INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Especialidad en Diseño de Sistemas en Chip

Reconocimiento de Validez Oficial de Estudios de nivel superior
según Acuerdo Secretarial 15018, publicado en el *Diario Oficial de la Federación*
el 29 de noviembre de 1976

DEPARTAMENTO DE ELECTRÓNICA, SISTEMAS E INFORMÁTICA

Test Module Design for ITESO TV1 *SerDes*

Tesina para obtener el grado de:

Especialista en diseño de sistemas en chip

Presenta *Ricardo Godínez Maldonado*

Asesores:

Víctor Avendaño Fernández, Alexandro Girón Allende, Esteban Martínez Guerrero

Guadalajara, Jalisco, Diciembre 2015

ACKNOWLEDGEMENTS

I would like to thank firstly my family, girlfriend and close friends which were of great support during the development of this project. Thank Conacyt for providing the scholarship, without it I would have not been able to achieve this goal. But most of all, thank for the guidance, counseling, patience and effort given by the teachers and counselors that helped me develop the skills needed to develop this project.

ABSTRACT

This document describes the work performed from January to November 2015 on the development of the Testing Module for the ITESO TV1 *SerDes* chip.

The objective of this project is to design, test and manufacture a *SerDes* (Serializer/Deserializer) system for high speed communications on a 180 nm technology using the **Cadence** tools for integrated circuit design.

The first section describes the background of the SerDes system; this section will explain the basic architecture of a SerDes Chip, its characteristics and applications.

Next, an explanation of the technology available, some testing concepts & methodologies and the specifications for designing the chip are presented.

Following this, a design proposal is presented by doing a summary of the requirements gathered for the implementation of this module and a description of its functionality.

After that, a description on how the Testing Module design is implemented, this section shows the RTL design specifications and verification results.

Lastly, the logic and physical synthesis of this module is explained, showing the results of the verification of this finalized RTL design.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The semiconductor industry is constantly evolving; every year the technologies are getting smaller, faster and more complex. Due to this, the task of testing a chip´s performance and reliability is critical. New testing methodologies are emerging in order to satisfy the testing needs of a chip. One of these methodologies consists on performing the testing inside the chip. The idea is to create testing blocks within the chip´s design, so the chip can test itself; this is called Design for Testability (DFT).

The project proposed is to design a SerDes communication system using an **IBM** technology kit and send it to a chip fabrication provider. This document will describe the implementation of DFT on the ITESO TV1 SerDes chip design

Specifications of this testing module were gathered from research and feedback from the designers working on the functional blocks of the SerDes. With this information, the testing module architecture was designed and verified. This was achieved by creating a Hardware Description Language (HDL) file that described the behavior of the circuit. Then this Register Transfer Level (RTL) description of the circuit was verified using a logic simulator, ***ModelSim***.

Following this, the Testing Module design was merged with the other RTL designs of the chip. The resulting *RTL* was verified and optimized on ***ModelSim*** by creating and running Test Benches on it.

The next step was to perform a *logic synthesis* of the design by loading the *RTL* files on ***Cadence's RC Compiler*** tool. This produced a finalized HDL file that is mapped directly to the Standard Digital Cells library contained on the ***IBM***'s design kit.

To verify the functionality of this finalized HDL, test benches were used to simulate the circuit at gate level using **ModelSim**.

Finally, with the help and collaboration of the digital block designers, the physical synthesis of the digital blocks was performed. This preliminary chip *layout* was created using ***Cadence´s Encounter*** tool.

To create a tape-out of the SerDes chip, this layout needs to be imported to **Virtuoso** and merged with the layouts of the analog blocks of the chip.

CHAPTER 1 - BACKGROUND OF *SERDES* SYSTEM AND TESTING

1.1 Definition of SerDes system and its advantages

A *SerDes* system is a high speed communication system used in today's industry. The term *SerDes* term comes from Serializer/Deserializer and the main function of this device is to receive serial binary data and convert it into parallel and vice versa [1].

One benefit of using these method vs a parallel data transmission is the reduction of complexity, cost, and space on an electronic board by reducing the number of buses needed to connect various devices [2].

A serial system can reach high transmission speeds and it´s certain that all the required bits will arrive at the given time, which ensures the integrity of the data at high speeds.

Another benefit of this system is that it only requires a four bus system for the implementation, two for transmission and two for reception, which reduces significantly the area occupied by the wiring in a circuit board among other benefits. Figure 1.1 shows a diagram where the difference between parallel and serial can be observed.

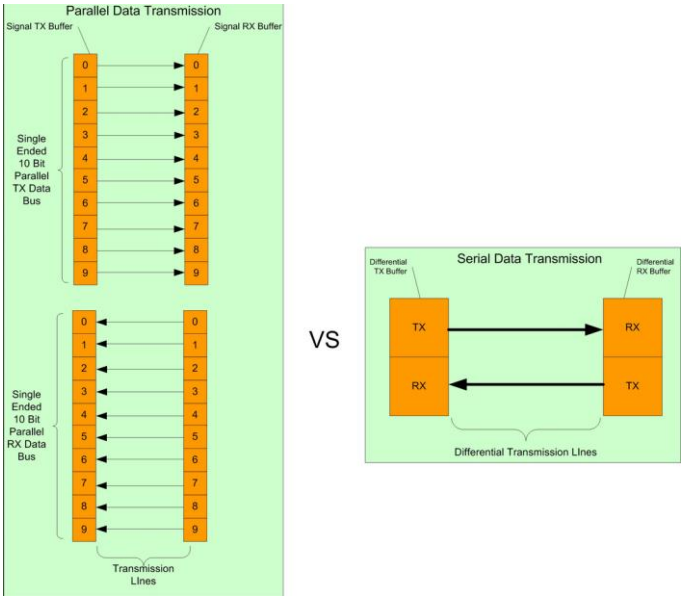

Figure 1.1 Serial VS Parallel Communications [2]

In short, the use of serial communication solved many of the problems caused by parallel communications, and so a standard protocol for serial communications was created, the **Peripheral Component Interconnect Express** protocol (**PCI Express**), which is the designated protocol that will be used on the *SerDes* design.

1.2.1 PCI Express communication protocol description

PCI Express, also known as **PCIE**, is currently used as the standard for serial communications between electronic devices. It was implemented by the **PCI-SIG** (**PCI Special Interest Group**) due to the need of improving the old Peripheral communications on a computer mother board that used parallel communication. This type of communication caused noise on the signal and propagation delay, so it represented a performance problem at higher clock speeds [3].

The main idea of this protocol is to take the parallel data and serialize it in order to be transmitted, this requires less wiring and avoid data loss at higher clock speeds since the data travels in a single stream.

It uses 2 wires for data transmission and another 2 wires for data reception, this is called Differential Transmission, in other words, a wire carries a positive signal, and the other wire carries a negative one, which is a mirrored version of the positive one.

The benefit of implementing a differential transmission line, is to avoid data corruption caused by electromagnetic interference using a technique called cancellation. At the time the differential signal arrives to the chip, it detects differences between the negative and positive lines; this difference between signals will be branded as noise and removed from the signal producing a noise free transmission [4]. Figure 1.2 shows an example on how a Differential Transmission handles noise and cleans the signal.

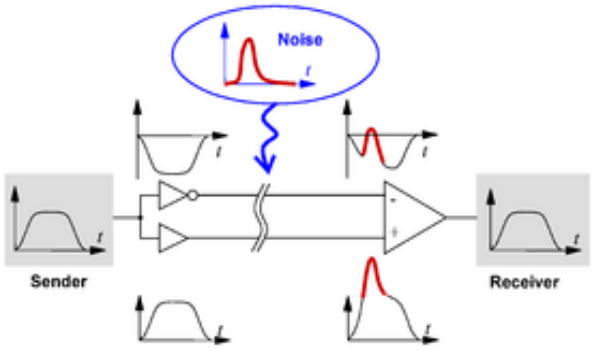

Figure 1.2 Noise Cancelation Example [4]

1.2.2 PCI Express Protocol Specifications

The selected PCI version to be implemented is the PCI Express Version 1.0. The characteristics in terms of bandwidth, transfer rate and encoding that the device must achieve to comply with the PCIE Version 1 are shown on Table 1.1.

PCI Express version	Line code	Transfer rate[a]	Bandwidth	
			Per lane[a]	In a ×16 (16-lane) slot[a]
1.0	8b/10b	2.5 GT/s	2 Gbit/s (250 MB/s)	32 Gbit/s (4 GB/s)

Table 1.1 PCIE Version 1.0 Transmission Specifications [5]

1.2.3 8b/10b encoding, structure and operation

Another **PCIE** protocol specification is the use of a binary data encoding called 8b/10b. The idea behind this encoding is to add 2 additional bits to the transmission package for every 8 bits of data, the 2 bits added are obtained using a lookup table. Figure 1.3 shows an example of a small portion of an 8b/10b encoding lookup table.

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D0.0	000 00000	100111 0100	011000 1011
D1.0	000 00001	011101 0100	100010 1011
D2.0	000 00010	101101 0100	010010 1011
D3.0	000 00011	110001 1011	110001 0100
D4.0	000 00100	110101 0100	001010 1011
D5.0	000 00101	101001 1011	101001 0100
D6.0	000 00110	011001 1011	011001 0100
D7.0	000 00111	111000 1011	000111 0100
D8.0	000 01000	111001 0100	000110 1011
D9.0	000 01001	100101 1011	100101 0100

Table 1.2 8b/10b encoding table example []

Using this code has several advantages for data transmission:

- Maintaining a DC – Transmitting to many ones (high) or to many zeroes (low), can cause data loss due to the intrinsic capacitance of the chip when attempting to transition from high to low or low to high voltage levels and not reaching the logic gate´s threshold.

- Detecting parity data errors – It's well defined by the 8b/10b encoding table the maximum number of ones (high) and zeroes (low) a data transmission must have, so it is pretty easy to identify when there is a mismatch on the parity of the data.

- Data Integrity – By design there are no more than 5 zeroes or 5 ones in a row in an 8b/10b encoding, so it ensures a good clock recovery.

1.3.1 System-level design specifications of SerDes

After describing the main features of a *SerDes* design, specifications for ITESO TV1 were created. These are the selected system requirements that the ITESO *SerDes* TV1 will have:

- Manufacturing technology: CMOS 180nm
- Communication protocol: PCIE 1.0
- VDD: 1.8V
- VSS: 0V
- Area: 1.5mm X 1.5mm
- Operation frequency: 1.5Gbps/2.5Gbps
- Clock frequency: 0.75Ghz/1.25Ghz
- Packaging: DIP40

1.3.2 Manufacturing Technology and Tools

The *CMOS* technology to be used to manufacture this system is 180 nm, which supply voltage is 1.8V. The tools available to design this chip on this technology are:

- Cadence Virtuoso – Schematic and *Layout* virtualization, analysis and simulation Tool.
- Model Sim – *RTL* simulation and debugging tool.

- IBM's cmrf7sf Design Kit – 180 nm technology kit available to manufacture a SerDes chip (Provided by Mosis)
- DIP 40 chip to package the SerDes chip (provided by Mosis).
- Cadence RC Compiler – *RTL* debugging and *Logic synthesis* generation tool.
- Cadence Encounter – Physical synthesis generation tool.

1.4 Description of a SerDes architecture

A *SerDes* system uses both Analog and Digital Blocks. The Analog Blocks are responsible for receiving and transmitting the signal entering and exiting the chip using analog devices designed to reduce jitter and produce a correct *CMOS* level signal with the lowest noise possible.

The Digital Blocks are responsible for handling incoming and outgoing binary data from the system by encoding, decoding, and transforming data from parallel to serial and vice versa. Figure 1.4 shows the hierarchy levels of the main blocks of the *SerDes* Chip Design.

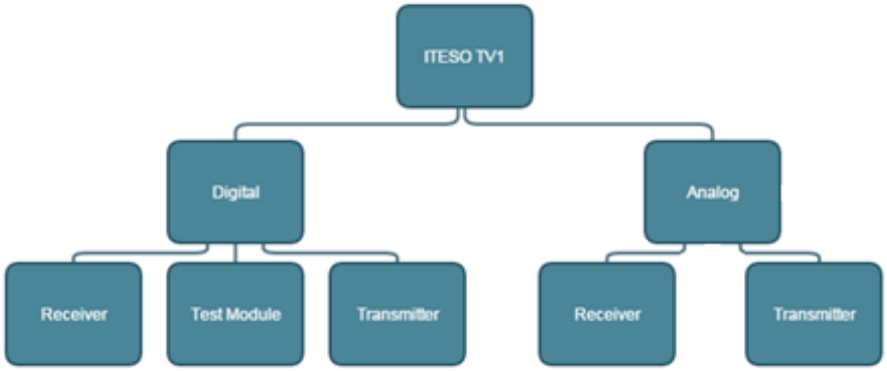

Figure 1.3 *SerDes* Block Hierarchies

The system will be divided into the following five main functional Blocks which can be found in Figure 1.5.

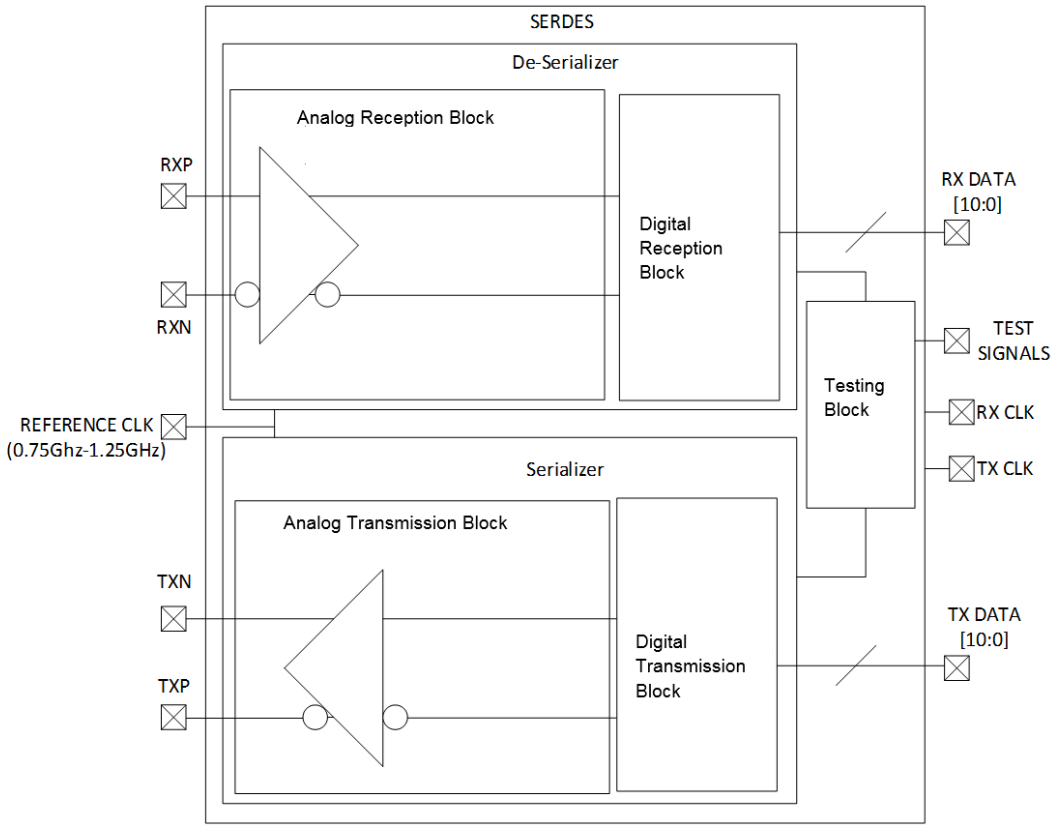

Figure 1.4 Top Level SerDes Block Diagram [6]

- Analog Reception Block: receives a differential serial input signal, amplify it to a correct *CMOS* signal level, filter noise and deliver a single ended serial signal to be processed by the Digital Receiver Module.

- Digital Reception Block: responsible for receiving data from the Analog Receiver Block and process it for its parallel conversion. At this stage a CDR (Clock Data Recovery) module is needed, which will determine the clock used to transmit the data and to set the reception frequency. Then this block takes the data received and decodes it, since the data comes in 8b/10b encoding. After decoding, the data will be converted from serial to parallel and output it outside the chip [7].

- Digital transmission block: receives parallel data input bits, encode the data in 8b/10b code and convert them to a train of serial data to be passed over to the Analog Transmission block to be transmitted.

- Analog Transmitter: in charge of receiving the serial data from the Digital Transmitter and treat it to be transmitted successfully outside the chip. It

amplifies, modules and equalizes de signal to maintain a correct DC level and appropriate signal strength.

- Testing Block: responsible of performing tests that are embedded on the chip´s architecture by using control signals to test the functionality of the device once it´s manufactured. Methodologies such as BIST (Explained on Chapter 2) can be used for designing the chip for testability. One of the tests that can be implemented is the ability to "bypass" any of the blocks of the system to test them independently of the other blocks. Another viable test is to connect the Digital Receiver Block output directly to the Digital Transmitter block input and inject data to the Analog Receiver to loop the data across the whole chip to do a full test of the design.

1.5 Integrated Circuits Testing - Basic concepts

The main purpose of performing tests on an Integrated Circuit (IC) is to detect design and fabrication errors to be able to correct them in a timely manner, this way ensuring that acceptable quality chips will be manufactured [8].

A chip may fail to pass a test for any of the following reasons:

- The test was wrong or designed incorrectly.
- The fabrication Process was faulty.
- The chip design was incorrect.
- The chip´s specifications were incorrect.

Figure 2.1 illustrates how the testing takes part on every step of an IC design flow and gives guidance to the IC designers on where to make adjustments in order to comply with the product specifications, this evaluation process is called Failure Mode Analysis (FMA).

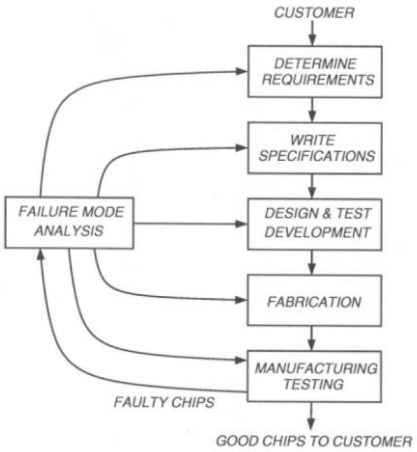

Figure 1.5 IC design & fabrication realization process (FMA) [8]

Testing has a very important role in chips design and manufacturing, it will greatly influence the final product´s level of quality and cost, in other words, having a good testing plan will ensure product quality at lower costs [8].

1.6 Built-In Self-Test (BIST)

The set of design methods and techniques that are used for creating an architecture that enables testing capabilities is called “Design for Testability” (DFT). The idea behind the ITESO TV1 Testing Module is to create digital blocks that are capable of evaluating if the chip is performing as expected using DFT techniques.

One common method for implementing DFT on a Digital Circuit is to execute the process illustrated on Figure 2.2:

[8]”The role of testing is to detect whether something went wrong and the role of diagnosis is to determine exactly what went wrong, and where the process needs to be altered. Therefore, correctness and effectiveness of testing is most important for quality products (another name for perfect products)”.

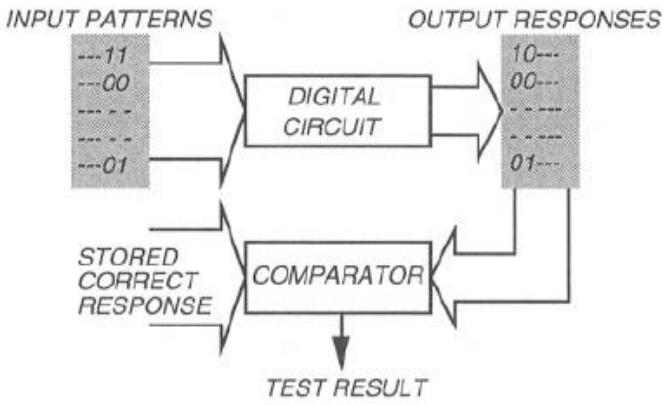

Figure 1.6 Digital Circuit Testing Process [8]

1. Apply test vectors (Binary Patterns) to the inputs of the digital circuit
2. Compare the response of the circuit to the expected response.
3. The circuit is good if the response matches the expected one.

Built-In Self-Test (BIST) is automating and embedding this process on a chip. BIST gives a chip the capability of performing this tests internally and determine if a block is working or not.

To add BIST functionality on a design, 2 Digital blocks can be implemented a Linear Feedback Shift Register (LFSR) and a Comparator. This blocks will be discussed in detail on section 2.3 and 2.4.

1.7 Linear Feedback Shift Register (LFSR) Concepts

A Linear Feedback Shift Register, also known as LFSR, is a digital hardware device capable of creating a pseudo random pattern sequence. This repeatable sequence of values is used as a test vector to verify the functionality of a digital circuit [8].

This device uses Flip-Flop Registers connected in series and linear XOR gates that feed back onto the registers [9]. A basic diagram of an LFSR appears on Figure 2.3.

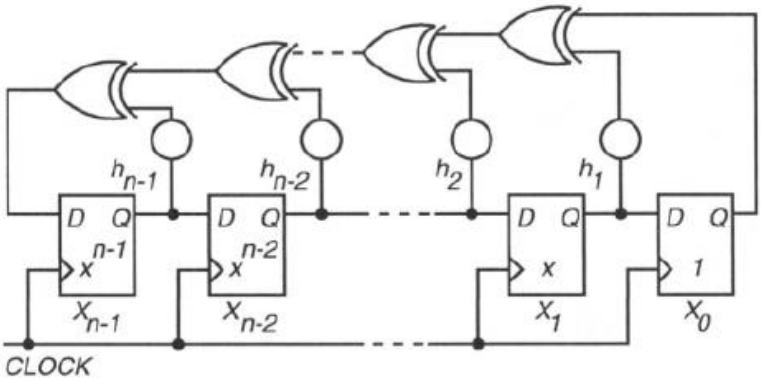

Figure 1.7 Linear Feedback Shift Register [8]

This is a finite state machine, so there are a determined number of values it can generate before starting the sequence over. This number can be calculated by raising 2 to the power of the number of Flip Flops on the System.

The pattern this device will follow is determined by the initial value the Flip Flop registers have at the beginning of the sequence generation, this initial value is called a *seed* [9].

The typical way of inserting a *seed* on the device is by connecting multiplexers that shift when the Registers need to obtain their initial value, Figure 2.4 shows a basic architecture of a 4 bit Register LFSR [10].

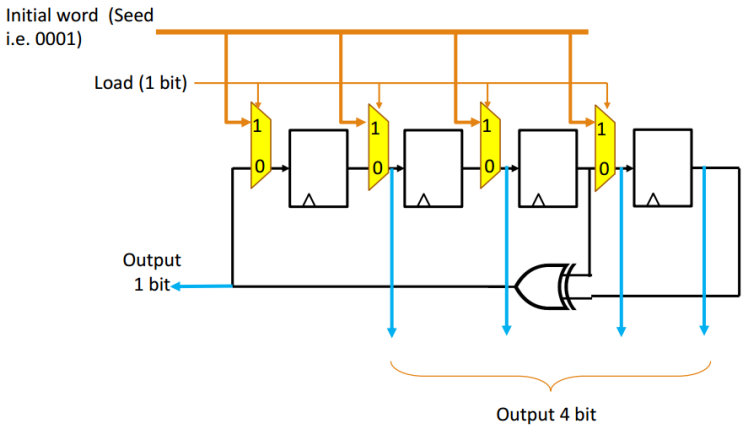

Figure 1.8 Four bit LFSR [10]

After the initial *seed* is injected, the multiplexer will shift and connect the output of the registers to the input of the next register. Each clock cycle the registers will shift their

value to the next register. The first register will acquire the value that the XOR gate is outputting, causing that the value of the first register to seem random. This will continue until all the registers obtain the value they had on the first cycle and repeat the process [10].

1.8 Comparator Concepts

A comparator is basically comprised of 2 memory blocks that need to be compared to each other and output if they are equal or not. The first memory block will be already loaded with all the values this digital block is expected to have.

With the help of a controller block, the second memory block will store the data output of the digital circuit being tested.

After storing the data, this device will compare the values between the 2 memory blocks and check if the expected result is equals to the value obtained from the block being tested and feed out the outcome. Figure 2.5 represents a basic comparator diagram showing its basic blocks [8].

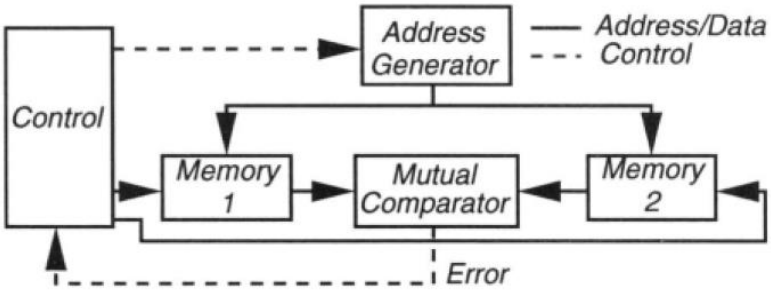

Figure 1.9 Comparator Diagram [8]

CHAPTER 2 - TESTING MODULE PLANNING DEVELOPMENT

From this chapter onward, figures containing diagrams or waveforms will be shown to explain how a functional block works and its verification results. To understand this figures better, the signals participating on this figures will be wrapped in "{}" and the block names will be surrounded by "||". This will help locate the signals and blocks that the text is referring to on these figures.

2.1 Testing module requirements gathering

The requirements obtained from the designers of the Analog Receiver, Digital Receiver, Analog Transmitter and Digital Transmitter were defined by the needs that each module has to be tested.

This section will explain the main concerns for testing the modules comprising the ITESO TV SerDes chip.

Analog Receiver:

The main interest for testing this module is to observe the signal before it reaches the next module. Since this is a high frequency signal, it was determined that the signal needed to be slowed down in order to observe it outside the chip using an oscilloscope. For this, a series of frequency dividers were implemented to processes this signal to output it at a low frequency so the amplitude and integrity of the signal delivered by the Analog Receptor could be observed. In order to prevent the frequency divider block from disrupting the signal, a series of buffers were put in place at the output port of this module.

Analog Transmitter:

The testing performed to this module consists of injecting a serial signal into the module and measure the result. For this, a Linear Feedback Shift Register (LFSR) block was implemented to generate a pseudo random pattern of serial data to be injected into the module and then measure the output to see if the transmitted data is what it was expected.

Digital Receiver:

For testing this module, the capability to bypass the rest of the system and observe the functionality of this module with no interference is needed. Another requirement is to be able to inject a series of pseudo random patterns and test if the resulting signal of the module is what it was expected. To achieve this goal, a series of multiplexers, the LFSR and the comparator were connected to the module.

Digital Transmitter:

The testing objective of this module is to bypass the rest of the system and observe the functionality of this module with no interference. It is also required to be able to inject a series of pseudo random patterns and test if the resulting signal of the module is what it was expected. To achieve this goal, a series of multiplexers, the LFSR and the comparator were implemented.

6 blocks were created to meet the testing requirements:

- Linear Feedback Shift Register Block (LFSR)
- Comparator Block
- Frequency Divider Block
- Bypass and Lopping Multiplexers
- Serial Settings Input Block
- Multiplexer Selection Decoder Block

2.2 Testing Module Implementation

In order to meet the requirements and specifications of the *SerDes* testing needs, a microarchitecture interconnecting the functional modules of the design was created. This architecture is shown on Figure 2.1.

A series of multiplexers |A, B, C, D, & E|, a Decoder |muxDecode|, a Linear Feedback Shift Register |LFSR|, a frequency divider |Freq Div|, and a comparator |Comp| that interact with the main 4 modules of the chip, the Analog Receiver |RX Analog|, the Digital Receiver |RX Digital|, the Digital Transmitter |TX Digital|, and the Analog Transmitter |TX Analog_P & TX Analog_N|. The main purpose of these multiplexers is to perform bypass, lopping and Built-In Self-Test (BIST) functionalities depending on the configuration mode settings (refer to section 2.4).

To operate the SerDes, the first step is to set the configuration settings of the chip. These settings will control the multiplexers |A, B, C, D, & E|, configure the Analog Transmitter |TX Analog_P & TX Analog_N| and the comparator |Comp|. The settings will be stored on the Serial Configuration Block |serConfig| (explained on section 2.2).

After configuring the chip, the SerDes will perform one of the 14 operation modes available by controlling the multiplexers and the testing blocks.

