physical size of the hardware is increased. Conventionally, in the older FPGA architectures, flip-flops would be used. With two flip-flops in each of the older-architecture CLBs, an n-bit LFSR will take up at least n/2 CLBs. LFSRs sequence through 2^N–1 states, where N is the number of flip-flops in the LFSR. At each clock edge, the contents of the flip-flops are shifted right by one position. There is a feedback path from predefined flip-flops to the leftmost flip-flop through an exclusive-NOR (XNOR) or an exclusive-OR (XOR) gate. A value of all "1's" is illegal in the case of an XNOR feedback, and a value of all "0's" is illegal for XOR feedback. The illegal state causes the counter to remain in its present state, locking out any further new values from being registered[13].

3.2 LFSR TERMINOLOGY

LFSRs sequence through 2^N –1 states, where N is the number of registers in the LFSR. The contents of the registers are shifted right by one position at each clock cycle. The feedbacks from predefined registers or taps to the leftmost register are XORed together. LFSRs have several variables:

- The number of stages in the shift registers
- The number of taps in the feedback path
- The position of each tap in the shift registers stage

• The initial starting condition of the shift register, often referred to as the FILL state

In the case of LFSRs with an XOR feedback, the FILL value must be non-zero to avoid the LFSR locking up in the next state.

3.2.1 Shift Register Length (*N*)

The shift register length is often referred to as the *degree*, and the longer the shift register, the longer the duration of the PN sequence before it repeats. For a shift register of fixed length *N*, the number and duration of the sequences it can generate are determined by the number and position of taps used to generate the parity feedback bit.

3.2.2 Shift Register Taps

The combination of taps and their location is often referred to as a polynomial, and expressed as

$$P(x) = X^7 + X^3 + 1$$

Various conventions are used to map the polynomial terms to register stages in the shift register implementation. In the polynomial $P(x) = X^7 + X^3 + 1$, the trailing "1" represents X^0, which is the output of the last stage of the shift register. X^3 is the output of register stage 3 and X^7 the output of the XOR. A few points to note about LFSRs and the polynomial used to describe them:

• The last tap of the shift register is the leading "1" and is always used in the shift register feedback path.

• The length of the shift register can be deduced from the exponent of the highest order term in the polynomial.

• The highest order term of the polynomial is the signal connecting the final XOR output to the shift register input. It does not feed back into the parity calculation along with the other taps identified in the polynomial [13, 14].

3.3 LFSR IMPLEMENTATION

There are two implementation styles of LFSRs, Galois implementation and Fibonacci implementation.

3.3.1 Galois Implementation

As shown in (Figure 3.1), the data flow is from left to right and the feedback path is from right to left. The polynomial increments from left to right with X^0 term (the "1" in the

polynomial) as the first term. This is referred to as a Tap polynomial, as it indicates which taps are to be fed back from the shift register. Since the XOR gate is in the shift register path, the Galois implementation is also known as an *in-line* or modular type (M-type) LFSR.

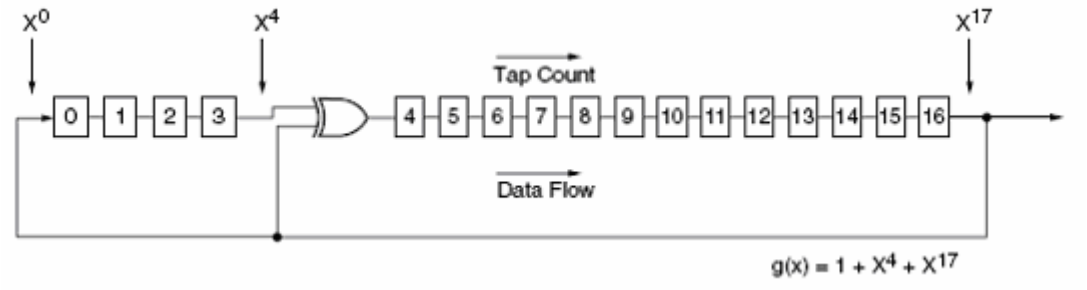


Figure 3.1 Galois Implementation

3.3.2 Fibonacci Implementation

In (Figure 3.2), the data flow is from left to right and the feedback path is from right to left, similar to the Galois implementation. However, the Fibonacci implementation polynomial decrements from left to right with X^0 as the last term in the polynomial. This polynomial is referred to as a Reciprocal Tap polynomial and the feedback taps are incrementally annotated from right to left along the shift register. Since the XOR gate is in the feedback path, the Fibonacci implementation is also known as an *out-of-line* or simple type (S-type) LFSR [14, 15].

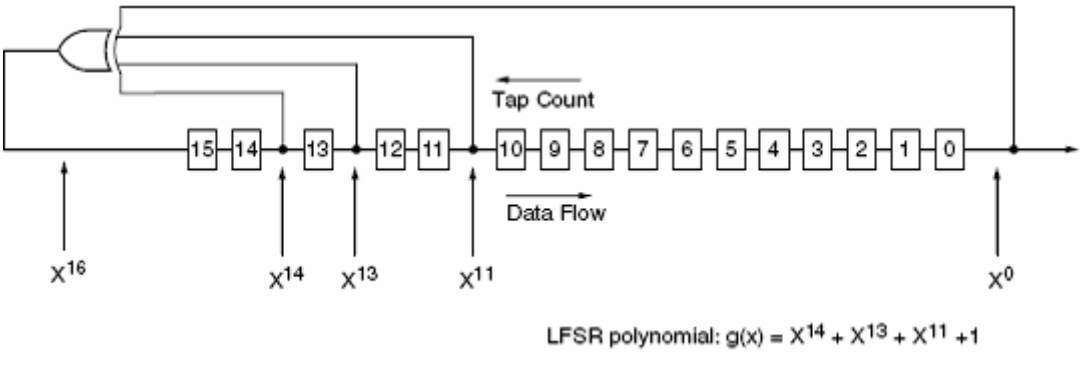


Figure 3.2 Fibonacci Implementation

3.3.3 MAXIMAL LENGTH SEQUENCES (*L*)

A maximal length sequence for a shift register of length N is referred to as an m-sequence, and is defined as:

$$L = 2^N - 1$$

An eight-stage LFSR, for example, will have a set of m-sequences of length 255[15].

3.4 SHIFT REGISTER LOOK-UP TABLE (LUT) MODE FOR AREA-EFFICIENT LFSRS

With the SRL16 primitive, it is possible to implement an *n*-bit LFSR in a fraction of the space used by a flip-flop design. A 16-bit LFSR would take up at least eight slices using flip-flops, since there are just two flip-flops per slice. The same 16-bit LFSR can be implemented in just four slices when using the SRL16s. Virtex-II devices have a new macro, SRLC16 in addition to the SRL16/E. Two outputs of the SRLC16 can be accessed simultaneously. One output is determined by the value of the 4-bit address line (i.e., A [3] –A [0]) and the other output is the cascadable output (the 16th bit of the shift register.) In Virtex devices, only a single tap or output of the SRL16s can be accessed at a time. The SRL16 output to be accessed is determined by the value of the 4-bit address line A [3] – A [0]. The SRL16 primitive will shift data on every clock cycle. A second primitive (SRL16E) provides the same shift register functionality, but adds a shift register Clock Enable. Both the SRL16 and SRL16E implement area-efficient shift registers in one LUT. It is worth noting, however, that parallel access to multiple taps is not possible, as the primitives have only one data output pin. Thus, for every single output that must be accessed, another SRL16 must be created if that output is in the same 16-bit set as the other. Because the number of taps rarely exceeds four, creating multiple instances of the SRL16 primitive is not a concern. It should be noted that because flip-flop based LFSRs will only consume as many flip-flops as there are stages in the shift register, the size at which it becomes more area efficient to use flip-flops is less than or equal to eight. To overcome the loss of parallel access, the following two approaches are reviewed [16].

3.4.1 MULTIPLE SHIFT REGISTERS WITH PARALLEL TAP ACCESS

(Figure3.3) demonstrates a 16-stage LFSR with four selectable tap points designed with four SRL16 primitives. An additional 4-input LUT is used to implement a parallel XOR parity calculation (Figure 3.4) that is fed back into the shift register as the new bit in the sequence. Tap D is the last stage in the shift register and so represents the LFSR output. This circuit is clocked at a frequency known as the *chip rate* [16].

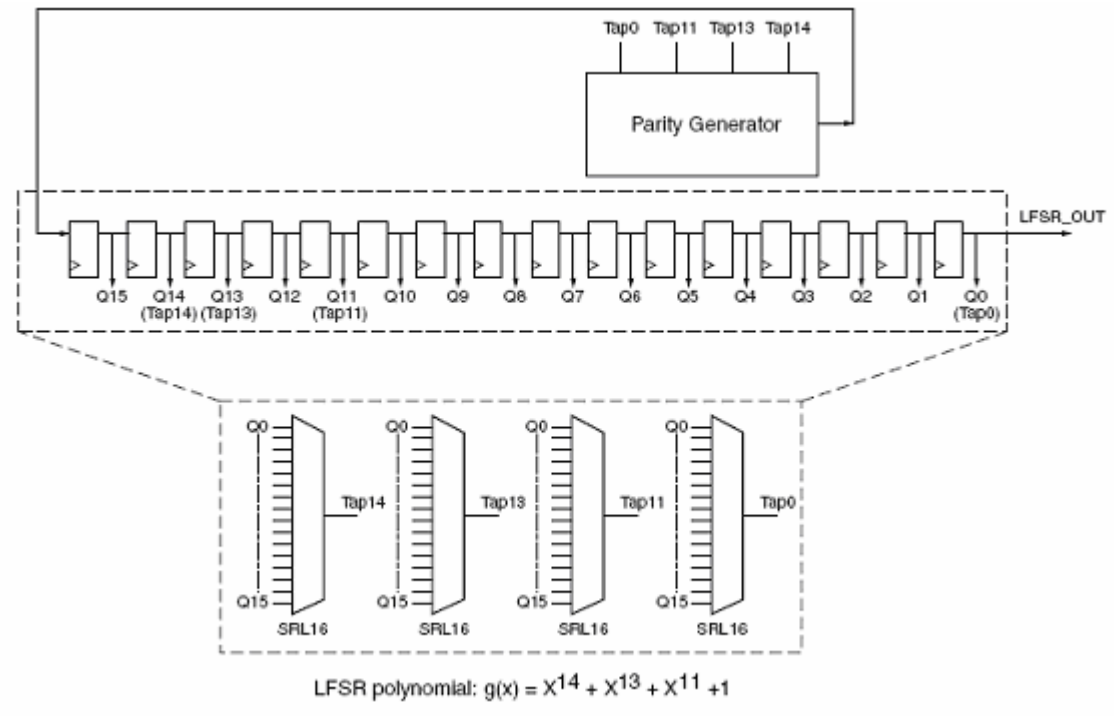


Figure3.3 Parity Calculation

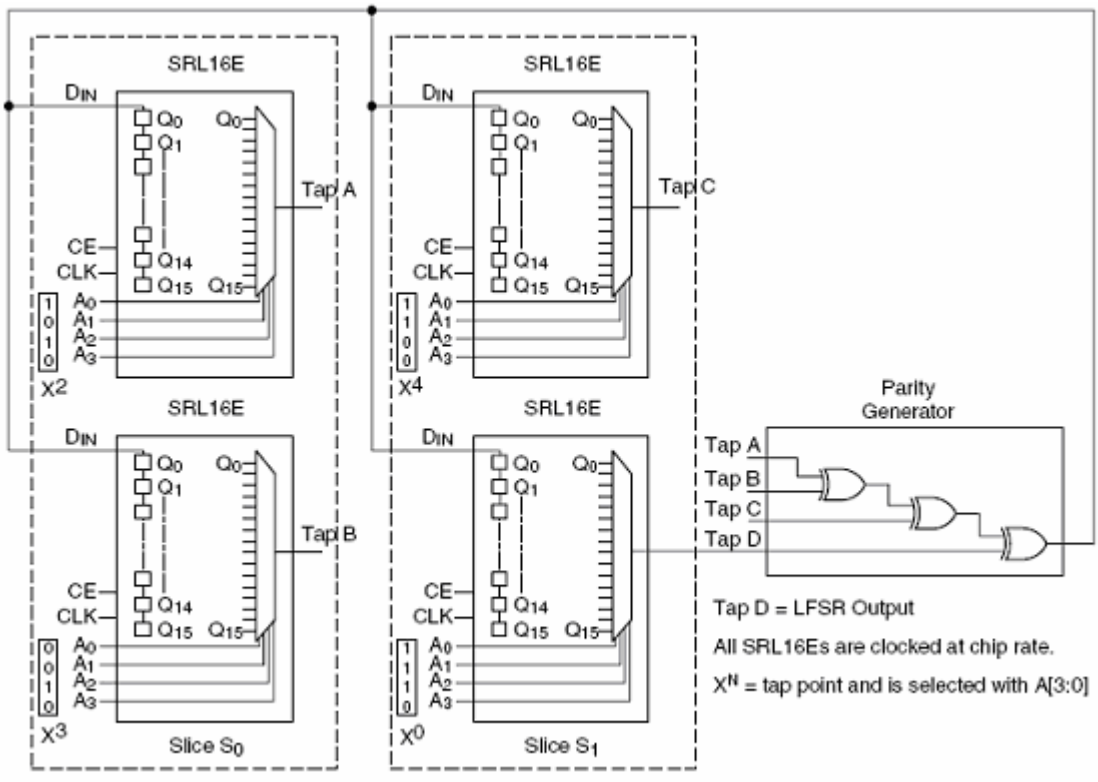


Figure 3.4 Four Tap Parallel LFSR

3.4.2 SINGLE SHIFT REGISTER WITH MULTICYCLE TAP ACCESS

(Figure 3.5) demonstrates how a single SRL16E primitive, with some additional logic, implements a 16-stage shift register that is clocked at a frequency called the *chip rate*. The SRL16 primitive address lines are multiplexed at four times the chip rate allowing four of the 16 shift register taps to be accessed during one chip rate period. The status of each accessed tap is input to a single XOR gate whose output is registered by a flip-flop also clocked at four times the chip rate. During one chip rate period, four taps are read and sequentially XORed to create the parity calculation. The final XOR state is available to the input of the shift register at the chip rate. This circuit enables any four of the sixteen shift register taps to be read, XORed together, and presented to the input of the shift register at the chip rate. Accessing the shift register taps over multiple cycles enables parallel access to four of the shift register taps at the chip rate. The multicycle clock rate should be twice the chip rate if only two taps are required. To access an odd number of taps during one chip rate period, the multicycle clock can be a binary-power-of-two multiple of the chip rate.

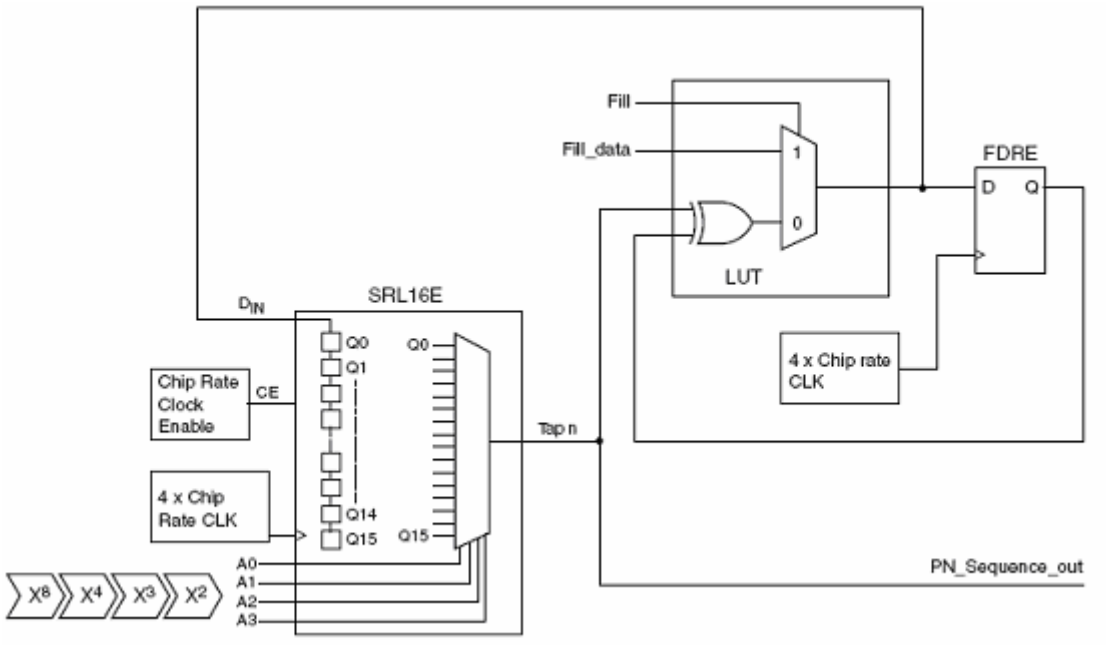


Figure 3.5 16 Stage Shift Register

Multi-cycle clock frequency = chip rate $x2^N$ (where N is an integer)

The FDRE flip-flop can be clock enabled for only as many clock cycles as there are taps to be accessed during the chip rate period. For example, to implement a polynomial with three feedback taps, the FDRE could be clocked with a 4□chip rate clock, and only clock enabled for three of the cycles.

• One SRL16 implements 16-stage shift register

• Selected taps are multiplexed to the output at 4xshift register clock rate

• LUT-based XOR implements a time-shared parity generator

3.4.3 FILL STATE

The *fill state* is defined as that point in a maximal length sequence at which the LFSR will start generating the subsequent states of that sequence. The fill is required to be completed in one cycle of the chip rate. With a parallel shift register, this requirement is easy to satisfy. It requires that each flip-flop in the chain be preceded by a multiplexer that can select between loading parallel FILL data or the shift-out data from the previous flip-flop. To implement an LFSR with this capability using flip-flops requires a 2:1 multiplexer at the input of every flip-flop. This means every shift register stage requires a LUT and flip-flop pair.The SRL16 primitive is not a parallel load shift register, but a solution to the parallel load problem comes through observing exactly what is happening during the last *N* chip periods of an *N*-stage LFSR prior to the FILL transition. Consider an eight-stage LFSR that has just reached a condition where it is eight chip rate clock cycles away from a FILL transition. During these last eight clock cycles, the contents of the shift register are shifted out as the last eight states of the sequence prior to the FILL signal. Also during this eight-cycle period, the XOR feedback path will be generating eight new bits to inject into the shift register as the next eight bits of the sequence.

However, the eight feedback bits calculated during the last eight cycles of the current sequence will not be shifted out as part of the sequence because they will be overwritten by the FILL bits. So rather than shift these eight feedback bits into the shift register, the eight clock cycles preceding the FILL command can be used to serially shift in the eight bits of new FILL data. This enables the SRL16 primitive to be used in LFSR applications where the LFSR has to be parallel loaded in one cycle of the chip rate clock. Note that this implementation is only applicable to instances where an occurrence of the FILL

command can be predicted. Implementing the serial load is achieved with a 2:1 multiplexer that routes either feedback data or new FILL data into the shift register at the chip rate. The multiplexer select line should be pulled high *N* chip rate clock cycles before the last sequence [16, 17].

CHAPTER-4
VHDL IMPLEMENTATION OF LFSR'S

4.1 INTRODUCTION

The heart of the PN generator is the LFSR. LFSRs sequence through $(2^N - 1)$ states, where N is the number of registers in the LFSR. The contents of the registers are shifted right by one position at each clock cycle. The feedback from predefined registers or taps to the left most register are XOR-ed together.

LFSRs have several variables:

- The number of stages in the shift register.
- The number of taps in the feedback path.
- The position of each tap in the shift register stage.
- The initial starting condition of the shift register often referred to as the "FILL" state

4.2 VHDL IMPLEMENTATION OF LFSR'S

The linear feedback shift register is made up of two parts: a shift register and a feedback function. The shift register is initialized with n bits (called the key), and each time a keystream bit is required, all of the bits in the register are shifted 1 bit to the right. So the least significant bit is the output bit. The new left-most bit is computed as the XOR of certain bits in the register. This arrangement can potentially produce a 2^n-1 bit-long pseudo-random sequence (referred to as the period) before repeating.

Following is the block diagram and simulation waveform of 8-bit LFSR and 13 bit LFSR's and than there is LFSR's with single tap and multicycle tap sequences.

4.2.1 8-BIT LFSR

The following example shows the implementation of 8-bit LFSR having bit 8, bit 4, bit 3, and bit 2 as the parity bits. So the tap sequence will look like;

$$x^8+x^4+x^3+x^2+1$$

This shift register is initialized with a key. Every clock cycle, all of the bits in the register are shifted such that the least significant bit is output. The new most significant

bit is computed from the XOR of certain bits in the register. This shift register is rising edge triggered with an active-high enable and active-high reset.

4.2.1.1 BLOCK DIAGRAM

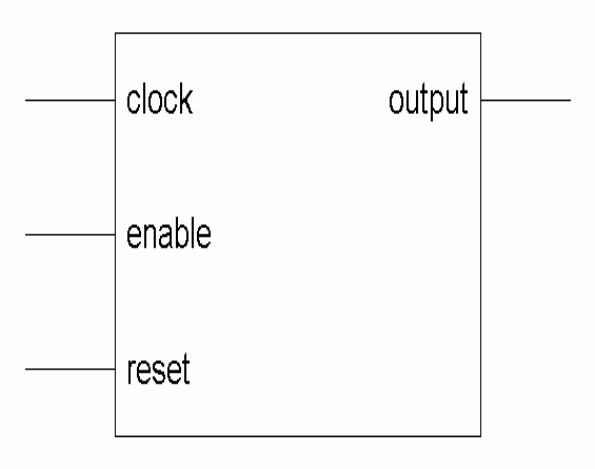


Figure 4.1 Block Diagram of 8-Bit LFSR

(Figure 4.1) shows the block diagram of 8-bit LFSR .Here clock is set all the time .Enable bit is '1' to enable the LFSR otherwise the chip is not enabled. Reset pin when '1' cause the output of LFSR to be in the previous state and when '0' cause the output of the LFSR to create the required pattern.

4.2.1.2 SCHEMATIC DIAGRAM

(Figure 4.2) shows the schematic of 8-bit LFSR. The diagram shows that from ex-or gates the required parity bits are fed back to the input. Here each block in (Figure 4.2(b)) contains the cell as shown in (Figure 4.1) .Thus it will have 8 cells as shown in (Figure 4.2(b)).

