INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

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

ESPECIALIDAD EN DISEÑO DE SISTEMAS EN CHIP

DESIGN FOR TESTABILITY IN A SERDES SYSTEM

Tesina para obtener el grado de:

ESPECIALISTA EN DISEÑO DE SISTEMAS EN CHIP

Presentan: Miguel Mihail Hoil Loria

Director: Víctor Avendaño Fernández, Manuel Salim Maza

San Pedro Tlaquepaque, Jalisco. Noviembre de 2017.

To my family, for their support and encouragement in this journey. No matter where we are, we will always be united by this bond.

Acknowledgements

First of all, I am very grateful to CONACYT for providing the economic support. Thank you for granting me this opportunity.

To my advisors, Victor Avendaño and Manuel Salim for all their time, advice, help, and encouragement to continue with the project. In the same way, I appreciate all the support from Professor Cuauhtémoc Aguilera.

I appreciate the assistance and advice of Professor Michele Brennan, the writing process of this work could not be possible without her help.

Last but not least, to my specialty partners Christian Aparicio, Oscar Toledo, and Alex Velásquez, whose help, support, and effort contributed to overcome the problems encountered and to develop the design flow. Thank you very much.

Abstract

Testing an IC after fabrication helps ensure chip functionality. The techniques that consider the creation and utilization of tests inside the design flow are called Design For Testability. The present work evaluates and improves the test modules implementing BIST techniques created by César Limones 2016 thesis. It is important to mention that this work reports the first effort to make the full SerDes analog and digital module integration at ITESO. It required all the designers to work together in order to complete the SerDes chip design flow. In particular, the comparison data module design structure was redefined after the data flow was analyzed. The test modules simulation demonstrated the correct functionality while the timing reports with a 156.25MHz clock frequency, showed that the design is timing compliant. The SerDes final layout, which also integrated the test modules, was created with the analog modules placement and routing. However, there were issues with the routing over the analog modules, which produced an overlap on the internal metal layers. For this reason, it is encouraged to further research about the association and outcomes between the layout of the analog modules, the LEF file generation, and the analog module routing in the design flow.

Resumen

Realizar pruebas en un chip luego de ser fabricado asegura la funcionalidad del circuito integrado. Las técnicas que contemplan la creación y aplicación de pruebas dentro del flujo de diseño del chip se llaman design for testability (diseño testable). Esta tesina evalúa y mejora los módulos de prueba que implementan técnicas de BIST, los cuales fueron creados por César Limones en la tesina del 2016. Es importante mencionar que este trabajo reporta los primeros esfuerzos realizados en el ITESO en integrar los módulos análogos y digitales que componen el SerDes. Se requirió del trabajo conjunto de todos los diseñadores para completar el flujo de diseño del chip del SerDes. En particular, la estructura del módulo de comparación fue totalmente modificada luego de que el flujo de datos fue analizado. Mediante la simulación de los módulos pruebas se demostró su correcto funcionamiento y los reportes de análisis de tiempo aplicados con un reloj a una frecuencia de 156.25MHz, mostraron que el diseño cumple con las restricciones de tiempo. El layout (diseño) final del SerDes, que integra también los módulos de pruebas, fue creado con la colocación y enrutamiento de los módulos análogos. Sin embargo, se presentaron problemas con el enrutamiento creado sobre el módulo análogo lo cual provocó un solapamiento con las capas de metal internas. Por esta razón, se exhorta a investigar sobre la asociación y resultados entre el layout de los módulos análogos, la generación del archivo LEF y el enrutamiento de los módulos análogos en el flujo de diseño.

List of figures

Figure 2.1. SerDes block diagram. 10
Figure 2.2. SerDes general structure. 10
Figure 2.3. 8b/10b encoding scheme in a system. 12
Figure 2.4. BIST system structure. 13
Figure 2.5. Standard linear feedback shift register general structure. 14
Figure 2.6. Mode 1, parallel loopback structure diagram. 16
Figure 2.7. Mode 2, serial loopback structure diagram. 17
Figure 2.8. Mode 3, RXA bypass structure diagram. 18
Figure 2.9. Mode 4, BIST with serial loopback structure diagram. 19
Figure 2.10. Mode 5, RXA bypass with parallel loopback structure diagram. 20
Figure 2.11. Mode 6, open BIST structure diagram. 20
Figure 2.12. Mode 7, RXA output with analog loopback structure diagram. 21
Figure 3.1 LFSR top module block diagram. 25
Figure 3.2. Data store finite state machine. 26
Figure 3.3. Comparison module finite state machine. 28
Figure 3.4. Preview of the comparator structural description with the integration of the store and comparison module. 28
Figure 3.5 CompartorP top module interaction with the serializer and deserializer. 34
Figure 3.6 Block diagram displaying the connections between CompartorP top module, serializer and deserializer. 35
Figure 3.7. Structure of the digital receiver input multiplexor. 36
Figure 3.8. Structure of the digital transmitter input multiplexor. 37
Figure 3.9. Structure of the analog transmitter input multiplexor. 38
Figure 3.10. Test_out multiplexor structure. 38
Figure 3.11. Structure of the digital receiver output multiplexor. 39
Figure 3.12.Test modules power grid. 41
Figure 3.13. Standard cells placed between the horizontal power stripes. 42
Figure 3.14 Test modules final layout. 42
Figure 3.15. New digital receiver output multiplexor. 45
Figure 3.16. Analog transmitter LEF file and Verilog file linking. 46
Figure 3.17. Logic synthesis log snippet. 46
Figure 3.18. SerDes pads and core after importing the design. 47
Figure 3.19. SerDes core size. 48
Figure 3.20. Horizontal stripes creation by zones. 48
Figure 3.21. Horizontal stripes placing process. 49
Figure 3.22. Analog modules before and after running the commands to move them. 50
Figure 4.1. LFSR Simulation 51
Figure 4.2 Test module worst slack path from the timing report. 52
Figure 4.3 Worst slack path taken from the timing report for the test module with combinational logic modification. 53

Figure 4.4 Worst slack for the test module where the comparison data module has registers between its combinational logic.. 54
Figure 4.5 Worst slack for the save data and comparison data modules with input data registers. 55
Figure 4.6 Comparison data module simulation for data transmission. 56
Figure 4.7 Delays generated by the registers in the comparison data module structure.............. 57
Figure 4.8 Store data module simulation. 57
Figure 4.9 Digital receiver input multiplexor simulation for three test operative modes............. 58
Figure 4.10 Digital transmitter input multiplexor simulation.. 58
Figure 4.11 Analog transmitter input multiplexor for two operating modes................................ 59
Figure 4.12 Simulation of the test_out pin outputs... 59
Figure 4.13 Digital receiver output multiplexor simulation for the modes 3b001 and 3b100...... 60
Figure 4.14 BIST modules are enabled with the test operating mode 3b100. 60
Figure 4.15 Simulation of the LSFR data generation after the enabling signa is delayed two clock cycles.. 61
Figure 4.16. Test operating mode 1, gate level simulation.. 62
Figure 4.17. Test operating mode 2, gate level simulation.. 63
Figure 4.18. Test operating mode 3, gate level simulation.. 63
Figure 4.19. Test operating mode 4, gate level simulation.. 64
Figure 4.20. Test operating mode 5, gate level simulation.. 65
Figure 4.21. Test operating mode 6, gate level simulation.. 66
Figure 4.22. Test operating mode 7, gate level simulation.. 66
Figure 4.23 Components of each design... 67
Figure 4.24. Statistics results of the comparison. 67
Figure 4.25. Worst slack path for the typical process.. 68
Figure 4.26. Worst slack path for the worst-case process.. 68
Figure 4.27. RTL to final gate level netlist comparison results.. 69
Figure 4.28. Connection error generated by the EDI tool... 70
Figure 4.29. Via errors generated and the correction.. 70
Figure 4.30. Last lines where the toll seems to not work. 71
Figure 4.31. SerDes final layout. 71
Figure 4.32. SerDes digital modules distribution in the layout. 72
Figure 4.33. Violation browser window and unconnected pin errors displayed. 73
Figure 4.34. Hold timing analysis summary.. 73
Figure 4.35. Setup timing analysis summary... 74
Figure 4.36. Layout in Virtuoso after the importation.. 75
Figure 4.37. Final layout after instance the analog modules. 75
Figure 5.1. Finite state machine transitions of the 2016 thesis... 78
Figure 5.2. Data flow in the COMP state.. 79
Figure 5.3. Error between the pads and the analog transmitter outputs....................................... 86

List of tables

Table 2.1. Overview of the available test operating modes. 16
Table 2.2. Test mode pins. 22
Table 3.1. Module inputs and outputs signals. 26
Table 3.2. Comparison module inputs and outputs. 27
Table 3.3 Comparison data module´s finite state machine transitions deployed in a table. 30
Table 3.4 Karnaugh map for flag_end_reg signal. 30
Table 3.5 Karnaugh map for the syncRst_registers_reg signal. 31
Table 3.6 Karnaugh map for the enable_registers_reg signal. 31
Table 3.7 Karnaugh map for the nxtSte_wire signal. 31
Table 3.8 Store data module´s finite state machine transitions deployed in a table. 32
Table 3.9 Karnaugh map for enable_register_reg signal. 32
Table 3.10 Karnaugh map for nxtSte_wire signal. 33
Table 3.11. Truth tables for each multiplexor selector for the digital receiver input. 36
Table 3.12 Truth tables for the digital transmitter input multiplexor´s selector bits. 37
Table 3.13.Truth table for the test_out multiplexor selector. 39
Table 3.14. SerDes pins. 44
Table 5.1. Control characters for 8-bit/10-bit codification (H-MSB, j-MSB). 77
Table 5.2. Worst slack path for each design modification. 81

List of acronyms and abbreviations

ATE	Automatic test equipment
ATPG	Automatic Test Pattern Generation
BCU	BIST Control Unit
BIST	Built-In Self-Test
Bit	Binary digit
CMOS	Complementary Metal-oxide Semiconductor
CMU	Clock multiplier unit
CTS	Clock Tree Synthesis
CUT	Circuit under test
DFT	Design For Testability
DRC	Design Rules Check
EDI	Encounter Digital Implementation
GUI	Graphical User Interface
HDL	Hardware Description Language
IC	Integrated circuit
LEC	Logic Equivalence Checker
LEF	Library Exchange Format
LFSR	Linear feedback shift register
LSSD	Level-sensitive scan design
LVS	Layout Versus Schematic
MSB	Most Significant Bit
ORA	Output Response Analyzer
PRBS	Pseudo random bit sequences
RTL	Register Transfer Level
SerDes	Serializer/Deserializer
STA	Static Timing Analysis
TPG	Test Pattern Generator