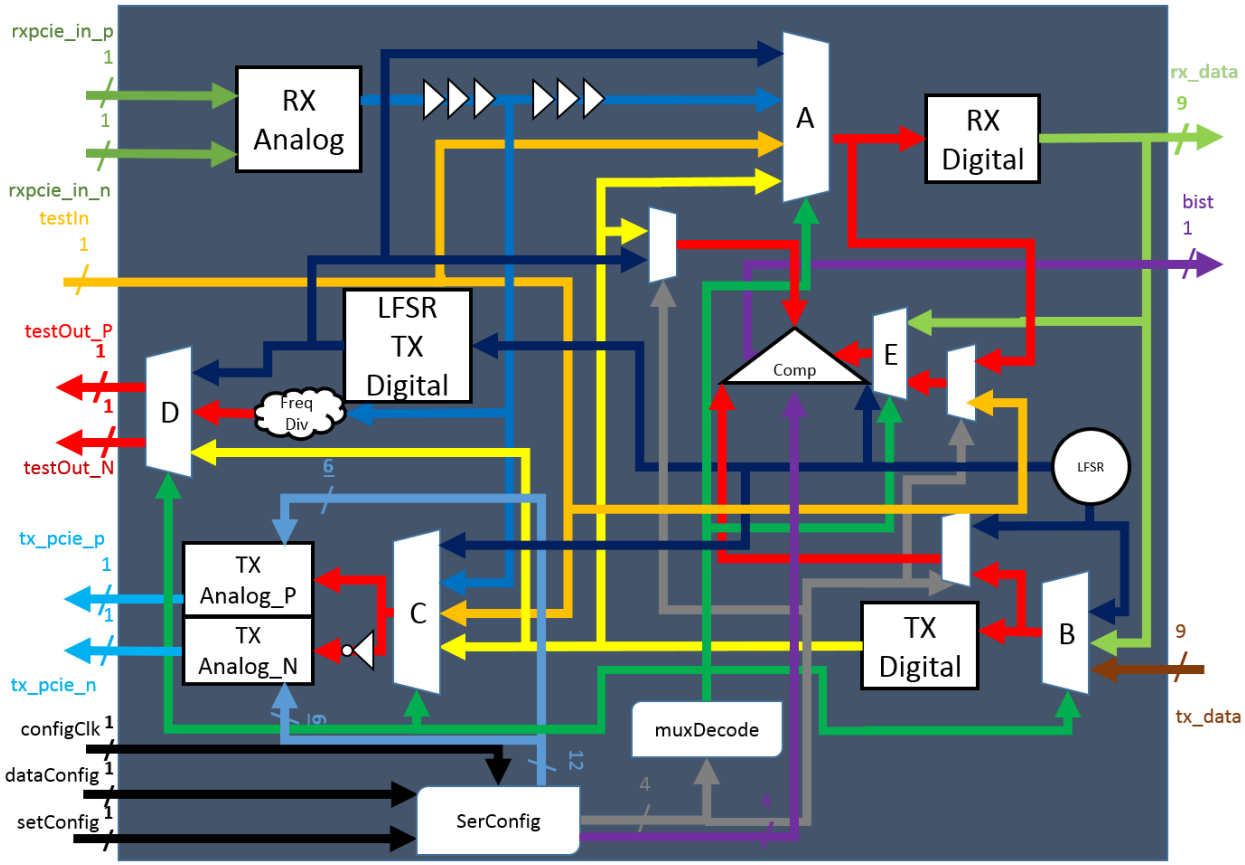

Figure 2.1 Testing Module Microarchitecture

2.2.1 Testing Module Serial Configuration Block

The maximum number of pins allowed by the packaging in which the chip will be fabricated is 40 pins, in order to avoid exceeding the maximum number of pins; a Digital I/O block was implemented to receive the necessary configurations and calibration settings of the chip serially on one of the input pins of the Chip. Figure 2.2 shows the microarchitecture of the Serial Settings Input Block, found as |SerConfig| on Figure 2.1.

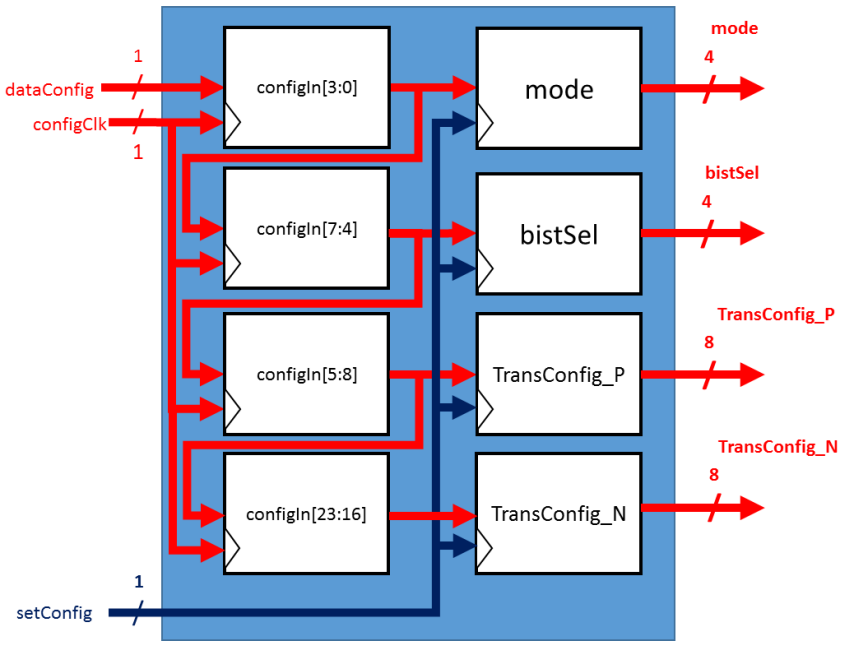

Figure 2.2 Serial Settings Input Block

This block will store in Flip-Flop Registers the configuration settings for 3 blocks on the *SerDes* chip:

- The Comparator Block BIST querying signal |bistSel| (explained on section 2.6)
- The Multiplexer Selection Decoder Block |mode| (explained on section 2.4)
- The Analog Transmitter |TransConfig_P & TransConfig_N| impedance, equalization and amplitude settings.

The block will receive 24 bits of serial data {dataConfig} which will be stored on a shift register array by clocking these registers externally {configClk}. Once the configuration data is completely received, a signal to save this configuration will be triggered {setConfig} and will be saved on their corresponding configuration registers to be used on their respective modules.

Figure 2.3 shows the testing results of this module, demonstrating that it behaves as expected. The test consists on injecting a serial input {dataConfig} with the following configuration settings: {mode}=4'd7, {bistSel}=4'd10, {TransConfig_P}= 8'b11111111, and {TransConfig_N}= 8'b01010101. These configurations were stored successfully {mode, bistSel, TransConfig_N & TransConfig_P}.

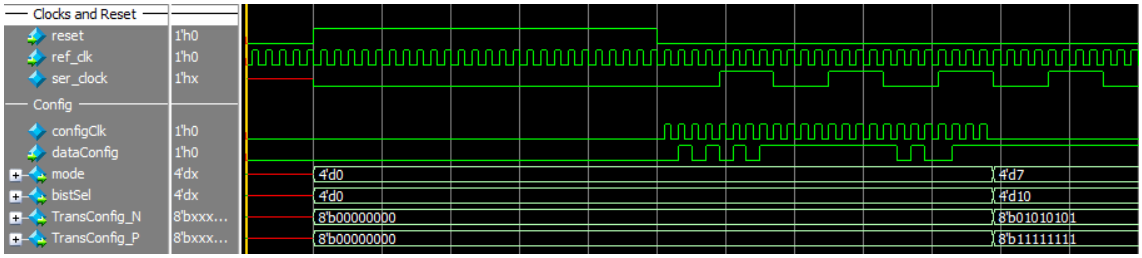

Figure 2.3 Serial Settings Input Block Test Waveform

2.3 Multiplexer Selection Decoder Block:

After storing the configurations of the multiplexers |mode|, as shown on Figure 2.2, these settings are passed over to the Multiplexer Selection Decoder block, referred as “muxDecode” on Figure 2.1.

This block will decode a 4 bit signal {mode} and output the resulting signal {sel} to be connected to the selector bits used by every multiplexer in the Bypass and Lopping Multiplexers block. Figure 2.4 shows the microarchitecture of this module.

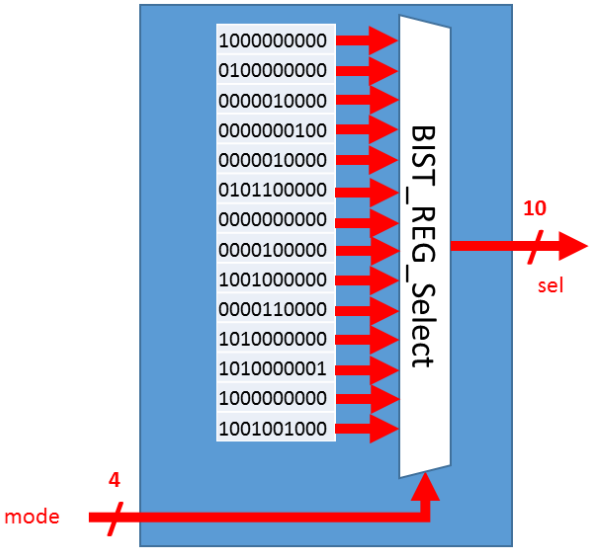

Figure 2.4 Multiplexer Selection Decoder Block (muxDecode)

Figure 3.5 shows the results of the simulation of this block. The selector bits {sel} of the multiplexers |A, B, C, D, & E| appear on the waveform, they shift their selector {sel} value each time the multiplexer configuration {mode} stored on the Serial Configuration Block

changes (see section 3.2.1). The value this selectors {sel} have, depend on the truth table on Table 2.1.

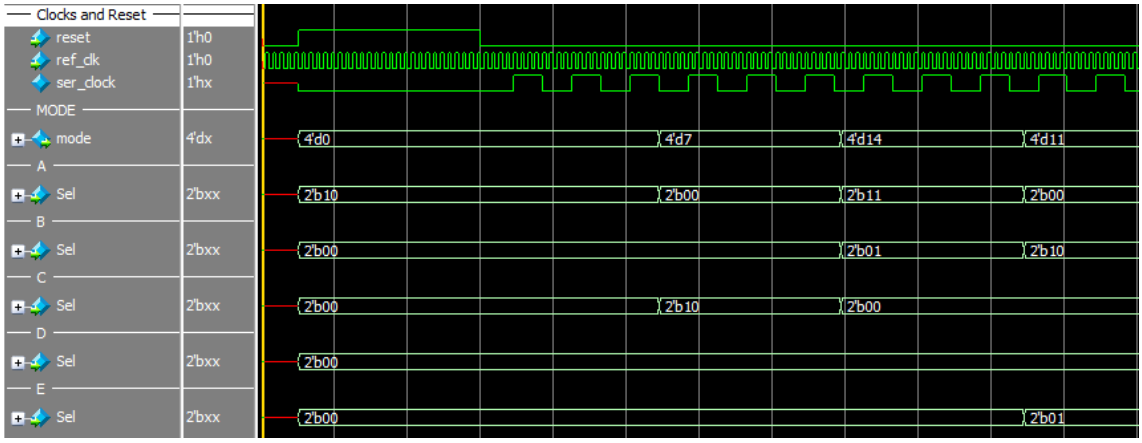

Figure 2.5 Multiplexer Selection Decoder Block test waveform

2.4 Bypassing and Lopping Multiplexers

A series of multiplexers interconnect the functional modules of the design to perform various operation modes. Each of these modes range from the chip´s normal operating mode, bypass modes and testing modes.

15 operating modes were created to meet the Testing Module specifications. These modes are defined by the truth table on Figure 2.6, which assign the selector value that each of the 5 multiplexers on the design must have in order to execute the desired mode, these multiplexers selector bits appear as MuxSelA, MuxSelB, MuxSelC, MuxSelD and MuxSelE on Figure 2.6.

MODE	MuxSelA	MuxSelB	MuxSelC	MuxSelD	MuxSelE
0	10	00	00	00	00
1	01	00	00	00	00
2	00	00	01	00	00
3	00	00	00	01	00
4	00	00	01	00	00
5	01	01	10	00	00
6	00	00	00	00	00
7	00	00	10	00	00
8	10	01	00	00	00
9	00	00	11	00	00
10	10	10	00	00	00
11	00	10	00	00	01
12	10	00	00	00	01
13	10	01	00	10	00
14	11	01	00	00	00

Table 2.1 *SerDes* Operating Modes Truth Table

These operating modes are activated based on the output signal of the |muxDecode| block. Depending of the operating mode, these multiplexers will route the signals on the *SerDes* in order to perform the tests described above.

The Figure 2.5 shown on the previous section, demonstrates a simulation on which the selector {sel} value of each multiplexer is influenced depending of the Operating Mode {mode}. A brief description for each Operation Mode is shown on Table 2.1 and an exhaustive explanation of every operation mode can be found on Chapter III of this document.

Mode	Mode Name	Description
0	Normal Mode	Normal Operation of the *SerDes*, Receiver and Transmitter Modules work independently.

1	RXD	Isolate the Digital Receiver for testing
2	TXA	Isolate the Analog Transmitter for testing
3	RXA Frequency Divider	Isolate the Analog Receiver and run the signal through a frequency divider, then observe the signal outside the chip.
4	TXD	Isolate the Digital Transmitter for testing
5	Digital RX Loopback	Bypass the Analog Receiver and the Analog Transmitter to test the Digital modules. Inject a signal to the Digital Receiver, pass it over to the Digital Transmitter and observe the result.
6	TX Digital Loopback	Bypass the Analog Receiver and the Analog Transmitter to test the Digital modules. Inject a signal to the Digital Transmitter, pass it over to the Digital Receiver and observe the result.
7	RX Analog Loopback	Bypass the Digital Receiver and the Digital Transmitter to test the Analog modules. Inject a signal to the Analog Receiver, pass it over to the Analog Transmitter and observe the result.
8	RXA Full Loop Back & BIST	Interconnect the modules to perform a full loop test. Starting with the Analog Receiver, then the Digital Receiver, the Digital Transmitter, and observing the output at the Analog Transmitter. The input & output signals are fed to the comparator to verify the output has the expected result which will be shown on the {bist} pin of the chip.
9	LFSR TXA	Inject the Analog Transmitter with a serial pseudo random pattern and observe the results.
10	LFSR TXD Full Loopback & BIST	Inject the Digital Transmitter with a parallel pseudo random pattern and loop the signal through the Analog Transmitter, the Analog Receiver and observe the result on the Digital Receiver. The input & output signals are fed to the comparator to verify the output has the expected result which will be shown on the {bist} pin of the chip.
11	LFSR TXD Digital Loopback & BIST	Inject the Digital Transmitter with a parallel pseudo random pattern and pass the signal to the Digital Receiver and observe the result.

		The input & output signals are fed to the comparator to verify the output has the expected result which will be shown on the {bist} pin of the chip.
12	TXD Full Loopback & BIST	Interconnect the Analog Receiver with the Analog Transmitter to perform a full loop test starting with the Digital Transmitter, then the Analog Transmitter, the Analog Receiver and the Digital Receiver and observe the result. The input & output signals are fed to the comparator to verify the output has the expected result which will be shown on the {bist} pin of the chip.
13	LFSR RXA Full Loopback & BIST	Inject the Analog Receiver with a serial pseudo random pattern and loop the signal through the other 3 modules to be verified when it exits through the Analog Transmitter. The input & output signals are fed to the comparator to verify the output has the expected result which will be shown on the {bist} pin of the chip.
14	LFSR RXD Digital Loopback & BIST	Inject the Digital Receiver with a serial pseudo random pattern and feed the signal to the Digital Transmitter, then observe the result. The input & output signals are fed to the comparator to verify the output has the expected result which will be shown on the {bist} pin of the chip.

Table 2.2 *SerDes* Operating Modes Description Table

2.5 Linear Feedback Shift Register Block (LFSR)

The Linear Feedback Shift Register block, also known as LFSR, is in charge of creating a Pseudo Random Pattern Signal which will be injected into the functional modules to evaluate if they are working properly, this block was implemented with the microarchitecture shown on Figure 2.6.

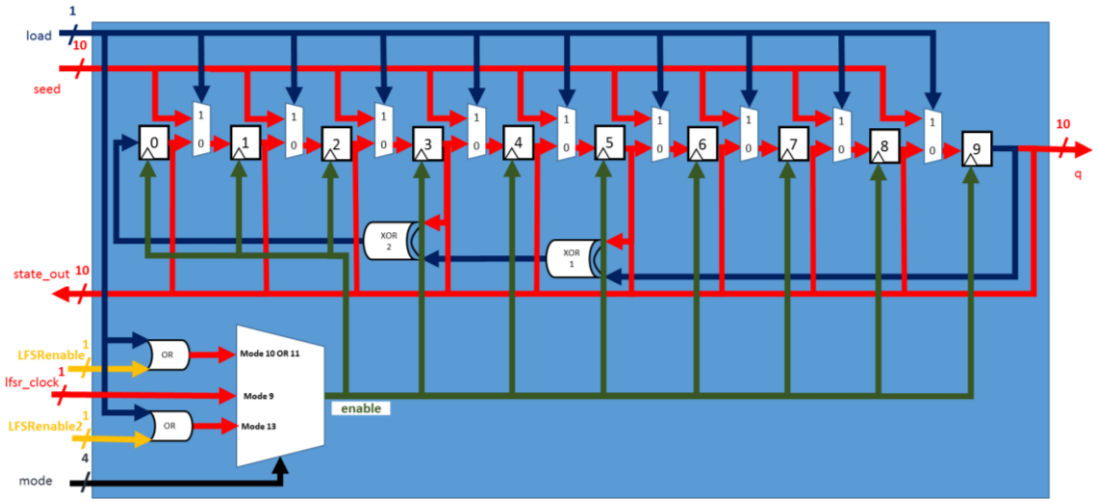

Figure 2.6 LFSR Architecture

This block consists of a series of interconnected Flip-Flops |0, 1, 2, 3, 4, 5, 6, 7, 8 & 9| that will be shifting their value each time they are enabled. The enable input of the registers {enable} is controlled depending of the operation mode of the *SerDes* {mode}.

When a pseudo random pattern is created for the Digital Transmitter, the LFSR needs to wait for the conversion from parallel to serial to conclude. Once the Digital transmitter finishes the conversion, an enable signal {enable} is used trigger a shift on the LFSR to the next pseudo random pattern value of the sequence.

The initial value of the registers is determined by feeding a *seed* {seed}. Once the initial value of the registers is set, a series of multiplexers will switch their selector bit {load} from 1 to 0 allowing the registers to connect on series creating a shift register.

After setting the initial value {seed} on the Flip-Flop Registers |0, 1, 2, 3, 4, 5, 6, 7, 8 & 9|, an XOR gate |XOR1|receives the output of 2 registers |5 & 9| and a second XOR gate |XOR2| receives the output of the first XOR gate |XOR1| and the value of one of the registers |3|. The output of the second XOR gate |XOR2| is fed back to the first register |1| of the series, creating a pattern of inputs that seems random. This will continue to happen indefinitely until the operation mode {mode} changes, which will either deactivate the LFSR or inject a new *seed*.

The LFSR´s Pseudo Random Pattern will be generated in 2 different formats, serial {q} and parallel {state_out}. The format used depends of the operation mode of the *SerDes*. The serial mode {q} is used to test the Analog Transmitter; this operation mode appears

as Mode 9 in Table 2.2. The parallel format {state_out} will be used for Transmission and Reception tests, these modes appear as 10, 11, 13 & 14 in Table 2.2.

For the reception tests (Modes 13 & 14 on Table 2.2), the LFSR output needs to be encoded in 8b/10 for the Digital Receiver to work properly. To encode the LFSR signal, the *SerDes* encoder and serializer blocks of the Digital Transmitter were instantiated for them to receive the parallel input of the LFSR (this instantiated module appears as "LFSR TX Digital" on Figure 2.1). When doing this, the reception modules receive a valid encoded serial transmission from the LFSR.

REFERENCE CITLALI´s WORK |LFSR TX Digital|

2.6 Comparator Block

Built-In Self-Test (BIST) functionality was added as well to the Testing Module (refer to section 2.2). A Comparator block is used to synchronize and compare 2 signals and determine if the signals have the expected result or not. The microarchitecture implemented for this module appears on Figure 2.7.

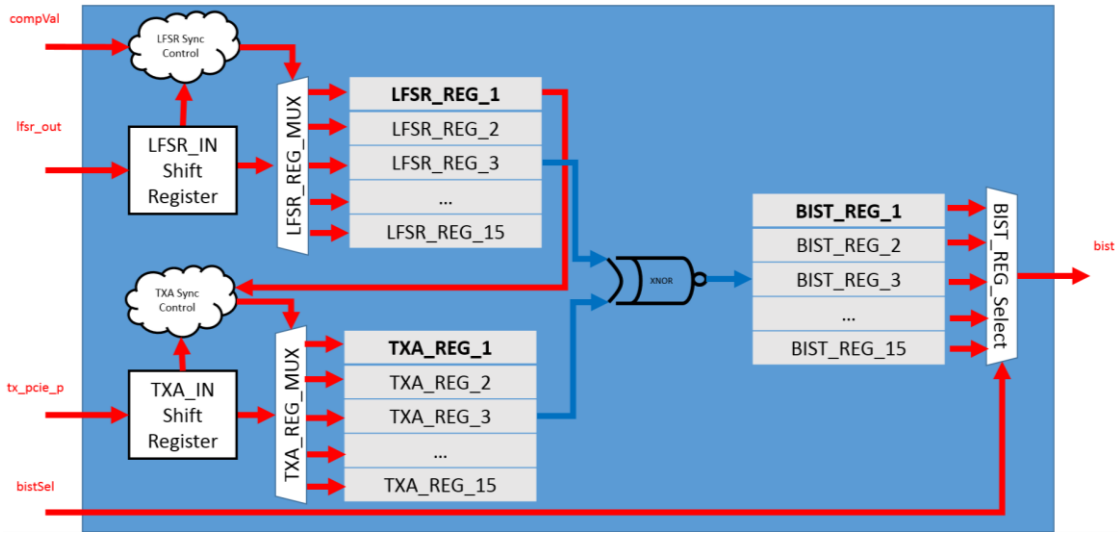

Figure 2.7 Comparator Microarchitecture

The first signal to enter the comparator {lfsr_out}, is the signal injected to the functional modules that are being tested (the tested modules depend on the operation mode, see section 2.4). A Control Block |LFSR Sync Control| will synchronize this signal and control

a multiplexer |LFSR_REG_MUX| to select the correct address on which this data will be stored on a register memory |LFSR_REG|.

The second signal to enter the comparator is the output signal {tx_pxie_p}. This output signal is generated by the functional blocks that are being tested. A Control Block |TXA Sync Control| will synchronize this signal and control a multiplexer |TXA_REG_MUX| to select the correct address on which this data will be stored on a register memory |TXA_REG|.

Then a comparison between the two register memories | LFSR_REG & TXA_REG | will occur: register 1 from memory 1 |LFSR_REG_1| will be compared with register 2 of memory 2 |TXA_REG_1|, if they are the same, a logic “1” will be stored on register 1 of a third register memory |BIST_REG_1|. The same process will be repeated through all the registers of the memories.

Once this process finishes, the third register memory |BIST_REG| will contain data of the successful and failed comparisons. In order to inquire the comparator about any of the results of the comparisons on memory 3 |BIST_REG|, a signal {bistSel} will be used to control a multiplexer |BIST_REG_Select| to decide which comparison result will be shown outside the chip by raising a flag on a pin of the *SerDes* {bist}.

CHAPTER 3 - VERIFICATION AND RESULTS OF THE TESTING MODULE IMPLEMENTATION

After finishing the *SerDes´* Testing Module *RTL* design, the next step is to verify that whole system behaves properly. To perform these verifications correctly, the RTL of the Testing Module was merged with the Digital Receiver's and Digital Transmitter's *RTL* designs creating a SerDes RTL top level design. This RTL contained the description of all the digital modules inside the ITESO TV1 SerDes chip.

To perform the verification on this top level design, fifteen test benches were created. These test benches were designed to verify the functionality of all the Operation Modes described on Table 2.2.

After the finishing the behavioral verification of each operation mode, a logic synthesis was performed (see Chapter 4). This synthesis produced an HDL file mapping the SerDes top level design to the standard cells of the **IBM**´s cmrf7sf Design Kit.

This logic synthesis HDL file was verified as well by using the 15 test benches created for the behavioral verification. These verifications will be found after the behavioral verifications of each operation mode.

3.1 Operation Mode 0 - Normal Mode

This is the typical mode of operation of the *SerDes*. This mode consists of 2 functions, Reception and Transmission as illustrated on Figure 3.1.

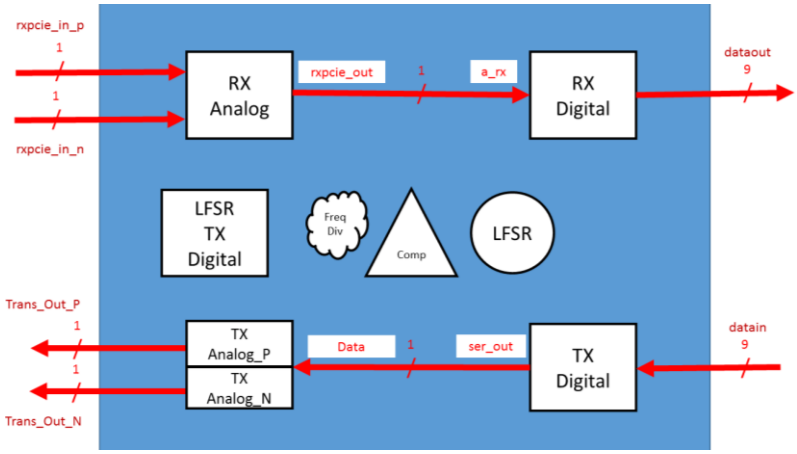

Figure 3.1 Operation Mode 0 – Normal Mode diagram

For the Reception function, a differential signal is injected to the RX Analog block {rxpcie_p and rxpcie_n}. After the RX Analog Block amplifies, eliminates noise, and transforms the signal {rxpcie_out} into a single ended signal, it passes it over to the RX Digital Block {a_rx} in order to Deserialize the data and decode it from a 8b/10b encoding to a 9 bit parallel output signal {dataout}.

The Reception function simulation appears on Figure 3.2, it shows the waveform obtained after simulating mode 0. It highlights how the input signal travels according to the logic described above.

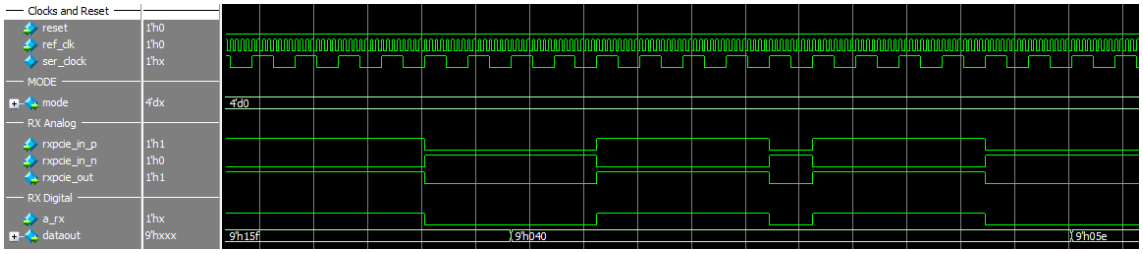

Figure 3.2 Operation Mode 0 – Normal Mode Waveform (Reception function)

Figure 3.3 shows the simulation of Transmission function. A 9 bit Parallel signal {datain} is injected to the Digital Transmitter |TX Digital| to be encoded in 8b/10b encoding and derialize it {ser_out}. This parallel signal {Data} is passed over to the Analog Transmitter

|TX Analog| to be amplified, modulated and equalized. Then it is transmitted as a serial differential signal outside the chip {Trans_Out_P and Trans_Out_N}.

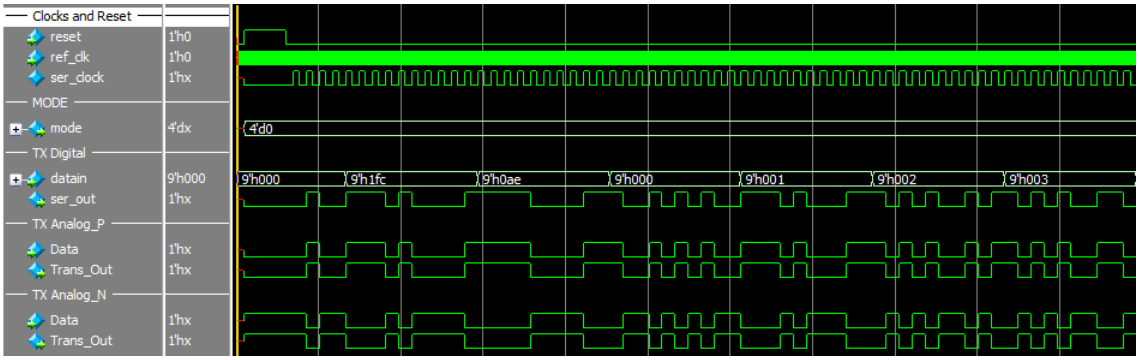

Figure 3.3 Operation Mode 0 – Normal Mode Waveform (Transmission function)

After completing the *logic synthesis*, which will be explained in more detail in Chapter 4, a second round of simulations was performed with the HDL file created by **RC Compiler** to check the functionality of the design after synthetizing it and mapping it to the **IBM**´s cmrf7sf Design Kit standard cells library, the results are shown in Figure 3.3.

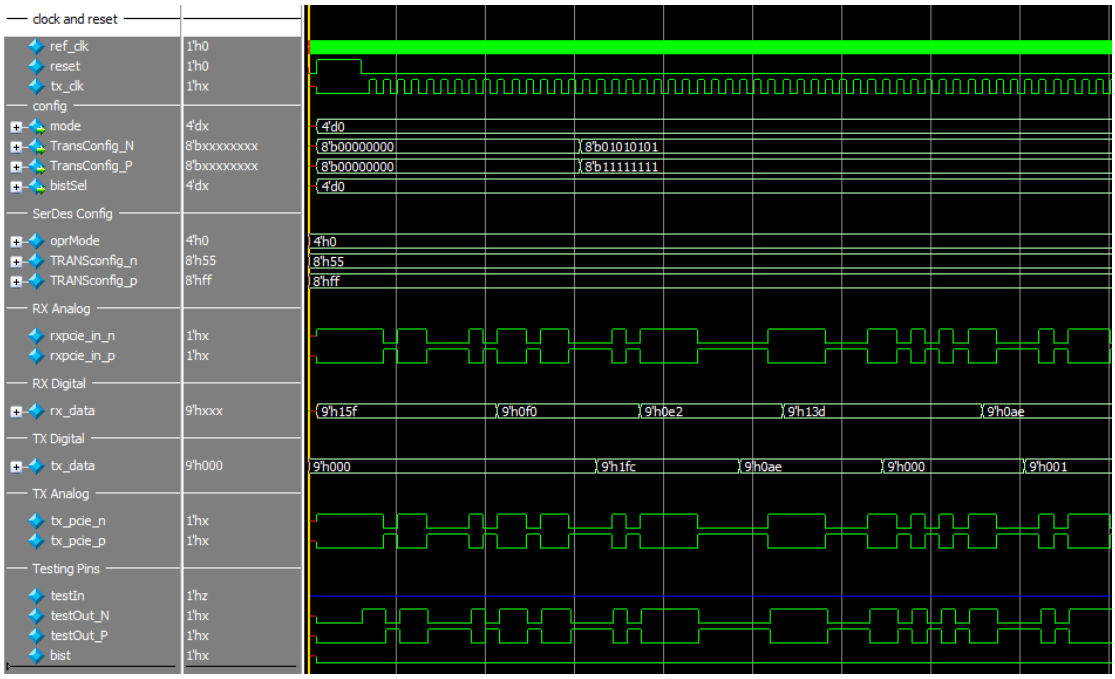

Figure 3.4 Operation Mode 0 – Normal Mode synthesis waveform

3.2 Operation Mode 1 - RXD

The objective of this mode is to test the Digital Receiver by doing a bypass on all the other functional modules of the *SerDes*. Figure 3.5 shows the diagram of this mode, the test bench waveform is presented on Figure 3.6.