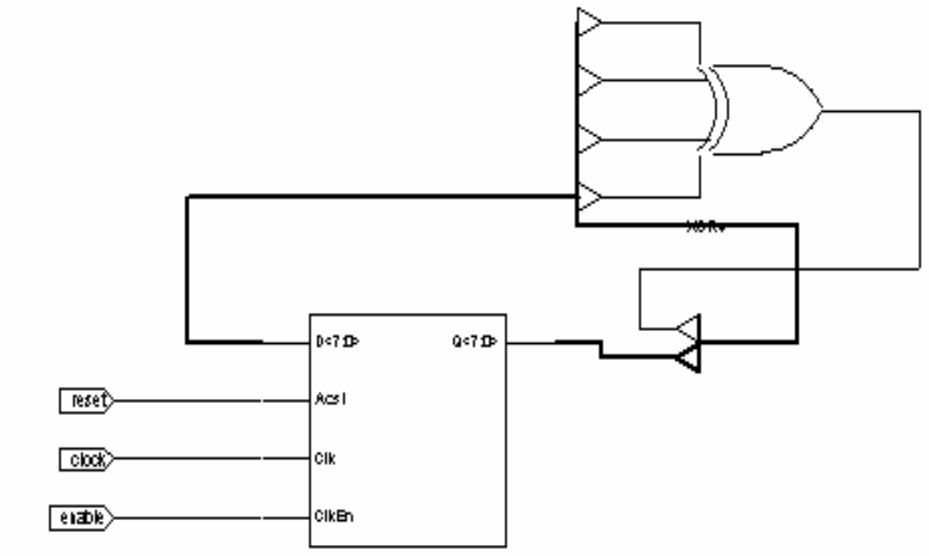


Figure 4.2(a) Circuit Diagram of 8-Bit LFSR

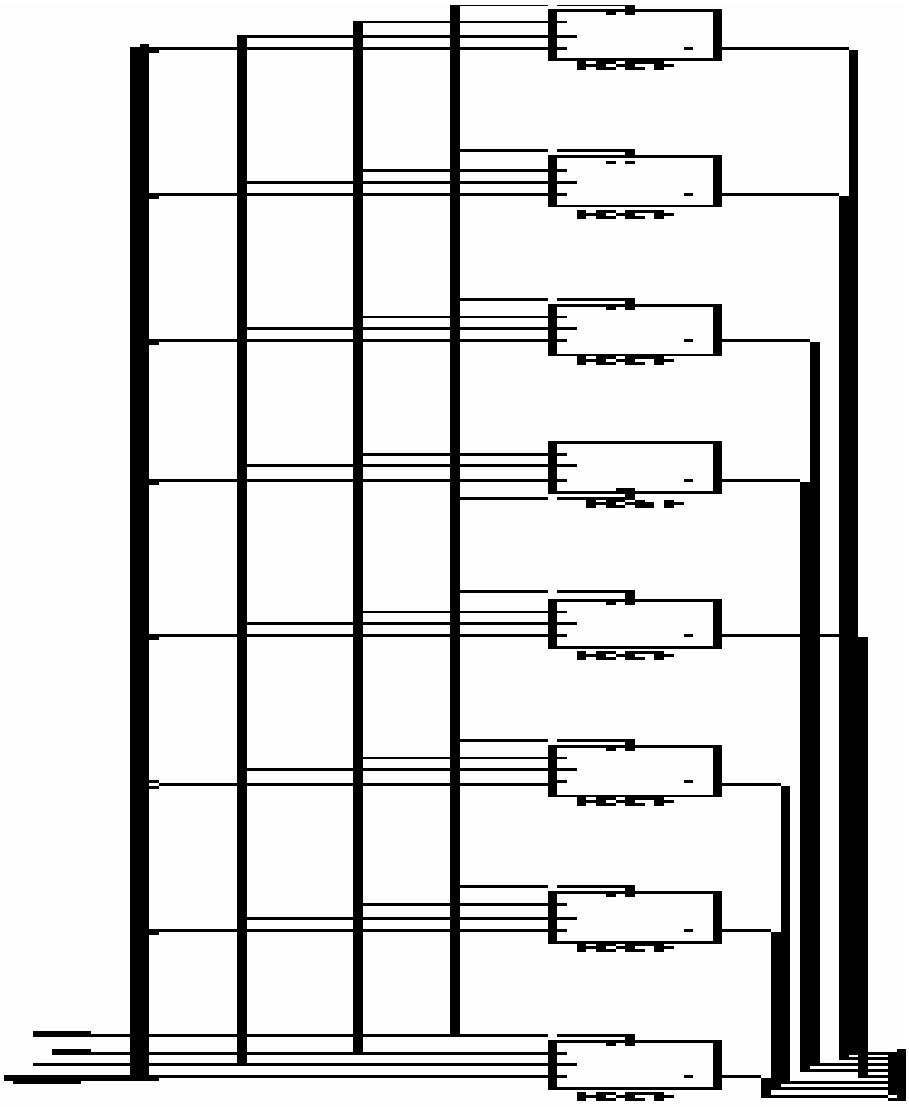


Figure 4.2(b) Schematic of 8-Bit LFSR

4.2.1.3 SIMULATION RESULTS

(Figure 4.3 (a)) shows the simulation results of an 8-bit LFSR .Here when reset is '1' the initial state will appear on both internal register and internal_reg_next which will show the next bits to be appear on the internal register. However when reset is "0" the LFSR starts doing generation of unique bit patterns which is clearly shown in (Appendix A.1).

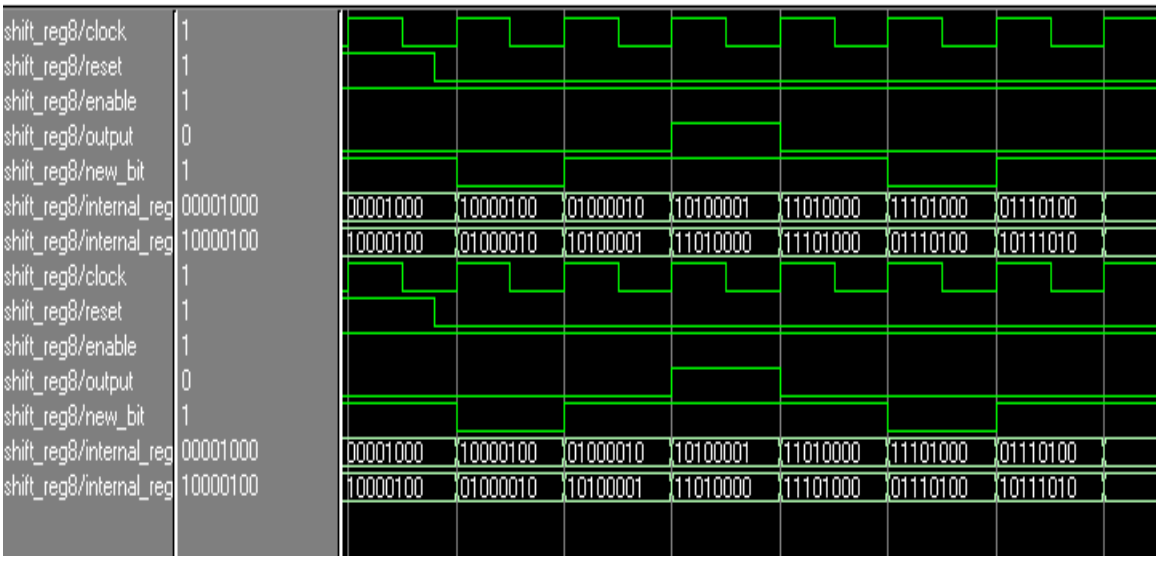


Figure 4.3 Simulation Waveforms of 8-Bit Register

4.2.1.4 DEVICE UTILISATION SUMMARY

Table 4.1 - Device Utilization by 8-bit LFSR

Selected Device	xqvr300cb228-4
Number of Slices	5
Number of Slice Flip Flops	8
Number of 4 input LUTs	1
Number of bonded IOBs	3
Number of GCLKs	1

Table 4.2 – Number of registers and logic gates in 8-bit LFSR

Registers	8
1-bit register	8
Xors	1

1-bit xor4	1

Table 4.3 – Cell Usage by 8-bit LFSR

BELS	1
LUT4_L	1
FlipFlops/Latches	8
FDCE	7
FDPE	1
Clock Buffers	1
BUFGP	1
IO Buffers	3
IBUF	2
OBUF	1

4.2.1.5 TIMING DETAIL SUMMARY

Table 4.4 – Timing Summary for 8-bit LFSR

Speed Grade	-4
Minimum period	4.349ns (Maximum Frequency: 229.938MHz)
Minimum input arrival time before clock	4.192ns
Maximum output required time after clock	8.289ns

4.2.2 13-BIT LFSR

The following example shows the implementation of 8-bit LFSR having bit 13,bit 4,bit 3 and bit 1 as the parity bits. So the tap sequence will look like;

$$x^{13}+x^{4}+x^{3}+x^{1}+1$$

This shift register is initialized with a key. Every clock cycle, all of the bits in the register are shifted such that the least significant bit is output. The new most significant bit is computed from the XOR of certain bits in the register. This shift register is rising edge triggered with an active-high enable and active-high reset**.**

4.2.2.1 CIRCUIT DIAGRAM

(Figure 4.4) shows the block diagram of 13-bit LFSR .Here clock is set all the time .Enable bit is '1' to enable the LFSR otherwise the chip is not enabled. Reset pin when

‘1’ cause the output of LFSR to be in the previous state and when ‘0’ cause the output of the LFSR to create the required pattern. Schematic diagram is shown in (Figure 4.5).

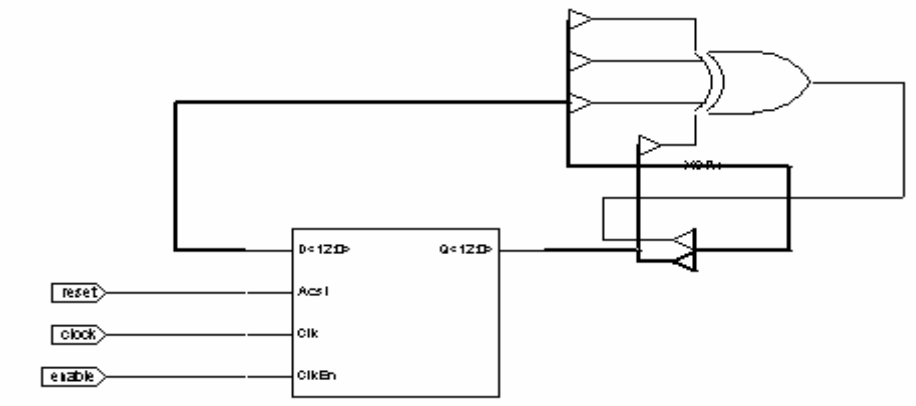


Figure 4.4 Circuit Diagram for 13-bit LFSR

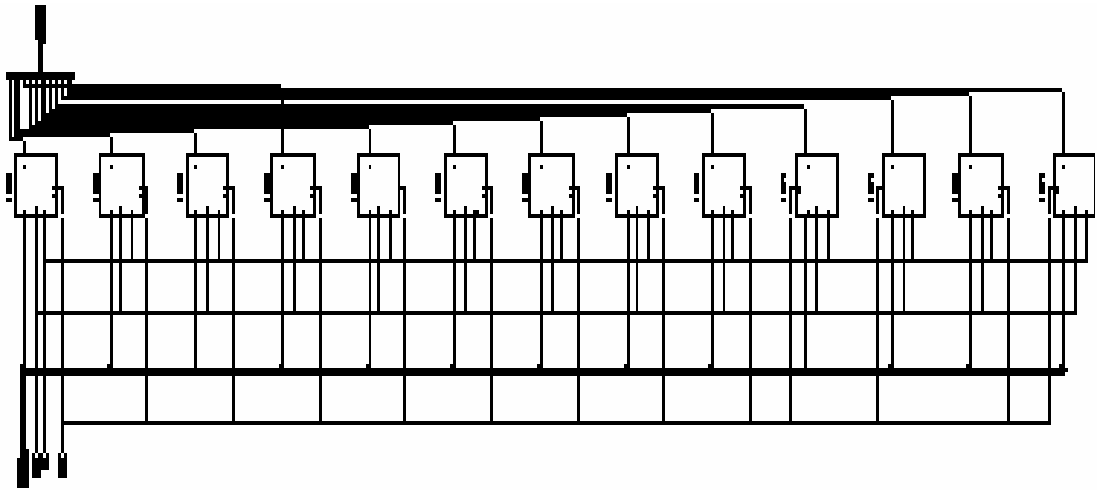


Figure 4.5 Schematic for 13-Bit LFSR

4.2.2.2 SIMULATION RESULTS

(Figure 4.6(a)) and (Figure 4.6(b)) shows the simulation results for 13-bit LFSR whose working is same as that of 8-bit LFSR that is when reset is ‘1’ initial state will appear on internal register otherwise it will generate the unique bit pattern depending upon the taps.

Here the output bit depends upon the least significant bit of the shift register. (Appendix A.2) shows the bit pattern.

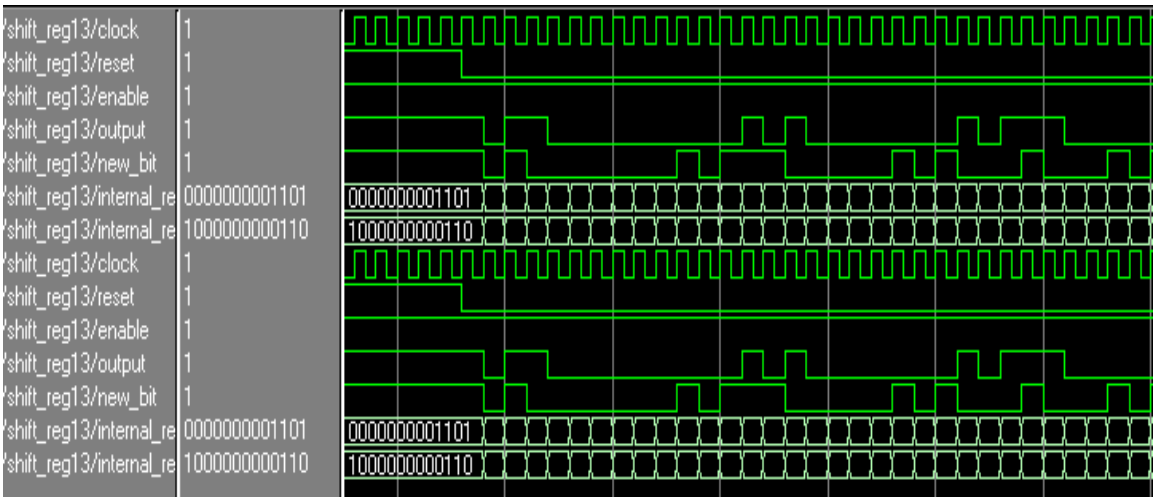


Figure 4.6(a) Simulation Waveforms of 13-bit LFSR

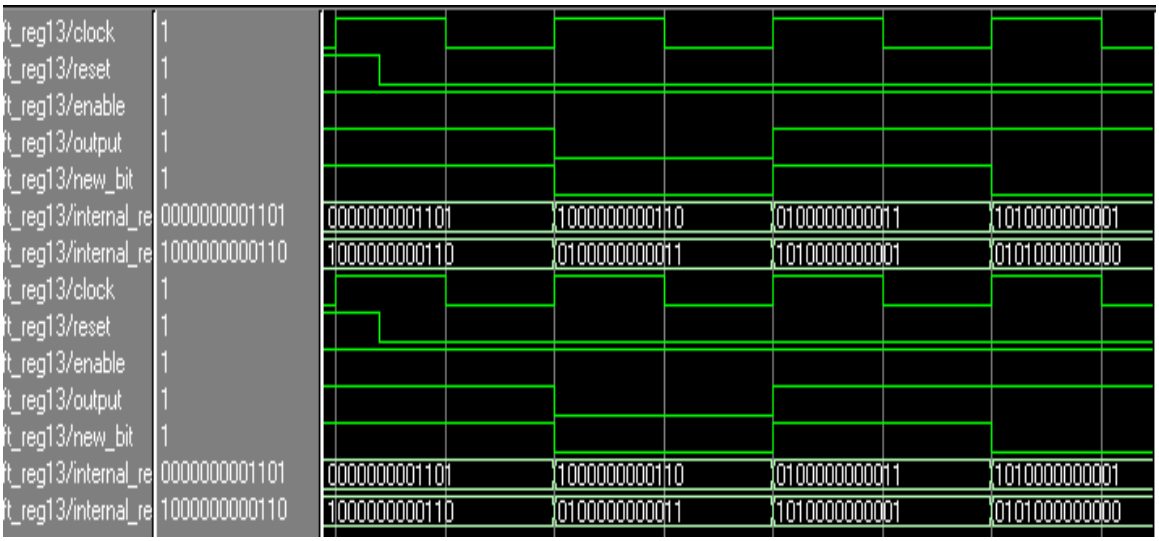


Figure 4.6(b) Simulation Waveforms of 13-bit LFSR (with pattern)

4.2.2.3 DEVICE UTILIZATION SUMMARY

Table 4.5 - Device Utilization by 13-bit LFSR

Selected Device	-4
Number of Slices	7
Number of Slice Flip Flops	13
Number of 4 input LUTs	1
Number of bonded IOBs	3
Number of GCLKs	1

Table 4.6 – Number of registers and logic gates in 13-bit LFSR

Registers	1
13-bit register	1
Xors	1
1-bit xor4	1

Table 4.7 – Cell Usage by 13-bit LFSR

BELS	1
LUT4_L	1
FlipFlops/Latches	13
FDCE	10
FDPE	3
Clock Buffers	1
BUFGP	1
IO Buffers	3
IBUF	2
OBUF	1

4.2.2.4 TIMING DETAILS

Table 4.8 – Timing Summary for 13-bit LFSR

Speed Grade	-4
Minimum period	4.349ns (Maximum Frequency: 229.938MHz)
Minimum input arrival time before clock	4.687ns
Maximum output required time after clock	8.498ns

4.2.3 LFSRS WITH PARALLEL TAP ACCESS AND PARITY CALCULATION

One method of creating an LFSR is creating multiple parallel SRL16s for each tap. The different outputs are then XNORed simultaneously using a single LUT and the output is then feedback into each SRL16. It is important to note that this code infers the SRL16 primitives, and thus is portable to any architecture. Because these SRL16s are inferred, they cannot be initialized to a known state on power-up (other than all zeros), nor can the shift registers be dynamically changed to a different length by altering the address lines. In this design, a single bit 2:1 multiplexer is inserted in order to enable the user to shift in the initial sequence.

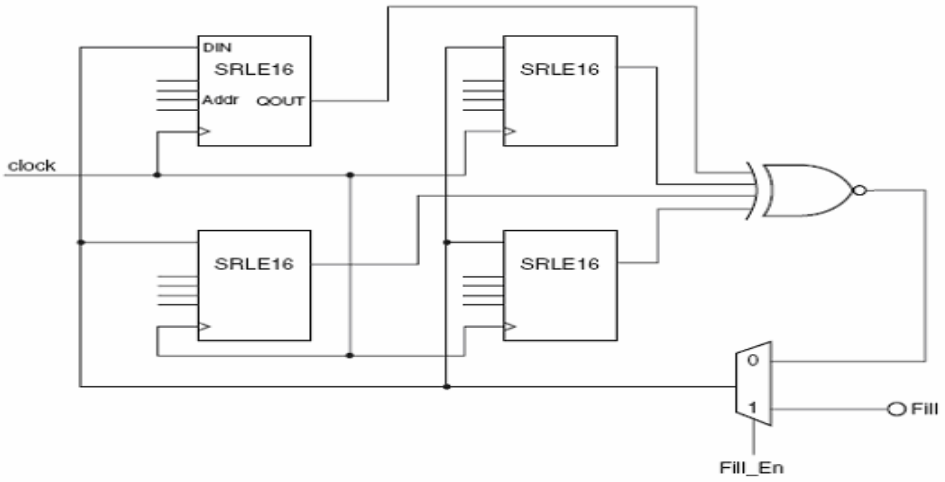


Figure 4.7 16-bit, 4 tap parallel LFSR

This implementation can be cascaded in order to implement larger designs. It is important to note that a LFSR which is more than twice as long will not necessarily use twice as many SRL16s. For example, a 32-bit LFSR which has bus taps on bits 32, 22, 2, and 1 will not use eight SRL16s even though the 16-bit implementation uses four as shown in (Figure 4.7). In the 32-bit LFSR it will only use five SRL16s. Likewise a 64-bit LFSR will not use ten SRL16s, it will only use seven. Thus the marginal cost of using SRL16s to implement LFSRs decreases with the size of the LFSR and the location of the taps.

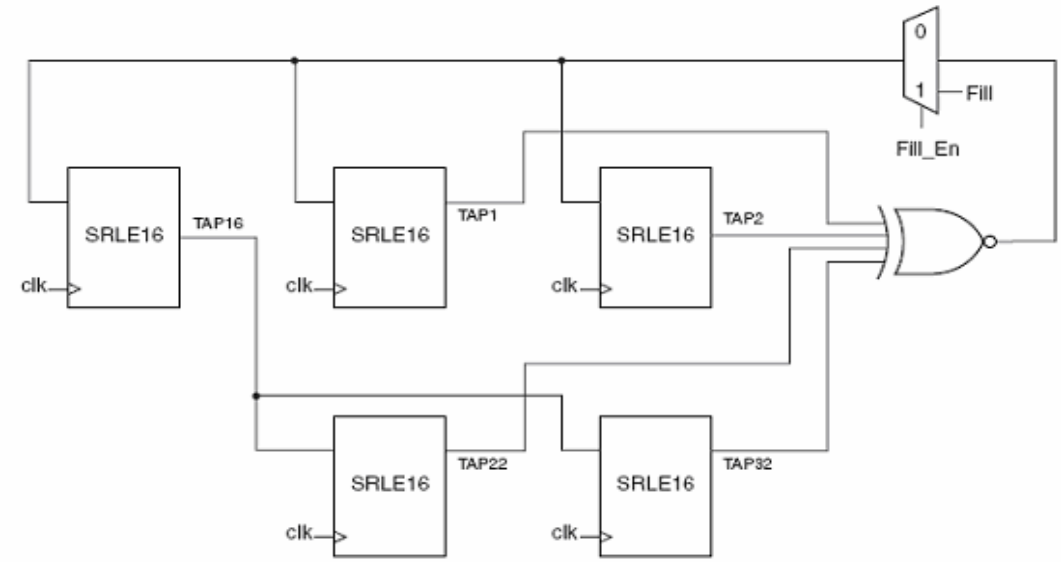


Figure 4.8 32-Bit, 4-Tap Parallel LFSR

In this thesis I will implement 16-bit, 4 tap parallel LFSR as shown in Figure 4.8 which is implemented by using Xilinx Project Navigator. Figures below shows its block diagram, circuit diagrams and simulation results.