TRA	Test Response Analyzer
VLSI	Very Large Scale Integration
XOR	Exclusive-OR

Table of Contents

Instituto Tecnológico y de Estudios Superiores de Occidente i

Acknowledgements v

Abstract vii

Resumen ix

List of figures xi

List of tables xiii

List of acronyms and abbreviations xv

Table of Contents xvii

Introduction 1

1 Background 5

2 Theoretical Framework 9

2.1 THE SERDES 9
2.1.1 Overview 9
2.1.2 Structure 9
2.2 8B/10B ENCODING 11
2.3 VLSI TESTING 12
2.3.1 Design for testability 12
2.3.2 Automatic Test Pattern Generation 14
2.3.3 Comparator 15
2.4 TEST OPERATING MODES 15
2.4.1 Mode 1 – Parallel Loopback 16
2.4.2 Mode 2 – Serial Loopback 17
2.4.3 Mode 3 – RXA bypass 18
2.4.4 Mode 4 – BIST with serial loopback 18
2.4.5 Mode 5 – RXA bypass with parallel loopback 19
2.4.6 Mode 6 – Open BIST 20
2.4.7 Mode 7 – RXA output with analog loopback 21
2.4.8 Test mode pins overview 22
2.5 CHIP DESIGN FLOW OVERVIEW 22
2.5.1 Front-end design 23
2.5.2 Back-end design 23

3 Methodology 25

3.1 LFSR 25
3.2 COMPARATOR MODULE 26
3.2.1 Store data module 26

3.2.2 Comparison data module 27
3.2.3 Integration of the store data and comparison data module 28
3.2.4 Behavioral finite state machine converted to a structural description with combinational logic 29
3.2.5 Comparison data module´s finite state machine combinational logic delaying. 33
3.2.6 Register input data 34
3.3 SIGNAL DRIVER IMPLEMENTATION 35
3.3.1 Digital receiver input multiplexor 35
3.3.2 Digital transmitter input multiplexor 37
3.3.3 Analog transmitter input multiplexor 38
3.3.4 Test_out multiplexor 38
3.3.5 Digital receiver output multiplexor 39
3.4 TOP MODULE INTEGRATION 40
3.5 LOGIC SYNTHESIS 40
3.6 TEST MODULES PHYSICAL DESIGN 41
3.6.1 Power grid 41
3.6.2 Placement 41
3.6.3 Final layout 42
3.7 SERDES INTEGRATION 43
3.7.1 Chip input and output pins 43
3.7.2 Signal driver modification 44
3.7.3 Logic synthesis 45
3.7.4 Physical design 46

4 Results 51

4.1 LFSR 51
4.2 COMPARATOR MODULE 51
4.2.1 Design with behavioral finite state machines 52
4.2.2 Finite state machine described with combinational logic 53
4.2.3 Comparison data module implemented with registers between the combinational logic 54
4.2.4 Store data and comparison data modules with registers for input data 55
4.2.5 Simulation 56
4.3 SIGNAL DRIVER 58
4.3.1 Digital receiver input multiplexor 58
4.3.2 Digital transmitter input multiplexor 58
4.3.3 Analog transmitter input multiplexor 59
4.3.4 Test_out multiplexor 59
4.3.5 Digital receiver output multiplexor 59
4.4 TOP MODULE 60
4.5 GATE LEVEL SIMULATION 61
4.5.1 Operating mode 1 (3’b001) – Parallel loopback 62
4.5.2 Operating mode 2 (3’b010) – Serial loopback 62
4.5.3 Operating mode 3 (3’b011) – RXA bypass 63
4.5.4 Operating mode 4 (3’b100) – BIST with serial loopback 64
4.5.5 Operating mode 5 (3’b101) – RXA bypass with parallel loopback 65
4.5.6 Operating mode 6 (3’b110) – Open BIST 65
4.5.7 Operating mode 7 (3’b111) – RXA output with analog loopback 66

4.6 Logical Equivalence Checker 67
4.7 SerDes integration 67
4.7.1 Logic synthesis 67
4.7.2 LEC 68
4.7.3 Physical design 69
4.7.4 Virtuoso importation 74

5 Discussion 77

5.1 LFSR 77
5.2 Comparator module design 78
5.2.1 Former code in the 2016 thesis 78
5.2.2 New comparator HDL code conception 80
5.2.3 Logic synthesis's timing reports results 81
5.2.4 Simulation 82
5.3 Signal driver 82
5.3.1 The former code 82
5.3.2 The improvements 83
5.4 Test modules top level 83
5.5 SerDes integration 84
5.5.1 Logic synthesis 84
5.5.2 LEC 85
5.5.3 Physical design 85
5.5.4 Virtuoso importation 86

Conclusion 87

Appendices 91

A1. Clock divider RTL 93
A2. Analog receiver RTL 94
A3. Analog transmitter RTL 95
A4. Analog transmitter wrapper RTL 96
A5. LFSR RTL 97
A6. Memory RTL 98
A7. Store data module RTL 99
A8. Register with synchronous reset RTL 101
A9. Comparison data module RTL 102
A10. Comparator top module RTL 105
A11. Signal driver RTL 107
A12. Test modules RTL 110
A13. Flip flop T RTL 113
A14. Deserializer RTL 114
A15. Digital receiver RTL 116
A16. SerDes core module RTL 117
A17. SerDes core with input/output pads RTL 120
B. Logic synthesis script 123
C. Logic synthesis constraints file 127
D. Pads positions file 128

E. VIEW DEFINITION FILE 130
F1. SETUP AND POWER GRID SCRIPT 131
F2. PLACE STANDARD CELLS AND MODULES SCRIPT 135
F3 CTS AND OPTIMIZATION 136
F4. ROUTING AND FILLERS 138
G1. TEST OPERATING MODE 1 TESTBENCH 139
G2. TEST OPERATING MODE 2 TESTBENCH 141
G3. TEST OPERATING MODE 3 TESTBENCH 143
G4. TEST OPERATING MODE 4 TESTBENCH 145
G5. TEST OPERATING MODE 5 TESTBENCH 147
G6. TEST OPERATING MODE 6 TESTBENCH 149
G7. TEST OPERATING MODE 7 TESTBENCH 151
H1. SIGNAL DRIVER VERSION 1 RTL 153
H2. TEST MODULES VERSION 1 RTL 155

References 157

Introduction

Electronic components known as Integrated Circuits (ICs) or chips, operate communicating information between one or multiple chips. The term binary digit (bit) is the elemental unit of electronic information in this field [1] and chip to chip communication involves transferring numerous bits through wires. The transmission of bits in multiple lines is named parallel transmission [2]. A key issue with parallel transmission is to synchronize the ongoing data due to the increasing speed of the data transaction [3].

The interaction between ICs through parallel transmission implies drawbacks as the load work becomes more demanding. For instance, factors like the increase in the data transmission rate and the extensive distance between chips, make a parallel bus demand more power and space, causing timing and integrity problems with the data [4]. An alternative solution is to employ a serial communication channel, that is, to transfer one bit at time [5]. This method requires fewer pins which, eases the routing between ICs.

In a particular data transmission style known as the differential signal method, the serial transmission is quick and safe as it allows a greater speed even for long transmissions and at the same time, guarantees no errors in the data and furthermore, with a lower cost [3]. Whether the communication is performed in a serial or parallel mode, there is a method to transit from serial to parallel and vice versa. The circuit involving the conversion between serial and parallel buses is the Serializer/Deserializer (SerDes).

A SerDes device is a transceiver, i.e. transmits and receives data. The deserializer function takes place when the SerDes receives a serial data and after gathering the incoming bits, outputs the information through a parallel bus. The serializer mode functions receiving parallel data and transmitting one bit at time. This device allows chip to chip communication for a large quantity of bits and decreases the complexity and problems generated by a parallel bus [4]. Although the SerDes assists the IC to IC interaction, there could be problems related with the circuit manufacturing, hence, the inspection of the circuit operation is required.

Introduction

Testing an IC is a very important step within the chip development and fabrication since it improves the quality and lowers the cost. A good testing method can detect an error if the circuit fails, then, the production process involving the specifications, design or manufacturing process could be defective [6]. Furthermore, when the number of circuits on a chip increases, the possibilities of a production error increase as well, since it only takes a defective component or wire to cause the chip to fail [7]. It is important to consider that the circuit efficiency can be predicted if a testing system is provided [8].

The techniques within the design flow that contemplate the creation and application of tests to detect design and specification problems, are defined as Design For Testability (DFT). In particular, Built-In Self-Test (BIST) is a DFT technique whose principal idea is to substitute an external check by developing and introducing additional circuits in the chip to activate and reproduce an automatic test. The basic BIST modules are a Test Pattern Generator (TPG) to create data in order to test the chip, a Test Response Analyzer (TRA), to check if the outcome produced within the TPG input is accurate, and a BIST Control Unit (BCU) to manage testing and the IC normal functionality [9]. For this project, a testing scheme is implemented to inspect and verify the overall chip operation.

As part of the specialty profile, the aim of this project is to fabricate a SerDes IC. The device elaboration complies with the system on chip design and integration as well as the mixed signal integration and verification, and the process of the physical design flow, which is the final layout required to make the chip. In addition to the SerDes functional structure, DFT techniques are applied to prove the device in real time in order to guarantee the correct assembly of the chip's internal modules after the manufacturing process has ended.

The current project is developed with a basic BIST structure. It includes a digital circuit for generating data to test the circuit, a comparator to act as the response analyzer and multiplexors that act as switches to make the SerDes internal connections to turn on the check process. The test functionality combined the BIST modules enable different test operating modes and allows checking the serialization and deserialization functions.

Introduction

Since 2015, the past two specialty generations have contributed to the test architecture for this SerDes project and as result, the actual structure consists of seven operative modes incorporating the BIST modules to test each internal module. The present document evaluates the advances and checks the accomplishment of project specifications and at the same time, points out some details and corrections in the work.

1 Background

The purpose of chip testing is to detect if there is an error and the process when the error is tracked down and a solution is proposed, is defined as diagnosis [6]. Chip testing includes test pattern generation and test application. An Automatic Test Pattern Generation (ATPG) tool produces test patterns, which are applied to a Circuit Under Test (CUT) during test application with an Automatic Test Equipment (ATE) [9].

The test application is enforced by an ATE operated with a computer. When the first ICs were produced in the 1960s, a need to automate tests also came along due to an elevated chip production. The ATE industry was developed in the mid-1960s and a test controlled by a minicomputer was implemented [7].

The considerable amount of time employed by engineers in creating test vectors resulted in producing a more testable IC design. Therefore, in the 1970s, the combination between design and test was denoted as DFT. The ad-hoc technique was the first established and its goal is to test the difficult sections of the circuit. Later, Level Sensitive Scan Design (LSSD) or scan design was the next DFT technique. The use of latches made LSSD a method to reduce the sequential logic testing problems. The last DFT technique was BIST, introduced around 1980 aiming to operate internal testing into the IC since CUT has an inner test circuitry [7].