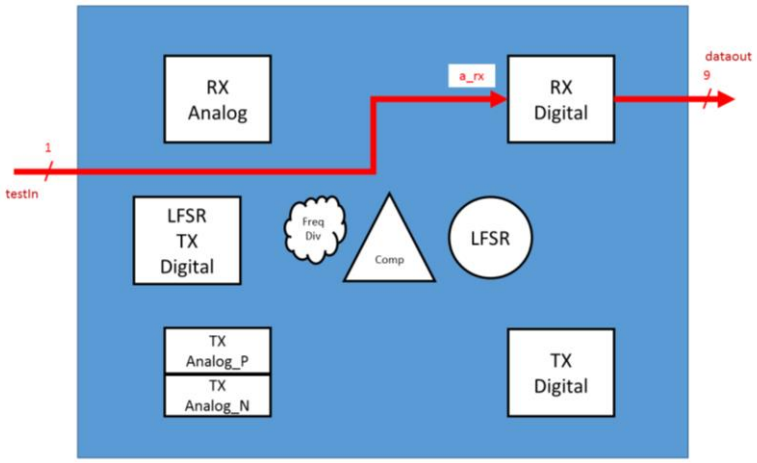

Figure 3.5 Operation Mode 1 - RXD diagram

A serial signal {testIn} is injected in one of the ports of the chip and it's routed directly to the input pin {a_rx} of the Digital Receiver |RX Digital|. Here is is deserialized and decoded. A decoded 9 bit parallel version of the injected signal should be measured on the parallel output ports of the chip {dataout}. The waveform on Figure 3.6 illustrates how the Analog Receiver |RX Analog| is bypassed and the Digital Receiver Block |RX Digital| receives its input signal directly from a test pin {testIn}.

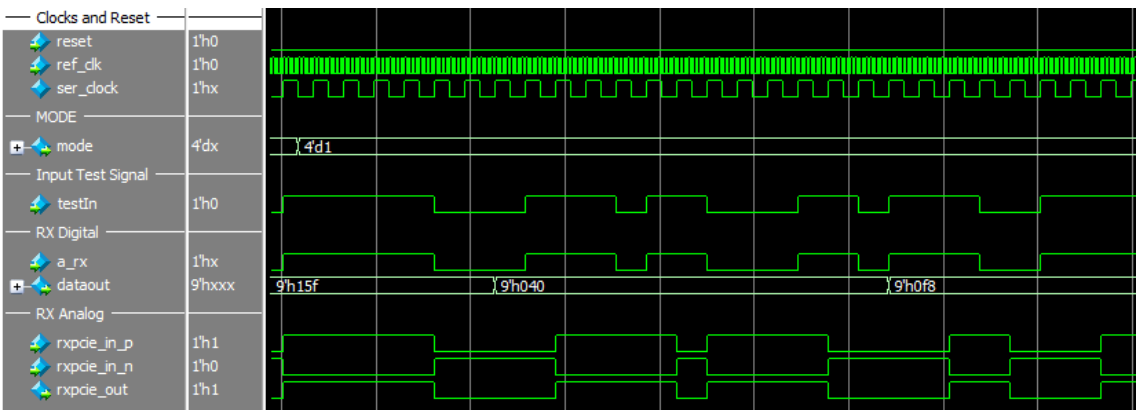

Figure 3.6 Operation Mode 1 - RXD waveform

The same operating mode was simulated on the HDL obtained from the logic synthesis. This simulation can be found on Figure 3.7.

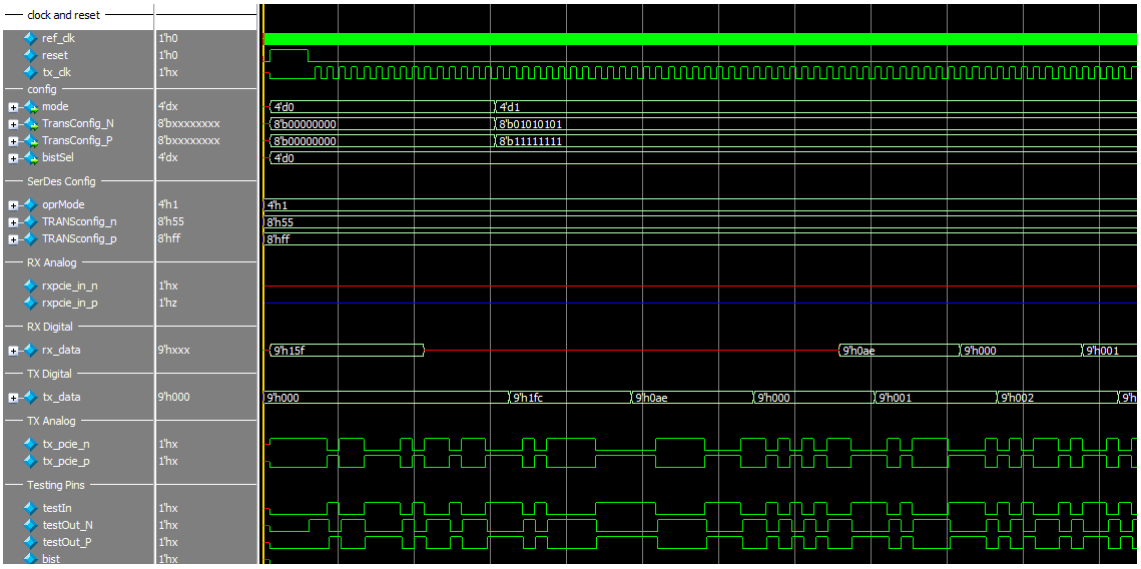

Figure 3.7 Operation Mode 1 - RXD synthesis waveform

3.3 Operation Mode 2 - TXA

The objective of this mode is to test the Analog Transmitter |TX Analog_P & TX Analog_N|. This mode does a bypass on all the other functional modules of the *SerDes*. Figure 3.8 shows a diagram of this mode, the test bench waveform is shown on Figure 3.9.

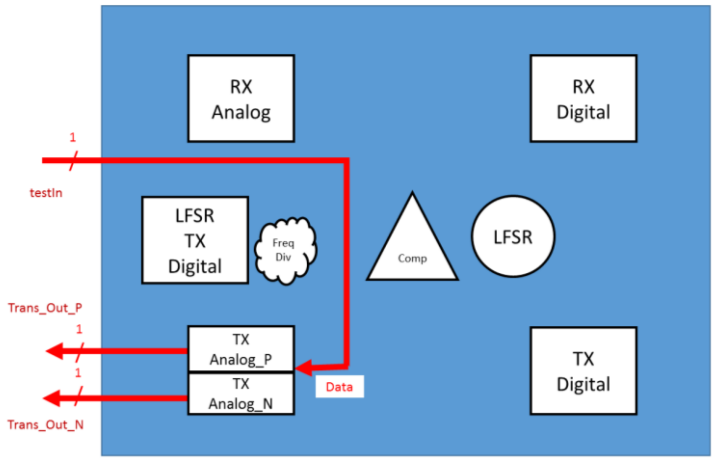

Figure 3.8 Operating Mode 2 – TXA diagram

A serial signal {testIn} is injected in one of the ports of the chip and routed directly to the input pin {Data} of the Analog Transmitter |TX Analog_P & TX Analog _N|. This block will amplify, modulate and equalize the signal. At the output of this block, there is a differential signal {Trans_Out_P & Trans_Out_N} that will be transmitted outside the *SerDes*. The waveform on Figure 3.9 shows how the Digital Transmitter |TX Digital| is bypassed and how the Analog Transmitter Block |TX Analog_P & TX Analog_N| receives its input signal directly from a test pin {testIn}.

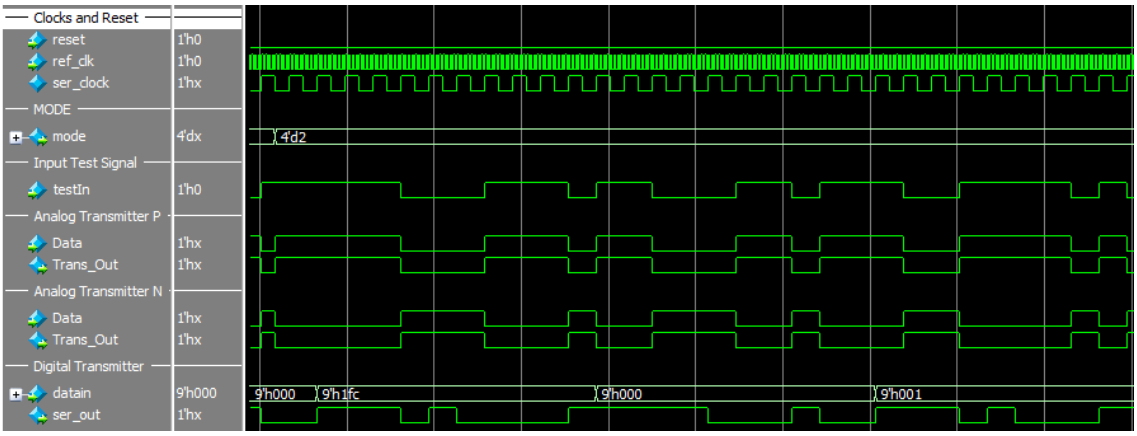

Figure 3.9 Operating Mode 2 – TXA waveform

The HDL file obtained from the logic synthesis was verified as well, Figure 3.10 shows how it performs Operation Mode 3 successfully.

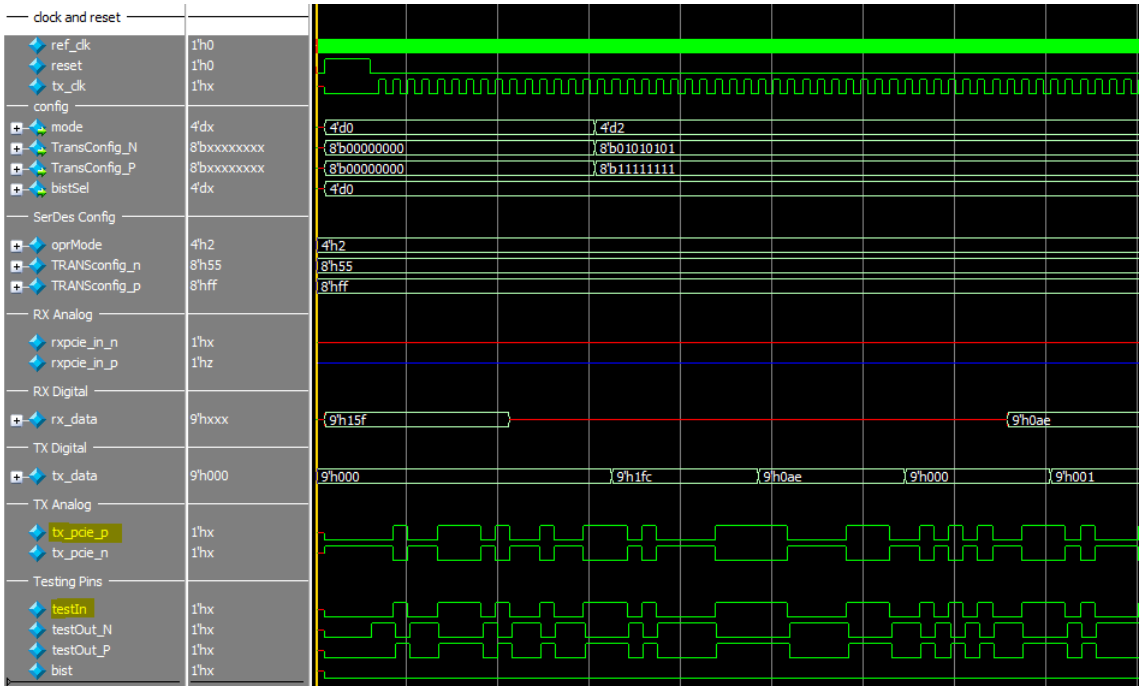

Figure 3.10 Operating Mode 3 - RXA Frequency Divider synthesis waveform

3.4 Operation Mode 3 - RXA Frequency Divider

This mode tests the Analog Receiver |RX Analog|. It bypasses all the other functional modules of the *SerDes* and outputs the resulting signal through a frequency divider |Freq Div|. This is to observe the module´s response at a slower speed |testOut_P|. Figure 3.11 shows the diagram of this Operating Mode.

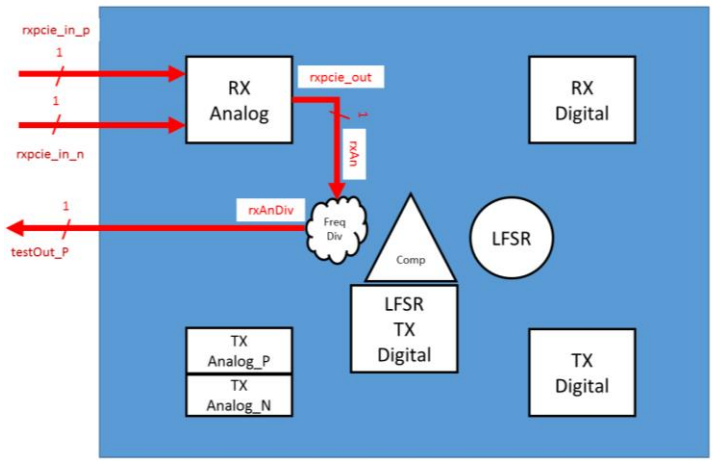

Figure 3.11 Operating Mode 3 - RXA Frequency Divider diagram

A differential signal {rxpcie_in_p & rxpcie_in_n} is fed to the Analog Receiver |RX Analog|. Then the signal is amplified, filtered and converted to a single ended signal {rxpcie_out}. This single ended signal signal {rxAn} is handed over to a frequency divider |Freq Div| to slow the signal frequency. The slowed signal is observed in one of the chip´s pins {testOut_P}.

The waveform on Figure 3.12 shows how the output of the Analog Receiver |RX Analog| is bypassing the Digital Receiver |RX Digital| and hands the signal over to the Frequency Divider |Freq Div| to be observed on one of the test pins {testOut_P} at a slower frequency.

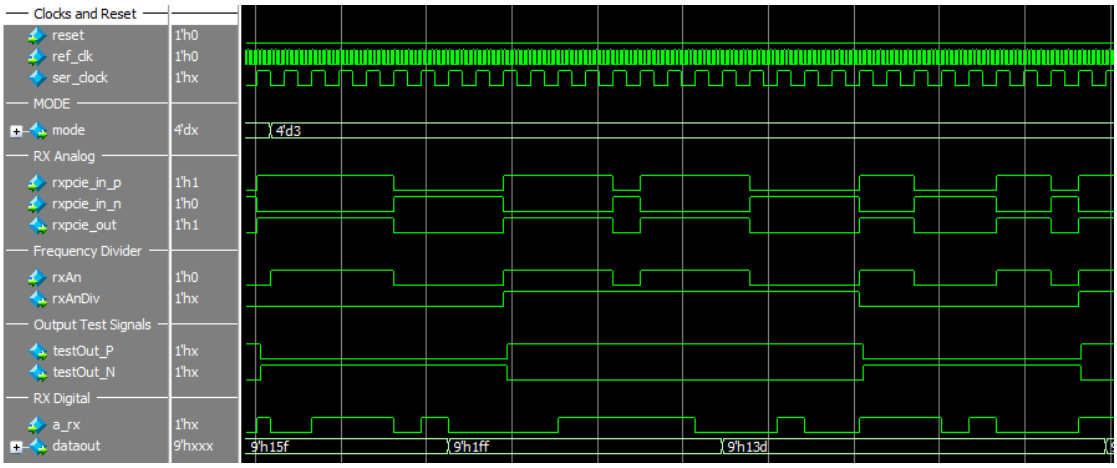

Figure 3.12 Operating Mode 3 - RXA Frequency Divider waveform

Figure 3.13 shows the verification of the HDL file obtained from the logic synthesis. The synthesis behaved as expected.

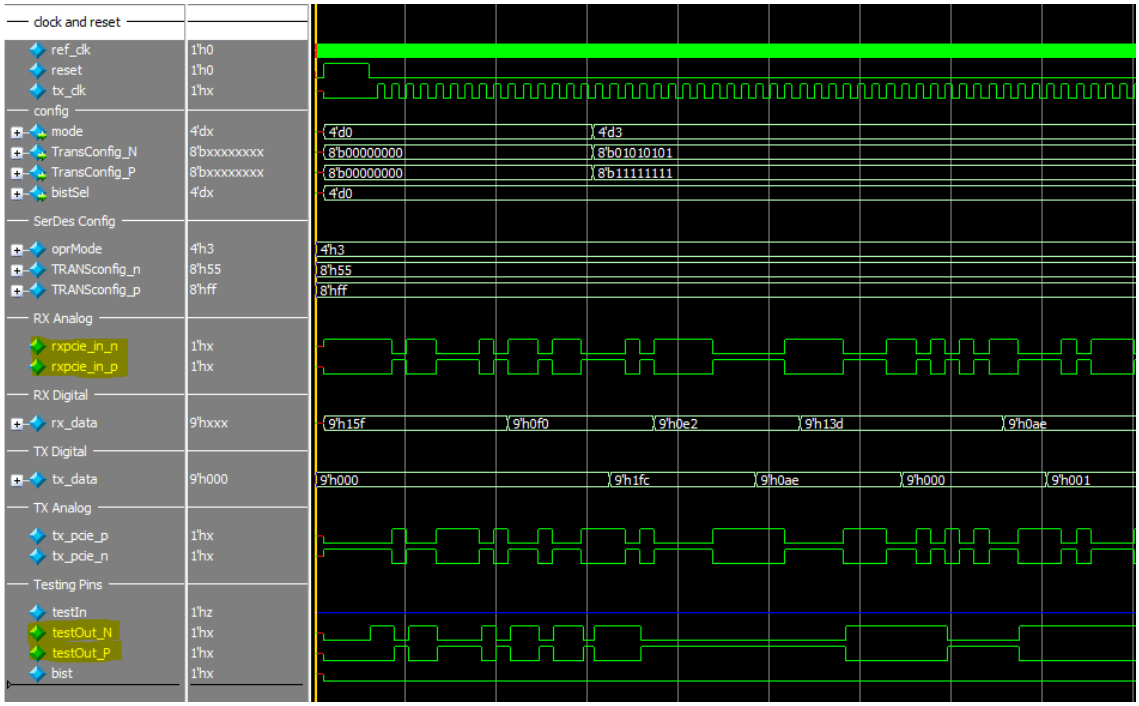

Figure 3.13 Operating Mode 3 - RXA Frequency Divider synthesis waveform

3.5 Operation Mode 4 - TXD

This mode tests the Digital Transmitter |TX Digital|. It performs a bypass on all the other functional modules of the *SerDes*. The diagram on Figure 3.14 shows how the bypass works.

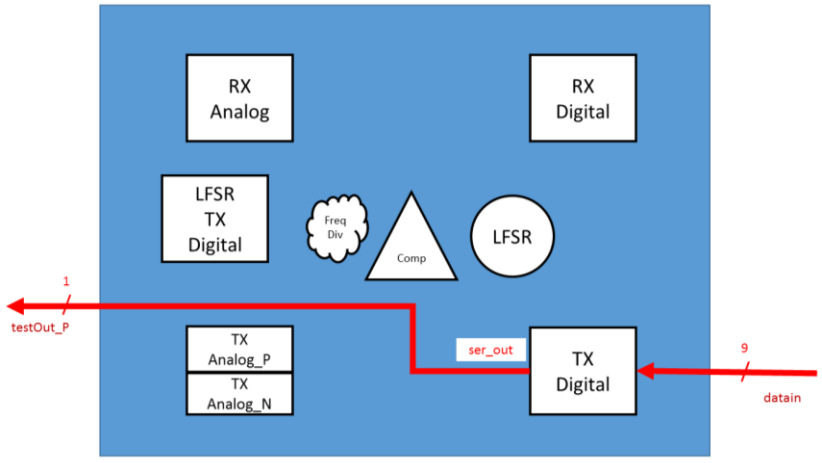

Figure 3.14 Operating Mode 4 - TXD diagram

A 9 bit parallel signal {datain} enters the Digital Transmitter |TX Digital|. After encoding the data in 8b/10b, the Digital Transmitter |TX Digital| serializes it and sends it out {ser_out} to one of the test pins of the chip {testOut_P}. This bypasses the Analog

Transmitter |TX Analog_p & TC Analog_n|. This is verified with the waveform on Figure 3.15.

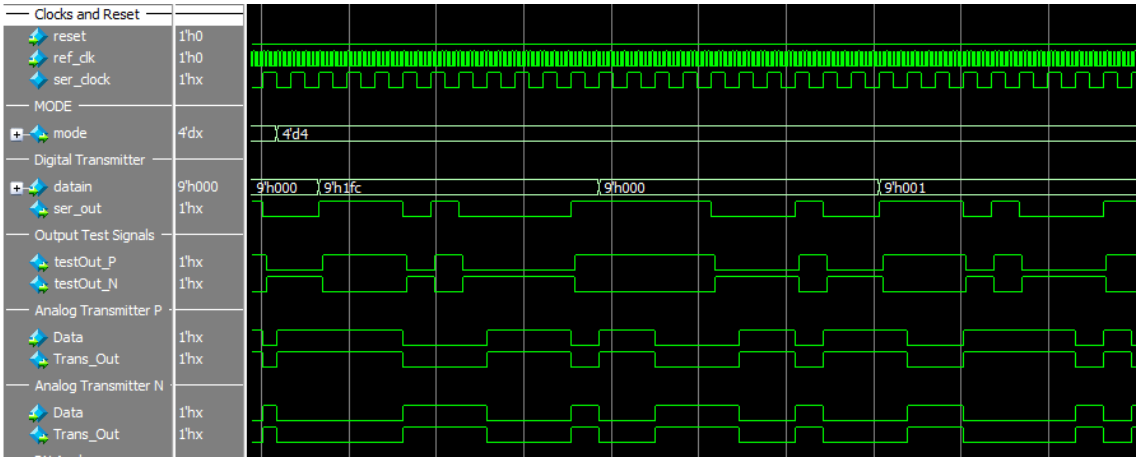

Figure 3.15 Operating Mode 4 – TXD waveform

The simulation on Figure 3.16 shows that the Operating Mode 4 works in the logic synthesis HDL as well.

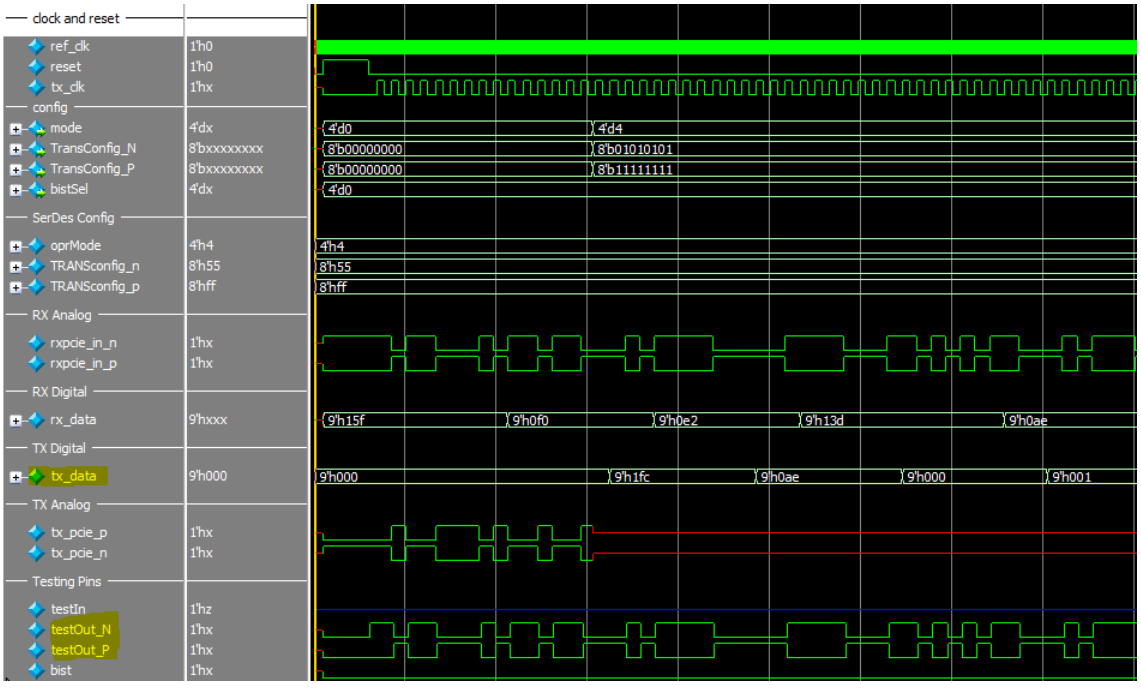

Figure 3.16 Operating Mode 4 – TXD synthesis waveform

3.6 Operation Mode 5 - Digital RX Loopback

The objective of this operating mode is to test the Digital Modules |RX Digital & TX Digital|. For this a bypass on the Analog Modules |TX Analog & RX Analog| of the *SerDes* is performed. The output of the Digital Receiver |RX Digital| si connected to the input of the Digital Transmitter |TX Digital| to perform a loop. The diagram of this mode appears on Figure 3.17.

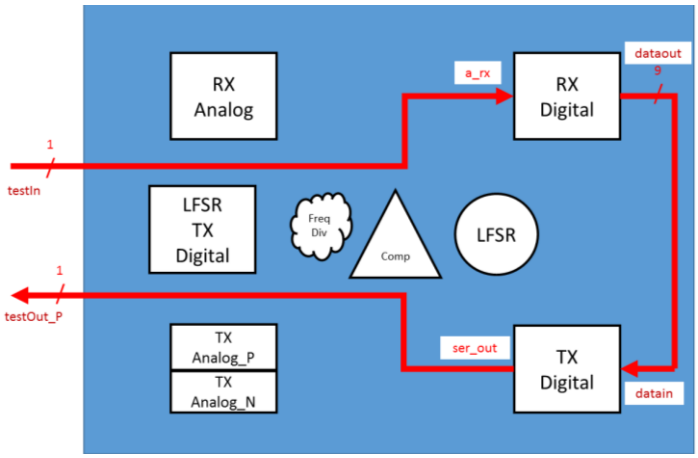

Figure 3.17 Operating Mode 5 - Digital RX Loopback diagram

A serial 8b/10b encoded serial signal {testIn} arrives at the Digital Receiver block |RX Digital| bypassing the Analog Receiver Block |RX Analog|. The signal {a_rx} gets deserialized and decoded. Then this parallel output {dataout} gets sent to de Digital Transmitter |TX Digital|. This block receives the parallel signal {datain} and encodes it to be serialized once more {ser_out}. This serial signal will be observed outside the chip on one of its pins {testOut_P}.

Figure 3.18 shows a waveform demonstrating that the serial transmission injected {testIn} to the Digital Receiver |RX Digital| is the same signal observed at the output {testOut_P}, but with a phase delay due to the deserialization and serialization processes.

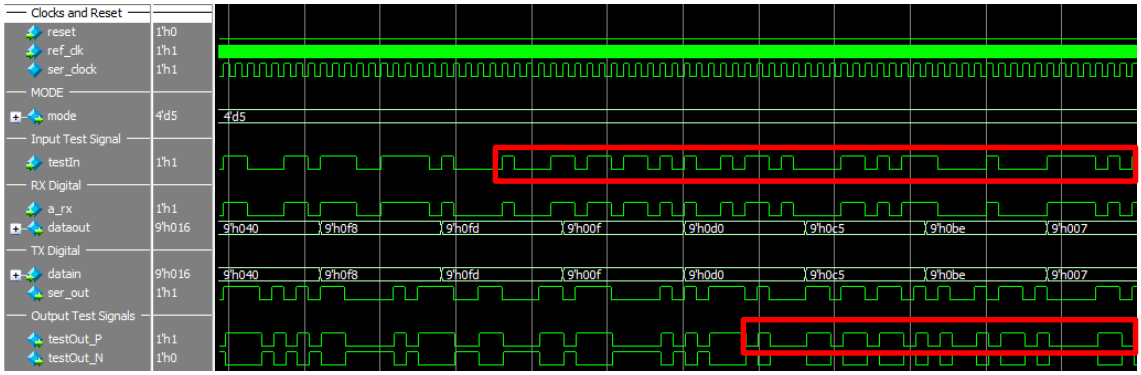

Figure 3.18 Operating Mode 5 - Digital RX Loopback waveform

The HDL obtained from the logic synthesis was verified as well. The gaps on the output pins {testOut_P & testOut_N} shown on Figure 3.19 are set up and hold issues happening inside the Digital Transmitter |TX Digital|.