4.2.3.1 BLOCK DIAGRAM

(Figure 4.9) shows the basic cell of srl16E look up table for 16-bit, 4 tap parallel LFSR. Here ‘CE’ is chip enable pin which when ‘1’ makes the chip enable otherwise chip is not enabled. ‘DIN’ is the input of first flip-flop which will be ‘1’ or ‘0’ depending on the fill state.

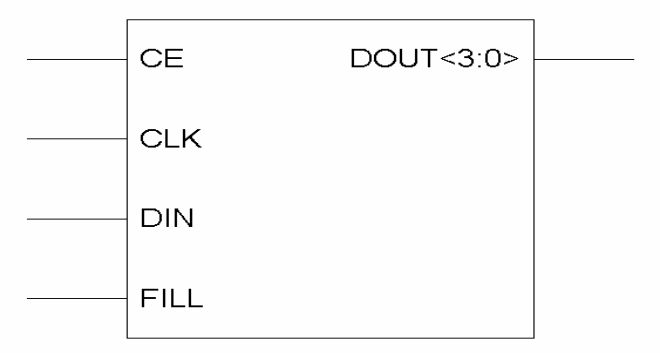


Figure 4.9 Block Diagram for 16-Bit, 4 Tap Parallel LFSR

4.2.3.2 SCHEMATIC DIAGRAM

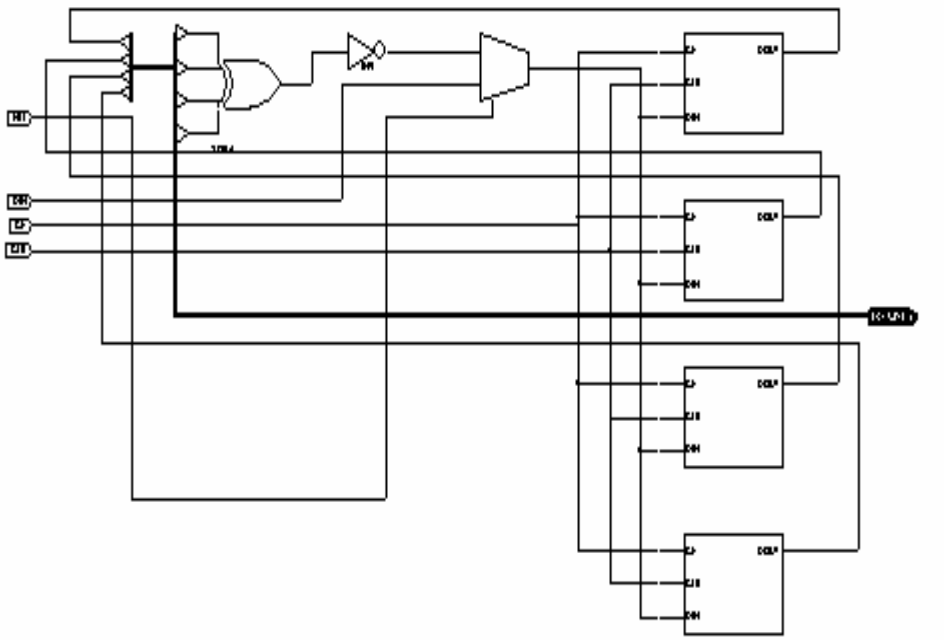


Figure 4.10 Schematic for 16-Bit, 4 Tap Parallel LFSR

(Figure 4.10) shows the schematic for 16-Bit, 4 Tap Parallel LFSR which use srl16e block as shown in figure 4.9 for each tap. The different taps are used in (Figure 4.10) which are shown below.

Here bit 14, 13, 11 and 0 are used as four taps whose blocks are shown as below;

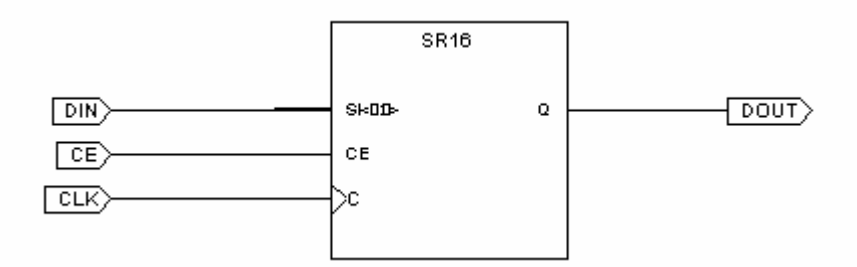


Figure 4.10(a) Register 0 as Tap U4

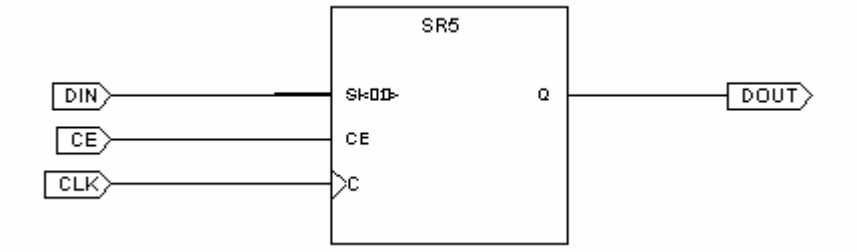


Figure 4.10(b) Register 11 as Tap U3

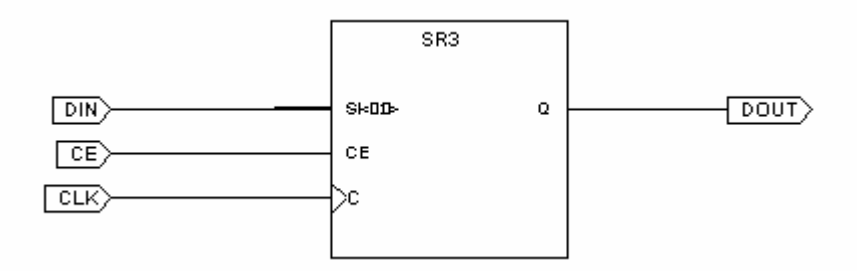


Figure 4.10(c) Register 13 as Tap U2

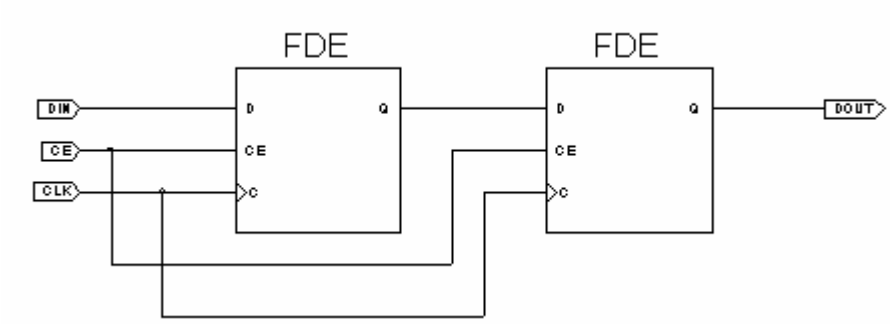


Figure 4.10(d) Register 14 as Tap U1 with Output Register 15

4.2.3.3 SIMULATION RESULTS

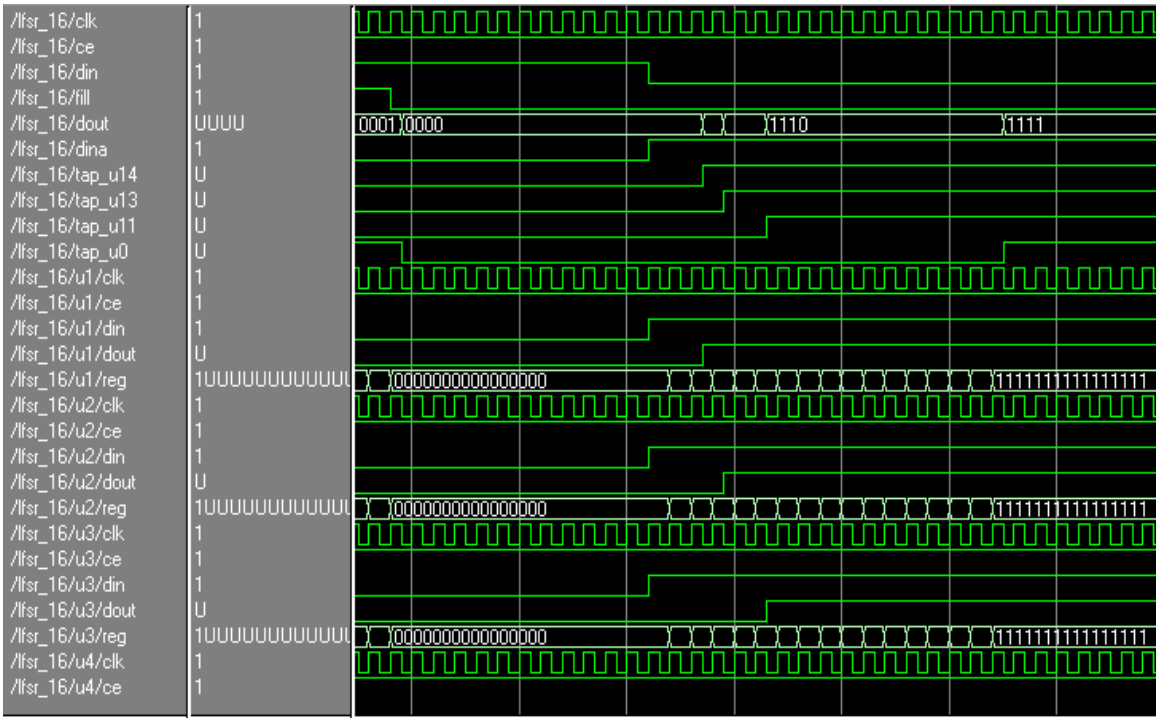


Figure 4.11(a) Simulation Waveforms for 16-Bit, 4 Tap Parallel LFSR

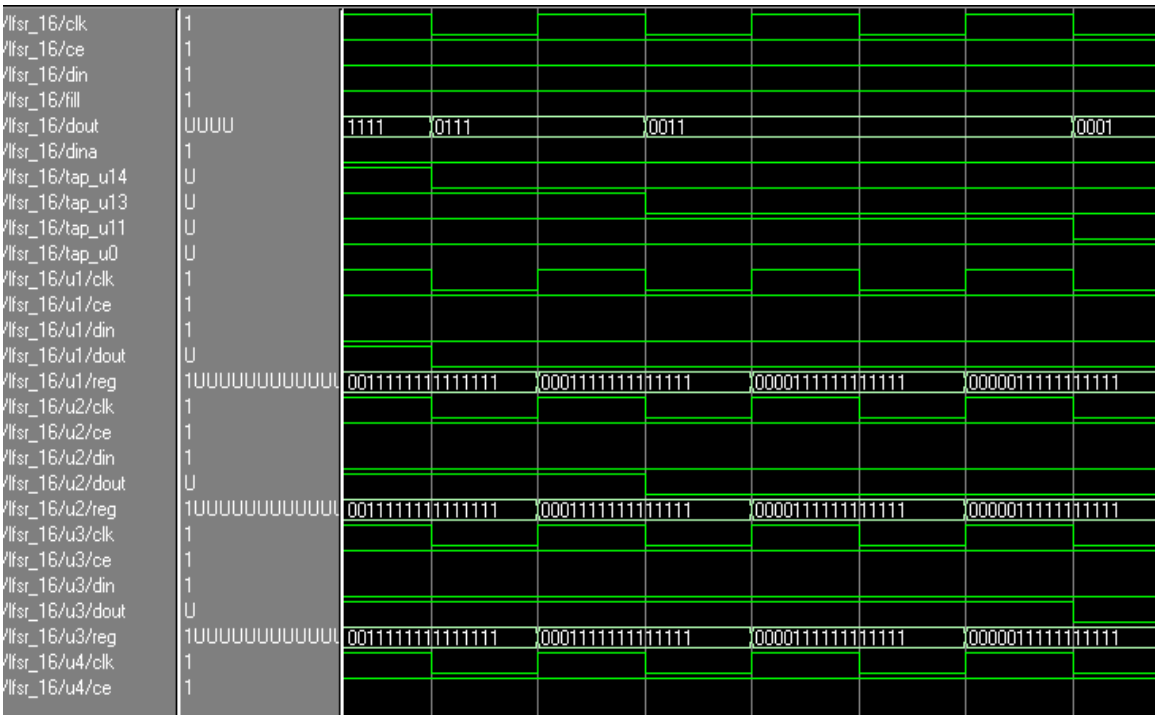


Figure 4.12(b) Simulation Waveform for 16-Bit, 4 Tap Parallel LFSR(with pattern)

4.2.3.4 DEVICE UTILIZATION SUMMARY

Table 4.9 - Device Utilization by 16-bit, 4 tap parallel LFSR

Selected Device	xqvr300cb228-4
Number of Slices	3
Number of Slice Flip Flops	5
Number of 4 input LUTs	5
Number of bonded IOBs	7
Number of GCLKs	1

Table 4.10 – Number of registers and logic gates in 16-bit, 4 tap parallel LFSR

Registers	2
1-bit register	2
Shift Registers	3
3-bit shift register	1
5-bit shift register	1
16-bit shift register	1

Multiplexers	1
1-bit 2-to-1 multiplexer	1
Xors	1
1-bit xor4	1

Table 4.11– Cell Usage by 16-bit, 4 tap parallel LFSR

BELS	4
GND	1
LUT3_L	1
LUT4_D	1
VCC	1
FlipFlops/Latches	5
FDE	5
Shifters	3
SRL16E	3
Clock Buffers	1
BUFGP	1
IO Buffers	7
IBUF	3
OBUF	4

4.2.3.5 TIMING DETAILS

Table 4.12 – Timing Summary for 16-bit, 4 taps parallel LFSR

Speed Grade	-4
Minimum period	6.549ns (Maximum Frequency: 152.695MHz)
Minimum input arrival time before clock	4.215ns
Maximum output required time after clock	8.498ns

4.2.4 LFSRS WITH MULTI-CYCLE TAP ACCESS AND SEQUENTIAL PARITY CALCULATION

Another method of creating an LFSR uses a single SRL16E, gaining access to the different taps by changing the address lines on the LUT that implements the SRL16E. In this case, the output will be delayed by the number of taps needed to implement it. As shown in (Figure 4.13), the output is produced on every rising edge of the chip rate clock, but the SRL16 and the output flip-flop are actually clocked at an increased clock rate—four times the chip rate for 4-tap LFSRs, and twice the chip rate for 2-tap LFSRs. The individual taps are then XNORed serially using the single output flip-flop. Note that during the last increased chip rate clock cycle, this flip-flop is synchronously reset to prepare it for the next increased clock cycle parity calculation. This method of implementation may use fewer resources in certain extremely long LFSRs; however, for shorter length LFSRs, the parallel LFSR implementation requires fewer resources. In the multicycle implementation, the SRL16E primitives have to be instantiated, since the synthesis tools cannot infer a dynamically changing output on the SRL16E.

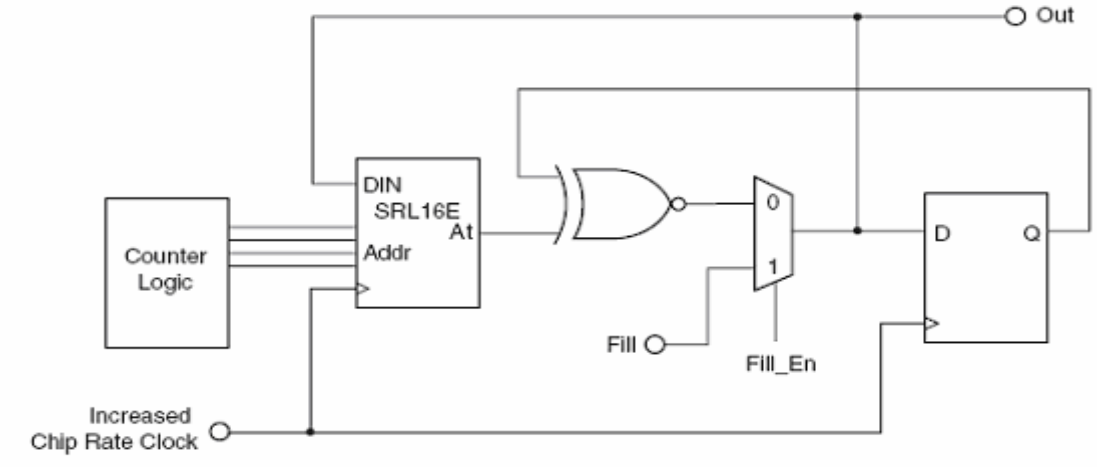


Figure 4.13 Multicycle Tap Access LFSR

4.2.4.1 BLOCK DIAGRAM

(Figure 4.14) shows the block diagram for multicycle tap access lfsr in which clk4x can be used .here we can generate bit pattern by initializing tap values as per requirement as well as they can do same functioning as 16-bit 4-tap LFSR.