The current work introduces a SerDes with a BIST structure based primarily on a comparator and a LFSR. Between the different test structures implementing BIST, S. Sunter, A. Roy, and J. F. Cote introduced a jitter technique combined with logic BIST and boundary scan in order to totally test the SerDes [10]. Another work done by W.-Z. Chen and G.-S. Huang presents a low power programmable Pseudo Random Bit Sequences (PRBS) and a low noise clock multiplier unit (CMU) for 10Gbps SerDes implemented in 180nm. The PBRS is a test pattern generator and the CMU determines the data jitter [11]. The BIST structure in this work, in comparison with the two previous works, does not introduce any new test structure but, instead, a basic and simple implementation.

1 Background

In the last two specialty's generations two students have contributed in developing the test modules in order to apply the acquired knowledge, aiming to integrate all the SerDes modules and send the SerDes chip for manufacturing. One of the key issues that have not allowed the fabrication of a SerDes chip so far, are the problems encountered in terms of compliance and timing constraints.

Ricardo Godínez Maldonado proposed the first architecture in "Test Module Design for ITESO TV1 SerDes" [12] designed for a CMOS 180nm technology. He introduces the BIST modules and analyses test requirements in function of every module responsible of the SerDes functionality; therefore, including the basic serialization/deserialization function, 14 test operating modes were available within the SerDes structure and handled by the BCU. With all these modules, a frequency divider, a comparator with two memories of 15 registers each and a Linear Feedback Shift Register (LFSR) with a serial and parallel data output were also implemented, in addition to the actual test structure, in order to work with an operative mode functionality.

The functionality of all the operating modes was verified in behavioral and gate level models with simulations. The behavioral models performed as expected according to the operating mode, however, not all testing modes at the gate level model performed as expected. Setup and hold time issues were found in the operating modes where the comparator was involved. Nevertheless, the work continued with the physical design flow in order to complete the full chip design, even though the functionality was defective.

César Fernando Limones Mora in "Test Modules Design for a SerDes Chip in 130 nm CMOS technology" [13] migrated Godínez's work from CMOS 180nm to CMOS 130nm technology and simplified the architecture instead of improving its functionality because "this type of design increased the complexity and debug capability of the code", in other words, the circuits for the test structure are larger than the SerDes circuitry. Hence, the test modules were redesigned with simpler modules. As a result, 8 test operating modes were implemented including the SerDes normal operating mode.

The new implementation passed the functionality simulation with behavioral and gate level models. However, a Static Timing Analysis (STA) result with a 125 MHz frequency did not accomplish the setup timings and, once again, the comparator's combinational logic did not achieve the specifications. The solution then, was to reduce frequency to 111 MHz, which allowed to satisfy timing constraints and carry on with the physical design flow.

The preliminary works support the test modules structure as well as the test methods implied, the points to improve and the purpose for each operating mode. An important goal is to improve the comparator and make it comply with time specifications since the previous works have not accomplished time constraints.

2 Theoretical Framework

2.1 The SerDes

2.1.1 Overview

In modern high-performance digital communication systems and most digital interfaces in computer systems high-speed SerDes are fundamental [14]. The SerDes allows chip to chip communication through serial data transmission making the transition from parallel data in order to perform a high-speed transaction. A SerDes circuit is implemented on the two sides of the serial transmission, the transmitting and receiving end, therefore, transition between parallel and serial data possible [15].

Originally, the telecommunication industry and specific businesses made use of high-speed SerDes. Today, it is used in every sector of the electronics industry [3]. A SerDes circuit can be found in DVI/HDMI, Serial ATA, USB 3.0, Gigabit Ethernet, PCI Express, and some interface electronics [16].

2.1.2 Structure

Within the SerDes architectures, four different structures can be distinguished: parallel clock SerDes, 8b/10 SerDes, embedded clock bits (alias start-stop bit) SerDes, and bit interleaving SerDes. The 8-bit/10-bit encodes each parallel data byte to send into a 10-bit code and is transmitted by the serializer. The receiver can detect the 10-bit encoded data upon receiving a comma character, a special symbol delimiting the transmission [17].

Figure 2.1 depicts a generic SerDes block Diagram with 8b/10b encoding. Each function has its own block, the serializer is followed by a Tx line interface and the deserializer is preceded by the Rx line interface. Both blocks have in common a clock manager and a large block with multiple functions. Since the image defines a generic block diagram, the largest block encloses the functions preceding the serialization or following the deserialization. For example, the encoding and the transmit buffer functions operate before the serializer block function.

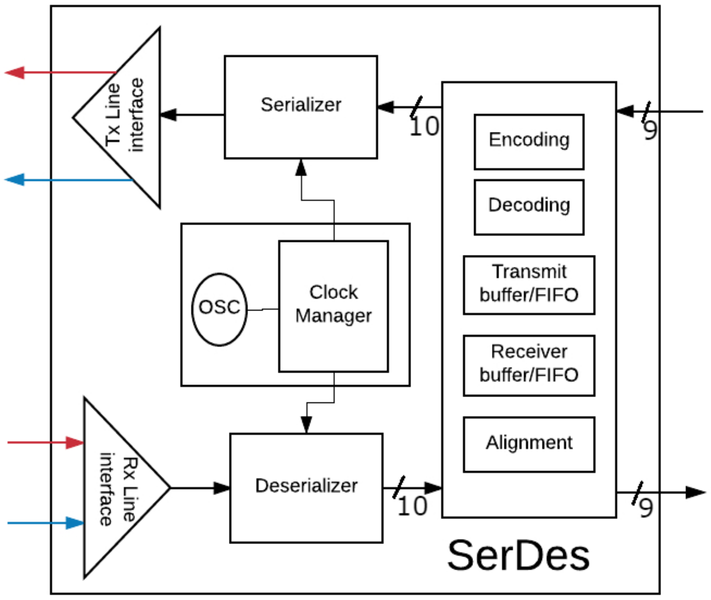


Figure 2.1. SerDes block diagram.

The SerDes structure in this project is composed of four modules classified in receiver blocks and transmitter blocks, and as analog blocks and digital blocks. In general, analog blocks process the electrical signals and digital blocks perform the serializer or deserializer function. Figure 2.2 shows the block diagram for the SerDes comprising the Analog Rx, Digital Rx, Analog Tx, and Digital Tx modules.

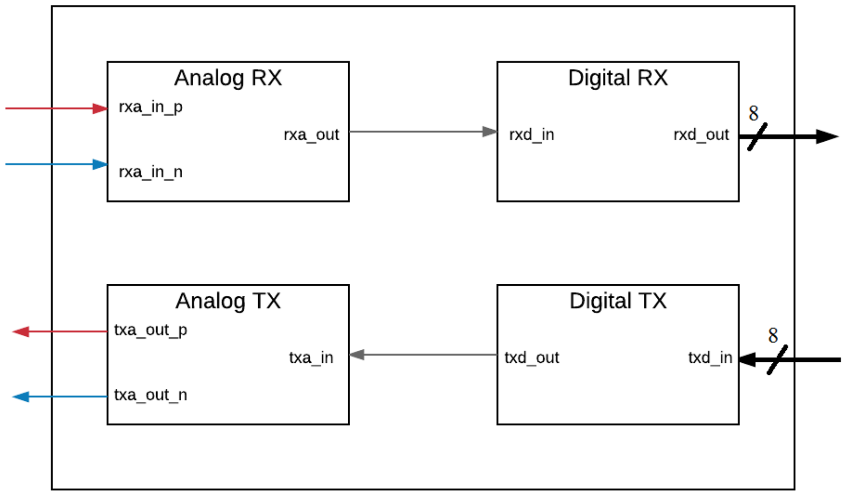


Figure 2.2. SerDes general structure.

The receiver function starts with the analog block known as Analog RX, which receives a differential signal as input and produces a digital signal as the output. The digital block, Digital RX, takes the receiver analog block output as input and recovers the data byte encoded and converts the serial data to parallel and finally, decodes the data.

The transmitter stage begins with the digital block Digital TX whose input is a parallel data byte, which is encoded and transmitted serially toward Analog TX input providing a differential signal as the output.

2.2 8b/10b encoding

The 8b/10b coding scheme was originated at IBM Corporation in 1983 by Albert X. Widmer and Peter A. Franaszek. It is used in many high-speed serial transmission standards allowing clock recovery due to the sufficient data transitions. Another advantage in clock recovery is the DC-balanced serial data stream, which contains the same quantity of 0s and 1s in the transmitted data. This coding scheme converts an 8-bit data into a 10-bit code, hence, there are 256 data characters called Dx.y and 12 control characters called Kx.y [18].

Figure 2.3 displays the simplified 8b/10b encoding and decoding system between two SerDes modules. The transmitter encodes an 8-bit data, serializes the 10-bit encoded data, and transmits bit to bit in a serial stream. The receiver deserializes the 10-bit data received and decodes it to the original byte. Although the image shows a four-data character transmission: D0.0, D31.5, D0.0, and ends with a control character K28.5, the data stream should start with a comma, thus the data stream could be the last data transmission.

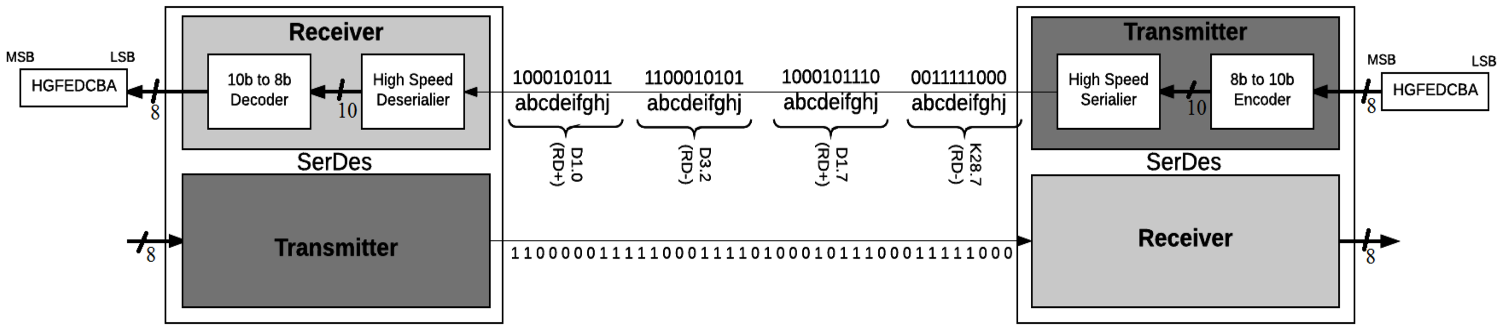


Figure 2.3. 8b/10b encoding scheme in a system.

2.3 VLSI testing

As the technology in chip fabrication advances, the production of circuits with thousands of transistors, also referred as Very Large Scale Integrated (VLSI) Circuits, improves as well. However, this enhancement comes along with increasing failure rates [19]. Therefore, testing the chip in every design flow step is important to reduce these faults.