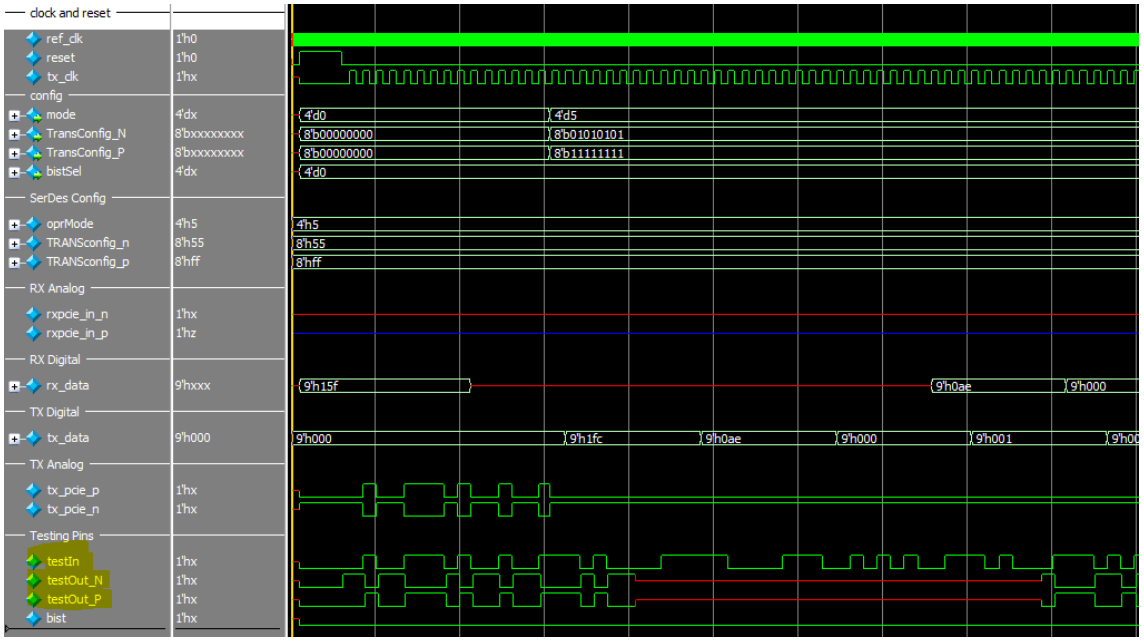

Figure 3.19 Operating Mode 5 - Digital RX Loopback synthesis waveform

3.7 Operation Mode 6 - TX Digital Loopback

The objective of this operation mode is to test the Digital Modules |RX Digital & TX Digital|. This mode does a bypass on the Analog Modules |TX Analog & RX Analog| and starts a loop on the Digital Transmitter |TX Digital|. The diagram of this mode appears on Figure 3.20.

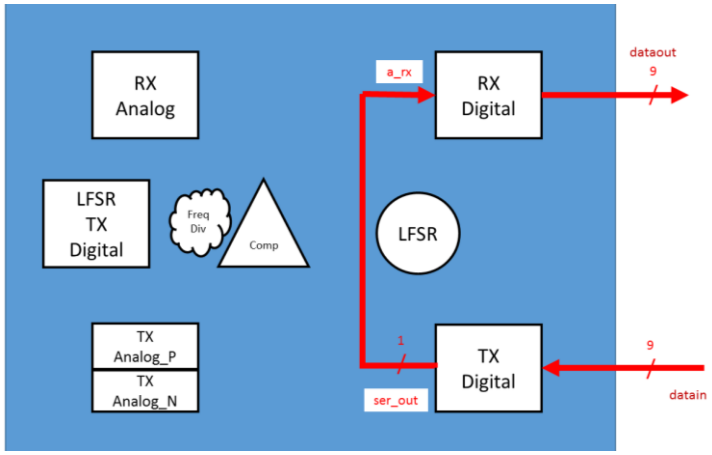

Figure 3.20 Operating Mode 6 - TX Digital Loopback

A 9 bit parallel signal {datain} enters the Digital Transmitter |TX Digital|. Here it is encoded and serialized {ser_out}. After this, the signal is forwarded to the input pin {a_rx} of the Digital Receiver |RX Digital| where it is deserialized and decoded from an 8b/10 encoding.

The Digital Receiver |RX Digital| delivers a decoded 9 bit parallel signal that can be observed on the parallel output ports of the chip {dataout}. The waveform on Figure 3.21 illustrates that the signal entering the Digital Receiver |RX Digital| is the same signal that exits the Digital Transmitter |TX Digital|.

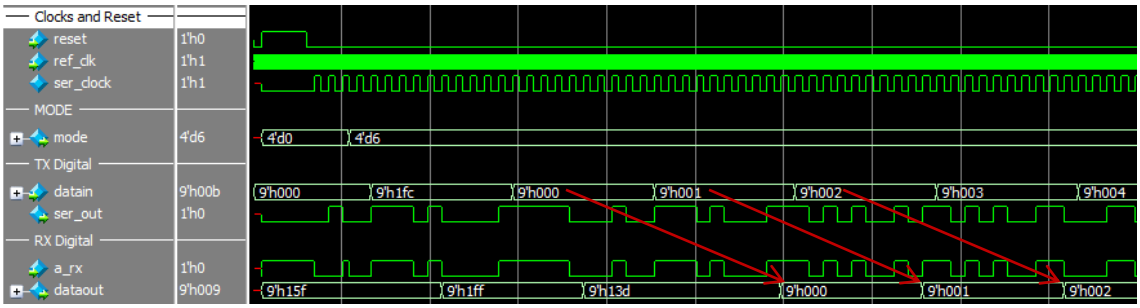

Figure 3.21 Operating Mode 6 - TX Digital Loopback waveform

This same verification was performed on the HDL file obtained from the logic synthesis (see Chapter 4). Figure 3.22 shows the resulting waveform.

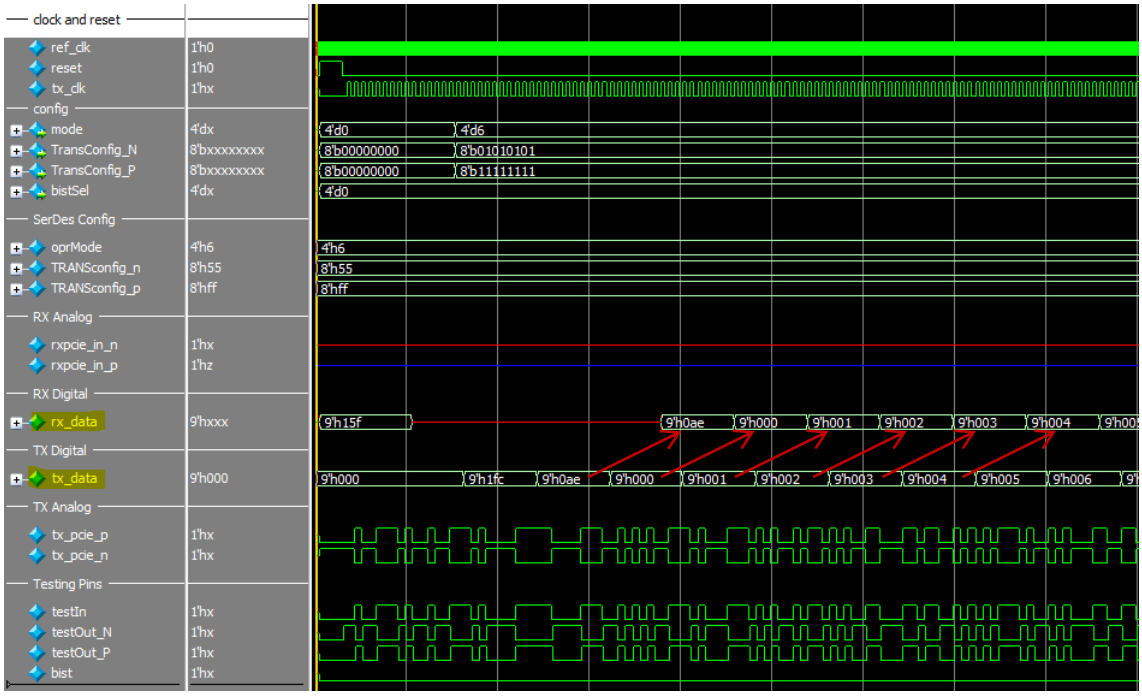

Figure 3.22 Operating Mode 6 - TX Digital Loopback synthesis waveform

3.8 Operation Mode 7 - RX Analog Loopback

This mode bypasses the Digital Modules |RX Digital & TX Digital| and connects the Analog Modules |RX Analog, TX Analog_P & TX Analog_N|. It performs a loop starting with the Analog Receiver |RX Analog|, the diagram on Figure 3.23 shows the diagram of this operation mode.

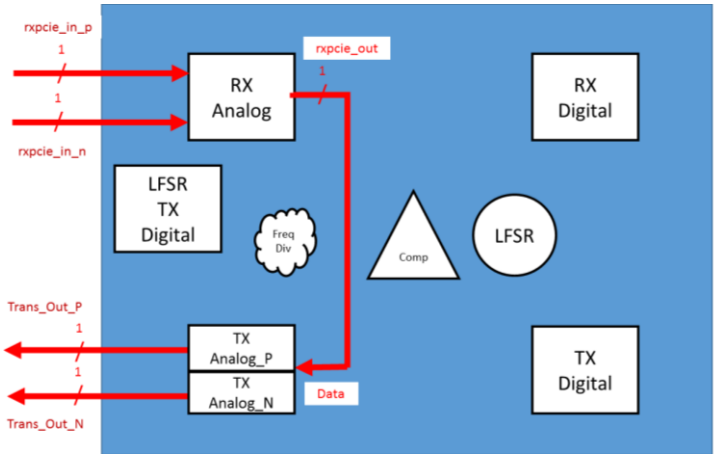

Figure 3.23 Operating Mode 7 - RX Analog Loopback diagram

A differential signal {rxpcie_in_p & rxpcie_in_n} enters the Analog Receiver |RX Analog|. The signal entering the block is amplified, filtered and converted to a single ended signal {rxpcie_out}. Then this signal {rxAn} is handed over to the Analog Transmitter |TX Analog_P & TX Analog _N| to be amplified, modulated and equalized. As a result, a differential signal {Trans_Out_P & Trans_Out_N} is transmitted from the *SerDes*. The waveform on Figure 3.24 shows how the Analog Loopback flows through the chip.

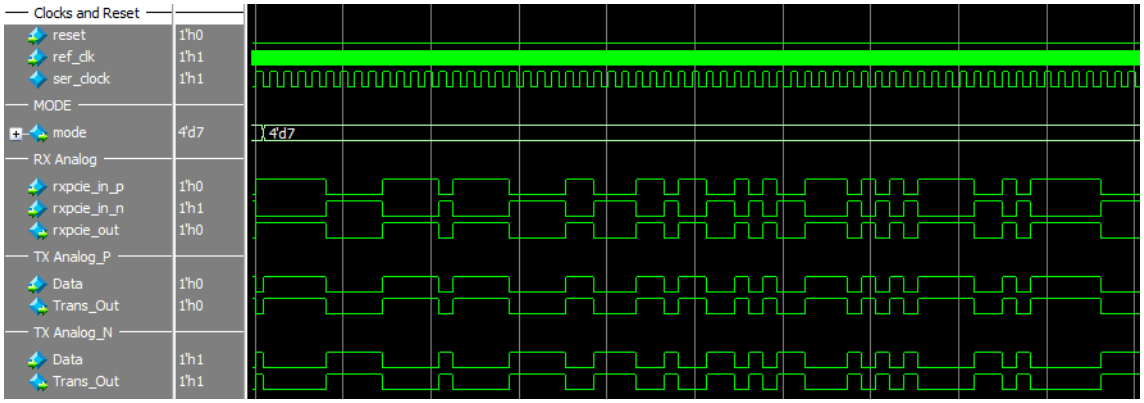

Figure 3.24 Operating Mode 7 - RX Analog Loopback waveform

Figure 3.25 shows a waveform that illustrates the results of the verification of this Operation Mode on the HDL file obtained from the logic synthesis.

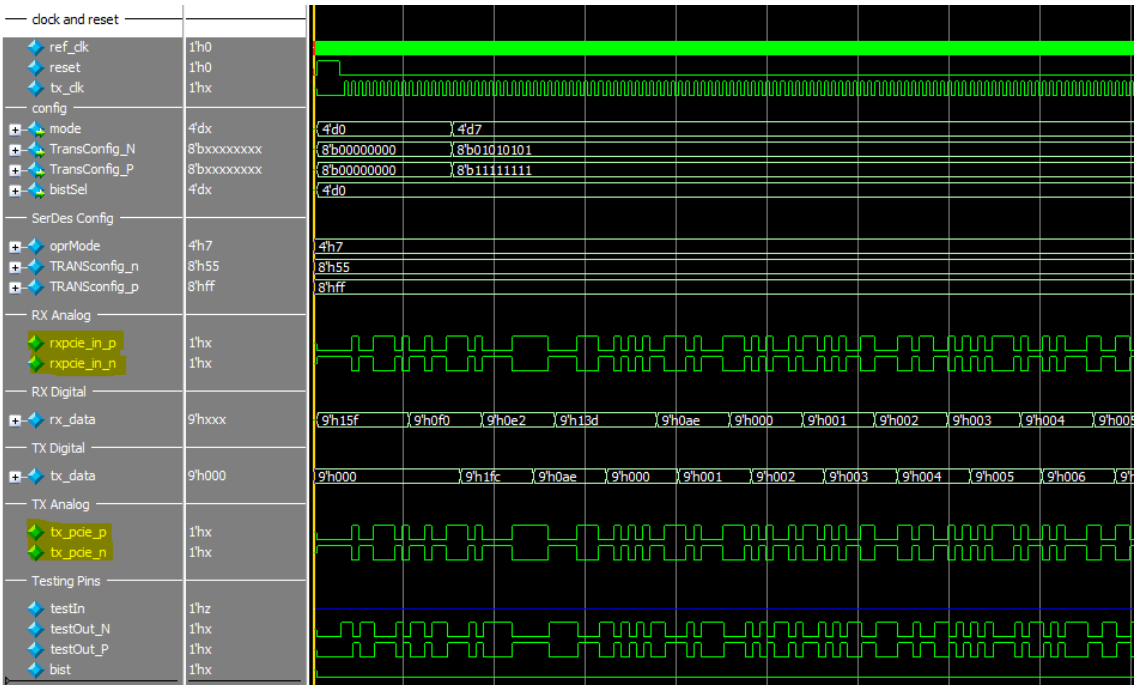

Figure 3.25 Operating Mode 7 - RX Analog Loopback synthesis waveform

3.9 Operation Mode 8 - RXA Full Loop Back & BIST

This mode loops all 4 modules. The loop starts from the Analog Receiver |RX Analog| all the way to the Analog Transmitter |TX Analog| as shown on Figure 3.26. It also performs BIST functionality by feeding the input and output signals through a comparator |Comp|.

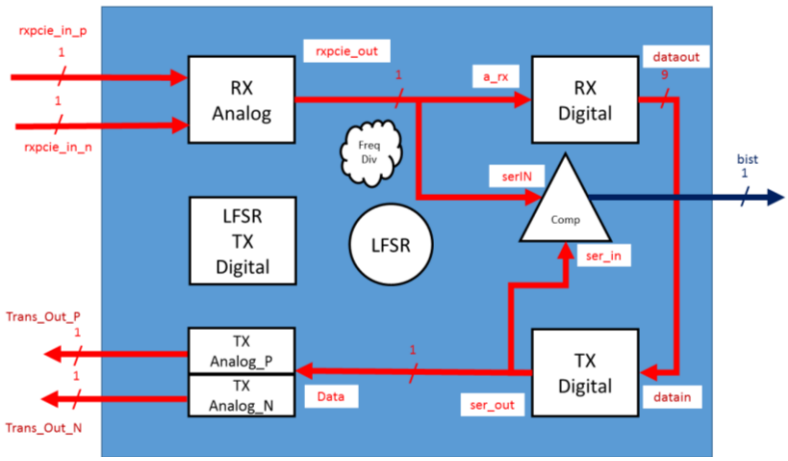

Figure 3.26 Operating Mode 8 - RXA Full Loop Back & BIST diagram

A differential signal {rxpcie_in_p & rxpcie_in_n} is injected to the Analog Receiver |RX Analog| to be amplified, filtered and converted to a single ended signal {rxpcie_out}. After this it is sent to the input pin {a_rx} of the Digital Receiver |RX Digital| to be deserialized and decoded. The parallel signal exiting this block {dataout} is routed to the input port {datain} of the Digital Transmitter |TX Digital|. Here the signal is encoded and serialized {ser_out}. Then this serial data {Data} is sent to the Analog Transmitter |TX Analog_P & TX Analog _N| to be amplified, modulated and equalized. A differential signal {Trans_Out_P & Trans_Out_N} will be transmitted from the *SerDes*. The waveform on Figure 3.27 shows how the signals travel across all the modules of the chip.

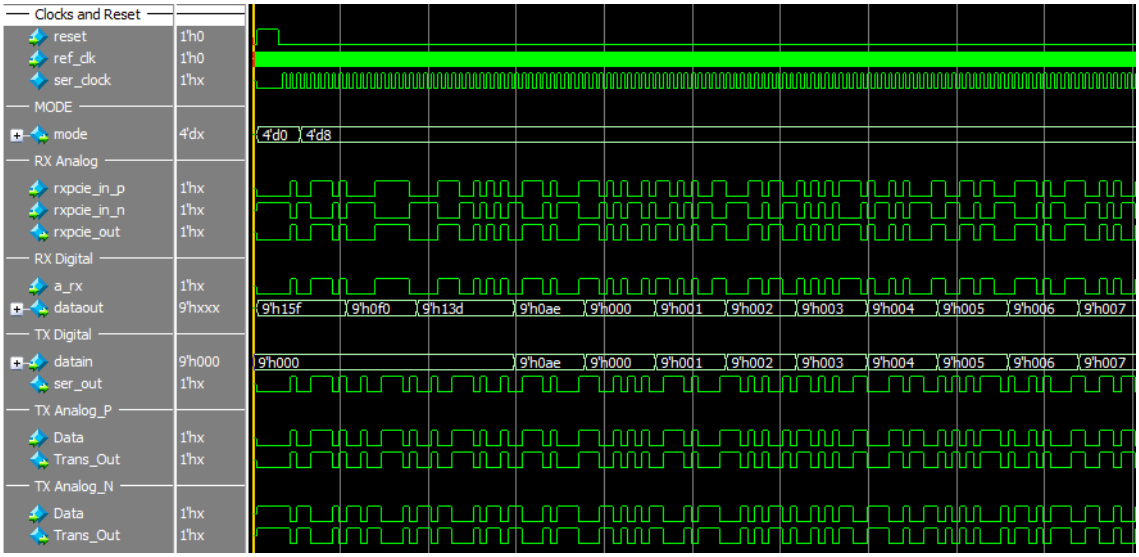

Figure 3.27 Operating Mode 8 – RXA Full Loopback waveform

Figure 3.28 shows how the input signal {a_rx} is passed to the Comparator |Comp|. It {ser_in} is synchronized and stored in a register memory {RegMem2}. In order to synchronize the transmission, two 9 bit data packages need to be received by the comparator. The first data package is LFSR *seed* and the second data package needs to be equals to “9’h001”. After the comparator detects that this two packages have been received, it will synchronize with the transmission and start storing the data in the correct order.

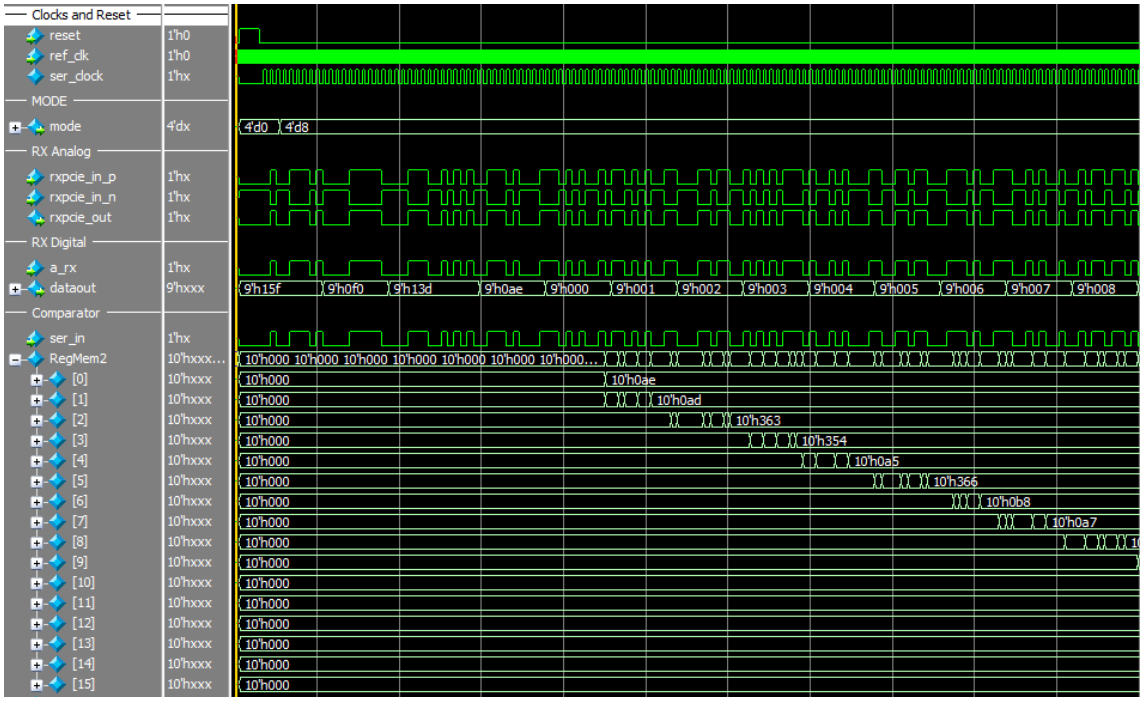

Figure 3.28 Operating Mode 8 – LFSR RXA Full Loopback input signal saving waveform

Figure 3.29 shows how the output signal {ser_out} is stored as well in a register memory {RegMem3}. This signal {serIN} is the resulting signal after looping {ser_out} through the Digital Receiver |RX Digital| and the Digital Transmitter |TX Digital|. The comparator |Comp| will be continuously comparing the output signal {serIN} to the first register of the input signal {RegMem2[0]} until they are the same. When this happens, the output signal {serIN} is stored on a register memory {RegMem3}.

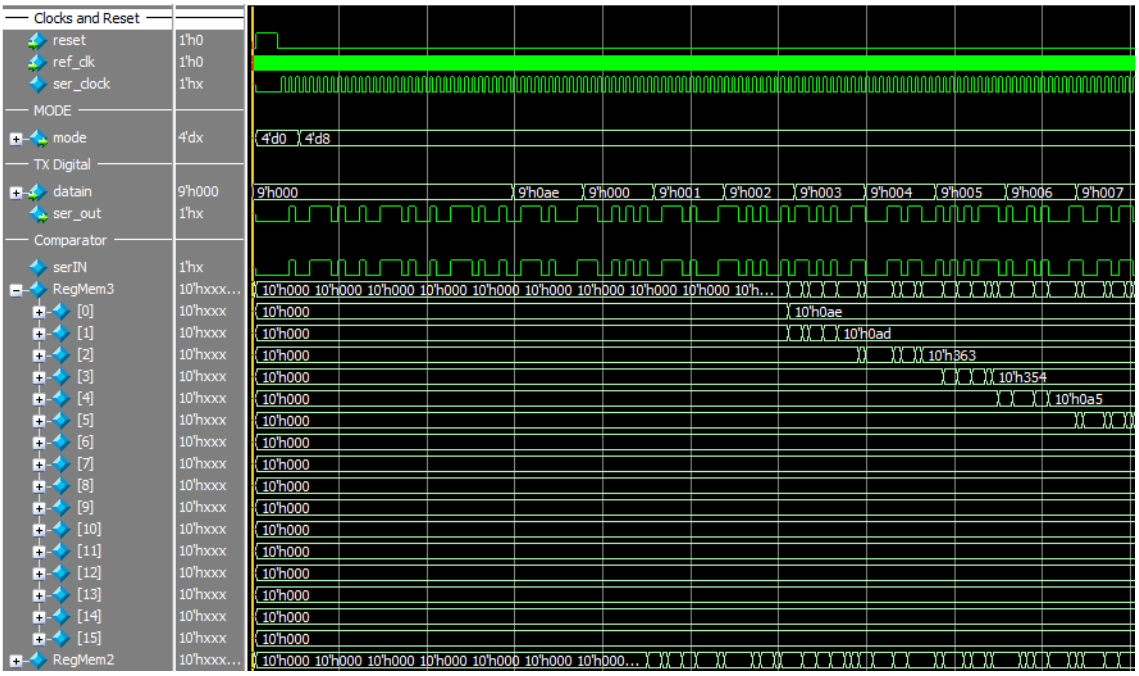

Figure 3.29 Operating Mode 8 – LFSR RXA Full Loopback output signal saving waveform

Figure 3.30 shows the waveform that demonstrates the BIST functionality in action. The register memory containing the input signals {RegMem2} and the one storing the output signals {RegMem3} are continuously compared to verify they are in fact the same. The result of the comparison will be stored to be queried later as described in Chapter 2 on section 2.6 where the comparator functionality is described.

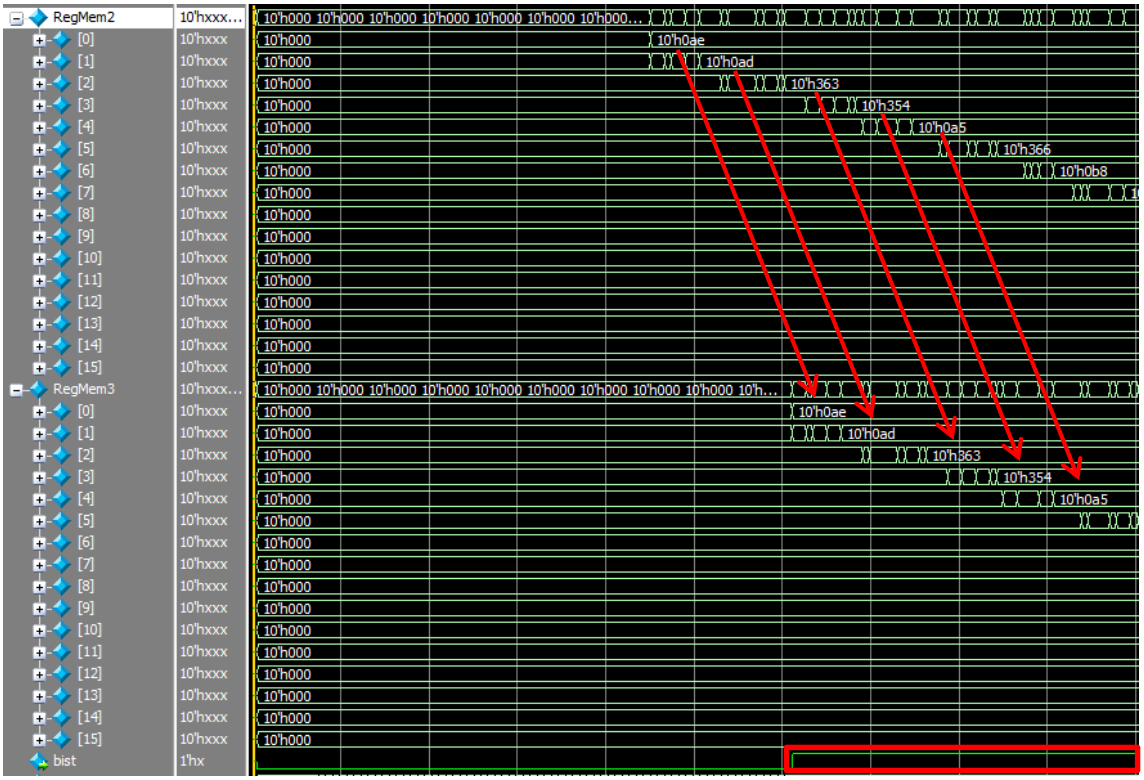

Figure 3.30 Operating Mode 8 – RXA Full Loopback BIST waveform

A verification of the HDL obtained from the logic synthesis described in Chapter 4 was performed. The waveform on Figure 3.31 shows the simulation of this HDL file.

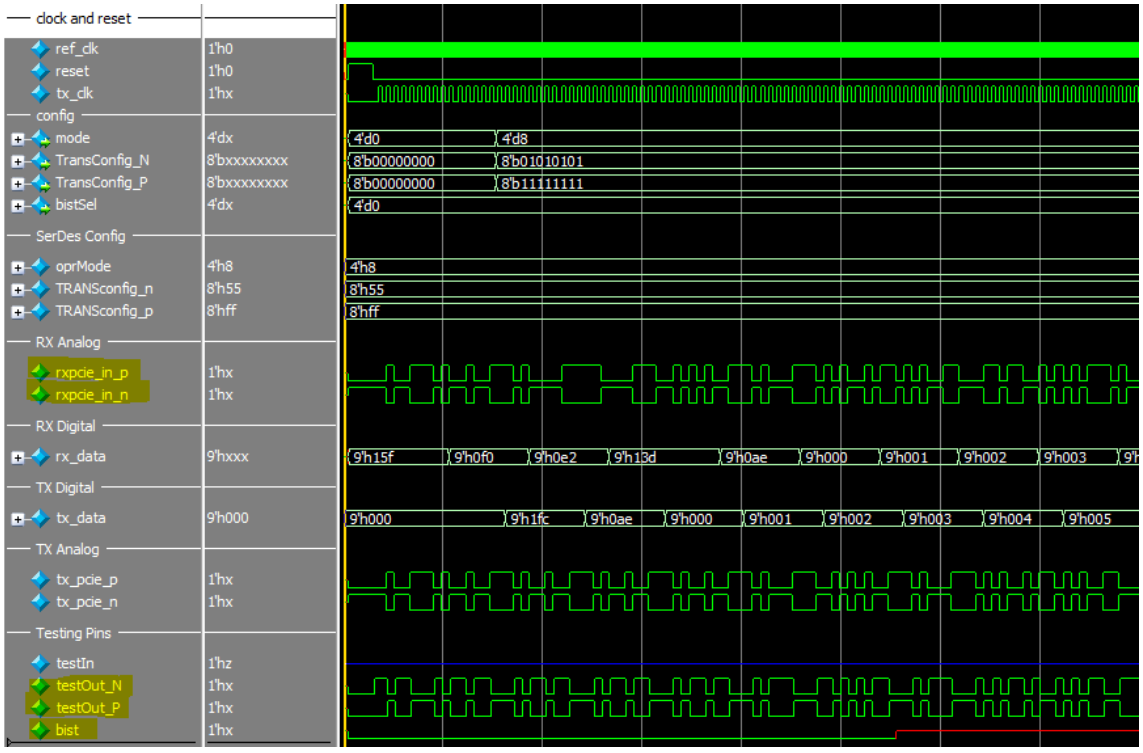

Figure 3.31 Operating Mode 8 – RXA Full Loopback & BIST synthesis waveform

The BIST functionality did not pass the logic synthesis verification. Hold issues on the registers of the comparator caused problems when storing the input and output signals.