Figure 4.14 Block Diagram of Multicycle Tap Access LFSR

4.2.4.2 SCHEMATIC DIAGRAM

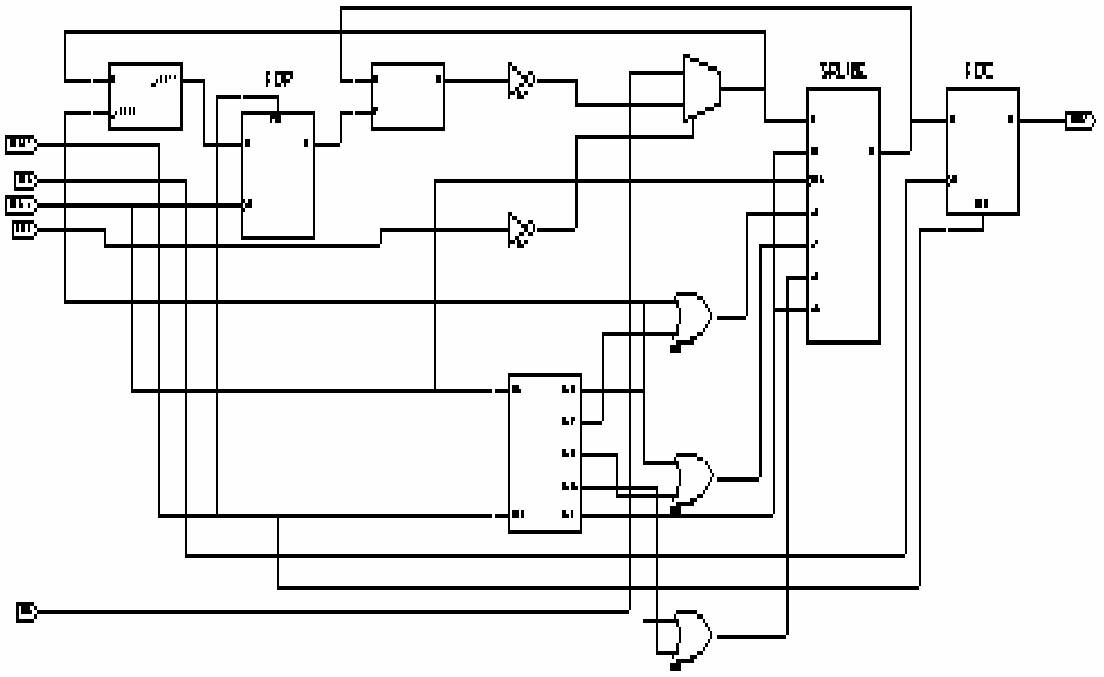


Figure 4.15 Schematic of Multicycle Tap Access LFSR Using Single SRL16E

4.2.4.3 SCHEMATIC FOR ADDRESS REGISTER

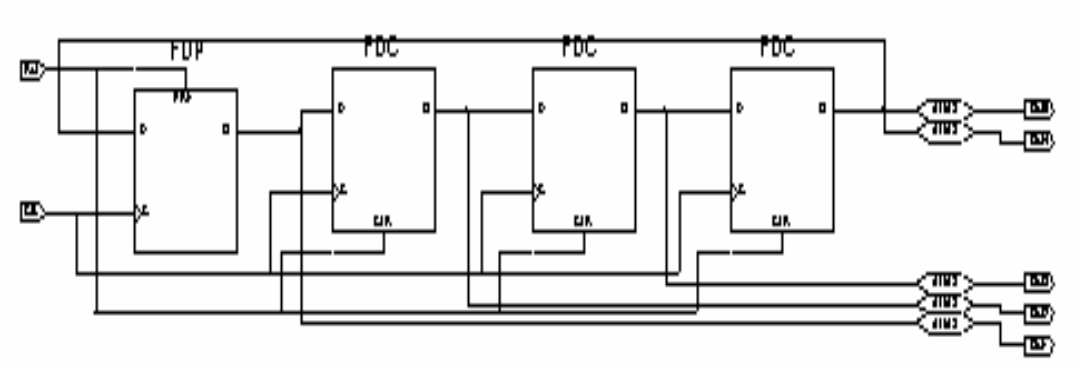


Figure 4.16 Schematic of address register for Multicycle Tap Access LFSR

4.2.4.4 XOR GATES CREATING RESULT

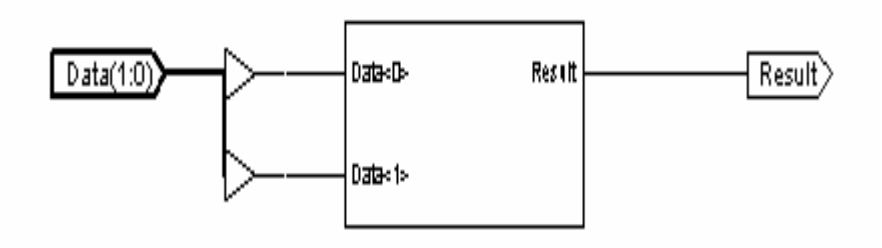


Figure 4.17(a) Block diagram for xor gate

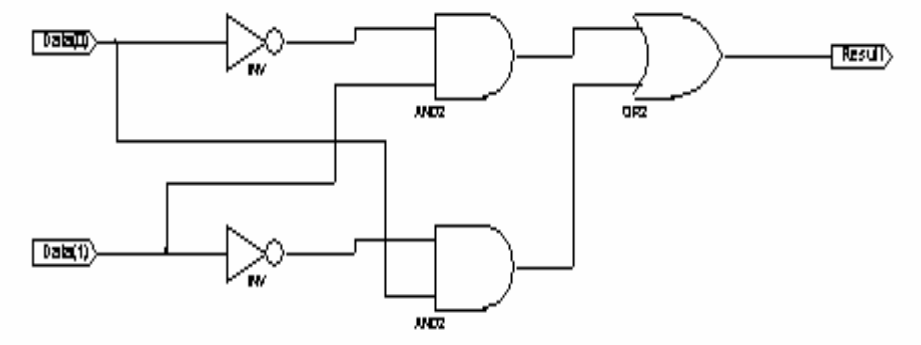


Figure 4.17(b) Schematic for Xor Gate

4.2.4.5 SIMULATION RESULTS

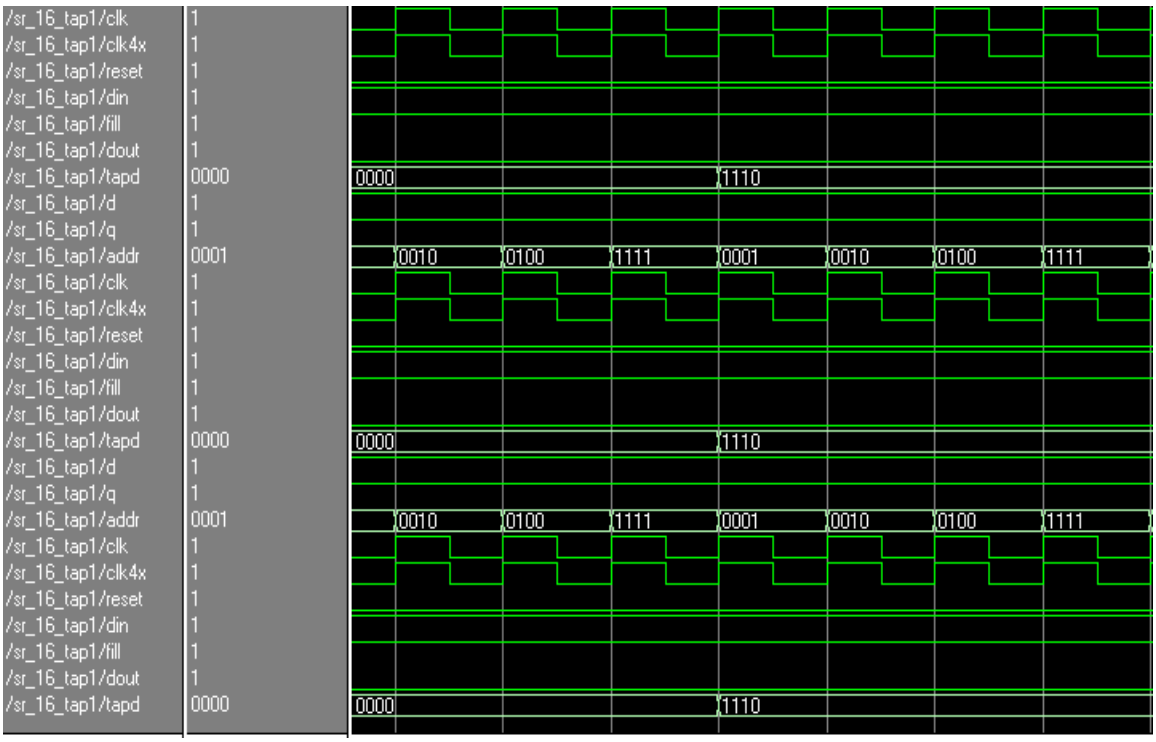


Figure 4.18 Simulation Waveforms for Multicycle Tap Access LFSR

4.2.4.6 DEVICE UTILISATION SUMMARY

Table 4.13 - Device Utilization by Multicycle Tap Access LFSR

Selected Device	xqvr300cb228-4
Number of Slices	4
Number of Slice Flip Flops	6
Number of 4 input LUTs	8
Number of bonded IOBs	4
Number of GCLKs	2

Table 4.14– Number of registers and logic gates in Multicycle Tap Access LFSR

FSMs	1
Registers	6
1-bit register	6
Multiplexers	1

1-bit 2-to-1 multiplexer	1
Xors	1
1-bit xor2	1

Table 4.15– Cell Usage by Multicycle Tap Access LFSR:

BELS	7
LUT2_L	3
LUT3	1
LUT3_L	1
LUT4_L	2
FlipFlops/Latches	6
FDC	4
FDP	2
Shifters	1
SRL16E	1
Clock Buffers	2
BUFGP	2
IO Buffers	4
IBUF	3
OBUF	1

4.2.4.7 TIMING DETAIL SUMMARY

Table 4.16 – Timing Summary for Multicycle Tap Access LFSR

Speed Grade	-4
Minimum period	7.734ns (Maximum Frequency: 129.299MHz)
Minimum input arrival time before clock	6.123ns
Maximum output required time after clock	8.289ns

CHAPTER-5
DESIGN APPLICATIONS OF WIRELESS COMMUNICATION SYSTEM USING LFSR

5.1 INTRODUCTION

A Linear Feedback Shift Register is a sequential shift register with combinational logic that causes it to pseudo-randomly cycle through a sequence of binary values. Linear feedback shift registers have multiple uses in wireless communication systems using LFSR as a functional block .Applications covered in this thesis include:

1. PN number generation by implementing PN generators with the help of LFSR.
2. RS code generators used for random number generation using LFSR.
3. Implementation of BIST with the help of LFSR for testing purposes

5.2PN GENERATOR IMPLEMENTATION

5.2.1 INTRODUCTION

Pseudo-random Noise (PN) generators are at the heart of every spread spectrum system. Many PN generators are required within Code Division Multiple Access (CDMA) base stations. PN generators are used to implement synchronization and uniquely code individual user signals across the transmission interface. PN generators are based upon Linear Feedback Shift Registers (LFSRs). Code Division Multiple Access (CDMA) systems are based upon several forms of spread spectrum techniques, the most popular being Direct Sequence Spread Spectrum (DS-SS).Within a DS-SS system, the data being transmitted is spread across a wide radio spectrum using a pseudo random binary sequence unique to each user. Every data bit of a user signal is multiplied by many bits of a pseudo random binary sequence. This sequence is created by a PN generator and often referred to as a PN-Code. The PN-Codes used within CDMA system posses mathematical properties that enable them to coexist in the same spectrum with minimal interference. It is these properties that enable multiple users to exist in the same radio spectrum and hence leads to the term Multiple Access in CDMA [18, 19].

5.2.2 PN GENERATOR DESIGN

A Pseudo-random Noise (PN) sequence/code is a binary sequence that exhibits randomness properties but has a finite length and is therefore deterministic. There are three uses for PN sequences in DS-SS applications:

1. Spreading the bandwidth of the modulated signal over a wide radio spectrum.

2. Uniquely coding the different user signals that occupy the same transmission bandwidth in a multi-access system.

3. Synchronization for W-CDMA systems where there is no global timing reference.

In order to achieve these objectives, the coding sequences require special correlation properties referred to as auto correlation, and cross correlation[20].

5.2.2.1 AUTO CORRELATION

Auto correlation is a measure of how well a signal *f* (t) can differentiate between itself and every time-shifted variant of itself. Consider a data word of period seven bits at time δt = 0 (Table 5.1). If the 7-bit code were to be repeating within a discrete system then there are only six time-shifted replicas of the word, (shown in Table 1 for time δt = 1 to δt = 6). If each bit of the original (δt = 0), is compared with each bit of every time-shifted replica, then there are a number of agreements (A), and disagreements (D), that when subtracted provide a measure of how closely the two words match (correlate).In (Table 5.1), the sequence "1110010" has a good auto correlation property as it provides a clear difference in the correlation value between itself and any time-shifted variant of itself. In (Table 5.2), the sequence "1111000" has the same number of bits, but the auto correlation property is not as good as there are some clear rejections of a match (correlation value = −5),and there are some "fuzzy" conditions where the time-shifted replica almost matches,(correlation value = 3).

Table 5.1- autocorrelation example

Sequence	Time Shift	(A)	(D)	(A–D)
1110010	δt = 0	7	0	7
0111001	δt = 1	3	4	−1
1011100	δt = 2	3	4	−1
0101110	δt = 3	3	4	−1
0010111	δt = 4	3	4	−1
1001011	δt = 5	3	4	−1
1100101	δt = 6	3	4	−1

Table 5.2- autocorrelation example

Sequence	Time Shift	(A)	(D)	(A–D)
1111000	δt = 0	7	0	7
0111100	δt = 1	5	2	3
0011110	δt = 2	3	4	–1
0001111	δt = 3	1	6	–5
1000111	δt = 4	1	6	–5
1100011	δt = 5	3	4	–1
1110001	δt = 6	5	2	3

5.2.2.2 CROSS CORRELATION

Cross correlation is defined as the correlation between two different signals. Cross correlation is also calculated by subtracting the disagreements from the agreements, between two different sequences as opposed to the time-shifted replicas of the same signal [20, 21].

5.2.2.3 UNIQUELY CODING THE DIFFERENT USER SIGNALS

In a CDMA system, each one of the multiple user signals in the receiver is assigned a unique PN code that behaves like a "key". From the examples in (Table 5.1) and (Table 5.2) it is evident that some sequences of the same length have better auto correlation properties than others and these special PN sequences are the ones used to code user signals in the system. It is important to use a set of PN sequences that have a small cross correlation between each other in order to reduce an effect called adjacent channel interference. If the cross correlation between two PN sequences or "keys" is not small, there is a possibility that data coded from one user is incorrectly identified and assigned to another user because the two keys had a reasonable correlation. Research on small cross correlation PN sequences, have been carried out by many individuals but code sets identified by Kasami, R. Gold and Walsh are used throughout the IS-95 and UMTS W-CDMA systems [22,23,24].

5.2.3 VHDL IMPLEMENTATION FOR PN GENERATORS

The PN Generator provides two spreading sequences for the "I" (In-Phase) and "Q" (Quadrature phase) channels used in Quadrature Phase Shift Keying (QPSK) modulation schemes. The PN Generator HDL code therefore implements two LFSRs, one for the "I" channel and one for the "Q" channel. The following is example code that implements two

LFSRs which can be used as part of pn generators.The number of taps, tap points, and LFSR width are parameratizable. When targetting Xilinx (Virtex) all the latest synthesis vendors (Leonardo, Synplicity, and FPGA Express) will infer the shift register LUTS (SRL16) resulting in a very efficient implementation [23].

I and Q Polinomials:

$$I(x) = X^{**}17 + X^{**}5 + 1$$

$$Q(x) = X^{**}17 + X^{**}9 + X^{**}5 + X^{**}4 + 1$$

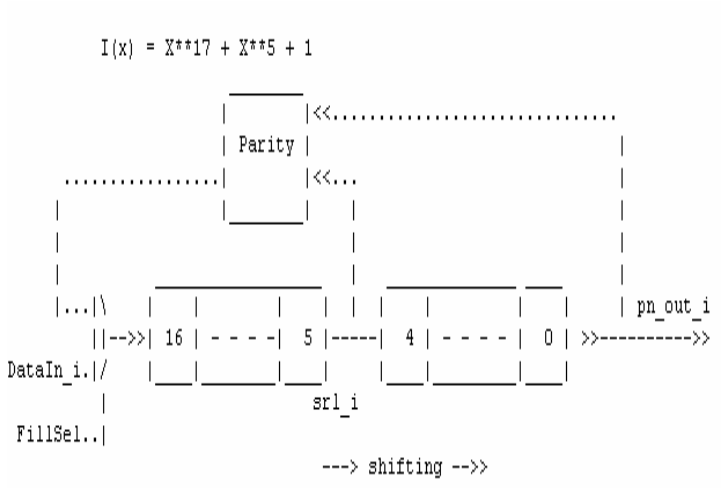


Figure 5.1 I(x) = X**17 + X**5 + 1

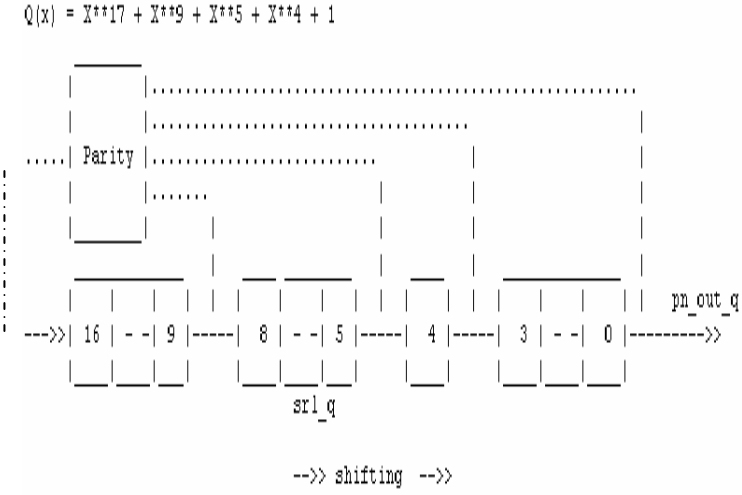


Figure5.2 Q(x) = X**17 + X**9 + X**5 + X**4 +1

5.2.3.1 BLOCK DIAGRAM

(Figure 5.3) shows the block diagram of PN generator which contains two inputs from 'I' and 'Q' polynomials and thus generates same bit patterns on both 'I'and 'Q' outputs. Here we take different taps for both inputs to show that it does not affect the outputs (inspite of delay to the output bit pattern).

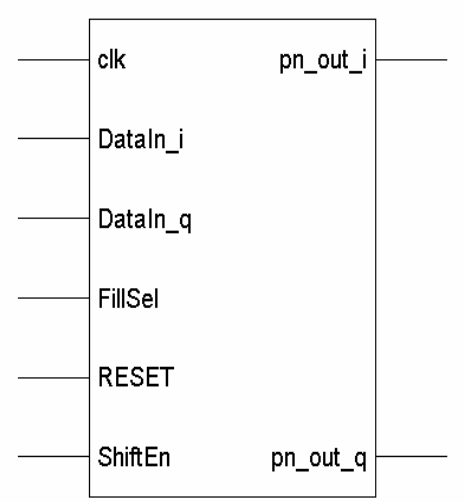


Figure 5.3 Block Diagram of PN Generator

5.2.3.2 TOP LEVEL SCHEMATIC OF PN GENERATOR

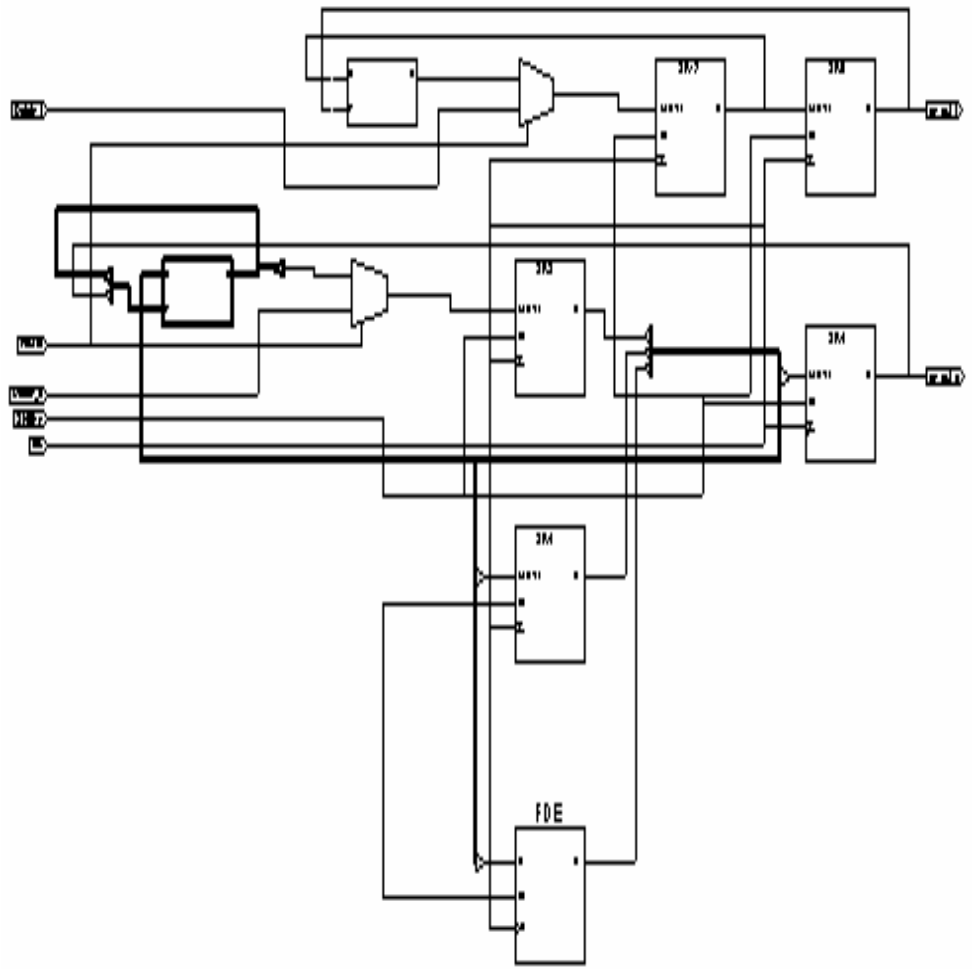


Figure 5.4 Schematic of PN Generator

5.2.3.3 SIMULATION RESULTS

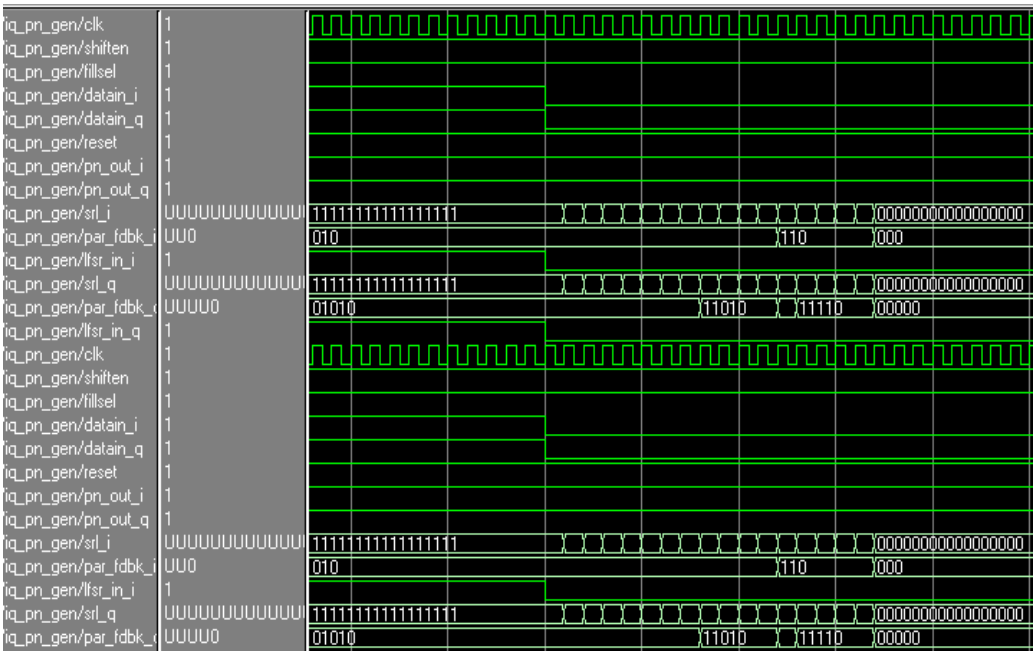


Figure 5.5 Simulation waveforms of PN generator

5.2.3.4 DEVICE UTILIZATION DETAIL

Table 5.3 - Device Utilization by PN Generator

Selected Device	xqvr300cb228-4
Number of Slices	5
Number of Slice Flip Flops	6
Number of 4 input LUTs	8
Number of bonded IOBs	6
Number of GCLKs	1

Table 5.4 – Number of registers and logic gates in PN generator

Registers	1
1-bit register	1
Shift Registers	5
5-bit shift register	1
12-bit shift register	1
4-bit shift register	2
8-bit shift register	1
Multiplexers	2
1-bit 2-to-1 multiplexer	2
Xors	4
1-bit xor2	4

Table 5.5 – Cell Usage by PN Generator

BELS	5
GND	1
LUT3_L	1
LUT4	2
VCC	1
FlipFlops/Latches	6
FDE	6
Shifters	5
SRL16E	5
Clock Buffers	1
BUFGP	1
IO Buffers	6
IBUF	4
OBUF	2

5.2.3.5 TIMING DETAILS

Table 5.6– Timing Summary for PN Generator

Speed Grade	-4
Minimum period	6.549ns (Maximum Frequency: 152.695MHz)
Minimum input arrival time before clock	4.490ns
Maximum output required time after clock	8.498ns

5.3 RS CODE GENERATOR IMPLEMENTATION

5.3.1 INTRODUCTION

RS code generators are used extensively in Code Division Multiple Access (CDMA) systems to generate code sequences with good correlation properties. The RS code generators use efficiently implemented Linear Feedback Shift Registers (LFSRs) in both the Virtex/Virtex-II series and Spartan-II family using the SRL16 macro. In a multi-user CDMA system several forms of "Spread Spectrum" modulation techniques are used. The most popular is the Direct Sequence Spread Spectrum (DS-SS). In this form of modulation each user signal is uniquely coded and spread across a wide band of transmission frequencies. Pseudo-random Noise (PN) sequences that are orthogonal to each other are used to code the user signals. Two sequences are considered orthogonal

when their cross correlation coefficient is zero. These PN sequences are generated using RS code generators. The basic functional blocks for RS code generators are LFSRs [25].

5.3.2 RS CODE GENERATORS

RS code generator is named after Reed Soloman. He suggested that sets of small correlation PN codes could be created by Modulo 2 addition of the results of two LFSRs, primed with factor codes. The result is a set of codes with correlation properties ideally suited to distinguish one code from another in a spectrum full of coded signals. They are generated by XORing the outputs of two same length LFSRs primed with specific Fill values from two factor codes. Figure 5.7 shows an implementation of a RS code generator. Two same-length LFSRs loaded with paired factor codes are XOR'd to create a new family of codes suited for use in CDMA systems [25, 26].