The basic principle in digital testing consists in applying test vectors in the circuit and comparing its response to the expected output. A chip can have two types of tests: parametric tests, to check technology-dependent parameters and, functional tests, to check the right performance in a circuit. Nevertheless, a circuit test plan can be different depending on the specific chip application [6].

2.3.1 Design for testability

Historically, chip design and testing were developed as isolated and different processes. As circuit complexity and the time employed increased in testing, the test was contemplated into the design flow in order to make the circuit more testable and it was defined as DFT. The techniques developed since its conception can be classified into these categories: ad hoc DFT techniques, LSSD or scan design, and BIST [7].

The objective of considering testing in the design flow is to improve controllability and observability [9]. The term controllability is the ability to set up a specific signal in the circuit nodes after feeding it with input values. Observability is defined as the ability to figure out the value in any circuit node by controlling the inputs and observing the outputs [20].

2.3.1.1 BIST

BIST is the chip circuitry that enables an internal process to test its own correct performance. BIST techniques can be divided into two categories, online BIST and offline BIST. Online BIST techniques are employed when the chip is operating its main functionality. Concurrent online BIST test occurs at the same time with normal function operation and nonconcurrent online BIST is enforced when the circuit is in an idle state [20].

Offline BIST testing operates when the chip is not in its dedicated functionality, hence, it does not check any real time error but is employed in functionality tests. Functional offline BIST is considered usually as diagnostic software or firmware and is used to test the functional specifications of the circuit. The last technique, structural offline BIST, tests the chip structure with external or internal circuitry [7].

As quoted before, the basic BIST structure has a TPG, TRA, and BCU. Figure 2.4 illustrates a similar structure, there is a Logic BIST Controller, Output Response Analyzer (ORA) and a CUT, however, the Logic BIST Controller performs the same functionality as the BCU and the ORA performs like a TRA. Therefore, the figure depicts the interaction with the CUT and the Logic BIST Controller handling all modules to perform the testing process.

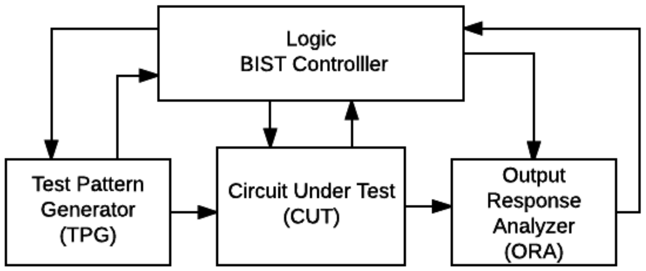


Figure 2.4. BIST system structure.

2.3.2 Automatic Test Pattern Generation

When an ATE is employed in manufacturing tests, it applies test patterns to a CUT and compares the result against the expected outcome. These test patterns are automatically generated by an Automatic Test Pattern Generator (ATPG) [21]. Test vectors can be applied in an exhaustive testing approach applying all the 2^n possible patterns to the n inputs circuit. Another approach divides the circuit into smaller parts containing less than n primary inputs and hence, each divided block is tested exhaustively [6].

2.3.2.1 Linear Feedback Shift Register

Between the different pattern generators, the LFSR is applied as TPG because it takes a reasonable area in chip space [21] and has a good performance in combinational circuits [22]. An LFSR is composed of D flip-flops and linear exclusive-OR (XOR) gates. A standard LFSR or also called external XOR LFSR has in its feedback a series of XOR gates fed on D flip-flop outputs. When the XOR gates are between the D flip-flops the structure is defined as a modular LFSR or internal XOR LFSR. A good LFSR design can be employed as a near-exhaustive TPG since it can generate 2^n-1 different states excluding the zero state [6]. Figure 2.5 shows a standard LFSR where every D flip-flop feeds a XOR gate that comprises the loopback.

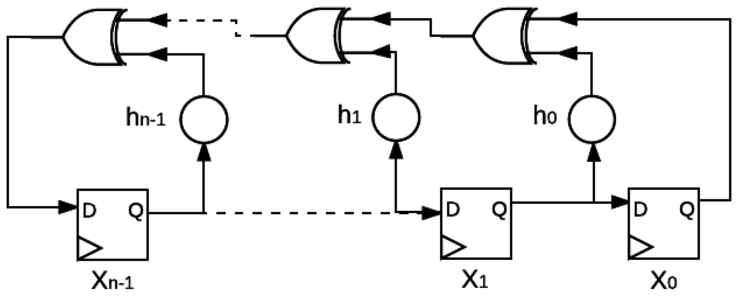


Figure 2.5. Standard linear feedback shift register general structure.

An LFSR is a stand-alone circuit where the only input required is the clock signal for D flip-flops composing the shift register. The LFSR output is the test pattern generated, which is integrated by each D flip-flop output. The quantity and locations of XORs determine the number of states in the circuit and the number of unique test patterns and, furthermore, it can be expressed with a polynomial equation called *characteristic* or *feedback polynomial*. Although these test vectors appear to be random due to random number properties, the sequence can be predicted with the LFSR current state and its characteristic polynomial. Therefore, the vectors are called *pseudorandom* vectors and the LFSR a *pseudorandom pattern generator* [9].

2.3.3 Comparator

The comparator applied in this BIST structure performs as TRA since it compares the test pattern against the system response and therefore, records the errors if there is no match. In the comparison scheme, there is a memory block whose objective is to record all the incoming test patterns in the transmitter. When the receiver processes the incoming data, the comparator performs the comparison with the first test pattern recorded. This comparison process performs a basic test since the process introduces a test pattern and checks the system response.

2.4 Test operating modes

Test operating modes are composed of the SerDes modules and the BIST modules: LFSR, comparator, and signal driver. The signal driver provides different configurations for interconnecting the SerDes modules, which is not possible with its normal function, enabling the different test operating modes to test the functional blocks in different arrangements.

Although the SerDes normal functionality is not a test mode, it is included between the test operating modes in order to cover all the combinations to *config_in[2 : 0]*. The codes for each mode are listed in the Table 2.1 and the next sections describe each operative mode structure.

Table 2.1. Overview of the available test operating modes.

Available test operating modes	
Mode	**config_in[2 : 0]**
0 – SerDes operation	000
1 – Parallel loopback	001
2 – Serial loopback	010
3 – RXA bypass	011
4 – BIST with serial Loopback	100
5 – RXA bypass with parallel loopback	101
6 – Open BIST	110
7 – RXA output with analog loopback	111

2.4.1 Mode 1 – Parallel Loopback

In Figure 2.6, the block diagram shows an internal parallel loopback created between the digital receiver output and the digital transmitter output. This connection defines the operative mode name.

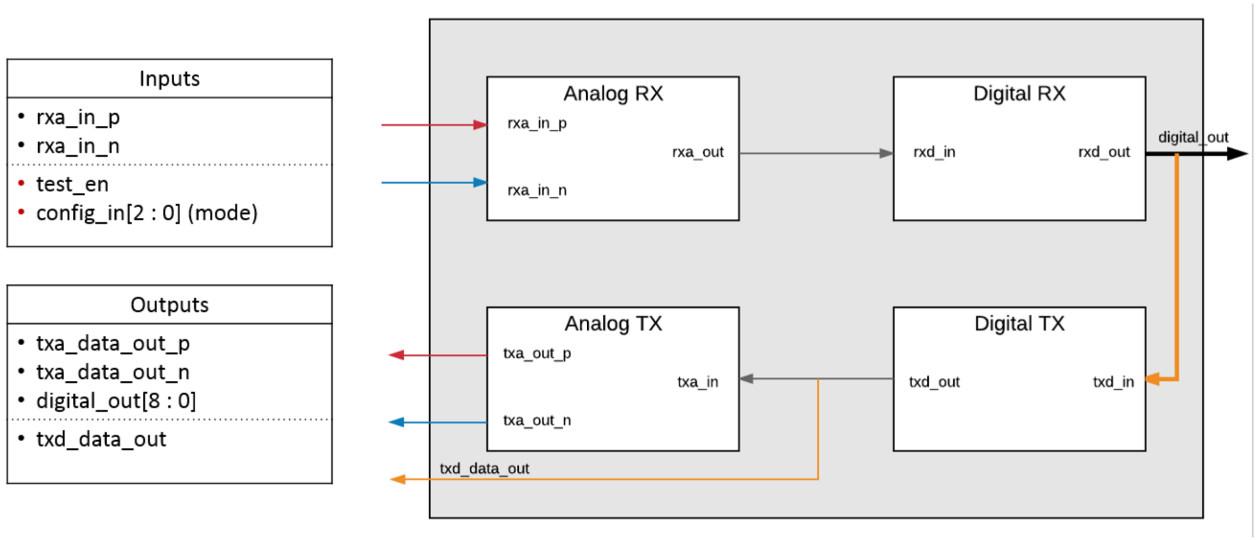


Figure 2.6. Mode 1, parallel loopback structure diagram.

Annotations:

- All the modules are in use.
- The input txd_in is multiplexed with the output digital_out.

- The system response is expected to have the output data equal to the input data.
- The parallel-serial data conversion process can be observed since the input and output data for the serialization and deserialization processes can be manually compared to ensure optimal system performance.

2.4.2 Mode 2 – Serial Loopback

Figure 2.7 illustrates mode 2 block diagram where the analog receptor module is isolated while the digital transmitter output is connected through an internal serial loop to the digital receiver input. The objective is to test the digital receiver and digital transmitter modules.

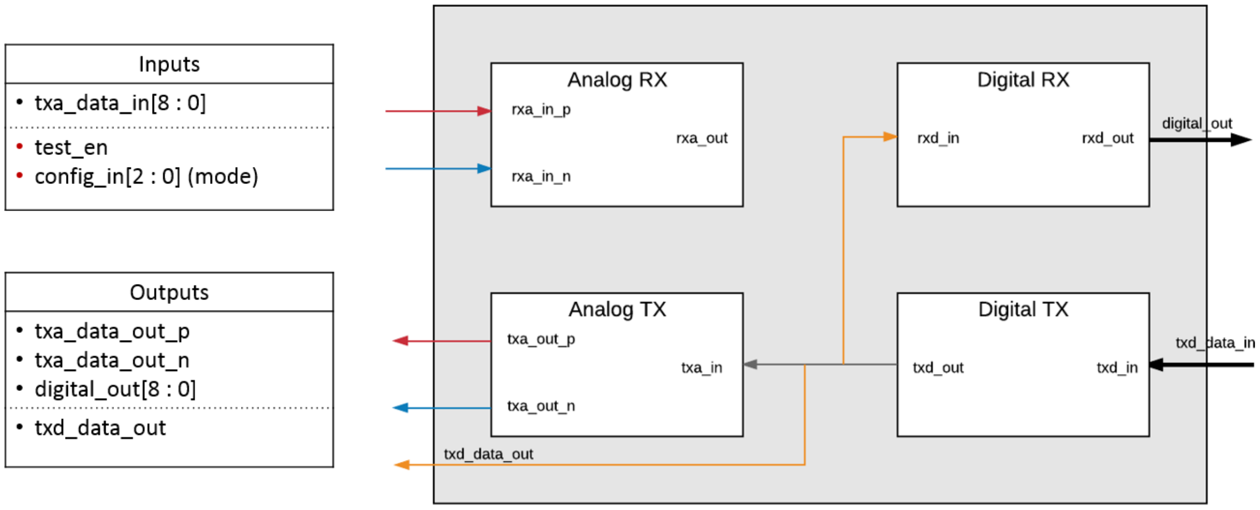


Figure 2.7. Mode 2, serial loopback structure diagram.