3.10 Operation Mode 9 - LFSR TXA

The objective of this mode is to test the Analog Transmitter |TX Analog_P & TX Analog_N|. This mode does a bypass on all the other functional modules of the *SerDes.* The input signals for this test are pseudo random patterns {q} generated by the LFSR. Figure 3.32 shows a diagram of the data path of this mode, the test bench simulation appears on Figure 3.33.

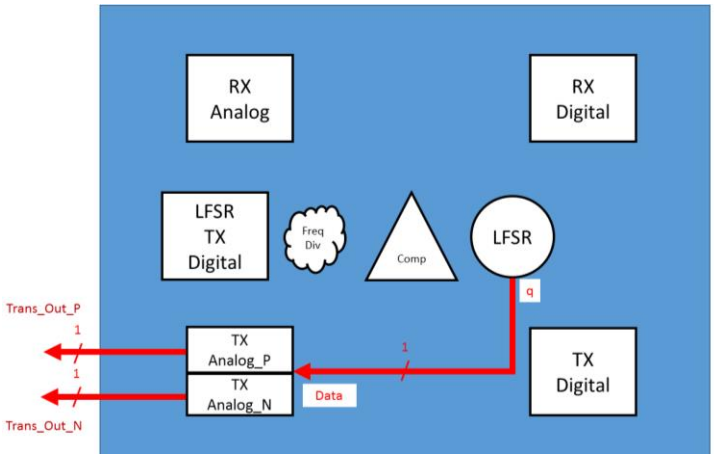

Figure 3.32 Operating Mode 9 - LFSR TXA diagram

The Linear Feedback Shift Register block |LFSR| generates a serial pseudo random pattern {q}. This pattern {Data} is injected to the Analog Transmitter |TX Analog_P & TX Analog_N| to test that it is working properly at the circuit´s normal operation frequency.

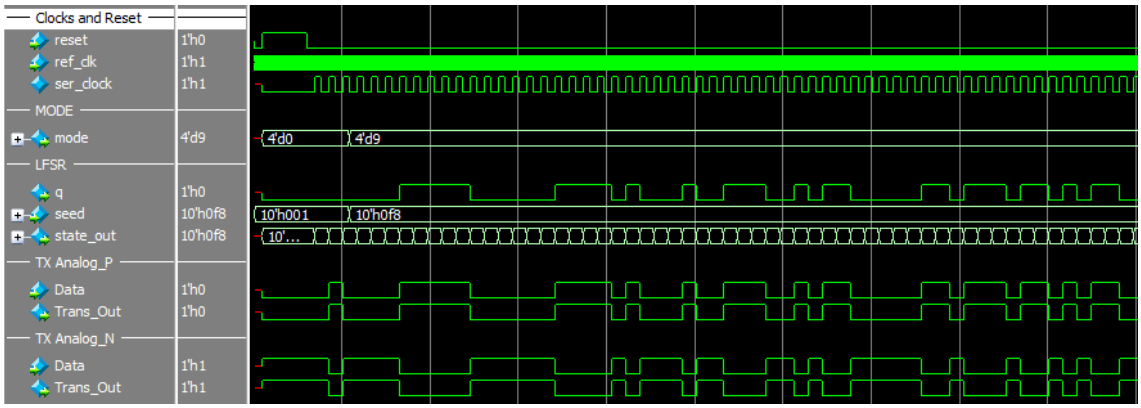

Figure 3.33 Operating Mode 9 - LFSR TXA waveform

The functionality of the design after the logic synthetizing it was verified, Figure 3.34 shows the results.

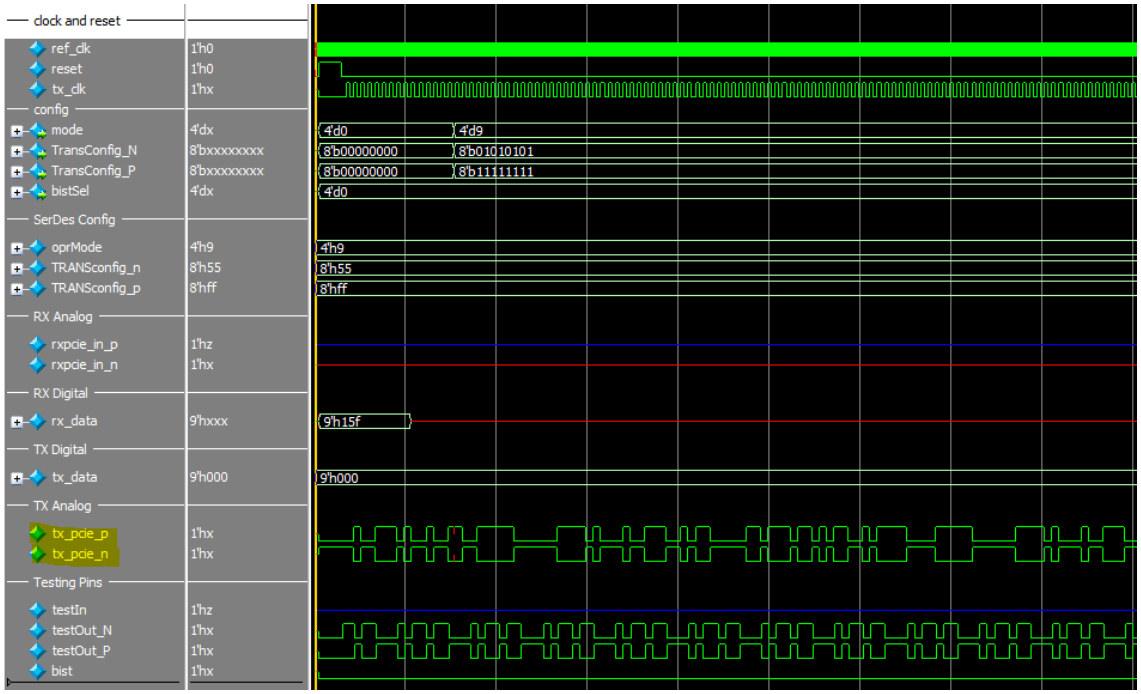

Figure 3.34 Operating Mode 9 - LFSR TXA synthesis waveform

3.11 Operation Mode 10 - LFSR TXD Full Loopback & BIST

This mode tests the SerDes in a full loop. The loop starts with the Digital Transmitter |TX Digital|, then the Analog Transmitter |TX Analog_P & TX Analog_N|, followed by the Analog Receiver |RX Analog| and exiting through the Digital Receiver |RX Digital|. The input signal of this test is generated by the LFSR. The diagram of this mode appears on Figure 3.35.

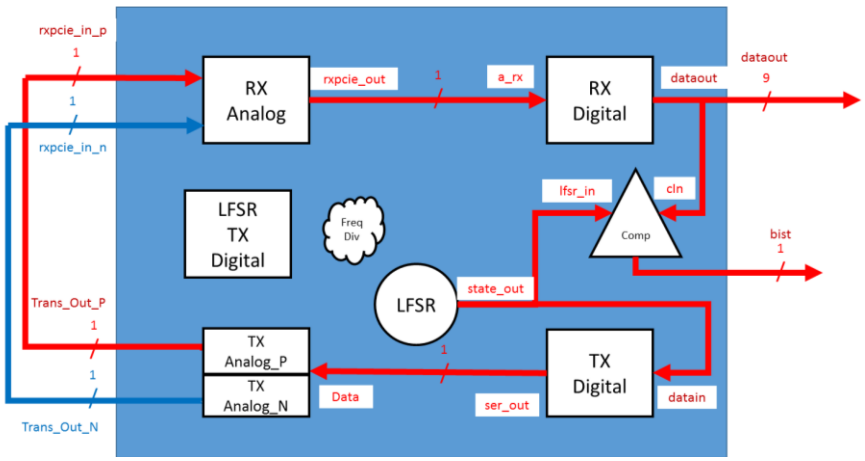

Figure 3.35 Operating Mode 10 -LFSR TXD Full Loopback & BIST diagram

The Linear Feedback Shift Register block |LFSR| generates a parallel pseudo random pattern {state out} which is injected to the Digital Transmitter |TX Digital| input {datain}. Here the signal is encoded and serialized {ser_out}. Then this serial data {Data} is sent to the Analog Transmitter |TX Analog_P & TX Analog _N| to be amplified, modulated and equalized. A differential signal {Trans_Out_P & Trans_Out_N} will be transmitted on two of the *SerDes* pins.

This output will be wired to the input pins {rxpcie_in_p & rxpcie_in_n} of SerDes' receiver |RX Analog|. In this block, the signal is amplified, filtered and converted to a single ended signal {rxpcie_out}. After this, it is sent to the input pin {a_rx} of the Digital Receiver |RX Digital| to be deserialized and decoded. The parallel signal exiting this block {dataout} should be the same as the signal injected {datain} to the Digital Transmitter |TX Digital| but with a delay due to the time it took to loop through the system. The results of this loop can be found on the waveform on Figure 3.36, where the matching of the output and input signals can be appreciated.

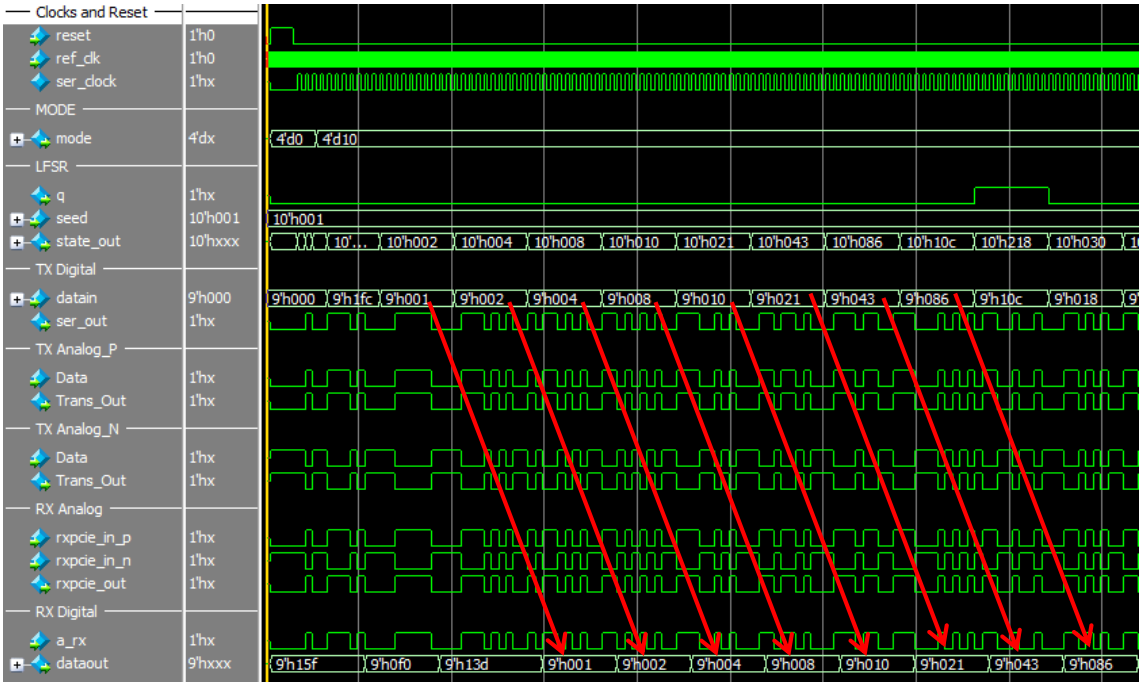

Figure 3.36 Operating Mode 10 -LFSR TXD Full Loopback waveform

Figure 3.37 shows the waveform that demonstrates the BIST functionality in action. The input signal {lfsr_in} and the output signals {cIn} are fed to the Comparator |Comp|. The input signal {lfsr_in} is stored in a register memory {RegMem[15:0]} and is continuously compared to the output signal {cIn} to verify they are in fact the same.

The result of the comparison will be stored to be queried later as described in Chapter 2 on Section 2.6.

Figure 3.37 Operating Mode 10 -LFSR TXD Full Loopback BIST waveform

The verification of the HDL obtained after performing the logic synthesis appears on Figure 3.38.

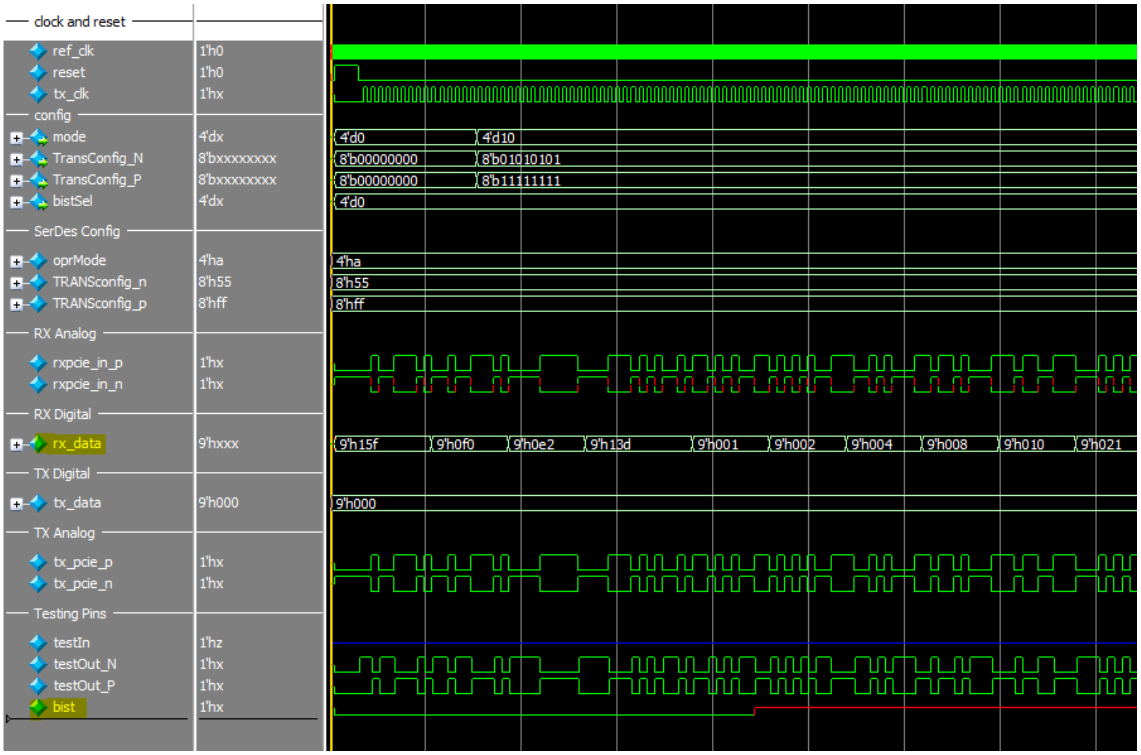

Figure 3.38 Operating Mode 10 -LFSR TXD Full Loopback & BIST synthesis waveform

Hold issues on the registers of the comparator caused problems when storing the input and output signals causing the comparator to fail the logic synthesis verification.

3.12 Operation Mode: 11 – LFSR TXD Digital Loopback & BIST

This mode tests the *SerDes* in a digital loop starting with the Digital Transmitter |TX Digital|. The input signal is a pseudo random parallel pattern created by the LFSR. BIST functionality was implemented as well using the Comparator block to check if the output meets the expected results. The diagram appears in Figure 3.39.

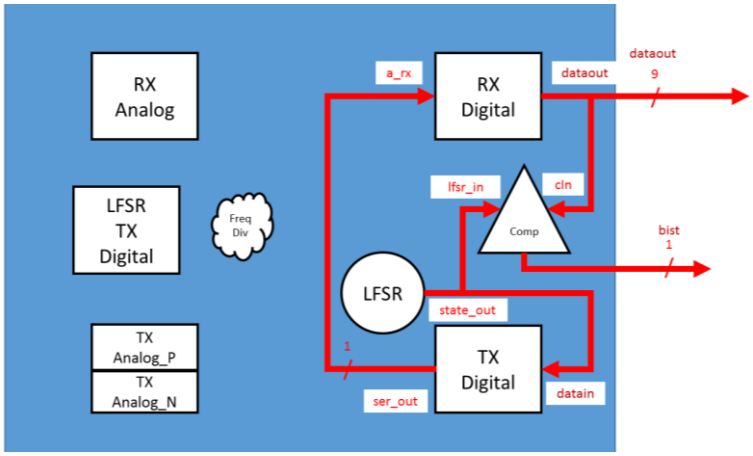

Figure 3.39 Operating Mode 11 – LFSR TXD Digital Loopback & BIST diagram

The Linear Feedback Shift Register block |LFSR| generates a parallel pseudo random pattern {state_out} which is injected to the Digital Transmitter |TX Digital| input {datain}. This block encodes and serializes the injected signal {ser_out}. After this, it is sent to the input pin {a_rx} of the Digital Receiver |RX Digital| to be deserialized and decoded. The parallel signal exiting this block {dataout} should be the same as the signal injected {datain} to the Digital Transmitter |TX Digital|. A phase delay between the input and output signals is due to the time it took to loop through the system. This behavior appears simulated on the waveform of Figure 3.40.

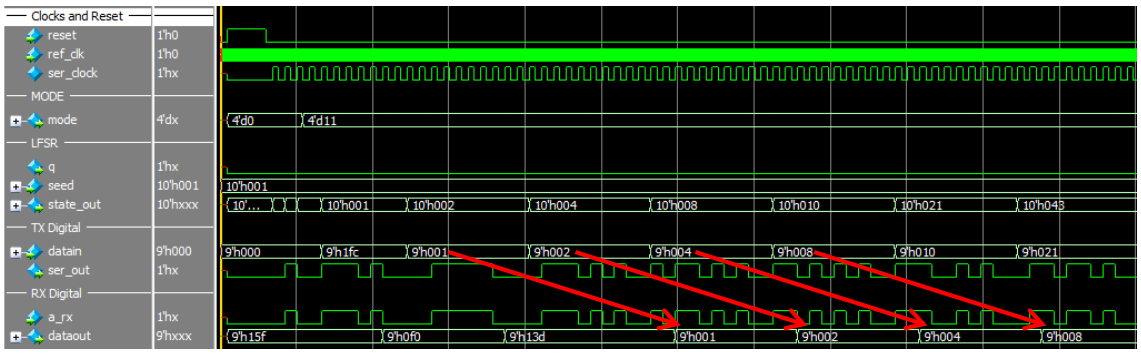

Figure 3.40 Operating Mode 11 – LFSR TXD Digital Loopback waveform

Figure 4.41 shows a waveform representing the BIST functionality. The input signal {lfsr_in} and the output signal {cIn} are fed to the Comparator |Comp| to be stored and compared. The input signal {lfsr_in} is stored in a register memory {RegMem[15:0]} and is continuously compared to the output signal {cIn} to verify that they are same.

The result of the comparison will be stored to be queried later as described in Chapter 3 on section 3.6 where the comparator functionality is described.

Figure 3.41 Operating Mode 11 – LFSR TXD Digital Loopback BIST waveform

After completing the *logic synthesis*, the resulting HDL was verified. Figure 3.42 shows how the logic synthesis HDL behaved when running the Operating Mode 11.

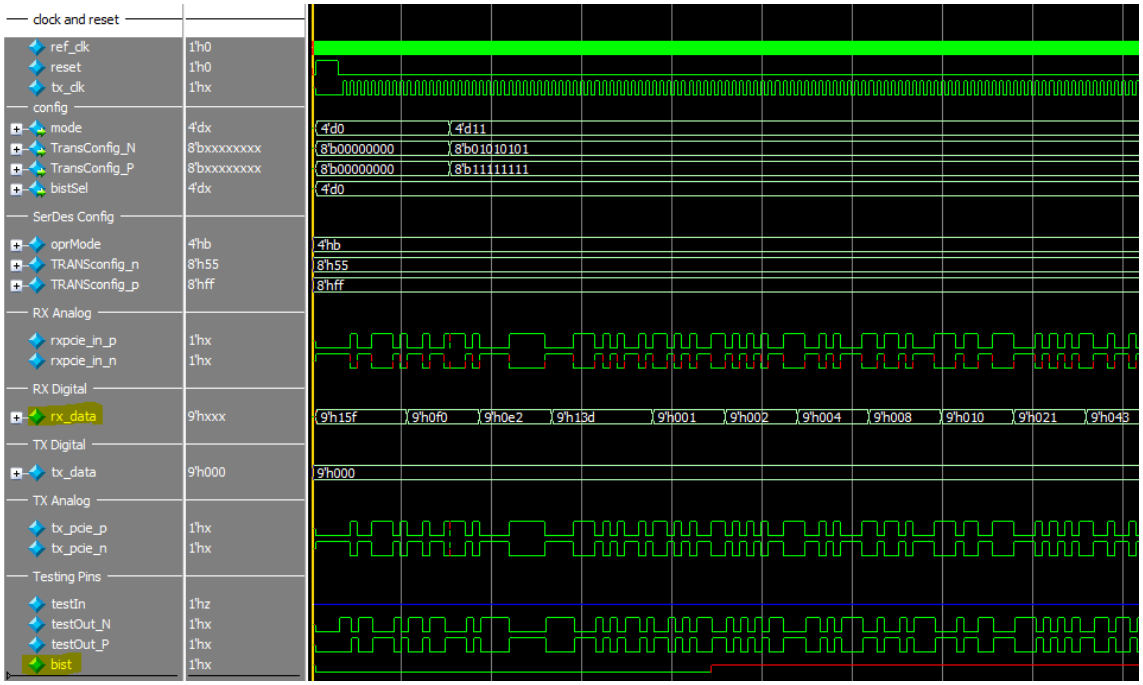

Figure 3.42 Operating Mode 11 – LFSR TXD Digital Loopback & BIST synthesis waveform

The comparator block |Comp| failed the verification due to hold issues, the rest of the blocks passed the verification.

3.13 Operation Mode 12 - TXD Full Loopback & BIST

AS Figure 3.43 illustrates, this mode loops all 4 modules starting from the Digital Transmitter |TX Digital|. After the signal travels on all the blocks of the SerDes it exits through the Digital Receiver |RX Digital|. It also incorporates the comparator |Comp| to verify if the output signal {dataout} is the expected one.

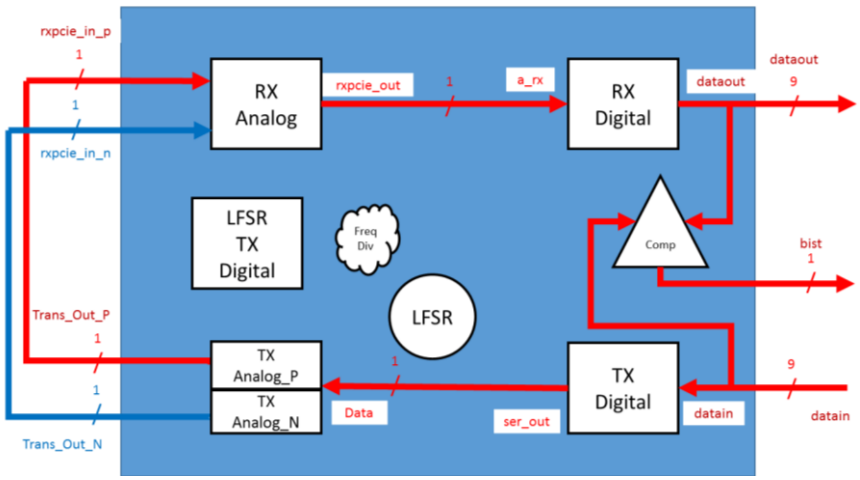

Figure 3.43 TXD Full Loopback & BIST diagram

The Digital Transmitter |TX Digital| receives a parallel input {datain} to be encoded and serialized {ser_out}. Then this serial data {Data} is sent to the Analog Transmitter |TX Analog_P & TX Analog _N| to be amplified, modulated and equalized. A differential signal {Trans_Out_P & Trans_Out_N} will be transmitted outside the *SerDes* on 2 of the chips pins.

This pins will be connected directly to the input {rxpcie_in_p & rxpcie_in_n} of the Analog Receiver |RX Analog|. Here the signal will be amplified, filtered and converted to a single ended signal {rxpcie_out}. Then the signal is sent to the input pin {a_rx} of the Digital Receiver |RX Digital| to be deserialized and decoded. The parallel signal exiting this block {dataout} should be the same as the signal injected {datain} to the Digital Transmitter |TX Digital| but with phase delay due to the time it took to loop through the system. The simulation results of this loop can be found on the waveform on Figure 3.44, where the matching of the output and input signals can be appreciated.

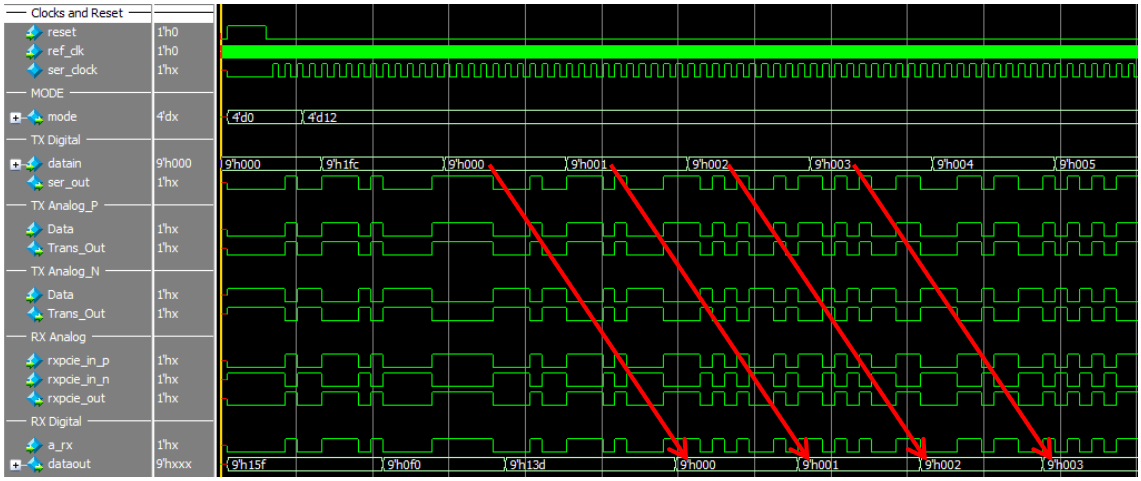

Figure 3.44 Operating Mode 11 – TXD Full Loopback waveform

For the BIST functionality, Figure 3.45 shows how the Comparator |Comp| works. The input signal {lfsr_in} and the output signals {cIn} are fed to the Comparator |Comp|. The input signal {lfsr_in} is stored in a register memory {RegMem[15:0]}. The first register of register memory {RegMem} is continuously compared to the output signal {cIn} to verify they are the same. When there is a match, the computation synchronizes and remaining registers {RegMem [15:1]} are compared the output signal {cIn}.

The result of these comparisons will be stored and can be queried outside the chip as described in Section 2.6.

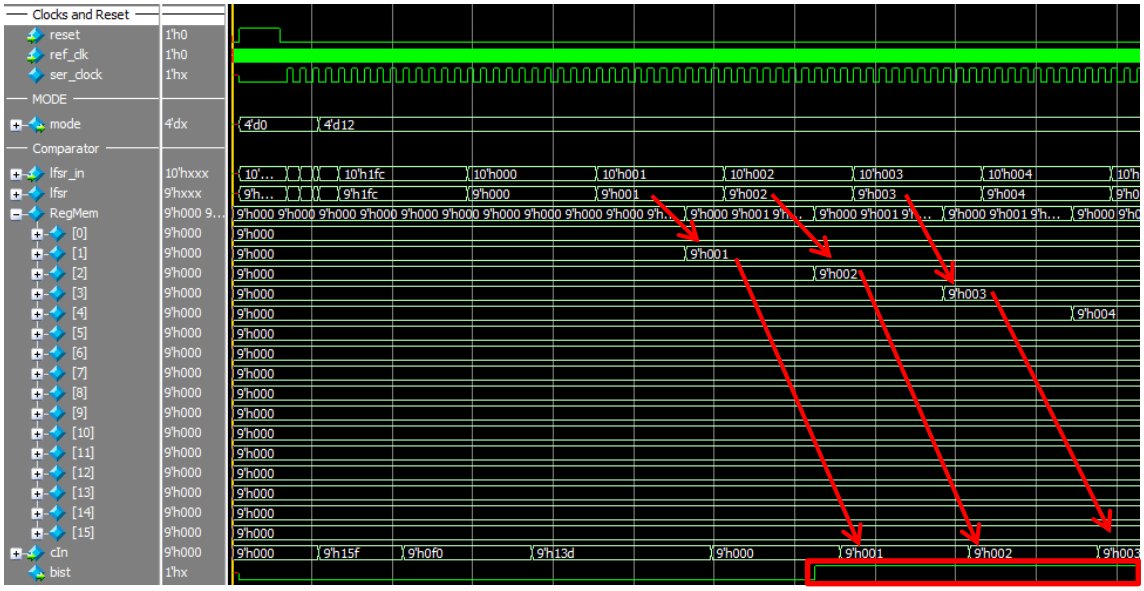

Figure 3.45 Operating Mode 11 – TXD Full Loopback BIST waveform

After completing the *logic synthesis*, which will be explained in more detail in Chapter 5, an HDL file was created by **RC Compiler**. This file was simulated using Operation Mode 11, the results appear in Figure 3.46.

Figure 3.46 Operating Mode 11 – TXD Full Loopback & BIST synthesis waveform

The BIST functionality did not pass the logic synthesis verification. Hold issues on the registers of the comparator caused problems when storing the input and output signals.

3.14 Operation Mode 13 – LFSR RXA Full Loopback & BIST

Figure 3.47 shows how this mode tests the *SerDes* in a full loop. It starts by injecting a pseudo random serial pattern {ser_out} in the Analog Receiver |RX Analog|. Then this signal loops through the rest of the Blocks of the SerDes and exits through the Analog Transmitter |TX Analog|. The Comparator |Comp| will check if the blocks behaved as expected.