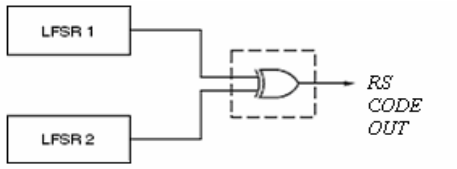


Figure 5.7 RS Code Generator

5.3.3 VHDL IMPLEMENTATION FOR RS CODE GENERATOR

Figure 5.8 demonstrates a 41-stage, 2-tap Gold code generator implementation in a Virtex-II device.The output of the RS code generator is obtained by XORing Tap 41 of the two LFSRs as shown in figure5.8 [25].

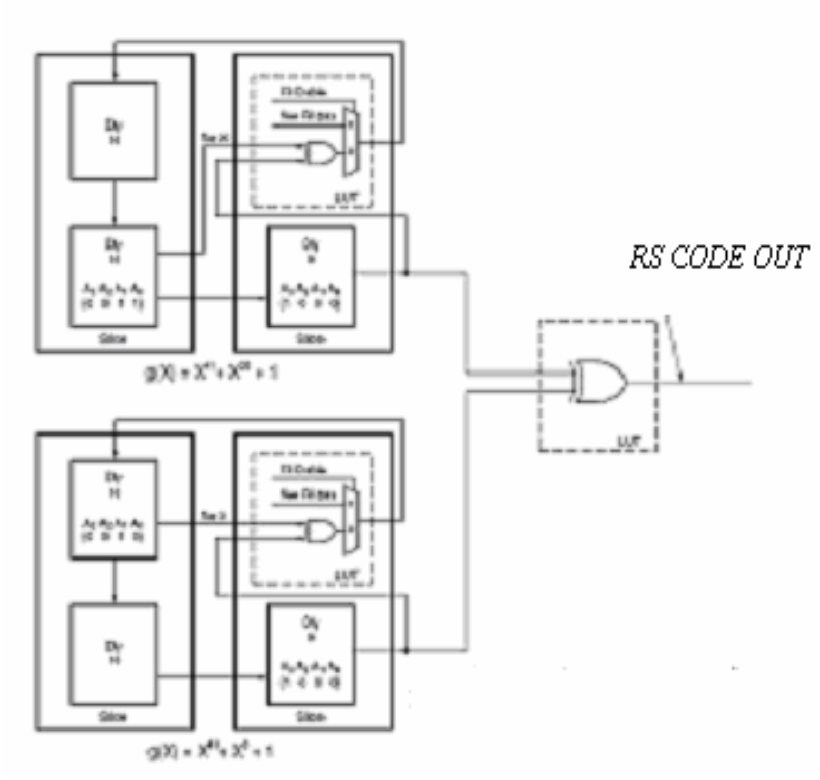


Figure 5.8 41-stages, 2-tap RS code generator.

5.3.3.1 BLOCK DIAGRAM

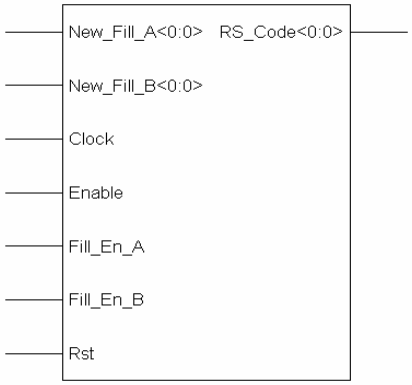


Figure 5.9 Block Diagram for RS Code Generator

5.3.3.2 XOR GATE PARITY FEEDBACK BLOCK DIAGRAM

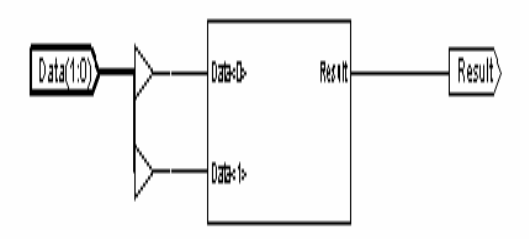


Figure 5.10 Block Diagram for XOR Gate

5.3.3.3 SCHEMATIC DIAGRAM

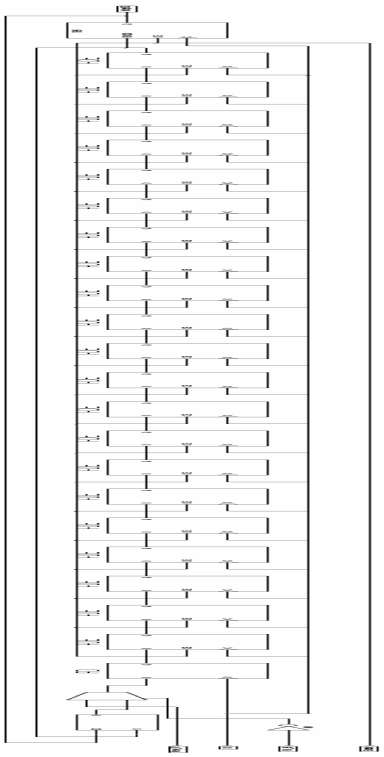


Figure 5.11 Schematic for RS Code Generator

5.3.3.4 SIMULATION RESULTS

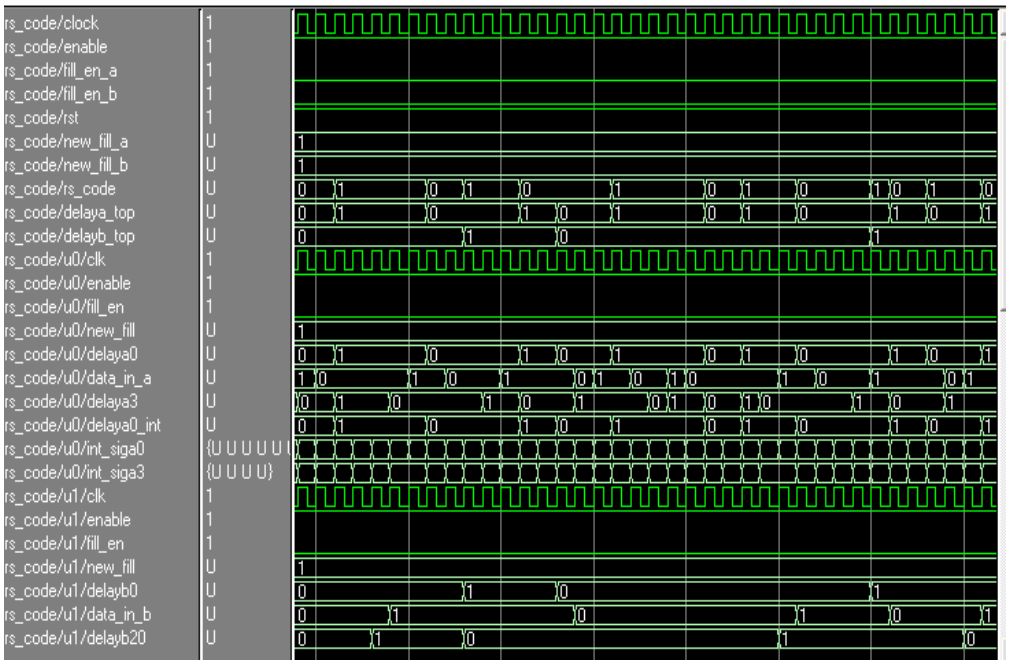


Figure 5.12 Simulation Waveforms for RS Code Generator

5.3.3.5 DEVICE UTILIZATION SUMMARY:

Table 5.7 - Device Utilization by RS Code Generator

Selected Device	xqvr300cb228-4
Number of Slices	17
Number of Slice Flip Flops	30
Number of 4 input LUTs	6
Number of bonded IOBs	6
Number of GCLKs	1

Table 5.8– Number of registers and logic gates in RS Code Generator

Registers	27
1-bit register	27
Shift Registers	2
22-bit shift register	1
5-bit shift register	1
Multiplexers	2
1-bit 2-to-1 multiplexer	2
Xors	3
1-bit xor2	3

5.3.3.6TIMING DETAILS

Table 5.9 – Timing Summary for RS Code Generator

Speed Grade	-4
Minimum period	6.549ns (Maximum Frequency: 152.695MHz
Minimum input arrival time before clock	6.085ns
Maximum output required time after clock	10.501ns

5.4 BIST IMPLEMENTATION

5.4.1 INTRODUCTION

The various test structures are proposed for BIST techniques. A typical structure used for gene-ration of pseudo-random test sets is the linear feedback shift register (LFSR). The BIST techniques have wide application in testing whole devices and embedded components. Built-in self-test (BIST) is a set of structured-test techniques for combinational and sequential logic, memories, multipliers, and other embedded logic blocks. In each case the principle is to generate test vectors, apply them to the circuit under test (CUT) or device under test (DUT), and then check the response [27].

5.4.2 BIST USING LFSR AND SIGNATURE ANALYZER

At the heart of this BIST approach, lie a pseudo-random binary sequence (PRBS) generator and a signature register. The PRBS generator is most easily implemented using a linear feedback shift register (LFSR). A PRBS generator allows us to generate all (well, almost all) of the required binary patterns for the circuit under test. The LFSR can be used to both generate the test sequence for the design that is to incorporate BIST and with slight modification can be used to capture the response of the design and generate a signature (the bit pattern held in the signature register).One approach for built-in self-test (BIST) of circuits with scan is to use a linear feedback shift register (LFSR) to shift a pseudo-random sequence of bits into the scan chain. When a pattern has been shifted into

the scan chain, it is applied to the circuit-under-test (CUT) and the response is loaded back into the scan chain and shifted out into a signature register for compaction as the next pattern is shifted into the scan chain. (Figure 5.13) shows a block diagram for this "test-per-scan" approach [27,28].

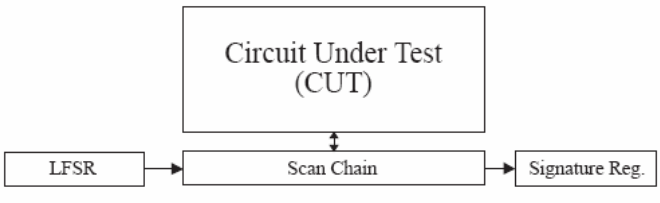


Figure 5.13 Block Diagram for Test-Per-Scan

5.4.2.1 LFSR

(Figure 5.14)shows a 3-bit maximal-length LFSR produces a repeating string of seven pseudorandom binary numbers: 7, 3, 1, 4, 2, and 5, 6. the exclusive-OR gates and shift register act to produce a pseudorandom binary sequence (PRBS) at each of the flip-flop outputs. By correctly choosing the points at which we take the feedback from an n -bit shift register we can produce a PRBS of length$2^n - 1$, a maximal-length sequence that includes all possible patterns (or vectors) of n bits, excluding the all-zeros pattern [29].

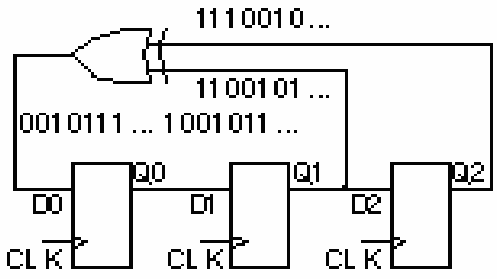


Figure 5.14 3-Bit LFSR

5.4.2.2 SIGNATURE ANALYSER

(Figure 5.15) shows the LFSR of (Figure 5.14) with an additional XOR gate used in the first stage of the shift register. If we apply a binary input sequence to IN, the shift register will perform data compaction (or compression) on the input sequence. At the end of the input sequence the shift-register contents, Q0Q1Q2, will form a pattern that we call a signature. If the input sequence and the serial-input signature register (SISR) are long

enough, it is unlikely (though possible) that two different input sequences will produce the same signature. If the input sequence comes from logic that we wish to test, a fault in the logic will cause the input sequence to change. This causes the signature to change from a known good value and we shall then know that the circuit under test is bad. This technique, called signature analysis, was developed by Hewlett-Packard to test equipment in the field in the late 1970s [29].

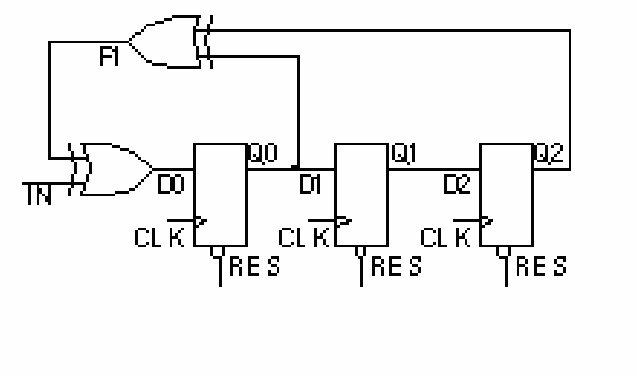


Figure 5.15 Signature Analyzer

5.4.2.3 A SIMPLE BIST EXAMPLE

We can combine the PRBS generator of (Figure5.14) together with the signature register of (Figure5.15) to form the simple BIST structure shown in (Figure 5.16) LFSR1 generates a maximal-length ($2^3 - 1 = 7$ cycles) PRBS. LFSR2 computes the signature ('011' for the good circuit) of the CUT. LFSR1 is initialized to '100' (Q0 = 1, Q1 = 0, Q2 = 0) and LFSR2 is initialized to '000'. The schematic in (Figure 5.16) shows the bit sequences in the circuit, both for a good circuit and for a bad circuit with a stuck-at-1 fault, F1[29,30].

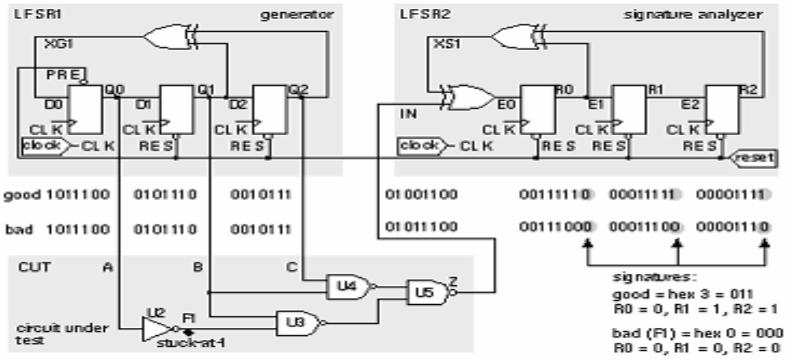


Figure 5.16 a Simple BIST Example

5.4.3 VHDL IMPLEMENTATION FOR BIST

5.4.3.1 BLOCK DIAGRAM FOR BIST

(Figure 5.17) shows the block diagram for BIST using 10-bit maximal length LFSR and 10-bit signature register. Here clock is available all the time for the circuit operation .Here if reset is "1" the circuit stay in its previous state, however if reset is "0" then it start operation.

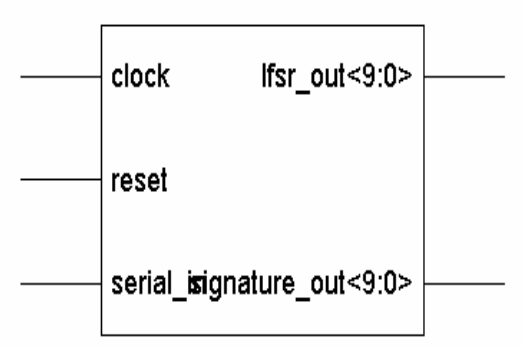


Figure 5.17 Block Diagram of BIST

5.4.3.2 CIRCUIT DIAGRAM

(Figure 5.18) shows the circuit diagram for BIST where we got two outputs, where one is from the LFSR and the other is from the signature register. For the proper BIST operation the output from signature analyzer should be exactly signature of the LFSR output.

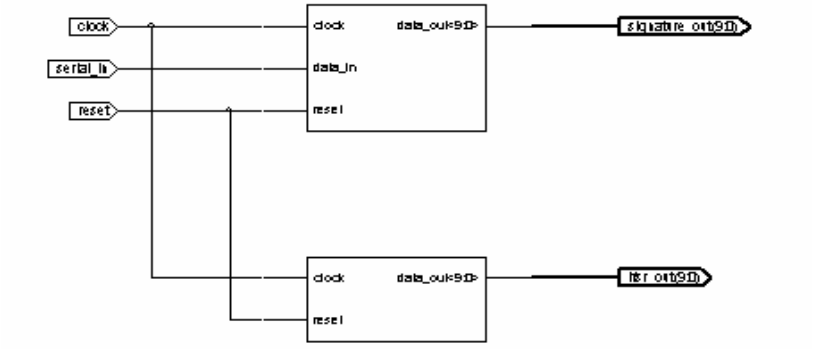


Figure 5.18 Circuit Diagram for BIST

5.4.3.3 SIGNATURE ANALYZER CIRCUIT DIAGRAM

(Figure 5.19) shows the signature analyzer circuit which is modified from the LFSR as shown below. It's a 10- bit signature register whose output should be exactly the signature of the LFSR. It consists of an FDR that is feedback register which fed back the required outputs back to the input through an exclusive-or gate.

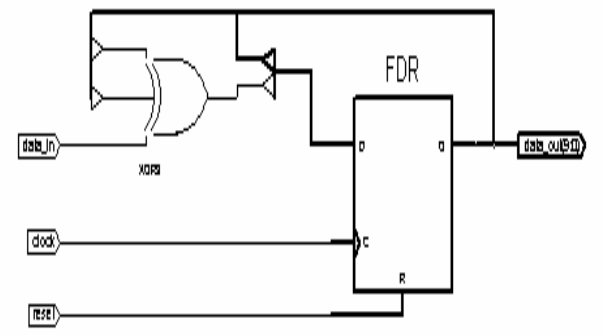


Figure 5.19 Schematic for Signature Analyzer

5.4.3.4 LFSR DIAGRAM

(Figure 5.20) shows a maximal length LFSR with 10-bit length where bits 6 and 9 are fed back to the input through the exclusive-or gate.

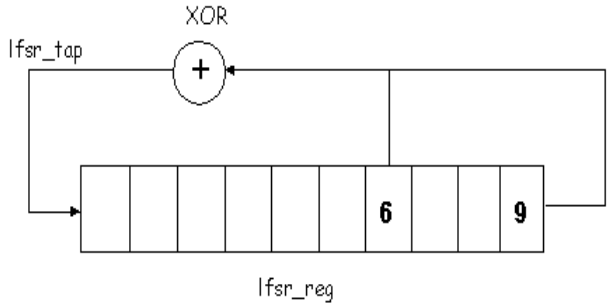


Figure 5.20 10-Bit LFSR for BIST Implementation

5.4.3.5 SIMULATION RESULTS FOR LFSR

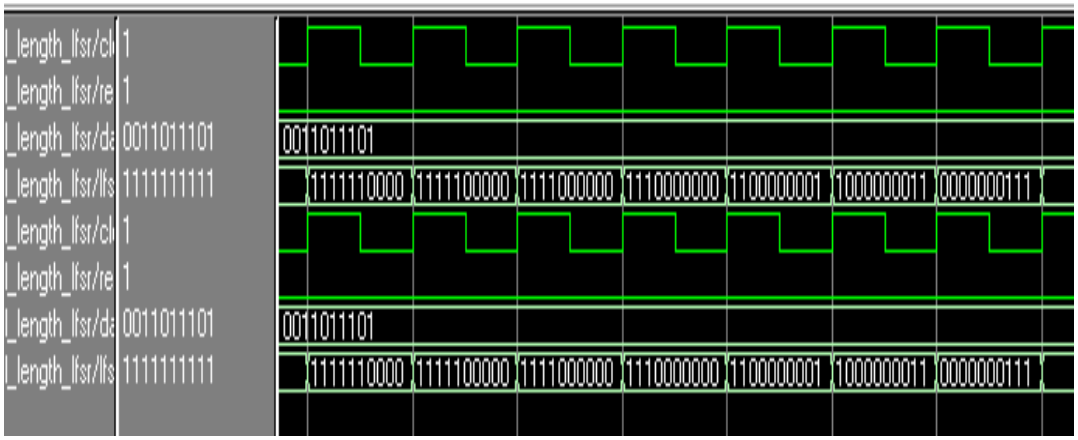


Figure 5.21 Simulation Waveforms for LFSR

5.4.3.6 SIMULATION RESULTS FOR SIGNATURE REGISTER

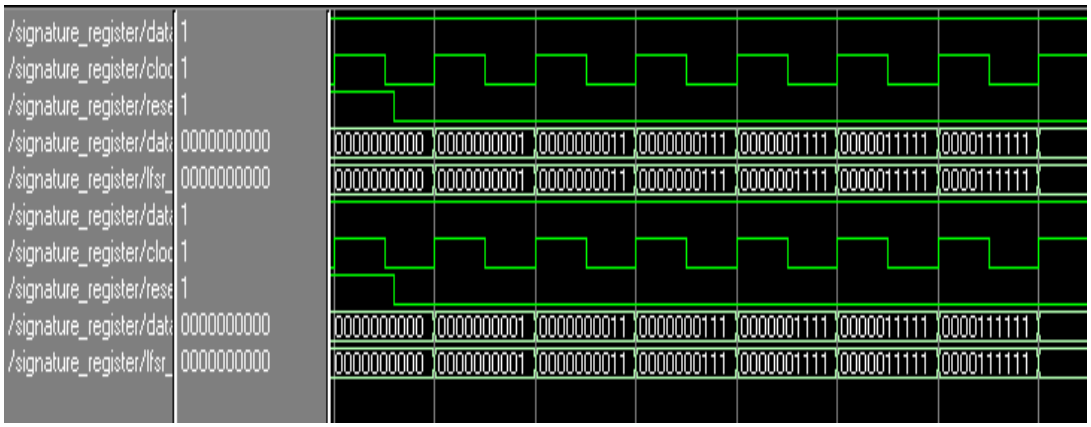


Figure 5.22 Simulation Waveforms for Signature Analyzer

5.4.3.7 SIMULATION RESULTS FOR BIST

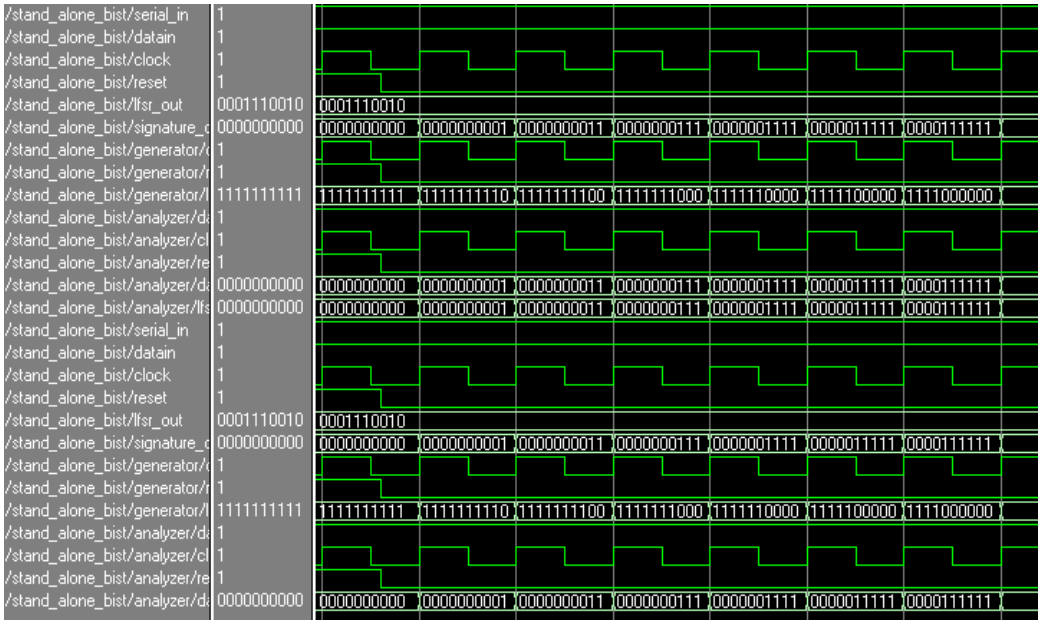


Figure 5.23 Simulation Waveforms for BIST

5.4.3.8 DEVICE UTILIZATION DETAILS

Table 5.10 - Device Utilization by BIST

Selected Device	xqvr300cb228-4
Number of Slices	6
Number of Slice Flip Flops	10
Number of 4 input LUTs	1
Number of bonded IOBs	12
Number of GCLKs	1