Annotations:

- The analog receiver is isolated.
- The analog receiver output is multiplexed with the digital transmitter output enabling the digital receiver to gather the transmitted data as input.
- The digital transmitter is tested directly having control over the data input and the possibility to check the output.
- With the test of the digital receiver and the analog transmitter their responses can be checked after providing an input signal which, is user controlled.

2.4.3 Mode 3 – RXA bypass

In this mode, the analog receiver is isolated while the digital receiver accepts input data directly from a pin enabled in this mode, *config_in[3]* in Figure 2.8.

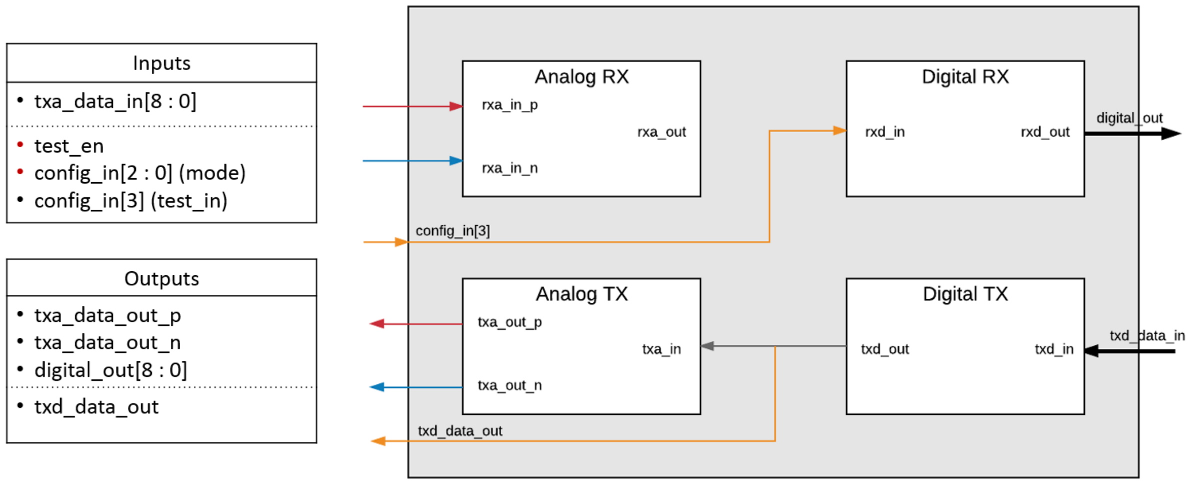


Figure 2.8. Mode 3, RXA bypass structure diagram.

Annotations:

- The analog receiver is isolated.
- The digital receiver is directly tested as result of having control of its input through the *config_in[3]* pin.
- The input data for the digital receiver must be encoded in 8b/10b.
- This mode is similar to mode 2, however, there is control over the digital receiver input data since there is no serial loop between the digital modules.

2.4.4 Mode 4 – BIST with serial loopback

This operating mode resembles mode 2 but with the addition of the BIST modules. In Figure 2.9, the LFSR, the comparator and implicitly, the BCU are shown. The LFSR generates pseudo-random patterns, which serve as input for the digital transmitter and the data generated by LSFR are compared with the digital receiver output applying the comparator module.

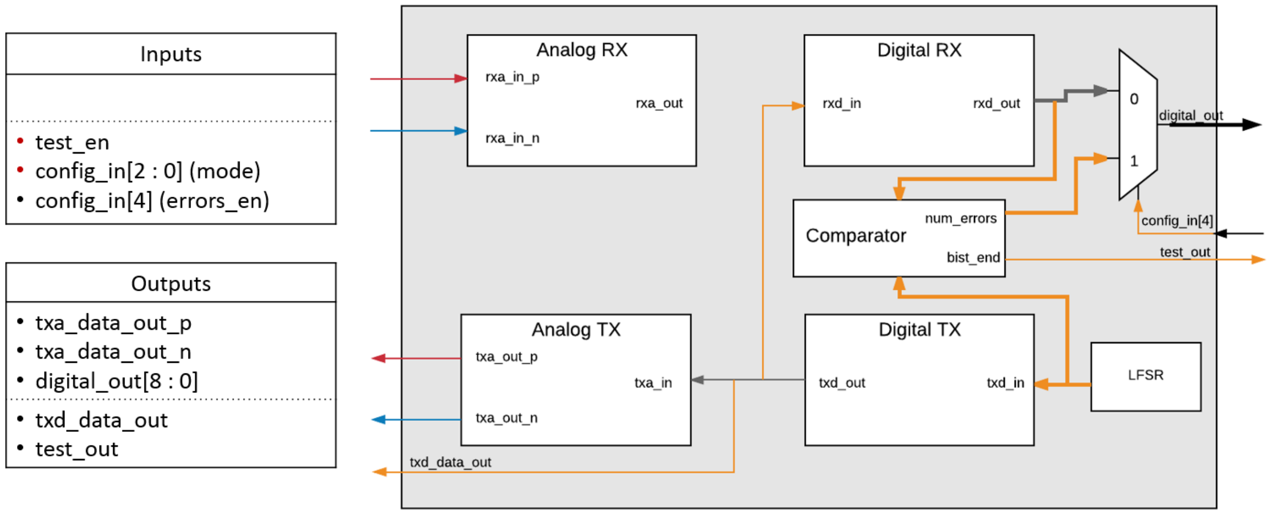


Figure 2.9. Mode 4, BIST with serial loopback structure diagram.

Annotations:

- The analog receiver is isolated.
- The BIST technique is used.
- There is no direct control over the digital transmitter data input, nevertheless it can be determined through the state of the pseudo-random data pattern.
- The comparison errors are observed by multiplexing them with the *digital_out* output.

2.4.5 Mode 5 – RXA bypass with parallel loopback

This mode, illustrated in Figure 2.10, resembles a combination of the operation mode 1 with the operation mode 3. Therefore, the analog receiver is isolated and there is a connection of the parallel data due to the digital receiver output tied to the digital transmitter input.

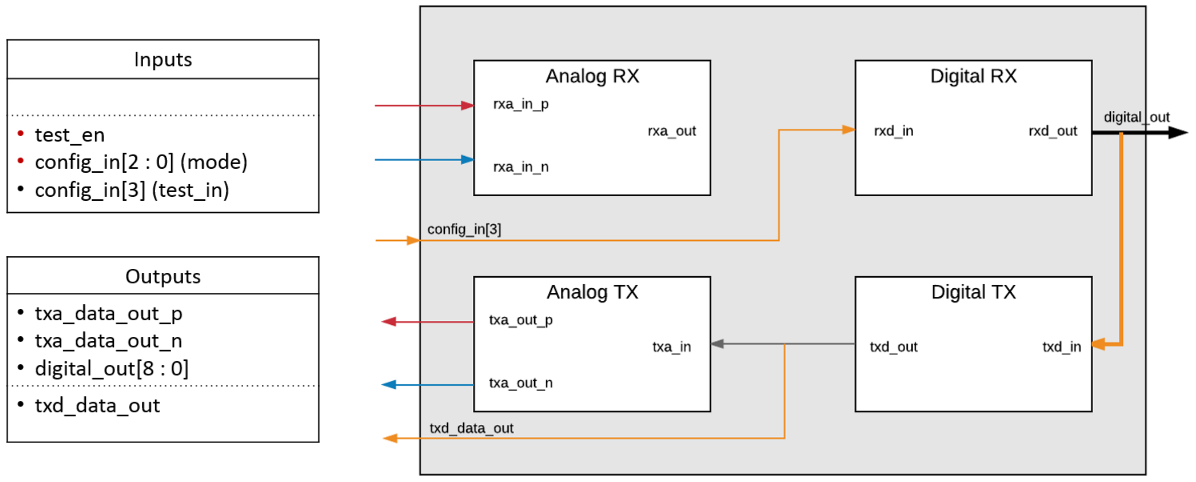


Figure 2.10. Mode 5, RXA bypass with parallel loopback structure diagram.

Annotations:

- It is a combination of the operating mode 3 and 1, but with the isolated analog receiver module.

2.4.6 Mode 6 – Open BIST

This mode performance resembles operating mode 4, however, as Figure 2.11 shows, there is no internal serial loopback and the analog receiver module operates as usual, i.e. its output is connected to the digital receiver input.

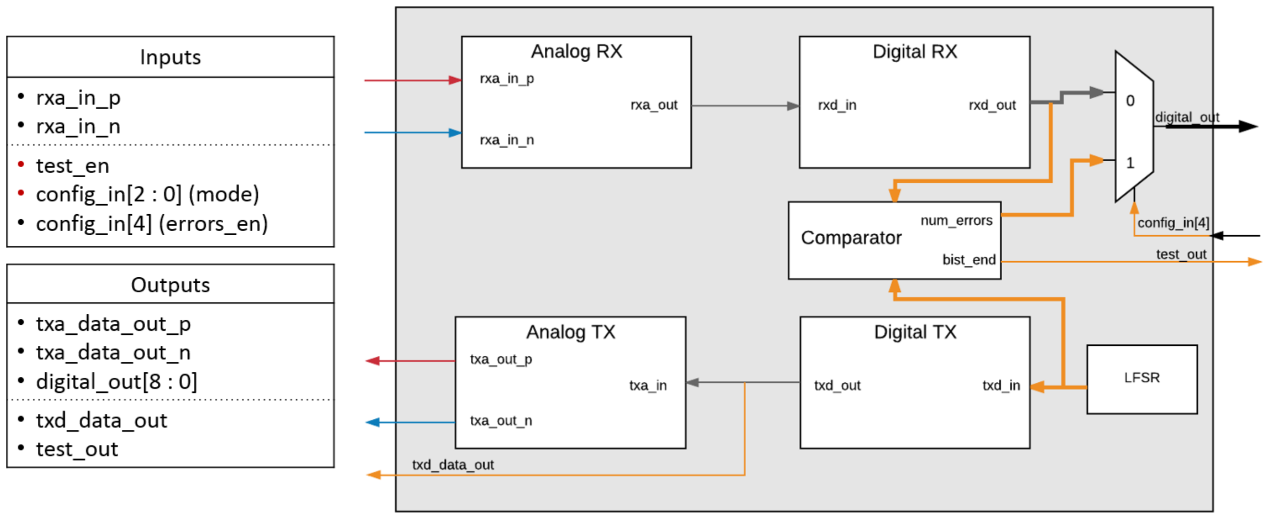


Figure 2.11. Mode 6, open BIST structure diagram.