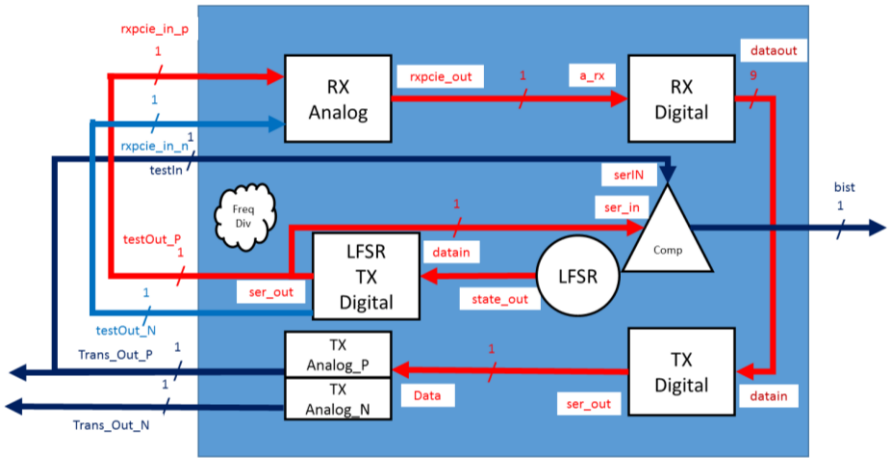

Figure 3.47 Operating Mode 13 – LFSR RXA Full Loopback & BIST diagram

The Linear Feedback Shift Register block |LFSR| generates a parallel pseudo random pattern {state_out}. This pattern {datain} is injected to the LFSR Encoder blcok |LFSR TX Digital| to be encoded and serialized. Then this signal is exposed outside the chip {testOut_P} along with a mirrored version of this signal {testOut_N}.

Then this differential signal {rxpcie_in_p & rxpcie_in_n} will be connected to the Analog Receiver |RX Analog| to be amplified, filtered and converted to a single ended signal {rxpcie_out}. After this, the input pin {a_rx} of the Digital Receiver |RX Digital| will receive the signal to deserialize and decode it. The parallel signal exiting this block {dataout} is routed to the input port {datain} of the Digital Transmitter |TX Digital| to be encoded and serialized {ser_out}. Then this serial data {Data} is sent to the Analog Transmitter |TX Analog_P & TX Analog _N| to be amplified, modulated and equalized.

A differential signal {Trans_Out_P & Trans_Out_N} will be transmitted through two of the *SerDes'* pins. The waveform on Figure 3.48 shows how the signals travel across the chip in this loop.

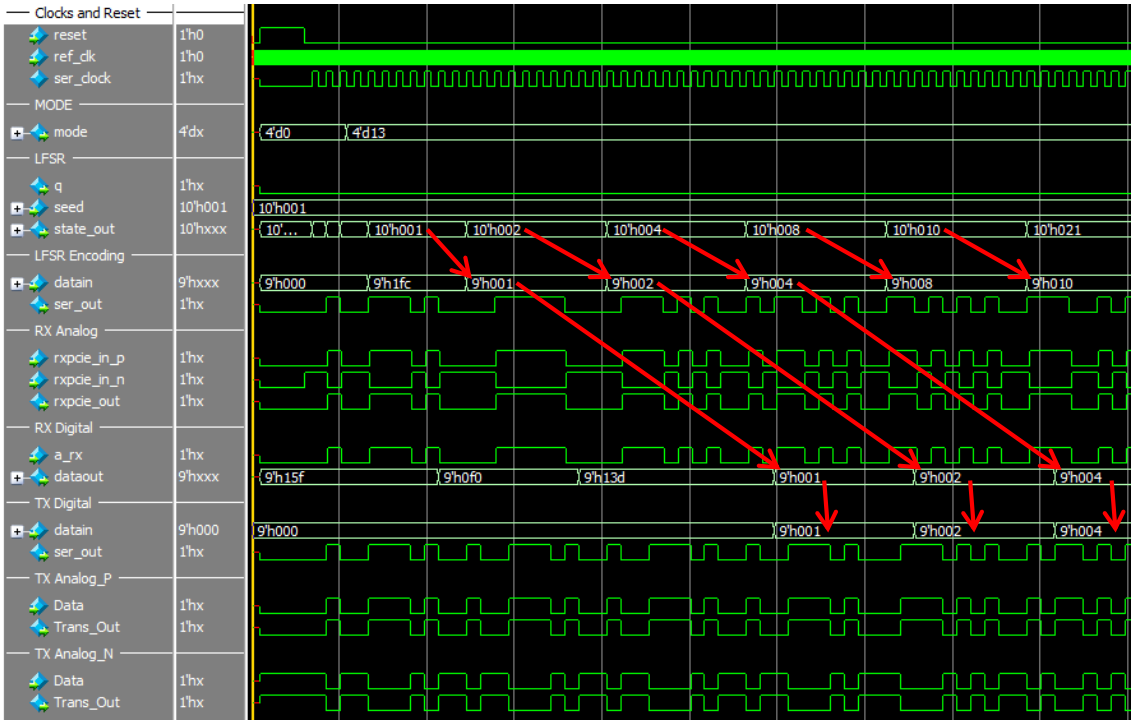

Figure 3.48 Operating Mode 13 – LFSR RXA Full Loopback waveform

The BIST functionality's verification for this operation mode appears on Figure 3.49. It shows how the input signal {ser_in} is passed to the Comparator |Comp| before it is looped through the chip. This signal {ser_in} will be stored in a register memory {RegMem2} when the first valid data is received.

In order to detect when the first valid data package arrives, two 9 bit data packages need to be received by the comparator. The first id the LFSR´s *seed* and the second has to be equals to "9'h001". After the comparator detects that this two packages have been received, it will synchronize the transmission and start storing the data in the correct order.

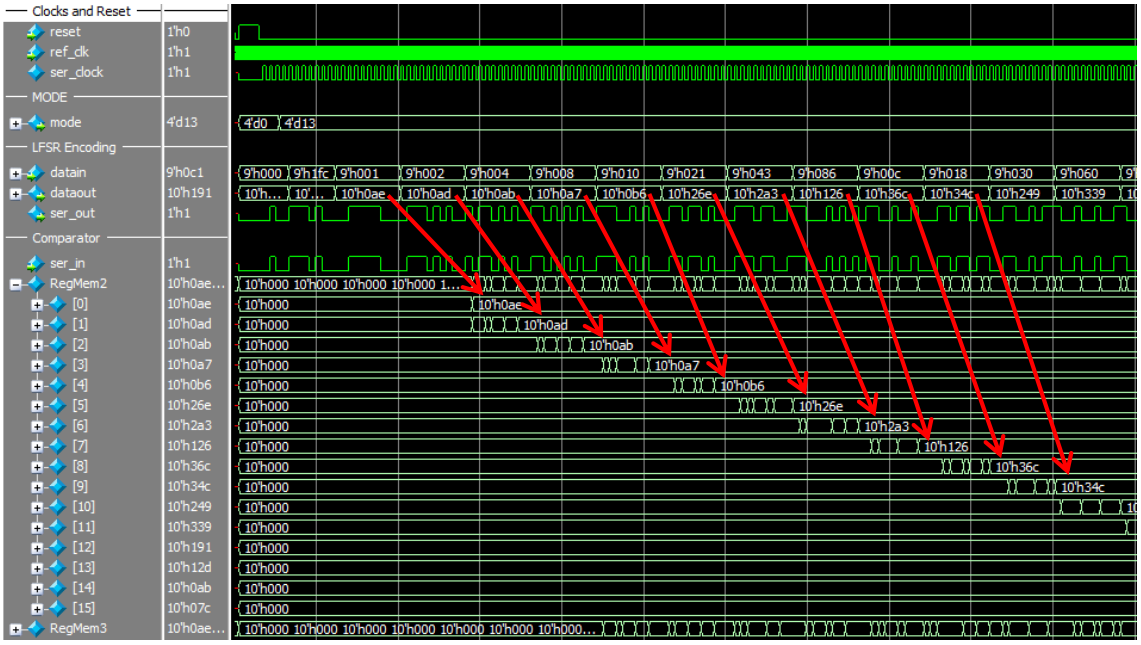

Figure 3.49 Operating Mode 13 – LFSR RXA Full Loopback input signal saving waveform

Figure 3.50 shows how the output signal {ser_out} is stored as well in a register memory {RegMem3}.The Comparator |Comp| will be continuously comparing the output signal {serIN} to the first register of the input signal {RegMem2[0]} until they are the same. When this happens, it will synchronize the storing procedure of the output signal {serIN} on a register memory {RegMem3}.

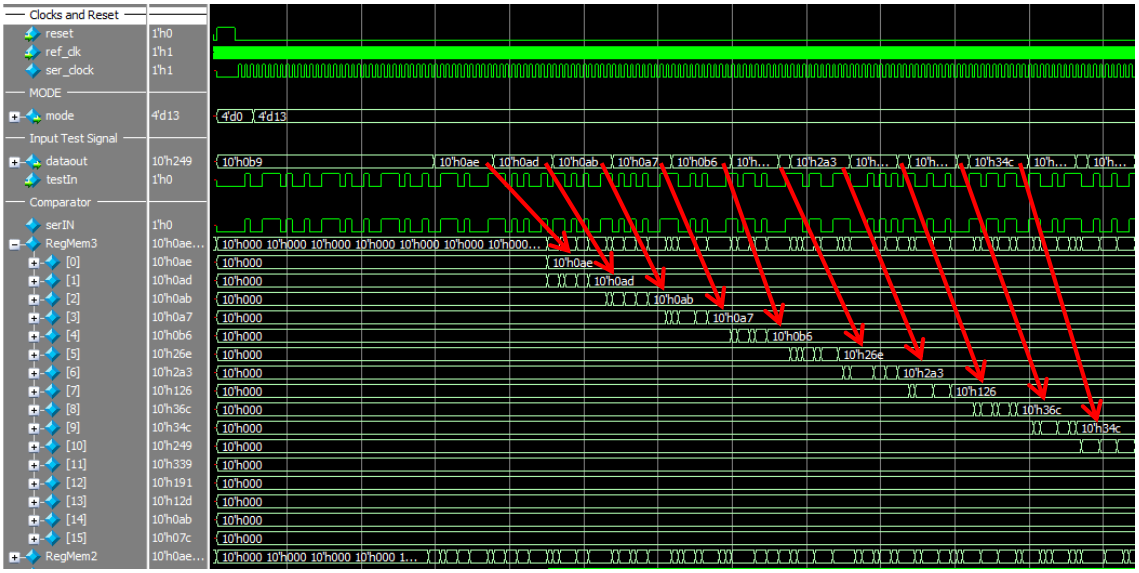

Figure 3.50 Operating Mode 13 – LFSR RXA Full Loopback output signal saving waveform

The waveform on Figure 3.51 shows the comparator performing the BIST functionality. The input signal {ser_in} that was stored in a register memory {RegMem2} and the output signal {serIN} stored in a register memory {RegMem3} will be continuously compared to verify they are the same. The result of the comparison will be stored inside the comparator and can be queried on one of the SerDes pins (see Section 2.6).

Figure 3.51 Operating Mode 13 – LFSR RXA Full Loopback BIST waveform

Figure 3.52 shows the waveform that resulted of verifying the logic synthesis´ HDL file.

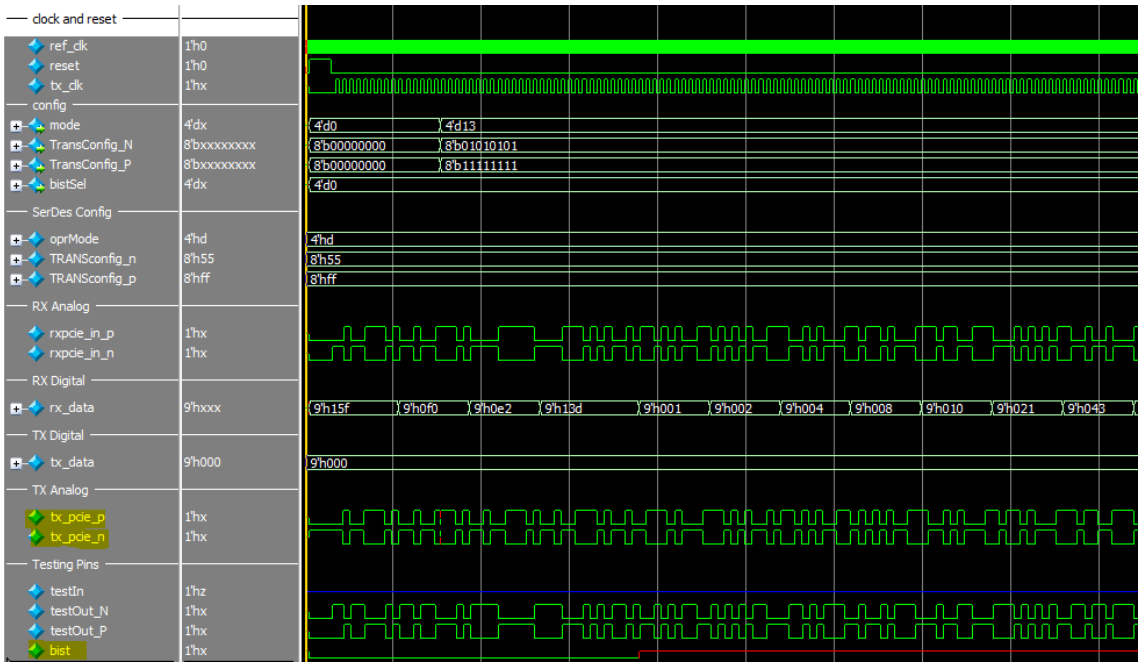

Figure 3.52 Operating Mode 13 – LFSR RXA Full Loopback & BIST synthesis waveform

Hold issues on the registers of the comparator caused problems when storing the input and output signals, causing the comparator to fail the logic synthesis verification.

3.15 Operation Mode 14 – LFSR RXD Digital Loopback & BIST

This Operation mode tests the *SerDes* in a digital loop bypassing the Analog Blocks. The loop starts from the Digital Receiver |RX Digital| and exits on the Digital Transmitter |TX Digital|. The LFSR generates a serial pseudo random pattern {ser_out} to be injected on the Digital Receiver |RX Digital|. BIST functionality is implemented as well feeding the Comparator |Comp| the input and output signals to check if the blocks behave the expected. Figure 3.53 show the diagram of this mode.

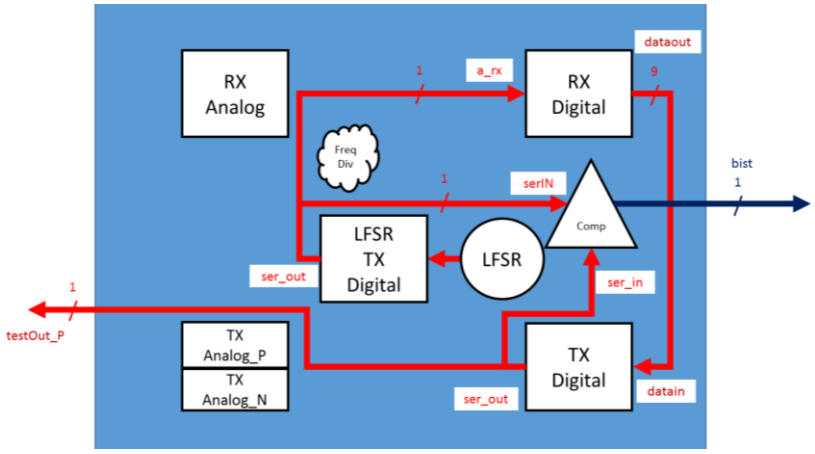

Figure 3.53 Operating Mode 14 – LFSR RXD Digital Loopback & BIST diagram

The Linear Feedback Shift Register block |LFSR| generates a parallel pseudo random pattern {state_out}. This pattern {datain} is injected to the LFSR Encoder |LFSR TX Digital| to encode and serialize it. Then this signal is sent to the input pin {a_rx} of the Digital Receiver |RX Digital| to be deserialized and decoded. The parallel signal exiting this block {dataout} is routed to the input port {datain} of the Digital Transmitter |TX Digital| to be encoded and serialized {ser_out}. The signal will be shown on one of the pins of the chip {testOut_P} to be observed. The waveform on Figure 3.54 shows how the signals travel across the chip´s digital blocks on this loop.

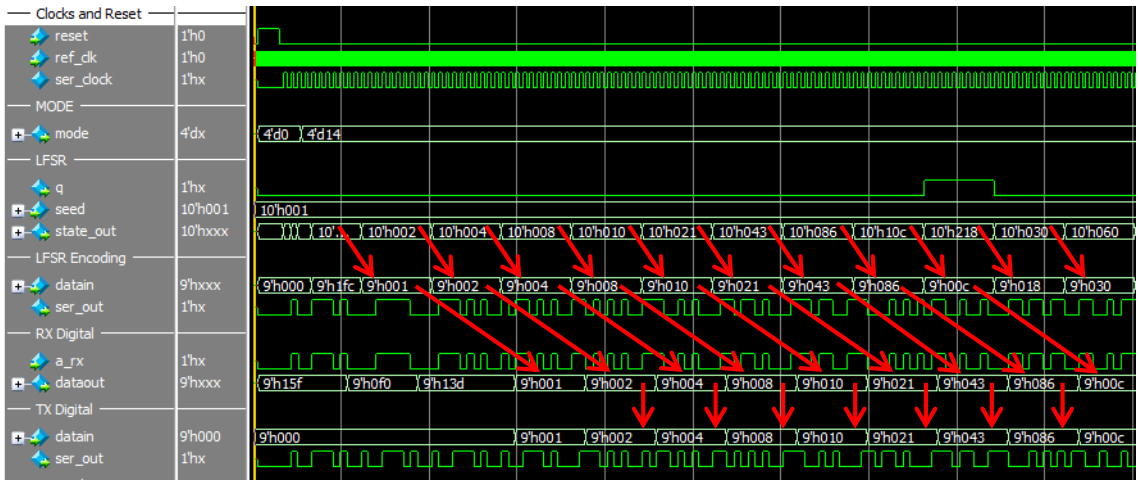

Figure 3.54 Operating Mode 14 – LFSR RXD Digital Loopback waveform

Figure 3.55 shows how the input signal {ser_in} is passed to the Comparator |Comp| and stored in a register memory {RegMem2}. Two 9 bit data packages need to be received on the comparator to synchronize this signal and start storing it. The first is the LFSR´s

seed and the second has to be equals to “9’h001”. After the comparator detects that this two packages have been received, the comparator will synchronize and will start storing the data in the correct order.

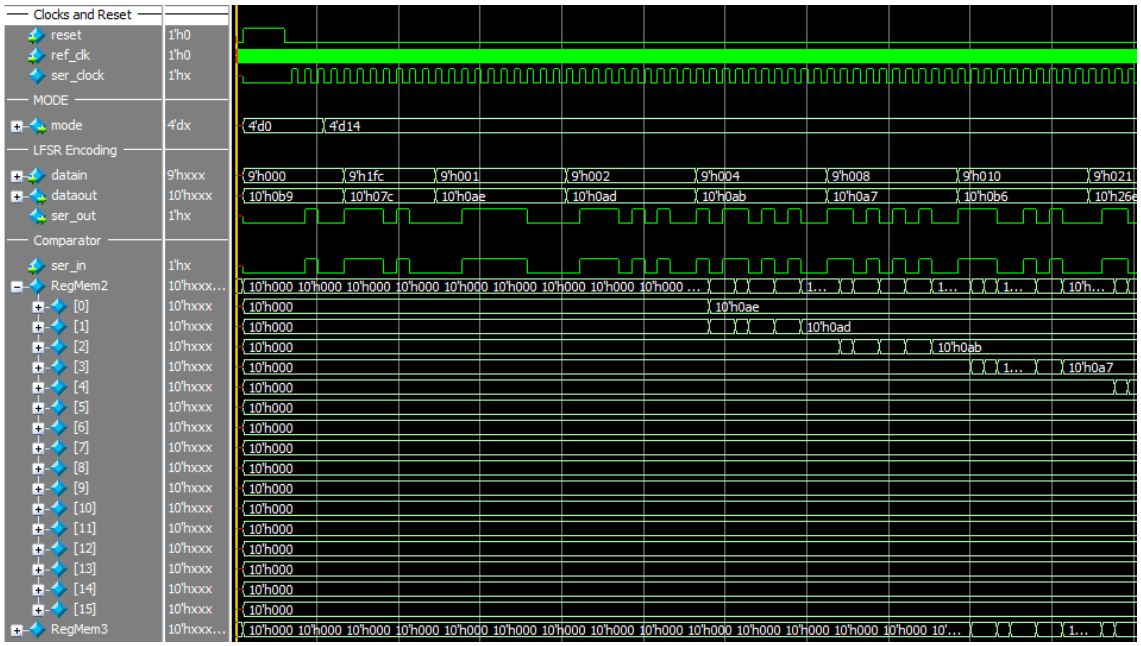

Figure 3.55 Operating Mode 14 – LFSR RXD Digital Loopback input signal saving waveform

Figure 3.56 shows how the output signal {serIN} is stored in a register memory {RegMem3}. The comparator |Comp| will be continuously comparing the output signal {serIN} to the first register of the input signal {RegMem2[0]} until they are the same. When this happens, the comparator will synchronize and the storing procedure of the output signal {serIN} on a register memory {RegMem3} will begin.

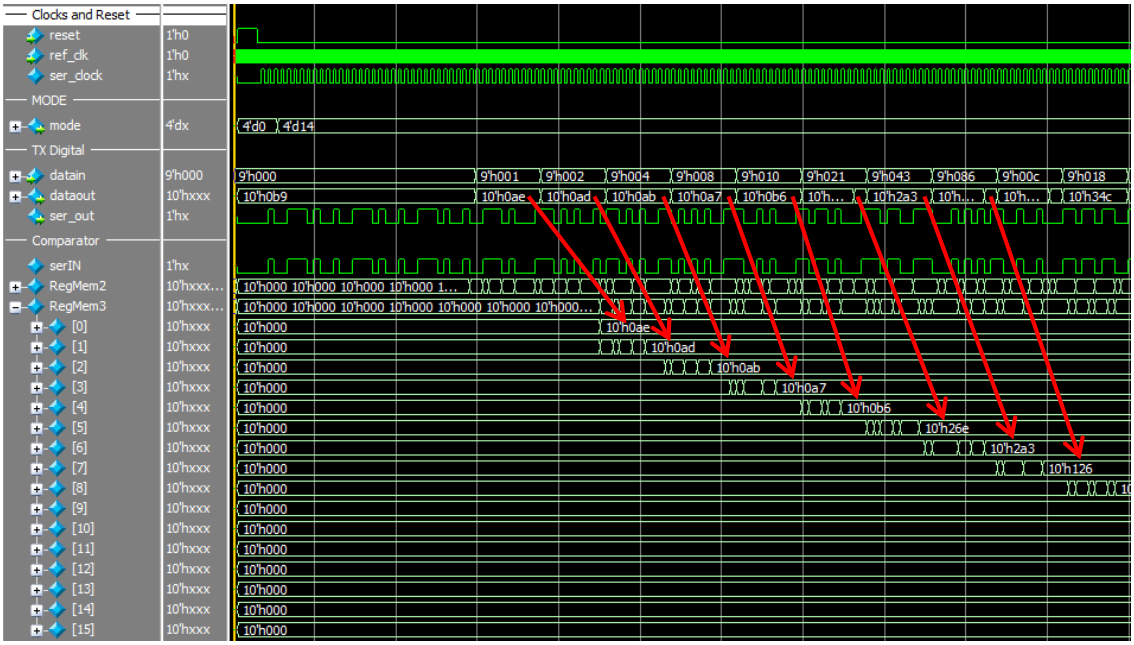

Figure 3.56 Operating Mode 14 – LFSR RXD Digital Loopback output signal saving waveform

Figure 3.57 shows the waveform that demonstrates the BIST functionality in action. The input signal {ser_in} that was stored in a register memory {RegMem2} and the output signal {serIN} stored in a register memory {RegMem3}, are continuously compared to verify if they are the same. The result of the comparison will be stored to be queried later, as described in Chapter 2 on section 2.6 .

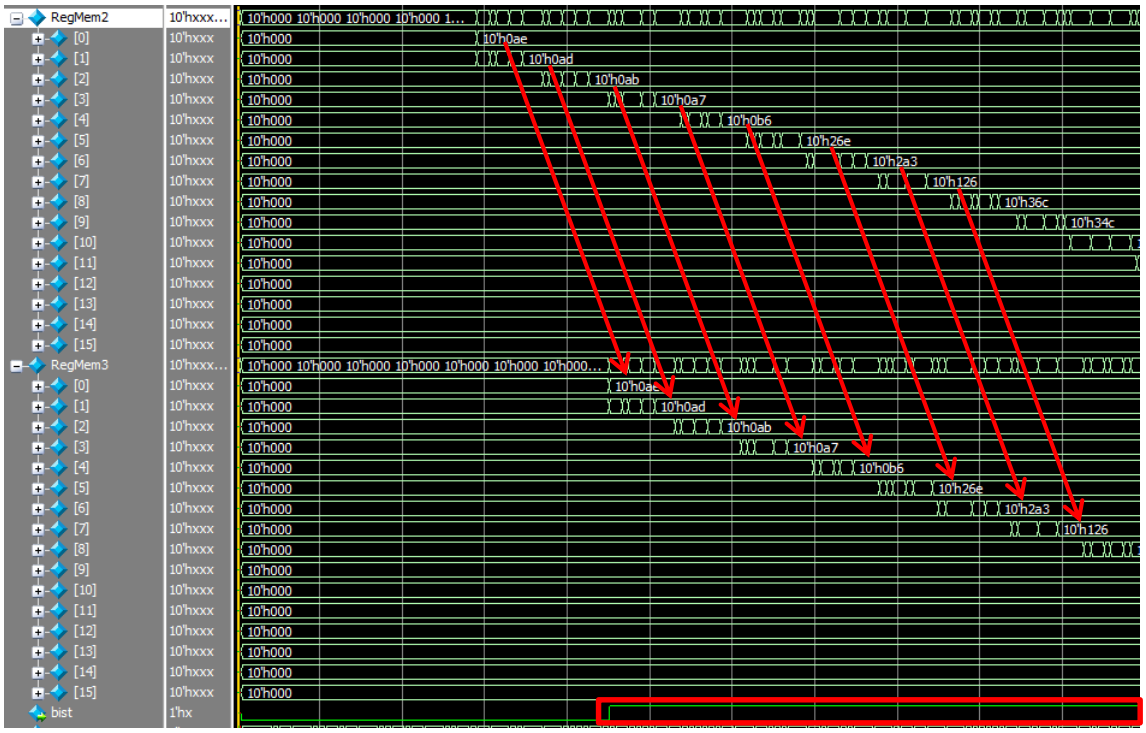

Figure 3.57 Operating Mode 14 – LFSR RXA Full Loopback BIST waveform

The logic synthesis HDL file was simulated on this Operation Mode as well. Figure 3.58 shows the waveform obtained.

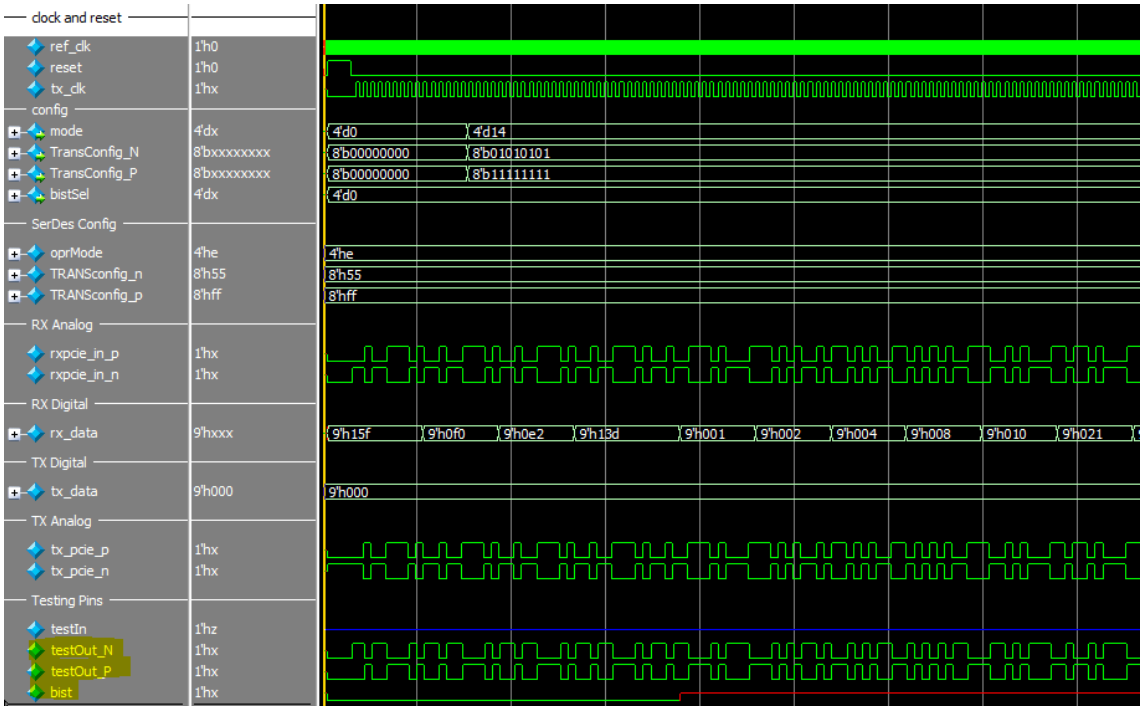

Figure 3.58 Operating Mode 14 – LFSR RXD Digital Loopback & BIST synthesis waveform

The comparator failed the logic synthesis verification due to hold issues presented on the register memories of this block.

CHAPTER 4 - LOGIC AND PHYSICAL SYNTHESIS

4.1 Logic synthesis

The logic synthesis is the translation of the RTL's behavioral model into logic gates. These gates are directly mapped to the **IBM**´s cmrf7sf Design Kit standard cells library.

For the logic synthesis, the **RC Compiler** tool was used. This tool needs input files such as HDL files, constraint files and liberty files that were specified in a TCL configuration file.