Table 5.11– Number of registers and logic gates in BIST

Registers	10
1-bit register	10
Xors	1
1-bit xor3	1

Table 5.12 – Cell Usage by BIST

BELS	1
LUT3_L	1
FlipFlops/Latches	10
FDR	10
Clock Buffers	1
BUFGP	1
IO Buffers	12
IBUF	2
OBUF	10

5.4.3.9 TIMING DETAILS:

Table 5.13 – Timing Summary for BIST

Speed Grade	-4
Minimum period	4.503ns (Maximum Frequency: 222.074MHz)
Minimum input arrival time before clock	4.503ns (Maximum Frequency: 222.074MHz)
Maximum output required time after clock	8.652ns
Maximum combinational path delay	No path found

CONCLUSION

This thesis has discussed the concept of pseudo random sequences as applicable to spread spectrum communications. Maximal length sequences were introduced, and used as an introduction to more complicated methods of PN code generation with the help of LFSR'S. The linear feedback shift register (LFSR) is a shift register which, using feedback, modifies itself on each rising edge of the clock. The feedback causes the value in the shift register to cycle through a set of unique values. The choice of LFSR length, gate type, LFSR type, maximum length logic, and tap positions allows the user to control the implementation and feedback of the LFSR, which, in turn, controls the sequence of repeating values the LFSR will iterate through. The PN Sequence Generator block generates a sequence of pseudorandom binary numbers. A Pseudonoise sequence can be used in a Mobile Communication.

The various test structures are proposed for BIST techniques. A typical structure used for generation of pseudo-random test sets is the linear feedback shift register (LFSR). The BIST techniques have wide application in testing whole devices and embedded components. We focus on the analysis of the state coverage, fault coverage, and optimal structure of BIST schemes. LFSR is better from the point of view of the number of cells. In realisation, where the number of cells is critical, it's better to use LFSR. The BIST techniques have wide application in testing whole devices and embedded components. The various test structures are proposed for BIST techniques but typical structure used for gene-ration of pseudo-random test sets is the linear feedback shift register (LFSR). The LFSR is better from the point of view of the number of cells. In realization, where the number of cells is critical, it's better to use LFSR. The BIST is everyday compromise between test ex-haustivity and adding hardware. It should be good to know how much hardware realization of LFSR needs.

Thus a Linear Feedback Shift Register is a sequential shift register with combinational logic that causes it to pseudo-randomly cycle through a sequence of binary values. Linear feedback shift registers have multiple uses in digital systems design. It can be mainly used for random pattern generation in PN-Generators and RS- Code

Generators. Also it is used in Build in Self Test (BIST) for testing the large patterns. Inspite of these applications the LFSR mainly cover many applications such as Data Encryption/Decryption, Digital Signal Processing, Data Integrity Checksums ,Data Compression Scrambler/Descrambler and Optimized Counters.

REFERENCES

[1] F.Simpson and J. Holtzman, “CDMA power control, interleaving, and coding", Proc. 41st IEEE Vehic. Technol. Conf., Saint Louis, MO, pp. 362-367,1991

[2] Neebel, D.J.,C.R.Kime, “Inhomogeneous Cellular Automata for Weighted Random Pattern Generation,” *Proc. of International Test Conference*, pp. 1013-1022, 1993

[3] R. L. Peterson, R. E. Ziemer and D. E. Borth, Introduction to Spread Spectrum Communications, Upper Saddle River, NJ, Prentice Hall, 1995

[4] R. L. Picholtz, D. L. Schilling, and L. B. Milstein, “Theory of spread-spectrum communications - A tutorial", IEEE Trans. Commun., vol. COM-30, pp. 855-884, May 1982

[5] G. L. Stuber and C. Kchao, “Analysis of a multiple-cell direct sequence CDMA cellular mobile radio system", IEEE J. Select. Commun., vol. 10, pp. 669-679,May 1992

[6] R.C. Dixon, Spread Spectrum Systems, 2nd Ed. John Wiley and Sons, NewYork, pp. 403-405, 1984

[7] A. K. Elhakeem, G. S. Takhar, and S. C. Gupta, “New code acquisition techniques in spread spectrum communications", IEEE Trans. Commun., vol. COM-28, pp. 246-259, Feb. 1980

[8] R. Ward, “Acquisition of pseudonoise signals by sequential estimation”, IEEE Trans. Commun. Tech., pp. 475-483, Dec. 1965.

[9] Spreading Codes for Direct Sequence for DS-CDMA Communication system magazine October 1996, pp.124-136, E.Dinan and B.Jabbari

[10] John Koeter “what’s an LFSR?”,Texas instruments, December1996.

[11] Peter Alfke, *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators,* Xilinx applications note **XAPP052.**

[12] Maria George and Peter Alfke “Linear Feedback Shift Registers in Vertex Devices”, Xilinx Application Note ,January 9,2001.

[13] D. J. Torrieri, “Performance of direct-sequence systems with long pseduonoise sequences", IEEE J. Select. Commun., vol. 10, pp. 770-781, May 1992.

[14] S. Fredricsson, “Pseudo-randomness properties of binary shift register se-quences”, IEEE Trans. Inform. Theory, vol. IT-21, pp. 115-120, Jan. 1975.

[15] Linear Feedback Shift Register, Xilinx Logicore, DS257 (v1.0) March 28, 2003

[16] Stephen Lim and Andy Miller “LFSRs as Functional Blocks in Wireless Applications” XAPP220 January 11, 2001

[17] Using Look-Up Tables as Shift Registers (SRL16) in Spartan-3 Devices, Application Note: Spartan-3, XAPP465 (v1.0) April 10, 2003

[18] S. Fredricsson, “Pseudo-randomness properties of binary shift register sequences”, IEEE Trans. Inform. Theory, vol. IT-21, pp. 115-120, Jan. 1975.

[19] E. A. Geraniotis, “Performance of noncoherent direct-sequence spread spectrum multiple-access communications”, Special Issue on Military Com-munications, IEEE J. Select. Areas Commun., vol. SAC-3, pp. 687-694, Sept.1985.

[20] H. Meyr and G. Poltzer, “Performance analysis for general PN-spread spectrum acquisition techniques”, IEEE Trans. Commun., vol. COM-31, pp. 1317-1319, Dec. 1983.

[21] D.V. Sarwate, M.B. Pursley, “Cross Correlation properties of Pseudo Random and related sequences”, Proc. IEEE, vol. 68, May 1980.

[22] W. A. Gardner, “The role of spectral correlation in the design and analysis of synchronizers”, IEEE Trans. Commun., vol. COM-34, pp. 1089-1095, Nov.1986.

[23] Esmael H. Dinan, and Bijan Jabbari, “Spreading codes for Direct Sequence CDMA and wideband CDMA cellular Networks”, IEEE Communications Magazine, pp. 48-50, September 1998.

[24 Ulrich *Walther and Gerhard P. Fettweis* “PN-Generators Embedded in High Performance Signal Processors” Dresden University of Technology Mannesmann Mobilfunk Chair for Mobile Communications Systems Mommsenstr. 13, D-01062 Dresden, Germany

[25] E.R. Berlekamp, “*Bit-serial Reed-Solomon encoders*”, IEEE Trans. Information Theory, vol. 28. pp. 120-126, Nov. 1982.

[26] R.E. Blahut, “*Theory and practice of error-control codes*”, Addison-Wesley, Reading, MA, 1983

[27] Principles of CMOS VLSI Design, A System Perspective Second Edition by Neil H.E Weste kanran Eshraghian

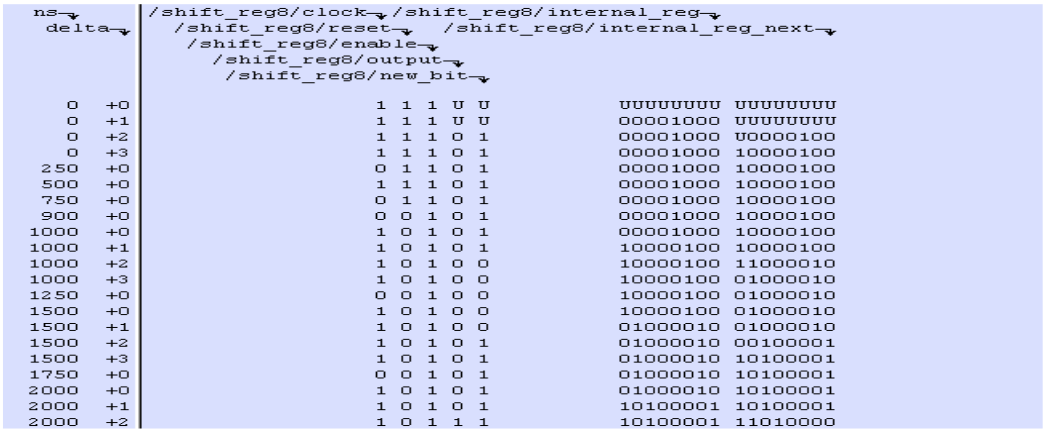
[28] Dufaza, C., and G. Cambon, "LFSR based Deterministic and Pseudo-Random Test Pattern Generator Structures," *Proc. of EuropeanTest Conference*, pp. 27-34, 1991.

[29] M. Abramovici, M. A. Breuer, A. D. Friedman: "Digital Systems Testing and Testable Design", Computer Science Press, 1990

[30] AGRAVAL, V. D., KIME, C. R., SALUJA, K. S. "A Tutorial on Built-in Self-test", Part 1: Principles. IEEE Design & Test of Computers. 1993, vol. 10, no. 3, p. 73 - 82.

APPENDIX A (BIT PATTERNS)

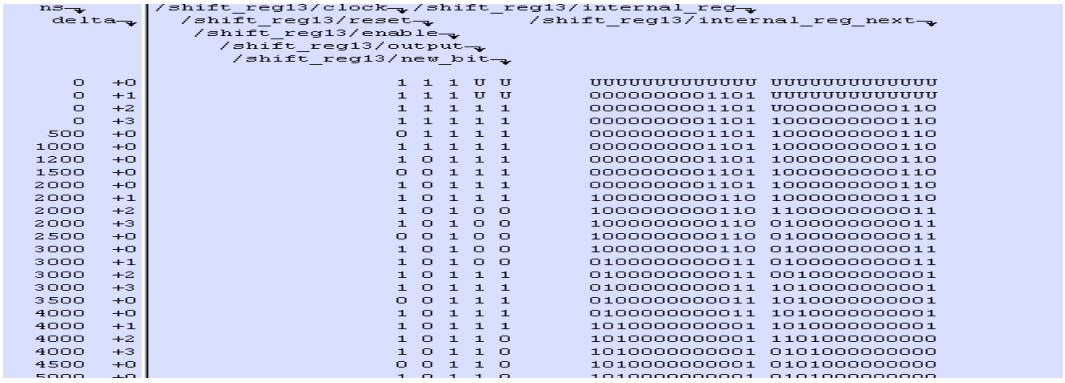
A.1 BIT PATTERN FOR 8-BIT LFSR



ns / delta	/shift_reg8/clock	/shift_reg8/reset	/shift_reg8/enable	/shift_reg8/output	/shift_reg8/new_bit	/shift_reg8/internal_reg	/shift_reg8/internal_reg_next
0 +0	1	1	1	U	U	UUUUUUUU	UUUUUUUU
0 +1	1	1	1	U	U	00001000	UUUUUUUU
0 +2	1	1	1	0	1	00001000	U0000100
0 +3	1	1	1	0	1	00001000	10000100
250 +0	0	1	1	0	1	00001000	10000100
500 +0	1	1	1	0	1	00001000	10000100
750 +0	0	1	1	0	1	00001000	10000100
900 +0	0	0	1	0	1	00001000	10000100
1000 +0	1	0	1	0	1	00001000	10000100
1000 +1	1	0	1	0	1	10000100	10000100
1000 +2	1	0	1	0	0	10000100	11000010
1000 +3	1	0	1	0	0	10000100	01000010
1250 +0	0	0	1	0	0	10000100	01000010
1500 +0	1	0	1	0	0	10000100	01000010
1500 +1	1	0	1	0	0	01000010	01000010
1500 +2	1	0	1	0	1	01000010	00100001
1500 +3	1	0	1	0	1	01000010	10100001
1750 +0	0	0	1	0	1	01000010	10100001
2000 +0	1	0	1	0	1	01000010	10100001
2000 +1	1	0	1	0	1	10100001	10100001
2000 +2	1	0	1	1	1	10100001	11010000

Figure -Bit pattern for Figure 4.3 (a)

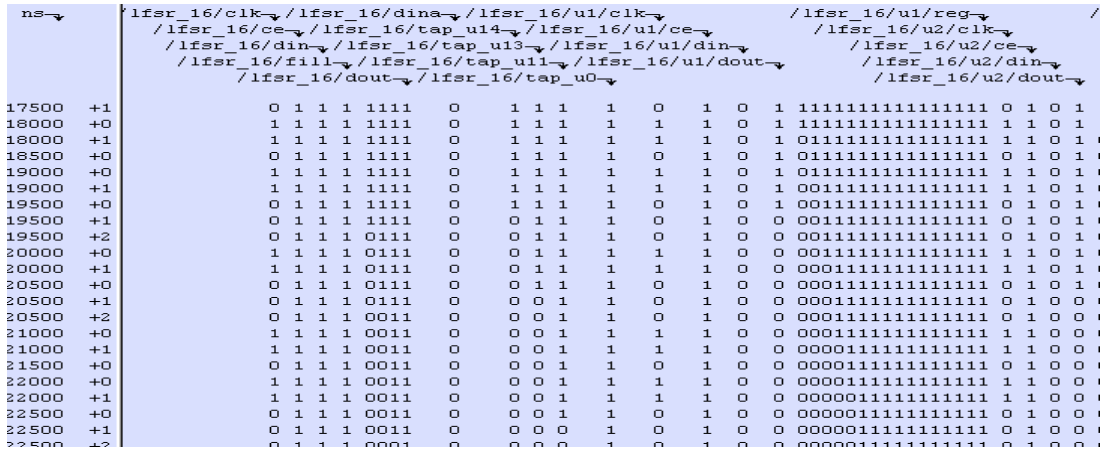
A.2 BIT PATTERN FOR 13-BIT LFSR



ns / delta	/shift_reg13/clock	/shift_reg13/reset	/shift_reg13/enable	/shift_reg13/output	/shift_reg13/new_bit	/shift_reg13/internal_reg	/shift_reg13/internal_reg_next
0 +0	1	1	1	U	U	UUUUUUUUUUUUU	UUUUUUUUUUUUU
0 +1	1	1	1	U	U	0000000001101	UUUUUUUUUUUUU
0 +2	1	1	1	1	1	0000000001101	U000000000110
0 +3	1	1	1	1	1	0000000001101	1000000000110
500 +0	0	1	1	1	1	0000000001101	1000000000110
1000 +0	1	1	1	1	1	0000000001101	1000000000110
1200 +0	1	0	1	1	1	0000000001101	1000000000110
1500 +0	0	0	1	1	1	0000000001101	1000000000110
2000 +0	1	0	1	1	1	0000000001101	1000000000110
2000 +1	1	0	1	1	1	1000000000110	1000000000110
2000 +2	1	0	1	0	0	1000000000110	1100000000011
2000 +3	1	0	1	0	0	1000000000110	0100000000011
2500 +0	0	0	1	0	0	1000000000110	0100000000011
3000 +0	1	0	1	0	0	1000000000110	0100000000011
3000 +1	1	0	1	0	0	0100000000011	0100000000011
3000 +2	1	0	1	1	1	0100000000011	0010000000001
3000 +3	1	0	1	1	1	0100000000011	1010000000001
3500 +0	0	0	1	1	1	0100000000011	1010000000001
4000 +0	1	0	1	1	1	0100000000011	1010000000001
4000 +1	1	0	1	1	1	1010000000001	1010000000001
4000 +2	1	0	1	1	0	1010000000001	1101000000000
4000 +3	1	0	1	1	0	1010000000001	0110100000000
4500 +0	0	0	1	1	0	1010000000001	0110100000000

Figure -Bit pattern for Figure 4.6

A.3 BIT PATTERN FOR 16-BIT, 4 TAP PARALLEL LFSR



ns	/lfsr_16/clk	/lfsr_16/ce	/lfsr_16/din	/lfsr_16/fill	/lfsr_16/dout	/lfsr_16/dina	/lfsr_16/tap_u14	/lfsr_16/tap_u13	/lfsr_16/tap_u11	/lfsr_16/tap_u0	/lfsr_16/u1/clk	/lfsr_16/u1/ce	/lfsr_16/u1/din	/lfsr_16/u1/dout	/lfsr_16/u1/reg	/lfsr_16/u2/clk	/lfsr_16/u2/ce	/lfsr_16/u2/din	/lfsr_16/u2/dout
17500 +1	0	1	1	1	1111	0	1	1	1	1	0	1	0	1	1111111111111111	0	1	0	1
18000 +0	1	1	1	1	1111	0	1	1	1	1	1	1	0	1	1111111111111111	1	1	0	1
18000 +1	1	1	1	1	1111	0	1	1	1	1	1	1	0	1	0111111111111111	1	1	0	1
18500 +0	0	1	1	1	1111	0	1	1	1	1	0	1	0	1	0111111111111111	0	1	0	1
19000 +0	1	1	1	1	1111	0	1	1	1	1	1	1	0	1	0111111111111111	1	1	0	1
19000 +1	1	1	1	1	1111	0	1	1	1	1	1	1	0	1	0011111111111111	1	1	0	1
19500 +0	0	1	1	1	1111	0	1	1	1	1	0	1	0	1	0011111111111111	0	1	0	1
19500 +1	0	1	1	1	1111	0	0	1	1	1	0	1	0	0	0011111111111111	0	1	0	1
19500 +2	0	1	1	1	0111	0	0	1	1	1	0	1	0	0	0011111111111111	0	1	0	1
20000 +0	1	1	1	1	0111	0	0	1	1	1	1	1	0	0	0011111111111111	1	1	0	1
20000 +1	1	1	1	1	0111	0	0	1	1	1	1	1	0	0	0001111111111111	1	1	0	1
20500 +0	0	1	1	1	0111	0	0	1	1	1	1	0	1	0	0001111111111111	0	1	0	1
20500 +1	0	1	1	1	0111	0	0	0	1	1	0	1	0	0	0001111111111111	0	1	0	0
20500 +2	0	1	1	1	0011	0	0	0	1	1	0	1	0	0	0001111111111111	0	1	0	0
21000 +0	1	1	1	1	0011	0	0	0	1	1	1	1	0	0	0001111111111111	1	1	0	0
21000 +1	1	1	1	1	0011	0	0	0	1	1	1	1	0	0	0000111111111111	1	1	0	0
21500 +0	0	1	1	1	0011	0	0	0	1	1	0	1	0	0	0000111111111111	0	1	0	0
22000 +0	1	1	1	1	0011	0	0	0	1	1	1	1	0	0	0000111111111111	1	1	0	0
22000 +1	1	1	1	1	0011	0	0	0	1	1	1	1	0	0	0000011111111111	1	1	0	0
22500 +0	0	1	1	1	0011	0	0	0	1	1	0	1	0	0	0000011111111111	0	1	0	0
22500 +1	0	1	1	1	0011	0	0	0	0	1	0	1	0	0	0000011111111111	0	1	0	0
22500 +2	0	1	1	1	0001	0	0	0	0	1	0	1	0	0	0000011111111111	0	1	0	0