Annotations:

- This mode is a mode 4 variation where there is no internal serial loopback so the digital receiver input is no longer connected to the digital transmitter output.
- There is a control over the data input for the analog receiver whose output is connected the digital receiver.
- The BISTs modules are in use so any input data will produce an error except if the data is the same as the generated by the LFSR, but encoded in 8b/10b.
- It is recommended to connect the analog transmitter output to the analog receiver input in order to detect errors generated by comparing the same data.

2.4.7 Mode 7 – RXA output with analog loopback

This module´s main characteristic is the internal analog loopback between the analog receiver output and the analog transmitter input, Figure 2.12 depicts the connection between Analog RX and Analog TX. The objective is to test the function of both analog modules. In addition, there is a dedicated pin to check the analog receiver output.

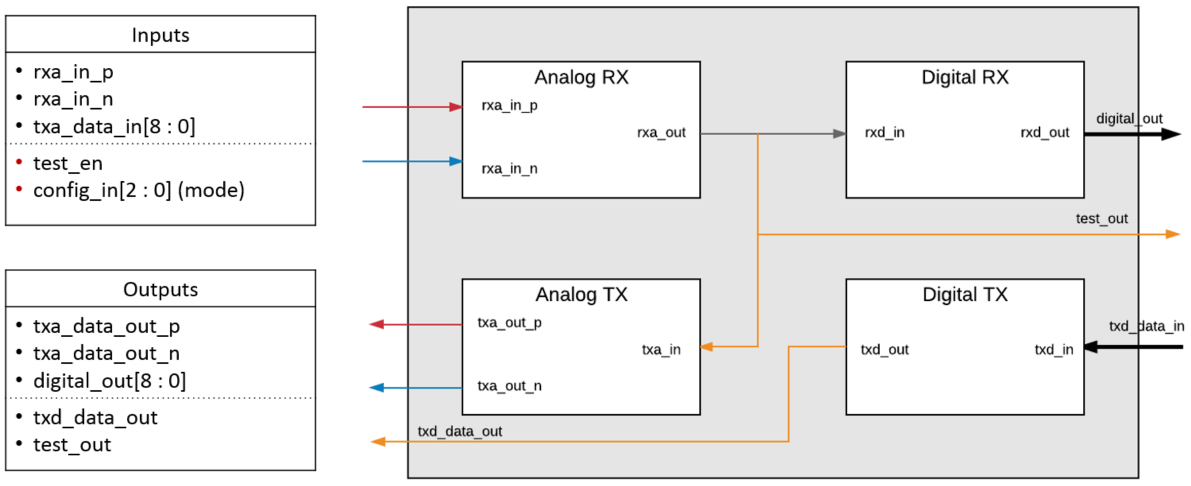


Figure 2.12. Mode 7, RXA output with analog loopback structure diagram.

Annotations:

- The digital transmitter is isolated but it is possible to test it directly.
- The analog functionality of the system is tested due to the interconnection between the analog blocks.

- The analog receiver module can be tested directly as a result of the control of its data input and a pin to analyze its output.
- The serial-parallel conversion response can be analyzed due to the possibility to check the input and output of each stage, from the analog to the digital stages.

2.4.8 Test mode pins overview

Table 2.2 summarizes the SerDes input and output pins required in order to enable the test operating modes. Only *test_en*, *txd_data_out* and *test_out* are test dedicated pins while *config_in* and *digital_out* are multiplexed for testing purposes.

Table 2.2. Test mode pins.

Test mode pins	
Inputs (6)	**Outputs (8)**
• test_en • config_in[2 : 0] (mode) • config_in[3] (test_in) • config_in[4] (errors_en)	• txd_data_out • test_out • digital_out[5 : 0] (errors_en = 1)

- The input ports ***test_en*** y ***config_in[2 : 0]*** are essential to activate the test mode.
- Five of the eight bits of ***config_in*** used by the analog transceiver are multiplexed and applied as input pins for the test mode.
- The ***errors_en*** only applies for mode 4 and mode 7 because the comparator is in use.

2.5 Chip design flow overview

Chip development is divided into two stages, front-end design, and back-end design. Front-end design starts from the device functionality, the implementation in a Hardware Description Language (HDL) until the Register Transfer Level (RTL) design is mapped in a specific transistor technology. Back-end design continues the design flow with the physical design flow, which comprises the layout development in order to send it to the manufacturing process.

2.5.1 Front-end design

Chip design starts with the device specifications. The architecture is planned after the functional specifications and translated to an HDL where the RTL is described. After a cycling process simulating the RTL, as well as verifying functionality and modifying the RTL design until complying with specifications, the next step is the logic synthesis.

The logic synthesis objective is to translate the RTL design into a particular transistor technology, like 130nm as in this project, employing standard cells. The standard cells are basic digital circuits with fixed height, provided by the design kit of the technology vendor. Logic synthesis takes the design files coded in an HDL, a constraints file and, a Library Exchange Format (LEF) file with the layout cell information in order to optimize the mapping process. To start, the design is translated to logic gates, then it is mapped to the standard cells and finally, optimized according to the constraints. As a result, a gate level netlist with standard cells is generated, however, it should be verified to check if the functionality remains.

Although the gate level netlist generated by the logic synthesis can be simulated and verified, at this point is hard to find an error since the netlist is mapped with a large number of standard cells making it difficult to track any signal. Logic Equivalence Checker (LEC) is a software tool that compares two designs, the original is called golden design and the other design to check for the same logic is the revised design. This comparison reports any mismatch and the key points in common, therefore, the LEC check verifies the functionality.

2.5.2 Back-end design

Back-end design is the physical design flow because the final layout is made and used for fabricating the chip. The gate level netlist from the logic synthesis is the main input for placing and routing all the standard cells in the design. The first step is the floorplan definition where the space for the layout is delimited followed by the power grid placement. When the layout space is reserved and the power lines are positioned in the layout, all standard cells composing the design are placed within the layout but not connected between them.

The next step in the physical flow is to define the clock route for every sequential element, this process is called Clock Tree Synthesis (CTS). The last layer to place in the layout are the standard cell interconnections and the process of linking the logic gates according the gate level netlist is called routing. When the layout is completed, it must be verified to ensure specification compliance. After the pads for inputs and outputs to communicate the chip with the exterior are placed within the layout, the tape-out is used for the manufacturing process.

3 Methodology

In the next sections, every test module component is reviewed and the process that leads to the improvements is developed.

3.1 LFSR

The LFSR is composed of ten D flip-flops and two XOR gates. In Figure 3.1, starting at zero, the ninth and fifth D flip-flops outputs are the input for a XOR whose output, together with the fourth D flip-flop output, are the inputs for a second XOR. This last XOR gate output is connected to the first D flip-flop completing the feedback in order to make an external XOR LFSR. With this structure, the LFSR polynomial is X^10+X^6+X^5+1 and 62 different patterns are generated between the seed and the restart of the sequence. The first value is the seed, the value of the comma 10’b11_1111_1100 or its equivalent in hexadecimal 10’h3FC.

The LFSR output is 10-bit and every value generated is compared against the 10-bit comma. If they are equal, then the first 9-bits are taken for the LFSR top module output, otherwise, only the first 8-bits are taken with the 9-bit set to zero.

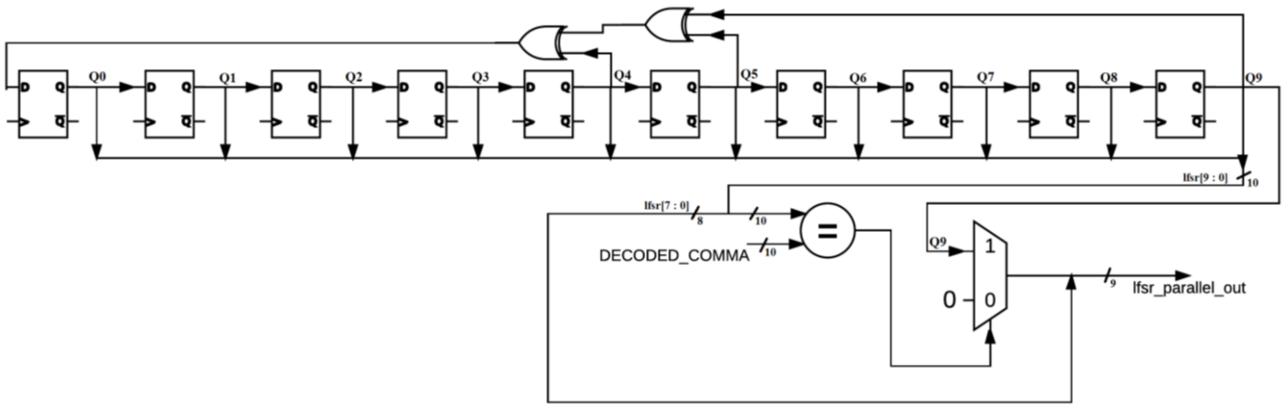


Figure 3.1 LFSR top module block diagram.

3.2 Comparator module

3.2.1 Store data module

The pointer to memory slot where the data being transmitted is saved, is controlled by this module. A finite state machine is implemented in order to track the decoded comma and handle the storing process. Module´s input and output ports are described in the table below.

Table 3.1. Module inputs and outputs signals.

dataA_save module			
Signal	Type	Bits	Description
clk	Input	1	The clock signal.
reset	Input	1	Global asynchronous reset.
lfsr_en	Input	1	Indicates when the digital transmitter starts the serialization.
dataA	Input	9	The digital transmitter input data.
cntA	Output	3	Points the memory slot where the data will be stored.
we	Output	1	Enables the store of data in the memory block.

Figure 3.2 illustrates the finite state machine transitions triggered by the *lfsr_en* signal and the value of the data to transmit. The *lfsr_en* signal indicates when there is data to transmit, therefore, after check if the data is a comma, then it is stored. When the finite state machine is at the SAVE state, the register for *cntA* signal is enabled and its count is incremented one unit for the next clock cycle. At the same time, data memory writing is enabled and the input data signal is recorded.