This TCL (.tcl) file had the following input settings:

- The path of a constraints (.sdc) file. This file contains timing constraints set in order to define the clocks of the system, the frequency of operation of the chip, hold and setup timing constraints, among others.
- The path of the HDL (.v) files containing the behavioral *RTL´*s description of the chip.
- The path of the .lib files of the **IBM**´s cmrf7sf Design Kit.

With this information, **RC Compiler** is able to produce a *logic synthesis*. Figure 4.1 shows the synthesis generated when importing the design of the LFSR into the tool.

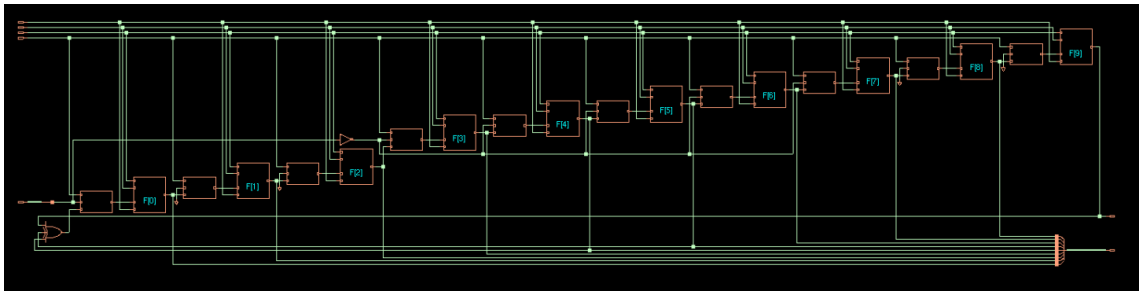

Figure 4.1 LFSR logic synthesis

After synthetizing the LFSR, the next step was to generate a logic synthesis of the whole SerDes top module. The RTL files of the top module were specified on the TCL script ant the synthesis was performed. Figure 4.2 and Figure 4.3 show the results.

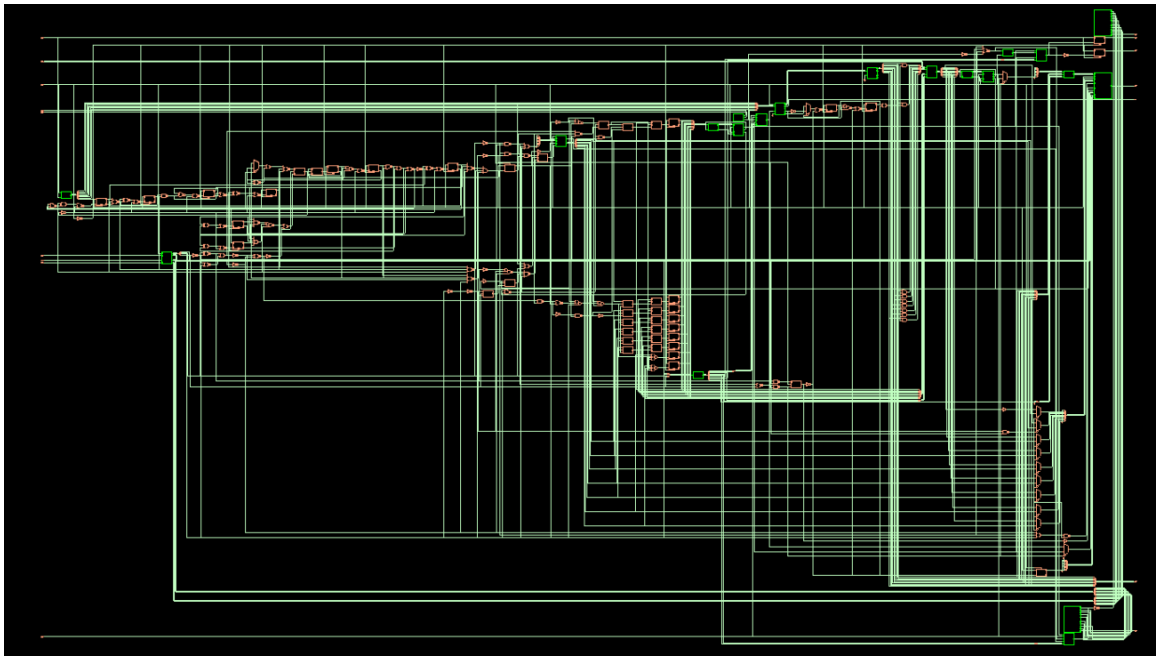

Figure 4.2 *SerDes* top level *logic synthesis* hierarchy breakdown

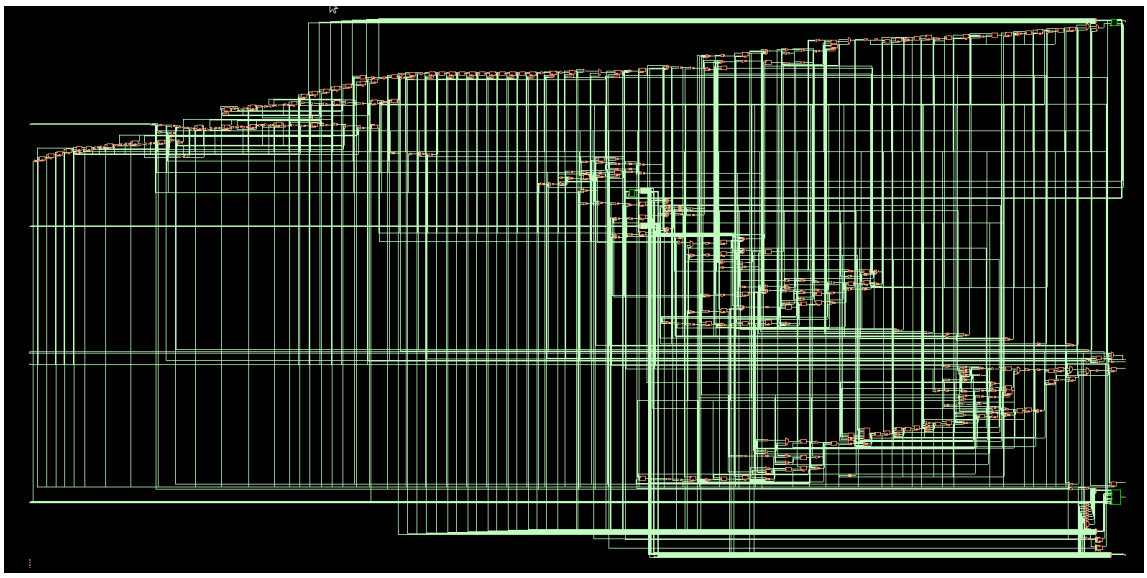

Figure 4.3 *SerDes top level logic synthesis* top view

After performing the *logic synthesis*, the compiling tool produces output files that are needed on the next step of the *VLSI* design flow. One of the primary outputs is an HDL file containing the description of the design mapped to the standard cells contained in the **IBM**´s cmrf7sf Design Kit. This HDL file was used to perform a second round of simulations on the *SerDes*, these simulations are included on Chapter 2, there is one for each operation mode and they appear at the end of each operation mode subsection.

4.2 Physical Synthesis

The physical synthesis was not completed due to timing constraints on the project, but a preliminary layout was done in order to run a full VLSI flow. This section will describe the steps performed to run a complete flow to obtain a first draft of the SerDes´ chip layout.

After obtaining the *logic synthesis* of the *SerDes* design, the resulting files were used as input for the next step of the design flow, the Physical Synthesis. To do this procedure, the **Encounter** tool was used.

This process was done with the collaboration and help of the digital design team of the ITESO TV1 SerDes chip design.

After creating a basic analyze view to instantiate the design, a floorplan was configured and a power grid was put in place. Following that, the place and route procedure was performed to create a preliminary *layout* of the digital blocks on the chip, this *layout* is shown on Figure 5.3 6 Figure 5.4.

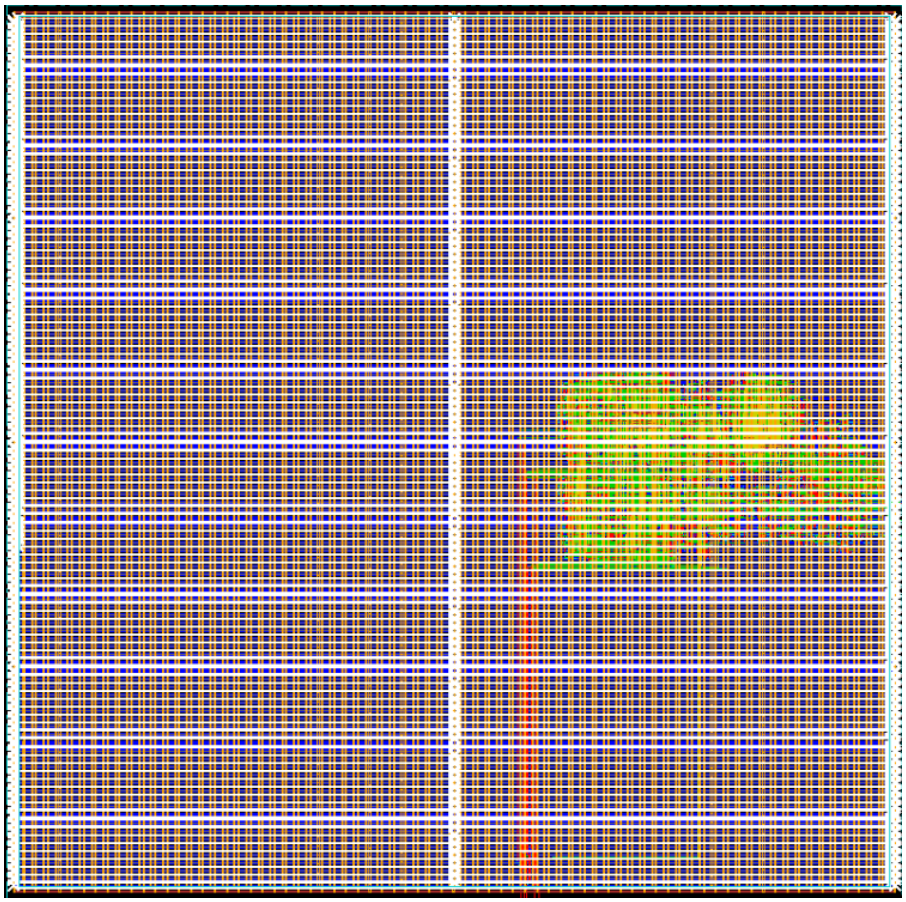

Figure 4.4 *SerDes* Digital Blocks Encounter *Layout*

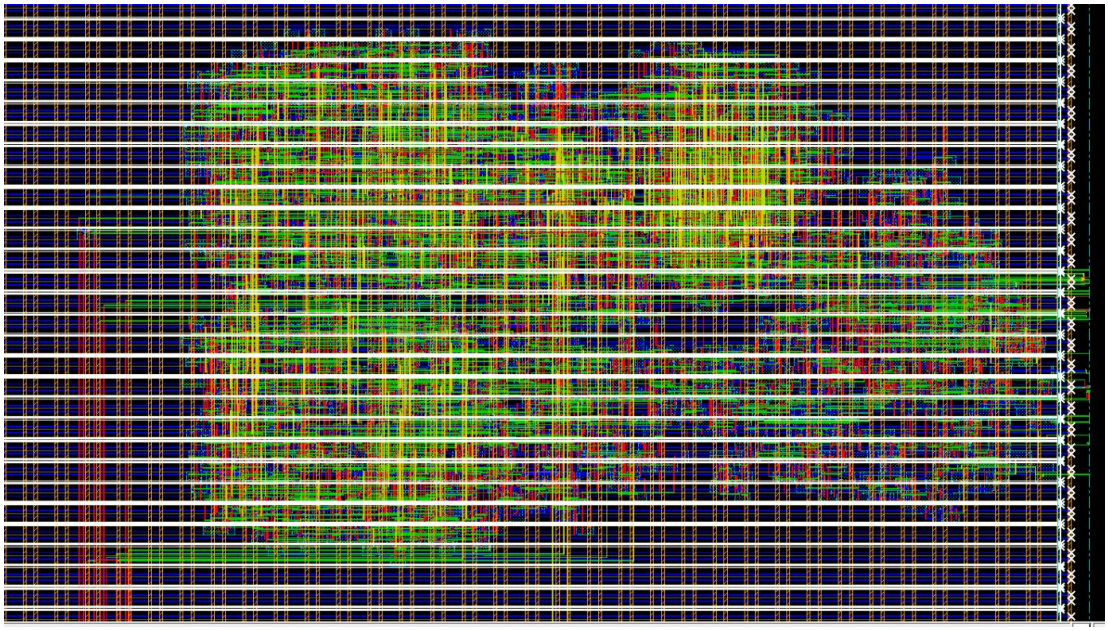

Figure 4.5 *SerDes* Digital Blocks Encounter *Layout* close up

Next a GDS file needed to be created. This is a file containing a translation of all the physical layers of the chip created by ***Encounter*** to be imported to ***Virtuoso***.

In order to create a valid GDS file, a mapping file was written to tell the export GDS feature in ***Encounter*** which **IBM**´s cmrf7sf Design Kit layer to map each of the preliminary layout physical layers.

This GDS file was imported to ***Virtuoso*** to create a layout file that will be able to connect to the Analog Modules of the chip and later generate a tape-out of the chip.

CONCLUSIONS

The development of this Testing Module will aid the Quality Assurance and functionality tests that will be performed on the ITESO TV1 SerDes chip once it is manufactured. It will provide insight into which modules are working properly and in some cases which specific part of a given module is failing. This document will serve as a guide and manual on how to operate the SerDes modes of operation.

As for the functional verifications, the behavioral model verifications were a success and show how the Testing Module interacts with the rest of the Digital Modules of the Chip. For example, the Operation Mode 8 shows that the loop of the 4 modules behaves as expected. It routes the input signals to the correct functional block and captures the input and output signals correctly on the comparator. The BIST functionality of this operation mode works properly since the comparison of the input and output signals raises the flag indicating they are the same.

As for the Logic Synthesis model verifications, the majority of the simulations were a success. All the operation modes routed the signals correctly to their corresponding functional blocks. Regarding the failed verifications, these errors were due to hold and set up errors happening on the flip-flop registers of the Comparator block. When these hold errors get fixed, then the design can be fully verified and integrated with the rest of the functional blocks of the chip.

The last portion of the work performed on the SerDes project is the physical synthesis, this section is incomplete due to timing constraints. But a complete VLSI flow was almost completed using dummy specifications and layer mappings to import the layout of the design to **Virtuoso**.

LIST OF REFERENCES

[1] A. Athavale y C. Christensen, “High-Speed Serial I/O Made Simple A Designers’ Guide, with FPGA Applications", 1.0 ed., San Jose, CA: Xcell Publications, 2005, pp. 26-28.

[2] Atul-pâté, article: “The basics of *SerDes* (serializers/deserializers) for interfacing”. Retrieved: February 4, 2015 Available at: http://www.planetanalog.com/document.asp?doc_id=528099

[3] Gabriel Torres and Cássio Lima, article: “Everything you need to know about the PCI Express”. Retrieved: February 4, 2015, Available at: http://www.hardwaresecrets.com/article/Everything-You-Need-to-Know-About-the-PCI-Express/190

[4] Unknown author, 2015-08-05, “Differential Signaling”, Online Article, Available at: https://en.wikipedia.org/wiki/Differential_signaling

[5] Unknown author, 2014-05-18, “PCI Express 4.0 Frequently Asked Questions”, Online Article, Available at: pcisig.com

[6] Unknown author: “Intervention project, Freescale for specialty in design of systems on Chip / ITESO”. Retrieved: 11 may 2015 Available at: http://cursos.iteso.mx/mod/resource/view.php?id=498817

[7] Sergio Arámbula Hampshire, "Data deserializer (*SerDes*) in *CMOS* technology of 0.5 microns," Department of electronics, systems and computer science at the Instituto Tecnológico y de Estudios Superiores de Occidente. Guadalajara, Jalisco, Feb. 2009

[8]Michael L. Bushnell y Vishwani D. Agrawal ,“15.2.3 BIST Pattern Generation” Essentials of Electronic Testing for Digital, Memory and Mixed-Signal *VLSI* Circuits, KLUWER ACADEMIC PUBLISHERS NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW, 2002

[9] Proffesor Chien-Mo James Li, 2014-26-11, “BIST - LFSR *seed* solving, Built in Self-Test-1”, Online Lecture, Available at: https://www.youtube.com/watch?v=8U_VPGjmKb0

[10] Victor Avendaño, 2015-06-28, “LFSR Implementation”

[11] Dave Lweis, article: “*SerDes* Architectures and Definitions”. Retrieved: 11 may 2015 Available at:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.173.4371 & rep rep1 & type = pdf

[12] Prof. M. Balakrishnan, 2012-02-17, “Signature Analysis & Built In Self-Test (BIST)”, Online Lecture, Available at: https://www.youtube.com/watch?v=bkA9FHQuzVU

APPENDICES

SerDes Top Verilog Code

```
module SerDes(
    input reset, //reset
    input ref_clk, //main system clock
    input rxpcie_in_p, //RXA In positive
    input rxpcie_in_n, //RXA In negative
    output rx_clk, //RXD clock
    output [8:0] rx_data, //RXD Out
    input[8:0]tx_data, //TXD parallel in
    output tx_pcie_p, //TXA Out Positive
    output tx_pcie_n, //TXA Out Negative
            input configClk,//clocking for config registers
            input dataConfig,//data input for config registers
            input setConfig,//saving configuration
    input testIn, //Bist test signal in
    output testOut_P, //Bist test signal out
    output testOut_N,
    output bist //Bist success/fail output pin
);

//Wire Declarations
            wire buffC;
            wire buffD;
            wire buffC2;
            wire buffD2;
    wire [3:0] bistSel;//bist register selector
    wire [3:0]mode;//Bist Configuration pins
    wire [7:0]TransConfig_P;//Bist Configuration pins
    wire [7:0]TransConfig_N;//Bist Configuration pins
    wire [7:0] clock_div8_phases; //clock divided in 8 phases
    wire [9:0] encoded_rx_data; //RXD internal wire (encoded data)
    wire [9:0] encoded_tx_data; //TXD internal wire
    wire  tx_disparityd; //TXD internal wire
    wire  tx_disparityq; //TXD internal wire
    wire  rx_disparityq; //RXD internal wire
    wire  rx_disparityd; //RXD internal wire
    wire  ser_clock; //serializer Clock (use this for LFSR Enable for mode 11)
    wire  lfsr_clock; //serializer Clock (use this for LFSR Enable for mode 11)
    wire rx_pcie; //RXA out //rxAn
    wire tx_pcie; //TXA In || TXD Out //txDOut
    wire c_data_valid;
    wire comma_detected;
    wire tx_disparityq2;
    wire [9:0] encoded_tx_data2;
    wire tx_disparityd2;
    wire tx_pcie2;
    wire LFSRenable; //Transmission Frame Signal from Digital Transmiter
    wire LFSRenable2; //Transmission Frame Signal from LFSR Encoded Signal
    wire soutA; //Buffer out A
    wire soutB; //Buffer out B
    wire q;//LFSR serial out
    wire [9:0]state_out;//LFSR parallel out
    wire muxA; //Multiplexer A output signal
    wire [8:0]muxB; //Multiplexer B output signal
    wire muxC; //Multiplexer C output signal
    wire muxD; //Multiplexer D output signal
    wire [8:0]muxE; //Multiplexer E output signal
    wire [9:0]sel; //Multiplexers selector signal
    wire rxAnDiv;//Frequency Divider Exit
//Wire Ceclarations

//Parameter Declarations
    parameter seedA = 10'b0011111000;    //selected Seed A
    parameter seedB = 10'b0000000001;    //selected Seed B
//Parameter Declarations
```

```
//Register Declarations
            reg FDen;
            reg OutCompS;
            reg InCompS;
            reg [9:0] InCompP;
   reg [8:0]cnt;//LFSR Seed Load Signal
   reg fFlag; //Mode 13 First Valid Data Received for Bist Syncronization
   reg [9:0]seed;
   reg [9:0]compVal;
   reg load;//activate lfsr
   reg [8:0]lfsr_out; //LFSR data out
   reg LFSRxenable;//Lfsr flip flop enable
   reg ldflag; //LFSR Load flag Signal
   reg compflag; //BIST comparator activation if mode 11
   reg compflagS; //BIST comparator activation if mode 13
   reg[8:0]rx_IN; //Digital Receiver output to connect to Mux B
   reg commaFlag;//First comma detected Flag (Mode 13)
   reg validFlag;//First Valid data decoded after the first commadetected Flag (Mode 13)
   reg FF_testOut_P; //Register Containing the Positive signal of the Testout
   reg FF_testOut_N; //Register Containing the Negative signal of the Testout
//Register Declarations

//Combinational Logic
   //Clocks Assign
   assign ser_clock = clock_div8_phases[0];
   assign lfsr_clock = clock_div8_phases[7];

   //Assign the Testout differential pins their value
   assign testOut_P=FF_testOut_P;
   assign testOut_N=FF_testOut_N;

   always@(*) begin

      //LFSR Seed Assign depending on the SerDes Operation Mode
      if(mode==4'd9) begin
         seed=seedA;
      end else begin
         seed=seedB;
      end

                  if(mode==4'd3 || mode==4'd7) begin
         FDen=soutA;
      end else begin
         FDen=0;
      end

                  if (mode==4'd8) begin
                        //InCompS=muxA;
                        //OutCompS=tx_pcie;
                        InCompS=tx_pcie;
                        OutCompS=muxA;
                  end else begin
                        InCompS=testIn;
                        OutCompS=tx_pcie2;
                  end

                  if (mode==12) begin
                        InCompP={1'b0,muxB};
                  end else begin
                        InCompP=state_out;
                  end


      //LFSR Seed Loading Depending on the Mode
      if(mode==4'd9 || mode==4'd10 || mode==4'd11 || mode==4'd13 || mode==4'd14) begin
            ldflag = 1'b1;
      end else begin
         ldflag = 1'b0;
      end

                  //Comparator parallel activation Depending on the Mode
                  if(mode==4'd10 || mode==4'd11 || mode==4'd12) begin
                        compflag=1'b1;
                  end else begin
                        compflag=1'b0;
                  end
```

```
                    //Comparator serial activation Depending on the Mode
                    if(mode==4'd8 || mode==4'd13 || mode==4'd14) begin
                            compflagS=1'b1;
                    end else begin
                            compflagS=1'b0;
                    end

        //Do not send data to the Digital Transmitter until valid data is decoded on the Digital Receiver
        if (mode==4'd13 || mode==4'd14 || mode==4'd8) begin
            if(validFlag) begin
                rx_IN=rx_data;
            end else begin
                rx_IN=0;
            end
        end else begin
            rx_IN=rx_data;
        end

        //Assign the LFSR´s Flip-Flops Enable signal depending on the mode
        if(mode==4'd11||mode==4'd10) begin
            if(LFSRenable || load) begin
                LFSRxenable=1;
            end else begin
                LFSRxenable=0;
            end
        end else if(mode==4'd9) begin
            if(lfsr_clock) begin
                LFSRxenable=1;
            end else begin
                LFSRxenable=0;
            end
        end else if(mode==4'd13 || mode==4'd14) begin
            if(LFSRenable2 || load) begin
                LFSRxenable=1;
            end else begin
                LFSRxenable=0;
            end
        end else begin
            LFSRxenable=1;
        end
    end
//Combinational Logic

//Sequential Logic
    //Testout Registers
    always @(posedge ref_clk, posedge reset)begin
        if (reset) begin
            FF_testOut_P<=0;
            FF_testOut_N<=0;
        end else begin
            FF_testOut_P<=muxD;
            FF_testOut_N<=~muxD;
        end
    end

    //Mode 13 - First Valid Data Transmition Detection from LFSR Encoded Data
    always @(posedge ser_clock, posedge reset) begin
        if (reset) begin
            commaFlag<=0;
            validFlag<=0;
        end else begin
            if (mode==4'd13 || mode==4'd14 || mode==4'd8) begin
                if (comma_detected)begin
                    commaFlag<=1;
                end
                if (commaFlag && c_data_valid) begin
                    validFlag<=1;
                    commaFlag<=0;
                end

            end else begin
                commaFlag<=0;
                validFlag<=0;
            end
        end
```

```
    end

    //LFSR output assign - Make sure the first output of the LFSR is a valid comma for the Digital Receiver
    always @(posedge ser_clock or posedge reset) begin
        if (reset) begin
            cnt <= 0;
            load <= 1;
            lfsr_out<=9'd0;
            compVal<=10'b1111111111;
            fFlag<=0;
        end else begin
            if(ldflag && cnt==4'b0000) begin
                load <= 1;
                cnt<=cnt+1'b1;
            end else if(!ldflag) begin
                cnt <=9'd0;
                load <= 0;
            end else begin
                load <= 0;
                cnt<=cnt+1'b1;
            end

                                        //Parallel seed insert with LFSR
            if (mode==4'd11||mode==4'd10) begin
                if (load) begin
                    lfsr_out<=9'b111111100;
                end else if(!load && LFSRenable) begin
                    lfsr_out<=state_out[8:0];
                end
            end

                                        //Serial Seed Insert with LFSR
            if (mode==4'd13 || mode==4'd14) begin
                if (load) begin
                    lfsr_out<=9'b111111100;
                end else if(!load && LFSRenable2) begin
                    lfsr_out<={1'b0,state_out[7:0]};
                end

                                        //Serial Seed Insert no LFSR
                                        if (mode==4'd8 || mode==4'd13 || mode==4'd14) begin
                compVal<=10'b0010101110;
            end else begin
                compVal<=10'b1111111111;
                                        end

        if(cnt==8'd159)begin
            cnt<=9'd0;
        end
        end
    end
//Sequential Logic

//Module Instantiation
    // clocking section
    clock_divider clock_dividerx(
        .a_rst(reset),
        .ref_clk(ref_clk),
        .clocks_out(clock_div8_phases));
    // Analog Receiver
    analogue_receiver analogue_receiverx(
        .rxpcie_in_p(rxpcie_in_p),
        .rxpcie_in_n(rxpcie_in_n),
        .rxpcie_out(rx_pcie));
    //Digital Receiver - Deserializer
    deserializer deserializerx(
        .a_rst(reset),
        .clocks_in(clock_div8_phases),
        .a_rx(muxA),//rx_pcie
        .c_parallel_out(encoded_rx_data),
        .clock_out(rx_clk),
        .disparity_d(rx_disparityd),
        .disparity_q(rx_disparityq),
        .c_data_valid(c_data_valid),
        .comma_detected(comma_detected));
    //Digital Receiver - Decoder (8b/10b)
```

```
decode decodex(
    .datain(encoded_rx_data),
    .dispin(rx_disparityq),
    .dataout(rx_data),
    .dispout(rx_disparityd),
    .code_err(),
    .disp_err());
//Digital Transmitter - Encoder (8b/10b)
encode encodex(
    .datain(muxB),
    .dispin(tx_disparityq),
    .dataout(encoded_tx_data),
    .dispout(tx_disparityd)
);
//Digital Transmitter - Serializer
serializer serializerx(
    .par_in(encoded_tx_data),
    .clk(ser_clock),
    .rst(reset),
    .disparity_d(tx_disparityd),
    .ser_out(tx_pcie),
    .disparity_q(tx_disparityq),
    .tx_frame_started(LFSRenable)
);
//Analog Transmitter
analog_transmitter analog_transmitterx_P(
    .Data(muxC), //tx_pcie //Inverter H
    .Eq_A(TransConfig_P[0]),
    .Eq_B(TransConfig_P[1]),
    .Imp_A(TransConfig_P[2]),
    .Imp_B(TransConfig_P[3]),
    .Imp_C(TransConfig_P[4]),
    .Imp_D(TransConfig_P[5]),
    .Amp_A(TransConfig_P[6]),
    .Amp_B(TransConfig_P[7]),
    .Trans_Out(tx_pcie_p)
        );
        INVERT_H INVERT_H1(
                .Z(muxCINV),
                .A(muxC)
        );

        analog_transmitter analog_transmitterx_N(
    .Data(muxCINV), //tx_pcie //Inverter H
    .Eq_A(TransConfig_N[0]),
    .Eq_B(TransConfig_N[1]),
    .Imp_A(TransConfig_N[2]),
    .Imp_B(TransConfig_N[3]),
    .Imp_C(TransConfig_N[4]),
    .Imp_D(TransConfig_N[5]),
    .Amp_A(TransConfig_N[6]),
    .Amp_B(TransConfig_N[7]),
    .Trans_Out(tx_pcie_n)
        );
//Test Module Instances
                //Serial Configuration Registers
                serialConfig serialConfig(
                        .rst(reset),
                        .clk(configClk),
                        .dataConfig(dataConfig),
                        .enable(setConfig),
                        .mode(mode),
                        .bistSel(bistSel),
                        .TransConfig_P(TransConfig_P),
                        .TransConfig_N(TransConfig_N)
                );
    //Mode Decoder
    muxDecode muxDecode(
        .mode(mode),
        .sel(sel)
    );
    //MUX A
    Mux_4a1 aMux(
        .A(tx_pcie),
        .B(testIn),
        .C(soutB),
```

```
    .D(tx_pcie2),
    .Sel(sel[9:8]),
    .Q(muxA)
);

//MUX B
Mux_3a1_8b bMux(
    .A(tx_data),
    .B(rx_IN),
    .C(lfsr_out),
    .Sel(sel[7:6]),
    .Q(muxB)
);
//MUX C
Mux_4a1 cMux(
    .A(tx_pcie),
    .B(testIn),
    .C(FDen),
    .D(q),
    .Sel(sel[5:4]),
    .Q(muxC)
);
//MUX D
Mux_3a1 dMux(
    .A(tx_pcie),
    .B(rxAnDiv),
    .C(tx_pcie2),
    .Sel(sel[3:2]),
    .Q(muxD)
);
//MUX E
Mux_2a1_8b eMux(
    //.A({8'b00000000,testIn}),
                        .A({8'b00000000,InCompS}),
    .B(rx_data),
    .Sel(sel[1:0]),
    .Q(muxE)
);
//BUFFER A
                BUFFER_C bufferC(
                        .Z(buffC),
                        .A(rx_pcie)
                );
                BUFFER_D bufferD(
                        .Z(buffD),
                        .A(buffC)
                );
                BUFFER_E bufferE(
                        .Z(soutA),
                        .A(buffD)
                );
freqDiv freqDiv(
    //.rxAn(soutA),
                        .rxAn(FDen),
    .rst(reset),
    .rxAnDiv(rxAnDiv)
);
                BUFFER_C bufferC2(
                        .Z(buffC2),
                        .A(soutA)
                );
                BUFFER_D bufferD2(
                        .Z(buffD2),
                        .A(buffC2)
                );
                BUFFER_E bufferE2(
                        .Z(soutB),
                        .A(buffD2)
                );
//Linear Feedback Shift Register
lfsr lfsr(
    .q(q),
    .clk(ser_clock),
    .rst(reset),
    .seed(seed),
    .load(load),
```

```
        .state_out(state_out),
        .enable(LFSRxenable)
    );
    //LFSR - Parallel out Encoder (8b/10b)
    encode encodex2(
        .datain(lfsr_out),
        .dispin(tx_disparityq2),
        .dataout(encoded_tx_data2),
        .dispout(tx_disparityd2)
    );
    //LFSR - Encoded data Serializer
    serializer serializerx2(
        .par_in(encoded_tx_data2),
        .clk(ser_clock),
        .rst(reset),
        .disparity_d(tx_disparityd2),
        .ser_out(tx_pcie2),
        .disparity_q(tx_disparityq2),
        .tx_frame_started(LFSRenable2)
    );
    //Comparator
    comparator comparator(
        .cIn(muxE),
        .clk(ser_clock),
        .rst(reset),
        //.lfsr_in(state_out[9:0]),
        .lfsr_in(InCompP),
        .compflag(compflag),
        .compflagS(compflagS),
        .LFSRenable(LFSRenable),
        .bist(bist),
        .bistSel(bistSel),
        .ser_in(OutCompS),
        //.ser_in(tx_pcie2),
        .compVal(compVal)
    );
  //Test Module Instances
//Module Instantiation
endmodule
```