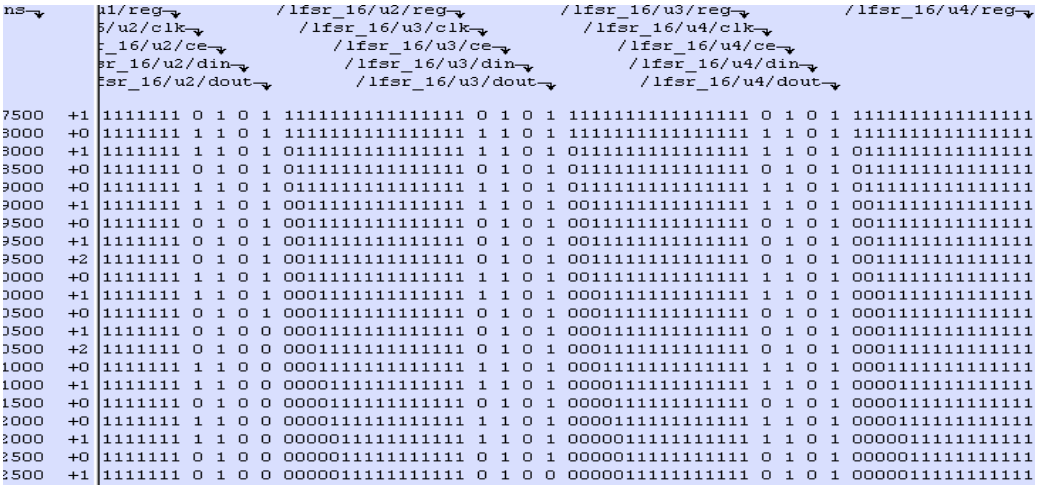


Figure -Bit pattern for Figure 4.12

A.4 BIT PATTERN FOR MULTICYCLE TAP ACCESS LFSR

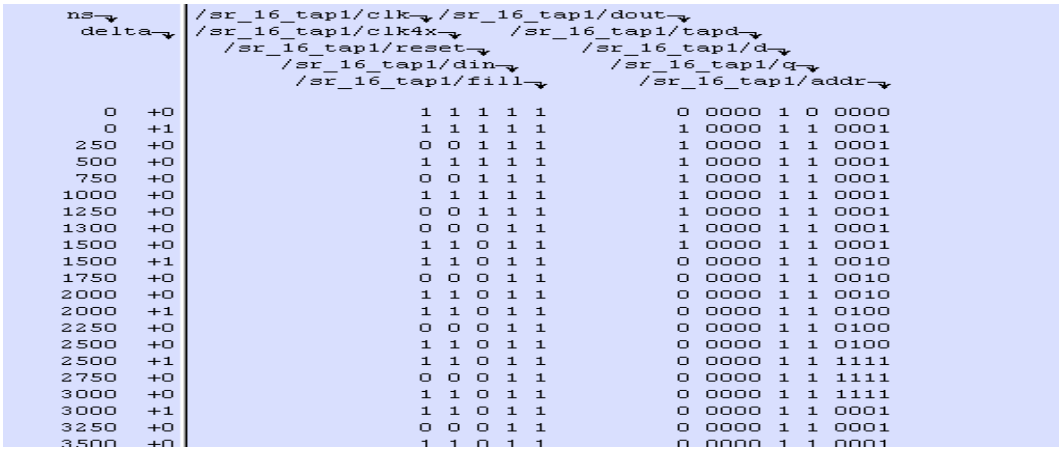
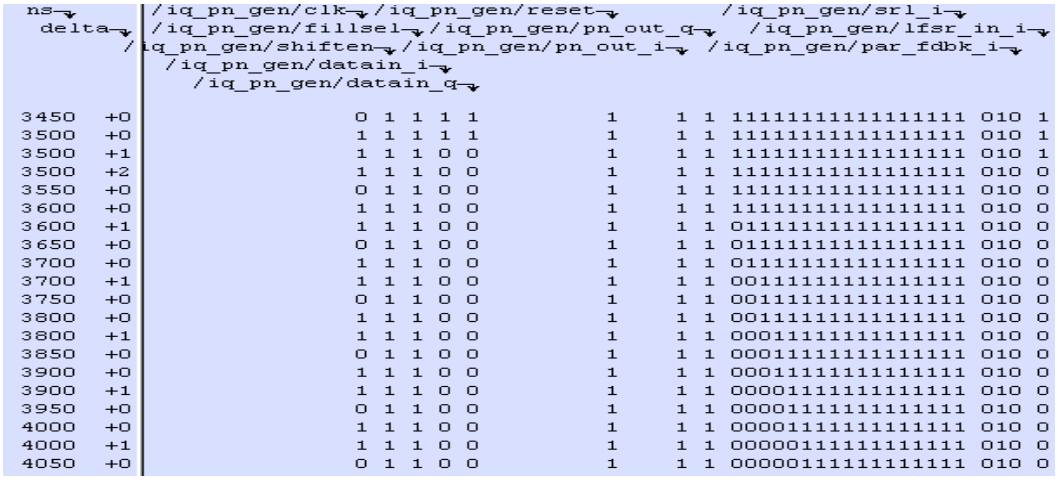


Figure -Bit pattern for Figure 4.18

A.5 BIT PATTERN FOR PN GENERATOR



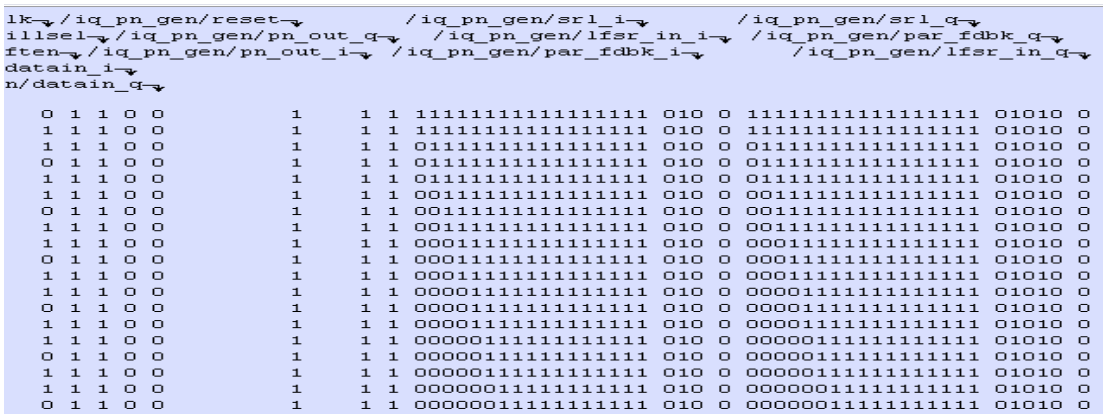


Figure -Bit pattern for Figure 5.5

A.6 BIT PATTERN FOR RS CODE GENERATOR

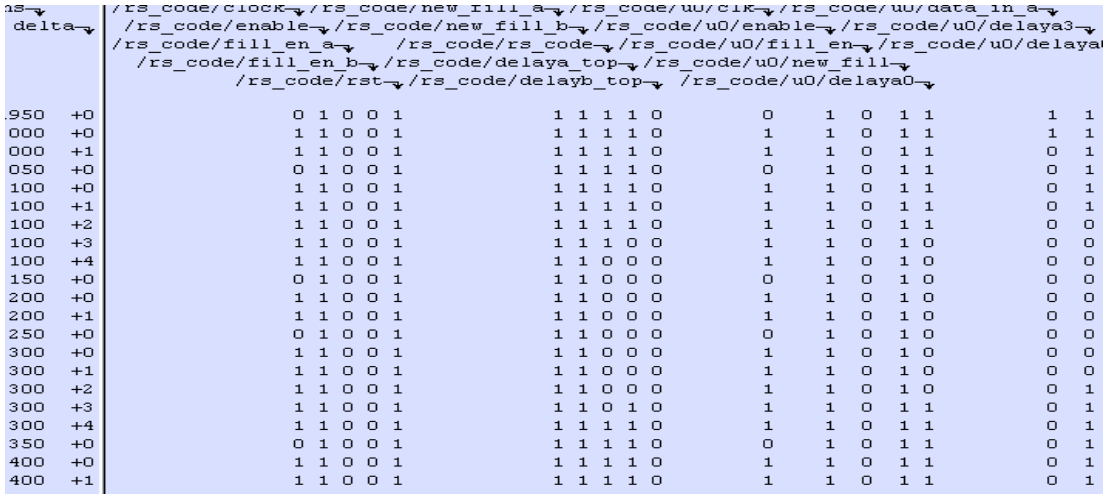


Figure -Bit pattern for Figure 5.12

A.7 BIT PATTERN FOR 10-BIT LFSR IN BIST

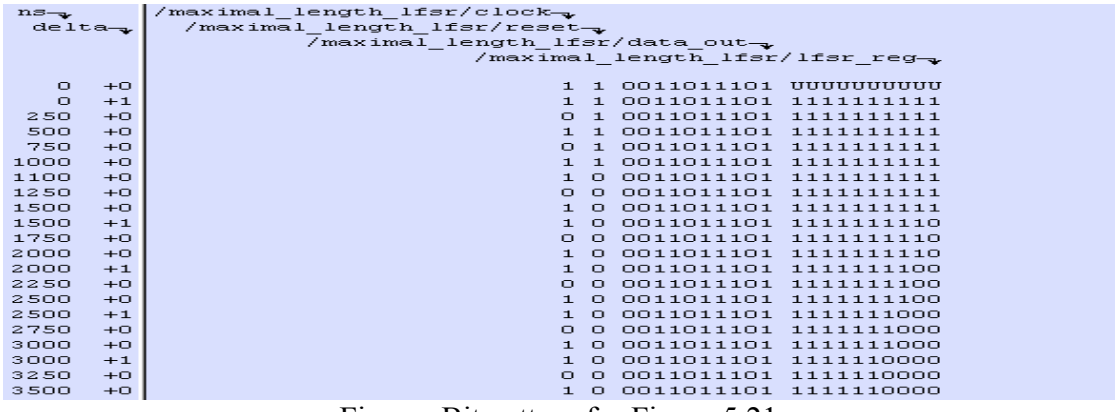
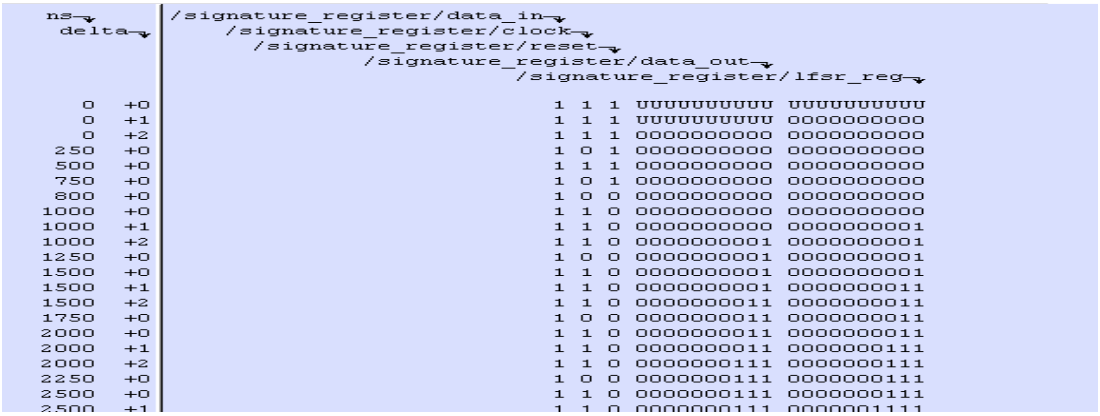


Figure -Bit pattern for Figure 5.21

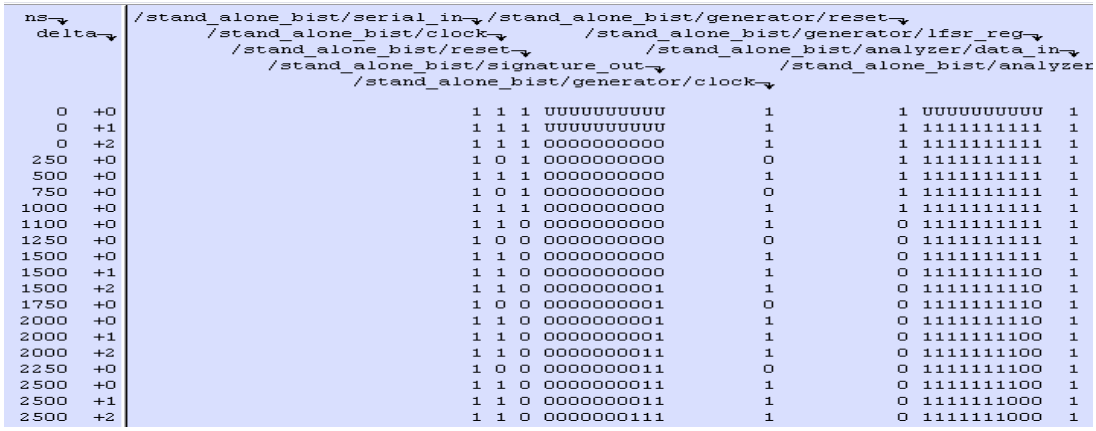
A.7 BIT PATTERN FOR SIGNATURE ANALYSER IN BIST



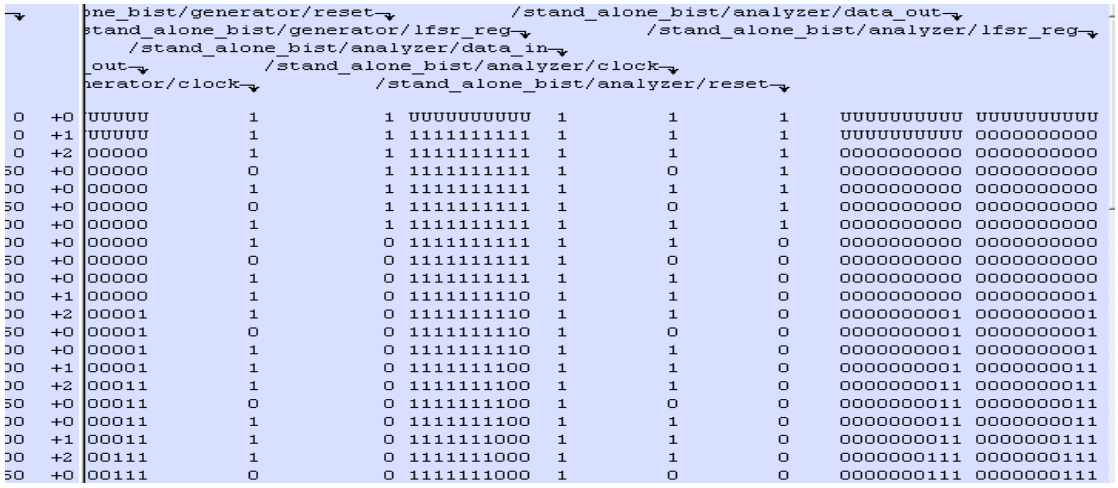
ns	delta	/signature_register/data_in	/signature_register/clock	/signature_register/reset	/signature_register/data_out	/signature_register/lfsr_reg
0	+0		1	1	UUUUUUUUUU	UUUUUUUUUU
0	+1		1	1	UUUUUUUUUU	0000000000
0	+2		1	1	0000000000	0000000000
250	+0		1	0	0000000000	0000000000
500	+0		1	1	0000000000	0000000000
750	+0		1	0	0000000000	0000000000
800	+0		1	0	0000000000	0000000000
1000	+0		1	1	0000000000	0000000000
1000	+1		1	1	0000000000	0000000001
1000	+2		1	1	0000000001	0000000001
1250	+0		1	0	0000000001	0000000001
1500	+0		1	1	0000000001	0000000001
1500	+1		1	1	0000000001	0000000011
1500	+2		1	1	0000000011	0000000011
1750	+0		1	0	0000000011	0000000011
2000	+0		1	1	0000000011	0000000011
2000	+1		1	1	0000000011	0000000111
2000	+2		1	1	0000000111	0000000111
2250	+0		1	0	0000000111	0000000111
2500	+0		1	1	0000000111	0000000111
2500	+1		1	1	0000000111	0000001111

Figure -Bit pattern for Figure 5.22

A.8 SIMULATION RESULTS FOR BIST



ns	delta	/stand_alone_bist/serial_in	/stand_alone_bist/clock	/stand_alone_bist/reset	/stand_alone_bist/signature_out	/stand_alone_bist/generator/clock	/stand_alone_bist/generator/reset	/stand_alone_bist/generator/lfsr_reg	/stand_alone_bist/analyzer/data_in	/stand_alone_bist/analyzer
0	+0		1	1	1	UUUUUUUUUU	1	1	UUUUUUUUUU	1
0	+1		1	1	1	UUUUUUUUUU	1	1	1111111111	1
0	+2		1	1	1	0000000000	1	1	1111111111	1
250	+0		1	0	1	0000000000	0	1	1111111111	1
500	+0		1	1	1	0000000000	1	1	1111111111	1
750	+0		1	0	1	0000000000	0	1	1111111111	1
1000	+0		1	1	1	0000000000	1	1	1111111111	1
1100	+0		1	1	0	0000000000	1	0	1111111111	1
1250	+0		1	0	0	0000000000	0	0	1111111111	1
1500	+0		1	1	0	0000000000	1	0	1111111111	1
1500	+1		1	1	0	0000000000	1	0	1111111110	1
1500	+2		1	1	0	0000000001	1	0	1111111110	1
1750	+0		1	0	0	0000000001	0	0	1111111110	1
2000	+0		1	1	0	0000000001	1	0	1111111110	1
2000	+1		1	1	0	0000000001	1	0	1111111100	1
2000	+2		1	1	0	0000000011	1	0	1111111100	1
2250	+0		1	0	0	0000000011	0	0	1111111100	1
2500	+0		1	1	0	0000000011	1	0	1111111100	1
2500	+1		1	1	0	0000000011	1	0	1111111000	1
2500	+2		1	1	0	0000000111	1	0	1111111000	1



ns	delta	/stand_alone_bist/generator/reset	/stand_alone_bist/generator/lfsr_reg	/stand_alone_bist/analyzer/data_in	/stand_alone_bist/analyzer/clock	/stand_alone_bist/analyzer/reset	/stand_alone_bist/analyzer/data_out	/stand_alone_bist/analyzer/lfsr_reg
0	+0	UUUUU	1	1	UUUUUUUUUU	1	1	1	UUUUUUUUUU	UUUUUUUUUU
0	+1	UUUUU	1	1	1111111111	1	1	1	UUUUUUUUUU	0000000000
0	+2	00000	1	1	1111111111	1	1	1	0000000000	0000000000
50	+0	00000	0	1	1111111111	1	0	1	0000000000	0000000000
00	+0	00000	1	1	1111111111	1	1	1	0000000000	0000000000
50	+0	00000	0	1	1111111111	1	0	1	0000000000	0000000000
00	+0	00000	1	1	1111111111	1	1	1	0000000000	0000000000
00	+0	00000	1	0	1111111111	1	1	0	0000000000	0000000000
50	+0	00000	0	0	1111111111	1	0	0	0000000000	0000000000
00	+0	00000	1	0	1111111111	1	1	0	0000000000	0000000000
00	+1	00000	1	0	1111111110	1	1	0	0000000000	0000000001
00	+2	00001	1	0	1111111110	1	1	0	0000000001	0000000001
50	+0	00001	0	0	1111111110	1	0	0	0000000001	0000000001
00	+0	00001	1	0	1111111110	1	1	0	0000000001	0000000001
00	+1	00001	1	0	1111111100	1	1	0	0000000001	0000000011
00	+2	00011	1	0	1111111100	1	1	0	0000000011	0000000011
50	+0	00011	0	0	1111111100	1	0	0	0000000011	0000000011
00	+0	00011	1	0	1111111100	1	1	0	0000000011	0000000011
00	+1	00011	1	0	1111111000	1	1	0	0000000011	0000000111
00	+2	00111	1	0	1111111000	1	1	0	0000000111	0000000111
50	+0	00111	0	0	1111111000	1	0	0	0000000111	0000000111

Figure -Bit pattern for Figure 5.23

APPENDIX B (SYNTHESIS REPORTS)

B.1 SYNTHESIS REPORT FOR 8-BIT LFSR

---- Source Parameters

Input File Name	:	shift_reg8.prj
Input Format	:	mixed
Ignore Synthesis Constraint File	:	NO

---- Target Parameters

Output File Name	:	shift_reg8
Output Format	:	NGC
Target Device	:	xqvr300-4-cb228

---- Source Options

Top Module Name	:	shift_reg8
Automatic FSM Extraction	:	YES
FSM Encoding Algorithm	:	Auto
FSM Style	:	lut
RAM Extraction	:	Yes
RAM Style	:	Auto
ROM Extraction	:	Yes
ROM Style	:	Auto
Mux Extraction	:	YES
Mux Style	:	Auto
Decoder Extraction	:	YES
Priority Encoder Extraction	:	YES
Shift Register Extraction	:	YES
Logical Shifter Extraction	:	YES
XOR Collapsing	:	YES
Resource Sharing	:	YES

Multiplier Style	:	lut
Automatic Register Balancing	:	No

---- Target Options

Add IO Buffers	:	YES
Global Maximum Fanout	:	100
Add Generic Clock Buffer(BUFG)	:	4
Register Duplication	:	YES
Equivalent register Removal	:	YES
Slice Packing	:	YES
Pack IO Registers into IOBs	:	auto

---- General Options

Optimization Goal	:	Speed
Optimization Effort	:	1
Keep Hierarchy	:	NO
Global Optimization	:	All Clock Nets
RTL Output	:	Yes
Write Timing Constraints	:	NO
Hierarchy Separator	:	_
Bus Delimiter	:	<>
Case Specifier	:	maintain
Slice Utilization Ratio	:	100
Slice Utilization Ratio Delta	:	5

---- Other Options

lso	:	shift_reg8.lso
Read Cores	:	YES
cross_clock_analysis	:	NO
verilog2001	:	YES
Optimize Instantiated Primitives	:	NO
tristate2logic	:	No

CPU: 5.89 / 8.06 s | Elapsed: 6.00 / 8.00 s

Total memory usage is 65532 kilobytes

TIMING DETAIL SUMMARY

Timing Summary for 8-bit LFSR

Speed Grade	-4
Minimum period	4.349ns (Maximum Frequency: 229.938MHz)
Minimum input arrival time before clock	4.192ns
Maximum output required time after clock	8.289ns

Timing Details for 8-bit LFSR

(All values displayed in nanoseconds (ns))

Timing constraint	Default period analysis for Clock 'clock'
Delay	4.349ns (Levels of Logic = 1)
Source	internal_reg_1 (FF)
Destination	internal_reg_7 (FF)
Source Clock	clock rising
Destination Clock	clock rising

8-bit LFSR Timing Details for Place and Route

Total REAL time to Placer completion	2 secs
Total CPU time to Placer completion	1 secs
Total REAL time to Router completion	3 secs
Total CPU time to Router completion	1 secs

DELAY DETAILS

Delay Details for 8-bit LFSR

The SCORE FOR THIS DESIGN is	147
The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is	0

The AVERAGE CONNECTION DELAY for this design is	1.212
The MAXIMUM PIN DELAY IS	2.047
The AVERAGE CONNECTION DELAY on the 10 WORST NETS is	1.271