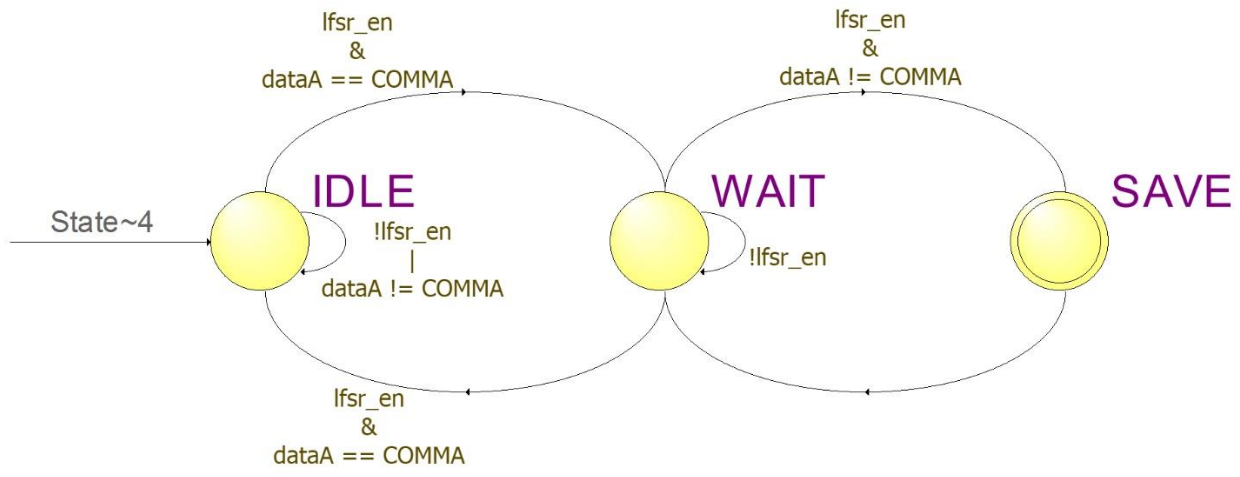


Figure *3.2. Data store finite state machine.*

3.2.2 Comparison data module

This module is responsible for the comparison process employing a counter to read the memory data and compare it with the digital receiver incoming data. When the comparison detects a mismatch, registered errors increase. The operation is controlled by a finite state machine, which also points out the end of the comparison process after it checks the received data and delimits the end when two decoded commas are captured.

This module has three registers whose output are used as the module outputs. Two of them have been described, the counter for signal *cntB* points the memory slot to read and an error counter for signal *num_errors* counts the number of mismatches in the comparison process. The last register is for the signal *bist_end*, which is set high when the comparison is over, i.e. the finite state machine has tracked two commas since the start at the IDLE state. The signal *bist_end* stays high until the process starts again when another comma is captured or when an asynchronous reset is made. All signals for this module are classified and described in Table 3.2.

Table 3.2. Comparison module inputs and outputs.

dataB_compare module			
Signal	Type	Bits	Description
clk	Input	1	The clock signal.
reset	Input	1	Global asynchronous reset.
data_valid_pipe	Input	1	Indicates when the digital receiver has a valid data.
dataB	Input	9	Digital receiver output.
MemData	Input	8	Memory block data to compare.
cntB	Output	3	Points the memory slot to read.
bist_end	Output	1	Indicates the end of the comparison between 2 commas.
num_errors	Output	6	Count the mismatches between the compared data.

The finite state machine whose transitions are depicted in Figure 3.3, controls three signals, which are set high or low depending on the current state. The signal *syncRst_registers_reg* is set high only in the CLEAR state and its purpose is to reset synchronously the *bist_end* register and the error counter register. The next signal is *flag_end_reg*, which is set high only in the COMPARE state to enable the *bist_end* register when the data is a comma, thus indicates the end of the process. The last signal is named *enable_registers_reg* and enables the counter pointing to memory in order to increase the count one unit. The signal *enable_registers_reg* also enables the error counter register when the compared data is not equal. The value for signal *enable_registers_reg* is defined in the COMPARE state, it is set low if the incoming data is a comma and set high if the data is not a comma.

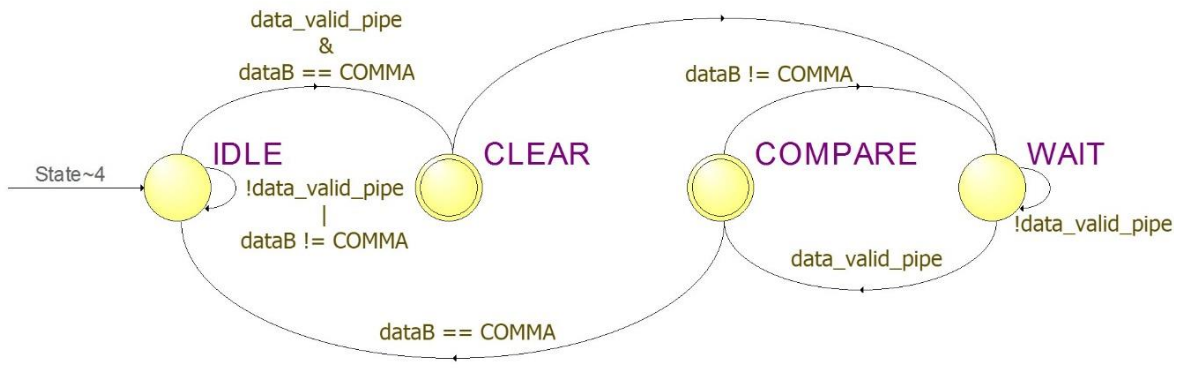


Figure 3.3. Comparison module finite state machine.

3.2.3 Integration of the store data and comparison data module

As seen in Figure 3.4, the comparator top module is composed of three modules, two of them have been described, one controls data store and another module reads data and makes the comparison. The last module that is in the middle is the block memory, which stores the data coming from the digital transmitter input. The memory writing process requires the data to write and two control signals: the write enable and the address to write, both of these signals come from the store module. The memory read function, which is performed in a synchronous way, only requires the address in order to obtain the data.

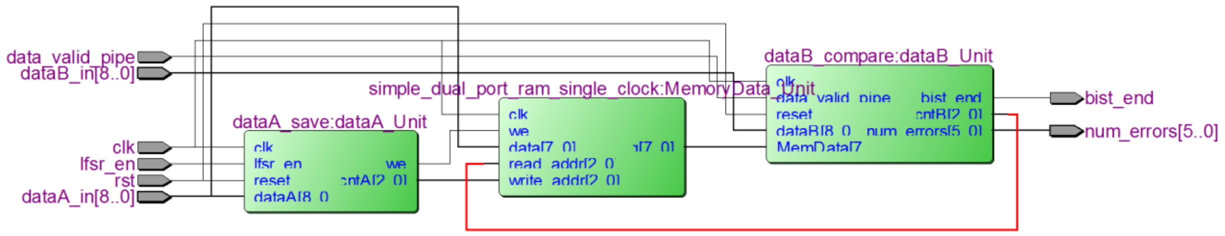


Figure 3.4. Preview of the comparator structural description with the integration of the store and comparison module.

In summary, the comparator is composed of two independent modules and a memory block. The comparison process works assuming that the data to compare comes first from the digital transmitter and later, the same data is captured by the digital receiver and the comparison takes place. This scheme follows the configuration for the BIST operating mode where the pseudo-random data to compare is generated by the LSFR and used as input in the digital transmitter.

3.2.4 Behavioral finite state machine converted to a structural description with combinational logic

Each finite state machine has been described applying a case statement block code for state transition and another case statement block code in order to control the different signal assignments for the current finite state machine state. This section shows how this behavioral description is converted to combinational logic.

First of all, every bit for current state variable is used with every input signal triggering the transitions. These signals declared as the inputs and outputs in the conversion process are defined for every input combination. The outputs are the next state variable and the signals defined with the state transitions. Integrating these data in a table eases the finite state machine analysis. The next step is to make a Karnaugh map for each output with the inputs and obtain a minimized logic equation. As a result, each signal assignment in the finite state machine will have a combinational logic equation comprising the current state variable and the triggering transition signals.

3.2.4.1 Comparison data module

For the comparison module, the input and output signals are listed below followed by Table 3.3, which shows the transitions according the state and triggering signals.

State[0] → S0	State[1] → S1
data_valid_pipe → dvp	data_is_comma_wire → dsc
nxtSte_wire[0] → NS0	nxtSte_wire[1] → NS1
flag_end_reg → fer	syncRst_registers_reg → srr
enable_registers_reg → enr	

Table 3.3 Comparison data module´s finite state machine transitions deployed in a table.

	Inputs				Outputs					
	S1	S0	dvp	dsc	fer	srr	enr	NS1	NS0	
IDLE	0	0	0	0	0	0	0	0	0	→IDLE
	0	0	0	1	0	0	0	0	0	
	0	0	1	0	0	0	0	0	0	
	0	0	1	1	0	0	0	0	1	→CLEAR
CLEAR	0	1	0	0	0	1	0	1	0	→WAIT
	0	1	0	1	0	1	0	1	0	
	0	1	1	0	0	1	0	1	0	
	0	1	1	1	0	1	0	1	0	
WAIT	1	0	0	0	0	0	0	1	0	→WAIT
	1	0	0	1	0	0	0	1	0	
	1	0	1	0	0	0	0	1	1	→COMPARE
	1	0	1	1	0	0	0	1	1	
COMPARE	1	1	0	0	1	0	1	1	0	→WAIT
	1	1	0	1	1	0	1	0	0	→IDLE
	1	1	1	0	1	0	1	1	0	→WAIT
	1	1	1	1	1	0	1	0	0	→IDLE

A minterm or maxterm can be defined from each output column but the objective is minimizing each logic equation, therefore, a Karnaugh map is created. The logic equations are listed below and the corresponding Karnaugh map is shown next to the result.

- $fer = \text{S1} * \text{S0}$ (created from the Karnaugh map in Table 3.4.

Table 3.4 Karnaugh map for flag_end_reg signal.

fer				
dvp dsc→ S1 S0↓	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	0	0	0
1 1	1	1	1	1
1 0	0	0	0	0

- $srr = \overline{S1} * S0$ (created from the Karnaugh map in Table 3.5)

Table 3.5 Karnaugh map for the syncRst_registers_reg signal.

srr				
dvp dsc→ S1 S0↓	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	1	1	1	1
1 1	0	0	0	0
1 0	0	0	0	0

- $enr = \mathrm{S1} * \mathrm{S0} * \overline{\mathrm{dsc}}$ (created from the Karnaugh map in Table 3.6.

Table 3.6 Karnaugh map for the enable_registers_reg signal.

enr				
dvp dsc→ S1 S0↓	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	0	0	0
1 1	1	0	0	1
1 0	0	0	0	0

- $\mathrm{NS1} = \overline{\mathrm{S1}} * S0 + S1 * S0 * \overline{\mathrm{dsc}} + S1 * \overline{\mathrm{S0}}$ (created from Table 3.7)

- $\mathrm{NS0} = \overline{\mathrm{S0}} * dvp * dsc + S1 * \overline{\mathrm{S0}} * dvp$ (created from Table 3.7)

Table 3.7 Karnaugh map for the nxtSte_wire signal.

nxtSte_wire									
bit1					bit0				
dvp dsc→ S1 S0↓	0 0	0 1	1 1	1 0	dvp dsc→ S1 S0↓	0 0	0 1	1 1	1 0
0 0	0	0	0	0	0 0	0	0	1	0
0 1	1	1	1	1	0 1	0	0	0	0
1 1	1	0	0	1	1 1	0	0	0	0
1 0	1	1	1	1	1 0	0	0	1	1

3.2.4.2 Store data module

Represented in Table 3.8, the store data module's finite state machine with three states is shown as well as the two triggering signals and one output signal. For three states, 2-bits are required, consequently, there is a fourth state, which is left and defined as an invalid. The invalid state is the combination 1 1 where each trigger signal cell has a 1/0 bit value indicating that signal can have a logical 1 or a logical 0.

lfsr_en → lfe dataA_is_dcdComma_wire → dac

Table 3.8 Store data module´s finite state machine transitions deployed in a table.