Serial Configuration Block Verilog Code

```
module SerDes(
  input reset, //reset
  input ref_clk, //main system clock
  input rxpcie_in_p, //RXA In positive
  input rxpcie_in_n, //RXA In negative
  output rx_clk, //RXD clock
  output [8:0] rx_data, //RXD Out
  input[8:0]tx_data, //TXD parallel in
  output tx_pcie_p, //TXA Out Positive
  output tx_pcie_n, //TXA Out Negative
        input configClk,//clocking for config registers
        input dataConfig,//data input for config registers
        input setConfig,//saving configuration
  input testIn, //Bist test signal in
  output testOut_P, //Bist test signal out
  output testOut_N,
  output bist //Bist success/fail output pin
);

//Wire Declarations
        wire buffC;
        wire buffD;
        wire buffC2;
        wire buffD2;
  wire [3:0] bistSel;//bist register selector
  wire [3:0]mode;//Bist Configuration pins
  wire [7:0]TransConfig_P;//Bist Configuration pins
  wire [7:0]TransConfig_N;//Bist Configuration pins
  wire [7:0] clock_div8_phases; //clock divided in 8 phases
  wire [9:0] encoded_rx_data; //RXD internal wire (encoded data)
```

```
    wire [9:0] encoded_tx_data; //TXD internal wire
    wire  tx_disparityd; //TXD internal wire
    wire  tx_disparityq; //TXD internal wire
    wire  rx_disparityq; //RXD internal wire
    wire  rx_disparityd; //RXD internal wire
    wire  ser_clock; //serializer Clock (use this for LFSR Enable for mode 11)
    wire  lfsr_clock; //serializer Clock (use this for LFSR Enable for mode 11)
    wire rx_pcie; //RXA out //rxAn
    wire tx_pcie; //TXA In || TXD Out //txDOut
    wire c_data_valid;
    wire comma_detected;
    wire tx_disparityq2;
    wire [9:0] encoded_tx_data2;
    wire tx_disparityd2;
    wire tx_pcie2;
    wire LFSRenable; //Transmission Frame Signal from Digital Transmiter
    wire LFSRenable2; //Transmission Frame Signal from LFSR Encoded Signal
    wire soutA; //Buffer out A
    wire soutB; //Buffer out B
    wire q;//LFSR serial out
    wire [9:0]state_out;//LFSR parallel out
    wire muxA; //Multiplexer A output signal
    wire [8:0]muxB; //Multiplexer B output signal
    wire muxC; //Multiplexer C output signal
    wire muxD; //Multiplexer D output signal
    wire [8:0]muxE; //Multiplexer E output signal
    wire [9:0]sel; //Multiplexers selector signal
    wire rxAnDiv;//Frequency Divider Exit
//Wire Ceclarations

//Parameter Declarations
    parameter seedA = 10'b0011111000;    //selected Seed A
    parameter seedB = 10'b0000000001;    //selected Seed B
//Parameter Declarations

//Register Declarations
                reg FDen;
                reg OutCompS;
                reg InCompS;
                reg [9:0] InCompP;
    reg [8:0]cnt;//LFSR Seed Load Signal
    reg fFlag; //Mode 13 First Valid Data Received for Bist Syncronization
    reg [9:0]seed;
    reg [9:0]compVal;
    reg load;//activate lfsr
    reg [8:0]lfsr_out; //LFSR data out
    reg LFSRxenable;//Lfsr flip flop enable
    reg ldflag; //LFSR Load flag Signal
    reg compflag; //BIST comparator activation if mode 11
    reg compflagS; //BIST comparator activation if mode 13
    reg[8:0]rx_IN; //Digital Receiver output to connect to Mux B
    reg commaFlag;//First comma detected Flag (Mode 13)
    reg validFlag;//First Valid data decoded after the first commadetected Flag (Mode 13)
    reg FF_testOut_P; //Register Containing the Positive signal of the Testout
    reg FF_testOut_N; //Register Containing the Negative signal of the Testout
//Register Declarations

//Combinational Logic
    //Clocks Assign
    assign ser_clock = clock_div8_phases[0];
    assign lfsr_clock = clock_div8_phases[7];

    //Assign the Testout differential pins their value
    assign testOut_P=FF_testOut_P;
    assign testOut_N=FF_testOut_N;

    always@(*) begin

        //LFSR Seed Assign depending on the SerDes Operation Mode
        if(mode==4'd9) begin
            seed=seedA;
        end else begin
            seed=seedB;
        end

                        if(mode==4'd3 || mode==4'd7) begin
```

```
    FDen=soutA;
end else begin
    FDen=0;
end

                if (mode==4'd8) begin
                        //InCompS=muxA;
                        //OutCompS=tx_pcie;
                        InCompS=tx_pcie;
                        OutCompS=muxA;
                end else begin
                        InCompS=testIn;
                        OutCompS=tx_pcie2;
                end

                if (mode==12) begin
                        InCompP={1'b0,muxB};
                end else begin
                        InCompP=state_out;
                end


//LFSR Seed Loading Depending on the Mode
if(mode==4'd9 || mode==4'd10 || mode==4'd11 || mode==4'd13 || mode==4'd14) begin
      ldflag = 1'b1;
end else begin
    ldflag = 1'b0;
end

                //Comparator parallel activation Depending on the Mode
                if(mode==4'd10 || mode==4'd11 || mode==4'd12) begin
                        compflag=1'b1;
                end else begin
                        compflag=1'b0;
                end

                //Comparator serial activation Depending on the Mode
                if(mode==4'd8 || mode==4'd13 || mode==4'd14) begin
                        compflagS=1'b1;
                end else begin
                        compflagS=1'b0;
                end

//Do not send data to the Digital Transmitter until valid data is decoded on the Digital Receiver
if (mode==4'd13 || mode==4'd14 || mode==4'd8) begin
    if(validFlag) begin
        rx_IN=rx_data;
    end else begin
        rx_IN=0;
    end
end else begin
    rx_IN=rx_data;
end

//Assign the LFSR´s Flip-Flops Enable signal depending on the mode
if(mode==4'd11||mode==4'd10) begin
    if(LFSRenable || load) begin
        LFSRxenable=1;
    end else begin
        LFSRxenable=0;
    end
end else if(mode==4'd9) begin
    if(lfsr_clock) begin
        LFSRxenable=1;
    end else begin
        LFSRxenable=0;
    end
end else if(mode==4'd13 || mode==4'd14) begin
    if(LFSRenable2 || load) begin
        LFSRxenable=1;
    end else begin
        LFSRxenable=0;
    end
end else begin
    LFSRxenable=1;
end
```

```
    end
//Combinational Logic

//Sequential Logic
    //Testout Registers
    always @(posedge ref_clk, posedge reset)begin
        if (reset) begin
            FF_testOut_P<=0;
            FF_testOut_N<=0;
        end else begin
            FF_testOut_P<=muxD;
            FF_testOut_N<=~muxD;
        end
    end

    //Mode 13 - First Valid Data Transmition Detection from LFSR Encoded Data
    always @(posedge ser_clock, posedge reset) begin
        if (reset) begin
            commaFlag<=0;
            validFlag<=0;
        end else begin
            if (mode==4'd13 || mode==4'd14 || mode==4'd8) begin
                if (comma_detected)begin
                    commaFlag<=1;
                end
                if (commaFlag && c_data_valid) begin
                    validFlag<=1;
                    commaFlag<=0;
                end

            end else begin
                commaFlag<=0;
                validFlag<=0;
            end
        end
    end

    //LFSR output assign - Make sure the first output of the LFSR is a valid comma for the Digital Receiver
    always @(posedge ser_clock or posedge reset) begin
        if (reset) begin
            cnt <= 0;
            load <= 1;
            lfsr_out<=9'd0;
            compVal<=10'b1111111111;
            fFlag<=0;
        end else begin
            if(ldflag && cnt==4'b0000) begin
                load <= 1;
                cnt<=cnt+1'b1;
            end else if(!ldflag) begin
                cnt <=9'd0;
                load <= 0;
            end else begin
                load <= 0;
                cnt<=cnt+1'b1;
            end

                                        //Parallel seed insert with LFSR
            if (mode==4'd11||mode==4'd10) begin
                if (load) begin
                    lfsr_out<=9'b111111100;
                end else if(!load && LFSRenable) begin
                    lfsr_out<=state_out[8:0];
                end
            end

                                        //Serial Seed Insert with LFSR
            if (mode==4'd13 || mode==4'd14) begin
                if (load) begin
                    lfsr_out<=9'b111111100;
                end else if(!load && LFSRenable2) begin
                    lfsr_out<={1'b0,state_out[7:0]};
                end

                                        //Serial Seed Insert no LFSR
                                        if (mode==4'd8 || mode==4'd13 || mode==4'd14) begin
```

```
                compVal<=10'b0010101110;
            end else begin
                compVal<=10'b1111111111;
                                    end

        if(cnt==8'd159)begin
            cnt<=9'd0;
        end
        end
    end
//Sequential Logic

//Module Instantiation
    // clocking section
    clock_divider clock_dividerx(
        .a_rst(reset),
        .ref_clk(ref_clk),
        .clocks_out(clock_div8_phases));
    // Analog Receiver
    analogue_receiver analogue_receiverx(
        .rxpcie_in_p(rxpcie_in_p),
        .rxpcie_in_n(rxpcie_in_n),
        .rxpcie_out(rx_pcie));
    //Digital Receiver - Deserializer
    deserializer deserializerx(
        .a_rst(reset),
        .clocks_in(clock_div8_phases),
        .a_rx(muxA),//rx_pcie
        .c_parallel_out(encoded_rx_data),
        .clock_out(rx_clk),
        .disparity_d(rx_disparityd),
        .disparity_q(rx_disparityq),
        .c_data_valid(c_data_valid),
        .comma_detected(comma_detected));
    //Digital Receiver - Decoder (8b/10b)
    decode decodex(
        .datain(encoded_rx_data),
        .dispin(rx_disparityq),
        .dataout(rx_data),
        .dispout(rx_disparityd),
        .code_err(),
        .disp_err());
    //Digital Transmitter - Encoder (8b/10b)
    encode encodex(
        .datain(muxB),
        .dispin(tx_disparityq),
        .dataout(encoded_tx_data),
        .dispout(tx_disparityd)
    );
    //Digital Transmitter - Serializer
    serializer serializerx(
        .par_in(encoded_tx_data),
        .clk(ser_clock),
        .rst(reset),
        .disparity_d(tx_disparityd),
        .ser_out(tx_pcie),
        .disparity_q(tx_disparityq),
        .tx_frame_started(LFSRenable)
    );
    //Analog Transmitter
    analog_transmitter analog_transmitterx_P(
        .Data(muxC), //tx_pcie //Inverter H
        .Eq_A(TransConfig_P[0]),
        .Eq_B(TransConfig_P[1]),
        .Imp_A(TransConfig_P[2]),
        .Imp_B(TransConfig_P[3]),
        .Imp_C(TransConfig_P[4]),
        .Imp_D(TransConfig_P[5]),
        .Amp_A(TransConfig_P[6]),
        .Amp_B(TransConfig_P[7]),
        .Trans_Out(tx_pcie_p)
            );
            INVERT_H INVERT_H1(
                    .Z(muxCINV),
                    .A(muxC)
            );
```

```
        analog_transmitter analog_transmitterx_N(
    .Data(muxCINV), //tx_pcie //Inverter H
    .Eq_A(TransConfig_N[0]),
    .Eq_B(TransConfig_N[1]),
    .Imp_A(TransConfig_N[2]),
    .Imp_B(TransConfig_N[3]),
    .Imp_C(TransConfig_N[4]),
    .Imp_D(TransConfig_N[5]),
    .Amp_A(TransConfig_N[6]),
    .Amp_B(TransConfig_N[7]),
    .Trans_Out(tx_pcie_n)
        );
//Test Module Instances
                    //Serial Configuration Registers
                    serialConfig serialConfig(
                            .rst(reset),
                            .clk(configClk),
                            .dataConfig(dataConfig),
                            .enable(setConfig),
                            .mode(mode),
                            .bistSel(bistSel),
                            .TransConfig_P(TransConfig_P),
                            .TransConfig_N(TransConfig_N)
                    );
    //Mode Decoder
    muxDecode muxDecode(
       .mode(mode),
       .sel(sel)
    );
    //MUX A
    Mux_4a1 aMux(
       .A(tx_pcie),
       .B(testIn),
       .C(soutB),
       .D(tx_pcie2),
       .Sel(sel[9:8]),
       .Q(muxA)
    );

    //MUX B
    Mux_3a1_8b bMux(
       .A(tx_data),
       .B(rx_IN),
       .C(lfsr_out),
       .Sel(sel[7:6]),
       .Q(muxB)
    );
    //MUX C
    Mux_4a1 cMux(
       .A(tx_pcie),
       .B(testIn),
       .C(FDen),
       .D(q),
       .Sel(sel[5:4]),
       .Q(muxC)
    );
    //MUX D
    Mux_3a1 dMux(
       .A(tx_pcie),
       .B(rxAnDiv),
       .C(tx_pcie2),
       .Sel(sel[3:2]),
       .Q(muxD)
    );
    //MUX E
    Mux_2a1_8b eMux(
       //.A({8'b00000000,testIn}),
                            .A({8'b00000000,InCompS}),
       .B(rx_data),
       .Sel(sel[1:0]),
       .Q(muxE)
    );
    //BUFFER A
                    BUFFER_C bufferC(
                            .Z(buffC),
```

```
                .A(rx_pcie)
            );
            BUFFER_D bufferD(
                .Z(buffD),
                .A(buffC)
            );
            BUFFER_E bufferE(
                .Z(soutA),
                .A(buffD)
            );
        freqDiv freqDiv(
          //.rxAn(soutA),
                .rxAn(FDen),
          .rst(reset),
          .rxAnDiv(rxAnDiv)
        );
            BUFFER_C bufferC2(
                .Z(buffC2),
                .A(soutA)
            );
            BUFFER_D bufferD2(
                .Z(buffD2),
                .A(buffC2)
            );
            BUFFER_E bufferE2(
                .Z(soutB),
                .A(buffD2)
            );
        //Linear Feedback Shift Register
        lfsr lfsr(
          .q(q),
          .clk(ser_clock),
          .rst(reset),
          .seed(seed),
          .load(load),
          .state_out(state_out),
          .enable(LFSRxenable)
        );
        //LFSR - Parallel out Encoder (8b/10b)
        encode encodex2(
          .datain(lfsr_out),
          .dispin(tx_disparityq2),
          .dataout(encoded_tx_data2),
          .dispout(tx_disparityd2)
        );
        //LFSR - Encoded data Serializer
        serializer serializerx2(
          .par_in(encoded_tx_data2),
          .clk(ser_clock),
          .rst(reset),
          .disparity_d(tx_disparityd2),
          .ser_out(tx_pcie2),
          .disparity_q(tx_disparityq2),
          .tx_frame_started(LFSRenable2)
        );
        //Comparator
        comparator comparator(
          .cIn(muxE),
          .clk(ser_clock),
          .rst(reset),
          //.lfsr_in(state_out[9:0]),
          .lfsr_in(InCompP),
          .compflag(compflag),
          .compflagS(compflagS),
          .LFSRenable(LFSRenable),
          .bist(bist),
          .bistSel(bistSel),
          .ser_in(OutCompS),
          //.ser_in(tx_pcie2),
          .compVal(compVal)
        );
    //Test Module Instances
//Module Instantiation
endmodule
```

Multiplexer Decoder Verilog Code

```
module muxDecode(
input [3:0] mode,
output reg [9:0] sel
);
always @ (*) begin
		sel=10'b1000000000;
		case (mode)
			4'd0:sel=10'b1000000000;
			4'd1:sel=10'b0100000000;
			4'd2:sel=10'b0000010000;
			4'd3:sel=10'b0000000100;
			4'd4:sel=10'b0000010000;
			4'd5:sel=10'b0101100000;
			4'd6:sel=10'b0000000000;
			4'd7:sel=10'b0000100000;
			4'd8:sel=10'b1001000000;
			4'd9:sel=10'b0000110000;
			4'd10:sel=10'b1010000001;
			4'd11:sel=10'b0010000001;
			4'd12:sel=10'b1000000001;
			4'd13:sel=10'b1001001000;
			4'd14:sel=10'b1101000000;
			default:sel=10'b1000000000;
		endcase
end
endmodule
```

1 bit, 4 to 1 Multiplexer Verilog Code

```
module Mux_4a1(
input A,
input B,
input C,
input D,
input [1:0]Sel,
output reg Q
);
always @(A or B or C or D or Sel) begin
		case (Sel)
			2'b00:Q=A;
			2'b01:Q=B;
			2'b10:Q=C;
			2'b11:Q=D;
			default:Q=A;
		endcase
end
endmodule
```

8 bit, 3 to 1 Multiplexer Verilog Code

```
module Mux_3a1_8b(
input [8:0]A,
input [8:0]B,
input [8:0]C,
input [1:0]Sel,
output reg [8:0]Q
);
always @(A or B or C or Sel) begin
		case (Sel)
			2'b00:Q=A;
			2'b01:Q=B;
			2'b10:Q=C;
			default:Q=A;
		endcase
end
```

```
endmodule
```

1 bit, 3 to 1 Multiplexer Verilog Code

```
module Mux_3a1(
input A,
input B,
input C,
input [1:0]Sel,
output reg Q
);
always @(A or B or C or Sel) begin
        case (Sel)
                2'b00:Q=A;
                2'b01:Q=B;
                2'b10:Q=C;
                default:Q=A;
        endcase
end
endmodule
```

8 bit, 4 to 1 Multiplexer Verilog Code

```
module Mux_2a1_8b(
input [8:0]A,
input [8:0]B,
input [1:0]Sel,
output reg [8:0]Q
);
always @(A or B or Sel) begin
        case (Sel)
                2'b00:Q=A;
                2'b01:Q=B;
                default:Q=A;
        endcase
end
endmodule
```

Top LFSR Verilog Code

```
module lfsr(q, clk, rst, seed, load, state_out, enable);
output q;
input [9:0] seed;
input load;
input rst;
input clk;
input enable;
output wire [9:0] state_out;
wire [9:0] state_in;

flipflop F[9:0] (state_out, clk, rst, state_in, enable);
mux M1[9:0] (state_in, load, seed, {state_out[8],
state_out[7],
state_out[6],
state_out[5],
state_out[4],
state_out[3],
state_out[2],
state_out[1],
state_out[0],
nextbit});
xor G1(nextbit, state_out[4], nextbit2);
xor G2(nextbit2, state_out[5], state_out[9]);
assign q = state_out[9];
endmodule
```

LFSR Multiplexer Verilog Code

```
module mux(
output reg q,
input control,
input a,
input b
);
wire notcontrol;

always @(control or notcontrol or a or b)
        q = (control & a) | (notcontrol & b);
        not (notcontrol, control);
endmodule
```

LFSR Flip-Flop Verilog Code

```
module flipflop(q, clk, rst, d, enable);
input clk;
input rst;
input d;
output q;
input enable;
reg q;
always @(posedge clk or posedge rst)
begin
        if (rst) begin
                q<= 1'b0;
        end
        else begin
                if (enable) begin
                        q<= d;
                end
        end
end
endmodule
```

Frequency Divider Verilog Code

```
module freqDiv(
input rxAn,
input rst,
output reg rxAnDiv
);
reg counter;
always @(posedge rxAn or posedge rst) begin
        if(rst)begin
                counter<=1'b0;
                rxAnDiv <= 1'b0;
        end
        else if(counter==1'b1)          begin
                counter<=1'b0;
                rxAnDiv <= ~rxAnDiv;
        end
        else begin
                counter<=counter+1'b1;
        end
end

endmodule
```

Comparator Verilog Code

```
module comparator(
input [8:0]cIn,
```

```
input clk,
input rst,
input [9:0]lfsr_in,
input compflag,
input compflagS,
input LFSRenable,
output wire bist,
input [3:0]bistSel,
input ser_in,
input [9:0]compVal
);
wire [8:0]lfsr;
assign lfsr=lfsr_in[8:0];
wire serIN;
assign serIN=cIn[0];
reg [14:0]bisty;
parameter MEM_SIZE = 16;
reg [6:0]i;
reg [8:0] RegMem [0:MEM_SIZE -1];
reg [9:0] RegMem2 [0:MEM_SIZE -1];
reg [9:0] RegMem3 [0:MEM_SIZE -1];
reg [6:0]cntA;
reg [6:0]cntB;
reg [6:0]cntC;
reg [6:0]cntD;
//reg flag;
reg matchFlag;
reg matchFlag2;
reg[9:0]ser_outC;
reg[9:0]ser_outC2;
reg work;
always @(posedge clk, posedge rst) begin
	if(rst)begin
		for (i=0; i<MEM_SIZE; i=i+1) begin
			RegMem[i] <= 0;
			RegMem2[i] <= 0;
			RegMem3[i] <= 0;
		end
		cntA<=7'd0;
		cntB<=7'd0;
		cntC<=7'b1111111;
		cntD<=7'd0;
		//flag<=0;
		matchFlag<=0;
		matchFlag2<=0;
		bisty<=15'd0;
		ser_outC<=10'd0;
		ser_outC2<=10'd0;
		work<=0;
	end else begin
		if (compflag) begin
			work<=1;
			if (LFSRenable & lfsr!= 9'b111111100) begin
			//if (!flag) begin
			//	flag<=1;
			//end else begin
				RegMem[cntA] <= lfsr;
				if (cntA < MEM_SIZE - 1) begin
					cntA<=cntA+1;
				end else begin
					cntA<=7'd0;
				end
				if (matchFlag) begin
					if (RegMem[cntB] == cIn) begin
						bisty[cntB]<=1;
					end	else begin
						bisty[cntB]<=0;
					end
					if (cntB < MEM_SIZE - 1) begin
						cntB<=cntB+1;
					end else begin
						cntB<=7'd0;
					end
				end else begin
					if (RegMem[cntB] == cIn) begin
						bisty[cntB]<=1;
```

```
                                        cntB<=cntB+1'b1;
                                        matchFlag<=1;
                                end     else begin
                                        bisty[cntB]<=0;
                                end
                        end
                end
        end else if (compflagS) begin
                work<=1;
                if(matchFlag) begin
                        if (cntC < MEM_SIZE || cntC==7'b1111111) begin
                                RegMem2[cntC][cntD]<=ser_in;
                                cntD<=cntD + 1'b1;
                                if (cntD==6'd9) begin
                                        cntD<=0;
                                        cntC<=cntC+1'b1;
                                end
                        end
                end else begin
                        //if(ser_outC==RegMem2[0]) begin
                        if(ser_outC==compVal) begin
                                matchFlag<=1;
                                RegMem2[0]<=ser_outC;
                                RegMem2[1][0]<=ser_in;
                                cntC<=1;
                                cntD<=1;
                        end else begin
                                ser_outC<={ser_in,ser_outC[9:1]};

                        end
                end
                if(matchFlag2) begin
                        if (cntA < MEM_SIZE) begin
                                RegMem3[cntA][cntB]<=serIN;
                                cntB<=cntB + 1'b1;
                                if (cntB==6'd0) begin
                                        if(RegMem3[cntA-1]==RegMem2[cntA-1])begin
                                                bisty[cntA-1]<=1;
                                        end else begin
                                                bisty[cntA-1]<=0;
                                        end
                                end
                                if (cntB==6'd9) begin
                                        cntB<=0;
                                        cntA<=cntA+1'b1;
                                end
                        end
                end else begin
                        //if(ser_outC==RegMem2[0]) begin
                        if(ser_outC2==RegMem2[0] && matchFlag) begin
                                matchFlag2<=1;
                                RegMem3[0]<=ser_outC2;
                                RegMem3[1][0]<=serIN;
                                cntA<=1;
                                cntB<=1;
                                bisty[cntA]<=1;
                        end else begin
                                ser_outC2<={serIN,ser_outC2[9:1]};

                        end
                end
        end else begin
                if (work==1) begin
                        for (i=0; i<MEM_SIZE; i=i+1) begin
                                RegMem[i] <= 0;
                                RegMem2[i] <= 0;
                                RegMem3[i] <= 0;
                        end
                        cntA<=7'd0;
                        cntB<=7'd0;
                        cntC<=7'b1111111;
                        cntD<=7'd0;
                        //flag<=0;
                        matchFlag<=0;
                        matchFlag2<=0;
                        bisty<=15'd0;
```

```
                                        ser_outC<=10'd0;
                                        work<=0;
                                end
                        end
                end
        end
end
assign bist = bisty[bistSel];

always @(negedge compflag) begin
        for (i=0; i<MEM_SIZE; i=i+1) begin
                RegMem[i] <= 0;
        end
        cntA<=7'd0;
        cntB<=7'd0;
        //flag<=0;
        //bist<=0;
        matchFlag<=0;
        bisty<=15'd0;
end
endmodule
```

GDS Layer Mapping Configuration file

```
CA          PIN          14   0
CA          LEFPIN       14   0
CA          FILL         14   0
CA          FILLOPC      14   0
CA          VIA          14   0
CA          VIAFILL      14   0
CA          VIAFILLOPC   14   0
M1          NET          15   98
M1          SPNET        15   0
M1          PIN          15   32
M1          LEFPIN       15   0
M1          FILL         15   36
M1          FILLOPC      15   0
M1          VIA          15   0
M1          VIAFILL      15   0
M1          VIAFILLOPC   15   0
M1          LEFOBS       15   0
V1          PIN          16   0
V1          LEFPIN       16   0
V1          FILL         16   0
V1          FILLOPC      16   0
V1          VIA          16   0
V1          VIAFILL      16   0
V1          VIAFILLOPC   16   0
M2          NET          17   98
M2          SPNET        17   0
M2          PIN          17   32
M2          LEFPIN       17   0
M2          FILL         17   35
M2          FILLOPC      17   0
M2          VIA          17   0
M2          VIAFILL      17   0
M2          VIAFILLOPC   17   0
M2          LEFOBS       17   0
V2          PIN          18   0
V2          LEFPIN       18   0
V2          FILL         18   0
V2          FILLOPC      18   0
V2          VIA          18   0
V2          VIAFILL      18   0
V2          VIAFILLOPC   18   0
M3          NET          19   98
M3          SPNET        19   0
M3          PIN          19   32
M3          LEFPIN       19   0
M3          FILL         19   35
M3          FILLOPC      19   0
M3          VIA          19   0
M3          VIAFILL      19   0
```

```
M3          VIAFILLOPC    19   0
M3          LEFOBS        19   0
V3          PIN           20   0
V3          LEFPIN        20   0
V3          FILL          20   0
V3          FILLOPC       20   0
V3          VIA           20   0
V3          VIAFILL       20   0
V3          VIAFILLOPC    20   0
M4          NET           21   98
M4          SPNET         21   0
M4          PIN           21   32
M4          LEFPIN        21   0
M4          FILL          21   35
M4          FILLOPC       21   0
M4          VIA           21   0
M4          VIAFILL       21   0
M4          VIAFILLOPC    21   0
M4          LEFOBS        21   0
V4          PIN           22   0
V4          LEFPIN        22   0
V4          FILL          22   0
V4          FILLOPC       22   0
V4          VIA           22   0
V4          VIAFILL       22   0
V4          VIAFILLOPC    22   0
MT          NET           25   98
MT          SPNET         25   0
MT          PIN           25   32
MT          LEFPIN        25   0
MT          FILL          25   35
MT          FILLOPC       25   0
MT          VIA           25   0
MT          VIAFILL       25   0
MT          VIAFILLOPC    25   0
MT          LEFOBS        25   0
FT          PIN           127  0
FT          LEFPIN        127  0
FT          FILL          127  0
FT          FILLOPC       127  0
FT          VIA           127  0
FT          VIAFILL       127  0
FT          VIAFILLOPC    127  0
AM          NET           53   98
AM          SPNET         53   0
AM          PIN           53   32
AM          LEFPIN        53   0
AM          FILL          131  35
AM          FILLOPC       53   0
AM          VIA           53   0
AM          VIAFILL       53   0
AM          VIAFILLOPC    53   0
AM          LEFOBS        53   0
COMP        ALL           63   0
DIEAREA     ALL           63   0
```