Asynchronous Delay Detail for 8-bit LFSR

Max Delay	**Netname**
2.047	reset_IBUF
1.754	internal_reg<1>
1.528	enable_IBUF
1.497	internal_reg<3>
1.380	internal_reg<7>
1.373	internal_reg<2>
1.052	internal_reg<4>
1.041	internal_reg<6>
1.039	internal_reg<5>
0.600	clock_BUFGP
0.008	clock_BUFGP/IBUFG

B.1 SYNTHESIS REPORT FOR 13-BIT LFSR

---- Source Parameters

Input File Name : shift_reg13.prj

Input Format : mixed

Ignore Synthesis Constraint File : NO

Verilog Include Directory : YES

---- Target Parameters

Output File Name : shift_reg13

Output Format : NGC

Target Device : xqvr300-4-cb228

---- Source Options

Top Module Name	:	shift_reg13
Automatic FSM Extraction	:	YES
FSM Encoding Algorithm	:	Auto
FSM Style	:	lut
RAM Extraction	:	Yes
RAM Style	:	Auto
ROM Extraction	:	Yes
ROM Style	:	Auto
Mux Extraction	:	YES
Mux Style	:	Auto
Decoder Extraction	:	YES
Priority Encoder Extraction	:	YES
Shift Register Extraction	:	YES
Logical Shifter Extraction	:	YES
XOR Collapsing	:	YES
Resource Sharing	:	YES
Multiplier Style	:	lut
Automatic Register Balancing	:	No

---- Target Options

Add IO Buffers	:	YES
Global Maximum Fanout	:	100
Add Generic Clock Buffer(BUFG)	:	4
Register Duplication	:	YES
Equivalent register Removal	:	YES
Slice Packing	:	YES
Pack IO Registers into IOBs	:	auto

---- General Options

Optimization Goal	:	Speed
Optimization Effort	:	1

Keep Hierarchy	:	NO
Global Optimization	:	All Clock Nets
RTL Output	:	Yes
Write Timing Constraints	:	NO
Hierarchy Separator	:	_
Bus Delimiter	:	<>
Case Specifier	:	maintain
Slice Utilization Ratio	:	100
Slice Utilization Ratio Delta	:	5

---- Other Options

lso	:	shift_reg13.lso
Read Cores	:	YES
cross_clock_analysis	:	NO
verilog2001	:	YES
Optimize Instantiated Primitives	:	NO
tristate2logic	:	No

CPU: 6.23 / 8.89 s | Elapsed: 7.00 / 9.00 s

Total memory usage is 65532 kilobytes

TIMING DETAILS

Timing Summary for 13-bit LFSR

Speed Grade	-4
Minimum period	4.349ns (Maximum Frequency: 229.938MHz)
Minimum input arrival time before clock	4.687ns
Maximum output required time after clock	8.498ns
Maximum combinational path delay	No path found

Timing Details for 13-bit LFSR

(All values displayed in nanoseconds (ns))

Delay	4.349ns (Levels of Logic = 1)
Source	internal_reg_0 (FF)
Destination	internal_reg_12 (FF)
Source Clock	clock rising
Destination Clock	clock rising

Timing Details for Place and Route for 13-bit LFSR

Total REAL time to Placer completion	2 secs
Total CPU time to Placer completion	1 secs
Total REAL time to Router completion	2 secs
Total CPU time to Router completion	1 secs

DELAY SUMMARY
Delay Details for 13-bit LFSR

The SCORE FOR THIS DESIGN is	220
The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is	0
The AVERAGE CONNECTION DELAY for this design is	1.841
The MAXIMUM PIN DELAY IS	4.489
The AVERAGE CONNECTION DELAY on the 10 WORST NETS is	1.791

Asynchronous Delay Detail for 13-bit LFSR

Max Delay	Netname
4.489	reset_IBUF
2.866	enable_IBUF
1.800	internal_reg<0>
1.698	internal_reg<2>
1.528	internal_reg<3>
1.360	internal_reg<12>
1.103	internal_reg<10>
1.075	internal_reg<8>
1.075	internal_reg<6>
1.064	internal_reg<9>
1.041	internal_reg<5>
1.039	internal_reg<1>
1.039	internal_reg<11>
1.023	internal_reg<4>

1.003	internal_reg<7>
0.615	clock_BUFGP
0.008	clock_BUFGP/IBUFG

B.3 SYNTHESIS REPORT FOR 16-BIT, 4 TAP PARALLEL LFSR

---- Source Parameters

Input File Name : lfsr_16.prj

Input Format : mixed

Ignore Synthesis Constraint File : NO

---- Target Parameters

Output File Name : lfsr_16

Output Format : NGC

Target Device : xqvr300-4-cb228

---- Source Options

Top Module Name : lfsr_16

Automatic FSM Extraction : YES

FSM Encoding Algorithm : Auto

FSM Style : lut

RAM Extraction : Yes

RAM Style : Auto

ROM Extraction : Yes

ROM Style : Auto

Mux Extraction : YES

Mux Style : Auto

Decoder Extraction : YES

Priority Encoder Extraction : YES

Shift Register Extraction : YES

Logical Shifter Extraction : YES

XOR Collapsing : YES

Resource Sharing : YES

Multiplier Style : lut

Automatic Register Balancing	:	No
--- Target Options		
Add IO Buffers	:	YES
Global Maximum Fanout	:	100
Add Generic Clock Buffer(BUFG)	:	4
Register Duplication	:	YES
Equivalent register Removal	:	YES
Slice Packing	:	YES
Pack IO Registers into IOBs	:	auto
---- General Options		
Optimization Goal	:	Speed
Optimization Effort	:	1
Keep Hierarchy	:	NO
Global Optimization	:	All Clock Nets
RTL Output	:	Yes
Write Timing Constraints	:	NO
Hierarchy Separator	:	_
Bus Delimiter	:	<>
Case Specifier	:	maintain
Slice Utilization Ratio	:	100
Slice Utilization Ratio Delta	:	5
---- Other Options		
lso	:	lfsr_16.lso
Read Cores	:	YES
cross_clock_analysis	:	NO
verilog2001	:	NO
Optimize Instantiated Primitives	:	NO
tristate2logic	:	No

CPU: 7.86 / 10.09 s | Elapsed: 8.00 / 10.00 s

Total memory usage is 60216 kilobytes.

TIMING DETAILS

Timing Details for 16-bit, 4 taps parallel LFSR

(All values displayed in nanoseconds (ns))

Delay	6.549ns (Levels of Logic = 0)
Source	U2_Mshreg_REG<13>_srl_0 (FF)
Destination	U2_Mshreg_REG<13>_0 (FF)
Source Clock	CLK rising
Destination Clock	CLK rising

Timing Details for Place and Route for 16-bit, 4 tap parallel LFSR

Total REAL time to Placer completion	2 secs
Total CPU time to Placer completion	1 secs
Total REAL time to Router completion	50 secs
Total CPU time to Router completion	38 secs

THE DELAY DETAIL SUMMARY

Delay Details for 16-bit, 4 tap parallel LFSR

The SCORE FOR THIS DESIGN is	157
The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is	0
The AVERAGE CONNECTION DELAY for this design is	1.303
The MAXIMUM PIN DELAY IS	2.274
The AVERAGE CONNECTION DELAY on the 10 WORST NETS is	1.341

Asynchronous Delay Detail for 16-bit, 4 taps parallel LFSR

Max Delay	Netname
2.274	CE_IBUF
2.260	U4_Mshreg_REG<0>_2
2.103	U3_Mshreg_REG<11>_1
1.580	U2_Mshreg_REG<13>_0
1.555	FILL_IBUF

1.516	U1_REG<14>
1.304	GLOBAL_LOGIC1_0
1.289	GLOBAL_LOGIC0_0
1.237	GLOBAL_LOGIC1
1.125	GLOBAL_LOGIC0
1.077	DINA
1.039	U1_REG<15>
0.588	CLK_BUFGP
0.511	DIN_IBUF
0.032	Mmux_DINA_Result_SW0/O
0.008	CLK_BUFGP/IBUFG

B.4 SYNTHESIS REPORT FOR MULTICYCLE TAP ACCESS LFSR

---- Source Parameters

Input File Name : sr_16_tap1.prj

Input Format : mixed

Ignore Synthesis Constraint File : NO

---- Target Parameters

Output File Name : sr_16_tap1

Output Format : NGC

Target Device : xqvr300-4-cb228

---- Source Options

Top Module Name : sr_16_tap1

Automatic FSM Extraction : YES

FSM Encoding Algorithm : Auto

FSM Style : lut

RAM Extraction : Yes

RAM Style : Auto

ROM Extraction : Yes

ROM Style : Auto

Mux Extraction : YES

Mux Style	:	Auto
Decoder Extraction	:	YES
Priority Encoder Extraction	:	YES
Shift Register Extraction	:	YES
Logical Shifter Extraction	:	YES
XOR Collapsing	:	YES
Resource Sharing	:	YES
Multiplier Style	:	lut
Automatic Register Balancing	:	No
---- Target Options		
Add IO Buffers	:	YES
Global Maximum Fanout	:	100
Add Generic Clock Buffer(BUFG)	:	4
Register Duplication	:	YES
Equivalent register Removal	:	YES
Slice Packing	:	YES
Pack IO Registers into IOBs	:	auto
---- General Options		
Optimization Goal	:	Speed
Optimization Effort	:	1
Keep Hierarchy	:	NO
Global Optimization	:	All Clock Nets
RTL Output	:	Yes
Write Timing Constraints	:	NO
Hierarchy Separator	:	_
Bus Delimiter	:	<>
Case Specifier	:	maintain
Slice Utilization Ratio	:	100
Slice Utilization Ratio Delta	:	5
---- Other Options		

lso	:	sr_16_tap1.lso
Read Cores	:	YES
cross_clock_analysis	:	NO
verilog2001	:	NO
Optimize Instantiated Primitives	:	NO
tristate2logic	:	NO

CPU: 4.77 / 6.70 s | Elapsed: 5.00 / 7.00 s

Total memory usage is 65532 kilobytes

TIMING DETAIL SUMMARY

Timing Details for Multicycle Tap Access LFSR

(All values displayed in nanoseconds (ns))

Delay	7.734ns (Levels of Logic = 1)
Source	U1 (FF)
Destination	U1 (FF)
Source Clock	CLK4x rising
Destination Clock	CLK4x rising

Timing Details for Place and Route for Multicycle Tap Access LFSR

Total CPU time to Placer completion	1 secs
Total REAL time to Router completion	3 secs
Total CPU time to Router completion	1 secs

THE DELAY DETAILS
Delay Detail for Multicycle Tap Access LFSR

The SCORE FOR THIS DESIGN is	170
The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is	0
The AVERAGE CONNECTION DELAY for this design is	1.367
The MAXIMUM PIN DELAY IS	2.814
The AVERAGE CONNECTION DELAY on the 10 WORST NETS is	1.669

Asynchronous Delay Detail for Multicycle Tap Access LFSR

Max Delay	Netname
2.814	ADDR_FFd1
2.132	DIN_IBUF
1.956	RESET_IBUF
1.879	TAPD
1.730	ADDR_FFd2
1.612	FILL_IBUF
1.450	ADDR_FFd3
1.399	ADDR_FFd4
1.373	Q
1.294	Mmux_D_Result1/O
1.194	_n00021/O
1.033	_n00031/O
0.850	_n00041/O
0.642	CLK4x_BUFGP
0.600	CLK_BUFGP
0.399	N300
0.032	Mmux_D_Result1_SW0/O
0.008	CLK4x_BUFGP/IBUFG
0.008	CLK_BUFGP/IBUFG

B.5 SYNTHESIS REPORT FOR PN GENERATOR

SYNTHESIS OPTIONS SUMMARY

---- Source Parameters

Input File Name : iq_pn_gen.prj

Input Format : mixed

Ignore Synthesis Constraint File : NO

Verilog Include Directory :

---- Target Parameters

Output File Name : iq_pn_gen

Output Format : NGC

Target Device : xqvr300-4-cb228

---- Source Options

Top Module Name	:	iq_pn_gen
Automatic FSM Extraction	:	YES
FSM Encoding Algorithm	:	Auto
FSM Style	:	lut
RAM Extraction	:	Yes
RAM Style	:	Auto
ROM Extraction	:	Yes
ROM Style	:	Auto
Mux Extraction	:	YES
Mux Style	:	Auto
Decoder Extraction	:	YES
Priority Encoder Extraction	:	YES
Shift Register Extraction	:	YES
Logical Shifter Extraction	:	YES
XOR Collapsing	:	YES
Resource Sharing	:	YES
Multiplier Style	:	lut
Automatic Register Balancing	:	No

---- Target Options

Add IO Buffers	:	YES
Global Maximum Fanout	:	100
Add Generic Clock Buffer(BUFG)	:	4
Register Duplication	:	YES
Equivalent register Removal	:	YES
Slice Packing	:	YES
Pack IO Registers into IOBs	:	auto

---- General Options

Optimization Goal	:	Speed
Optimization Effort	:	1
Keep Hierarchy	:	NO

Global Optimization	:	All Clock Nets
RTL Output	:	Yes
Write Timing Constraints	:	NO
Hierarchy Separator	:	_
Bus Delimiter	:	<>
Case Specifier	:	maintain
Slice Utilization Ratio	:	100
Slice Utilization Ratio Delta	:	5

---- Other Options

lso	:	iq_pn_gen.lso
Read Cores	:	YES
cross_clock_analysis	:	NO
verilog2001	:	YES
Optimize Instantiated Primitives	:	NO
tristate2logic	:	No

CPU: 6.13 / 7.81 s | Elapsed: 6.00 / 8.00 s

Total memory usage is 60216 kilobytes

TIMING DETAILS

Timing Details for PN Generator

(All values displayed in nanoseconds (ns))

Delay	6.549ns (Levels of Logic = 0)
Source	Mshreg_srl_q<0>_srl_0 (FF)
Destination	Mshreg_srl_q<0>_0 (FF)
Source Clock	clk rising
Destination Clock	clk rising

PN Generator Timing Details for Place and Route

Total REAL time to Placer completion	0 secs
Total CPU time to Placer completion	1 secs
Total REAL time to Router completion	37 secs
Total CPU time to Router completion	35 secs

THE DELAY SUMMARY REPORT

Delay Details for PN Generator

The SCORE FOR THIS DESIGN is	302
The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is	0
The AVERAGE CONNECTION DELAY for this design is	2.452
The MAXIMUM PIN DELAY IS	5.074
The AVERAGE CONNECTION DELAY on the 10 WORST NETS is	2.849

Asynchronous Delay Detail for PN Generator

Max Delay	Netname
5.074	ShiftEn_IBUF
3.966	FillSel_IBUF
3.513	DataIn_i_IBUF
3.470	Mshreg_srl_q<0>_0
3.342	DataIn_q_IBUF
3.246	Mshreg_srl_i<0>_2
1.708	Mshreg_srl_q<9>_1
1.480	Mshreg_srl_q<5>_4
1.377	Mshreg_srl_i<5>_3
1.311	srl_q<4>
1.305	GLOBAL_LOGIC1_1
1.286	GLOBAL_LOGIC0_3
1.271	GLOBAL_LOGIC0_0
1.235	GLOBAL_LOGIC0
1.230	GLOBAL_LOGIC0_2
1.081	GLOBAL_LOGIC1_0
0.999	GLOBAL_LOGIC1
0.872	GLOBAL_LOGIC0_1
0.838	lfsr_in_q
0.805	lfsr_in_i

B.5 SYNTHESIS REPORT FOR RS CODE GENERATOR

SYNTHESIS OPTIONS SUMMARY

---- Source Parameters

Input File Name	:	rs_code.prj
Input Format	:	mixed
Ignore Synthesis Constraint File	:	NO
---- Target Parameters		
Output File Name	:	rs_code
Output Format	:	NGC
Target Device	:	xqvr300-4-cb228
---- Source Options		
Top Module Name	:	rs_code
Automatic FSM Extraction	:	YES
FSM Encoding Algorithm	:	Auto
FSM Style	:	lut
RAM Extraction	:	Yes
RAM Style	:	Auto
ROM Extraction	:	Yes
ROM Style	:	Auto
Mux Extraction	:	YES
Mux Style	:	Auto
Decoder Extraction	:	YES
Priority Encoder Extraction	:	YES
Shift Register Extraction	:	YES
Logical Shifter Extraction	:	YES
XOR Collapsing	:	YES
Resource Sharing	:	YES
Multiplier Style	:	lut
Automatic Register Balancing	:	No
---- Target Options		
Add IO Buffers	:	YES
Global Maximum Fanout	:	100
Add Generic Clock Buffer(BUFG)	:	4

Register Duplication	:	YES
Equivalent register Removal	:	YES
Slice Packing	:	YES
Pack IO Registers into IOBs	:	auto

---- General Options

Optimization Goal	:	Speed
Optimization Effort	:	1
Keep Hierarchy	:	NO
Global Optimization	:	All Clock Nets
RTL Output	:	Yes
Write Timing Constraints	:	NO
Hierarchy Separator	:	_
Bus Delimiter	:	<>
Case Specifier	:	maintain
Slice Utilization Ratio	:	100
Slice Utilization Ratio Delta	:	5

---- Other Options

lso	:	rs_code.lso
Read Cores	:	YES
cross_clock_analysis	:	NO
verilog2001	:	YES
Optimize Instantiated Primitives	:	NO
tristate2logic	:	No

CPU: 6.63 / 8.30 s | Elapsed: 7.00 / 9.00 s

Total memory usage is 60216 kilobytes

TIMING DETAILS

RS Code Generator Timing Details for Place and Route

Total REAL time to Placer completion	2 secs
Total CPU time to Placer completion	1 secs

Total REAL time to Router completion	46 secs
Total CPU time to Router completion	37 secs

5.3.3.7DELAY DETAILS

Delay summary for RS Code Generator

Delay	6.549ns
Source	U1_Mshreg_int_sigB0<25><0>_srl_2 (FF)
Destination	U1_Mshreg_int_sigB0<25><0>_2 (FF)
Source Clock	Clock rising
Destination Clock	Clock rising

Asynchronous Delay Detail for RS Code Generator

Max Delay	**Netname**
5.063	Enable_IBUF
4.287	Fill_En_B_IBUF
4.224	New_Fill_B
3.759	Fill_En_A_IBUF
3.383	RS_Code
3.192	New_Fill_A
2.051	U0_Mshreg_int_sigA0<25><0>_1
1.997	U1_int_sigB0_20
1.580	U1_int_sigB0_19
1.566	U1_Data_In_B<0>
1.564	U0_int_sigA0_3
1.376	GLOBAL_LOGIC1
1.316	U1_int_sigB0_11
1.301	U1_int_sigB0_7
1.282	U1_int_sigB0_15
1.279	U1_int_sigB0_3

1.099	U1_Mshreg_int_sigB0<25><0>_2
1.080	U1_int_sigB0_9
1.075	U0_Mshreg_int_sigA0<25><0>_0
1.075	U1_int_sigB0_8

B.6 B.5 SYNTHESIS REPORT FOR BIST

SYNTHESIS OPTIONS SUMMARY

---- Source Parameters

Input File Name	:	stand_alone_bist.prj
Input Format	:	mixed
Ignore Synthesis Constraint File	:	NO
Verilog Include Directory	:	

---- Target Parameters

Output File Name	:	stand_alone_bist
Output Format	:	NGC
Target Device	:	xqvr300-4-cb228

---- Source Options

Top Module Name	:	stand_alone_bist
Automatic FSM Extraction	:	YES
FSM Encoding Algorithm	:	Auto
FSM Style	:	lut
RAM Extraction	:	Yes
RAM Style	:	Auto
ROM Extraction	:	Yes
ROM Style	:	Auto
Mux Extraction	:	YES
Mux Style	:	Auto
Decoder Extraction	:	YES
Priority Encoder Extraction	:	YES
Shift Register Extraction	:	YES
Logical Shifter Extraction	:	YES
XOR Collapsing	:	YES
Resource Sharing	:	YES
Multiplier Style	:	lut
Automatic Register Balancing	:	No

---- Target Options

Add IO Buffers	:	YES
Global Maximum Fanout	:	100
Add Generic Clock Buffer (BUFG)	:	4
Register Duplication	:	YES
Equivalent register Removal	:	YES
Slice Packing	:	YES
Pack IO Registers into IOBs	:	auto

---- General Options

Optimization Goal	:	Speed
Optimization Effort	:	1
Keep Hierarchy	:	NO
Global Optimization	:	AllClockNets
RTL Output	:	Yes
Write Timing Constraints	:	NO
Hierarchy Separator	:	_
Bus Delimiter	:	<>
Case Specifier	:	maintain
Slice Utilization Ratio	:	100
Slice Utilization Ratio Delta	:	5

---- Other Options

lso	:	stand_alone_bist.lso
Read Cores	:	YES
cross_clock_analysis	:	NO
verilog2001	:	YES
Optimize Instantiated Primitives	:	NO
tristate2logic	:	NO

CPU: 4.59 / 6.39 s | Elapsed: 5.00 / 6.00 s

Total memory usage is 60216 kilobytes

TIMING DETAILS

Timing Details for BIST

(All values displayed in nanoseconds (ns))

Delay	4.503ns (Levels of Logic = 1)
Source	analyzer_lfsr_reg_6 (FF)
Destination	analyzer_lfsr_reg_0 (FF)
Source Clock	clock rising
Destination Clock	clock rising

BIST Timing Details for Place and Route

Total REAL time to Placer completion	2 secs
Total CPU time to Placer completion	1 secs
Total REAL time to Router completion	2 secs
Total CPU time to Router completion	1 secs

THE DELAY DETAILS

Delay Detail for BIST

The SCORE FOR THIS DESIGN is	320
The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is	0
The AVERAGE CONNECTION DELAY for this design is	2.458
The MAXIMUM PIN DELAY IS	5.622
The AVERAGE CONNECTION DELAY on the 10 WORST NETS is	3.725

Asynchronous Delay Detail for BIST

Max Delay	Netname
5.622	reset_IBUF
4.320	analyzer_lfsr_reg<8>
4.057	analyzer_lfsr_reg<7>
3.786	analyzer_lfsr_reg<6>
3.577	serial_in_IBUF
3.494	analyzer_lfsr_reg<0>
3.418	analyzer_lfsr_reg<9>
3.232	analyzer_lfsr_reg<5>
3.058	analyzer_lfsr_reg<3>
2.714	analyzer_lfsr_reg<2>
2.682	analyzer_lfsr_reg<1>
2.549	analyzer_lfsr_reg<4>
0.630	clock_BUFGP
0.008	clock_BUFGP/IBUFG