	Inputs				Outputs			
	S1	S0	lfe	dac	enr	NS1	NS0	
IDLE	0	0	0	0	0	0	0	→IDLE
	0	0	0	1	0	0	0	
	0	0	1	0	0	0	0	
	0	0	1	1	0	0	1	→WAIT
WAIT	0	1	0	0	0	0	1	→WAIT
	0	1	0	1	0	0	1	
	0	1	1	0	0	1	0	→SAVE
	0	1	1	1	0	0	0	→IDLE
SAVE	1	0	0	0	1	0	1	→WAIT
	1	0	0	1	1	0	1	
	1	0	1	0	1	0	1	
	1	0	1	1	1	0	1	
?	1	1	1/0	1/0	0	0	0	→IDLE

- $enr = \mathrm{S1} * \overline{\mathrm{S0}}$ (created from the Karnaugh map in Table 3.9)

Table 3.9 Karnaugh map for enable_register_reg signal.

enr				
lfe dac→ S1 S0↓	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	0	0	0
1 1	0	0	0	0
1 0	1	1	1	1

- $\mathrm{NS1} = \overline{\mathrm{S1}} * S0 * lfe * \overline{\mathrm{dac}}$ (created from the Karnaugh map in Table 3.10)
- $\mathrm{NS0} = S1 * \overline{\mathrm{S0}} + \overline{\mathrm{S1}} * S0 * \overline{\mathrm{lfe}} + \overline{\mathrm{S0}} * lfe * dac$ (created from Table 3.10)

Table 3.10 Karnaugh map for nxtSte_wire signal.

nxtSte_wire									
bit1					bit0				
lfe dac→ S1 S0↓	0 0	0 1	1 1	1 0	lfe dac→ S1 S0↓	0 0	0 1	1 1	1 0
0 0	0	0	0	0	0 0	0	0	1	0
0 1	0	0	0	1	0 1	1	1	0	0
1 1	0	0	0	0	1 1	0	0	0	0
1 0	0	0	0	0	1 0	1	1	1	1

3.2.5 Comparison data module´s finite state machine combinational logic delaying.

The last element to be modified is the dataB_compare module, by adding supplementary registers to the structure in order to increase positive slack and work frequency, however, this change introduces latency since changes in signals due to state transitions are delayed. The change integrates two registers stages, each one placed between the finite state machine combinational logic.

Figure 3.5 shows the interaction between the serializer and deserializer with the CompartorP top module, which is integrated with dataA_save, the memory block, and dataB_compare submodules. Although the dataB_compare internal structure is not delimited, it shows the combinational logic delimited by the D flip-flops stages, as each input and output pass through a register. As a result, this arrangement delays dataB_compare outputs two clock cycles.

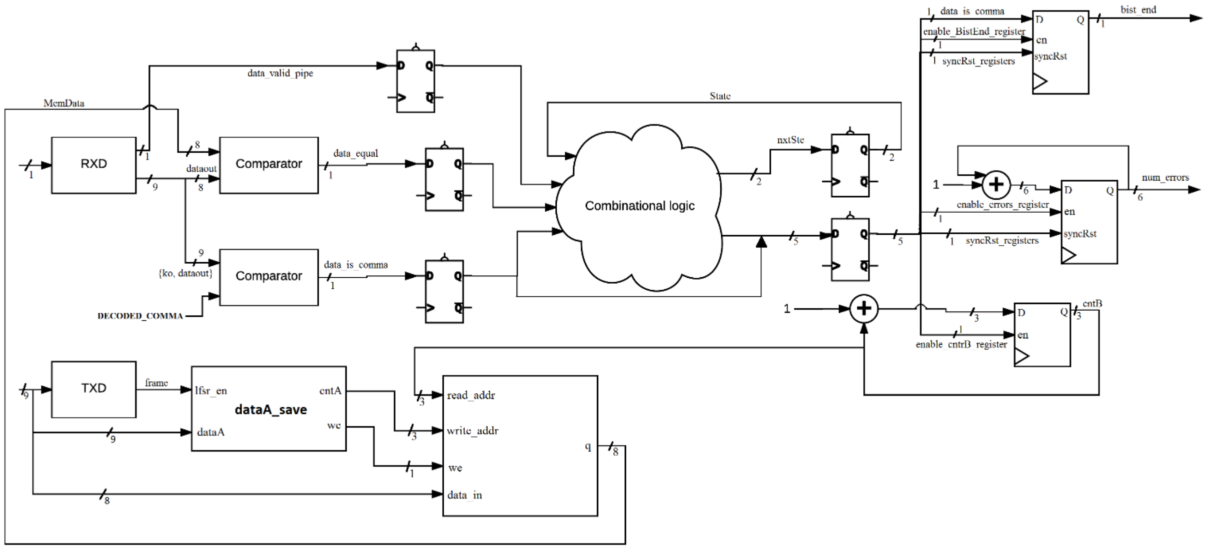


Figure 3.5 CompartorP top module interaction with the serializer and deserializer.

3.2.6 Register input data

Both modules, the data store and comparison data modules, were modified one last time with the addition of D flip-flops in order to register the input data. The *lfsr_en* and *dataA* input signals in the data store module (Table 3.1), and *data_valid_pipe*, *dataB,* and *MemData* input signals in the comparison data module (Table 3.2) pass first into a register before the module data processing.

With the addition of registers for input data in the comparison data module, the stage of D flip-flops placed before the combinational logic was removed. The final structure is displayed in Figure 3.6 where the comparison data module is delimited by a gray box.

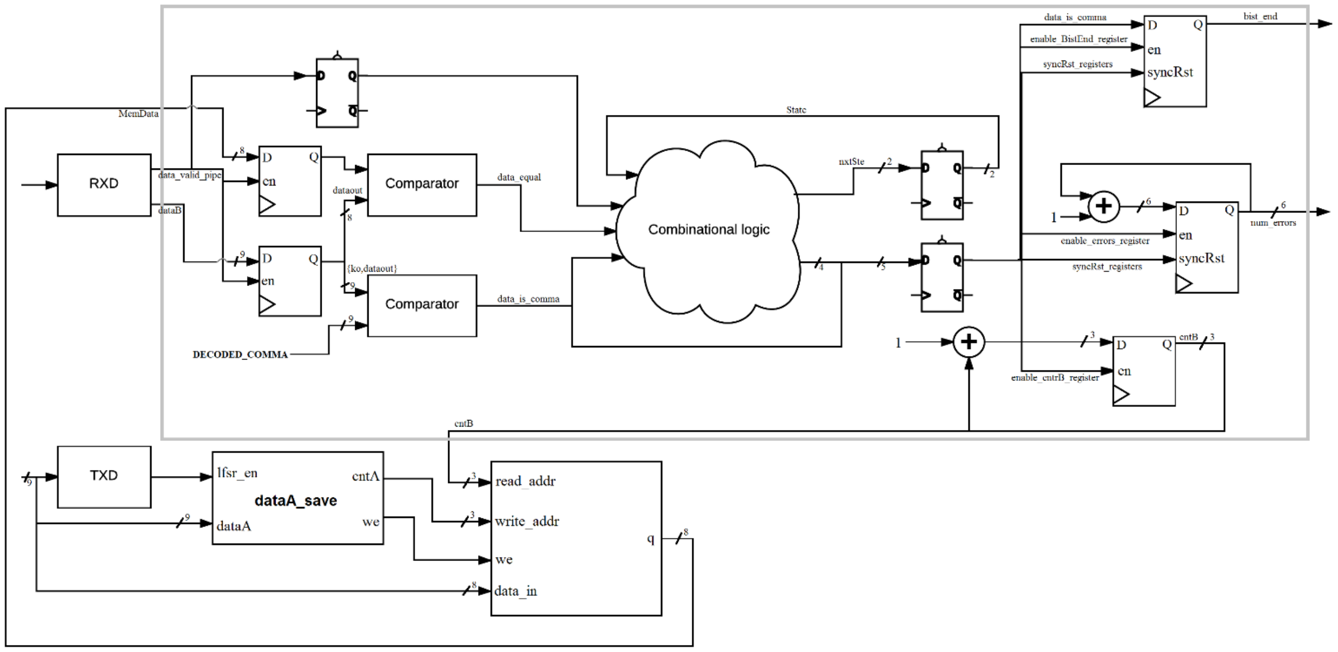


Figure 3.6 Block diagram displaying the connections between CompartorP top module, serializer and deserializer.

3.3 Signal driver implementation

The signal driver structural description was made around the data flowing in every operating mode to control five signals: the digital receiver input and output, the digital transmitter input, the analog transmitter input, and the *test_out* signal. From this, the combinational logic process for design every multiplexor selector has been explained. In each case, the 3-bit mode signal was used to get the Boolean equation.

3.3.1 Digital receiver input multiplexor

The digital receiver can have three different signals as input, these are the analog receiver output, the digital transmitter output, and the signal from the pin test_in. Therefore, a 3 to 1 multiplexor and two signals for the selectors, are necessary. Below, Figure 3.7 shows the connection for the multiplexors and the input signals.

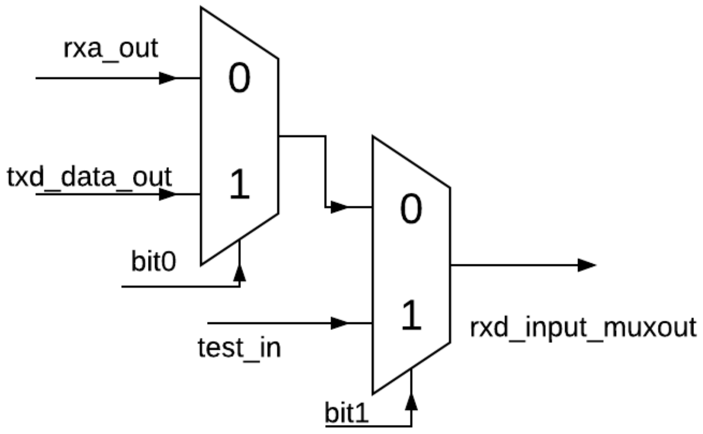


Figure 3.7. Structure of the digital receiver input multiplexor.

This multiplexor arrangement enables selecting the analog receiver output with bit1-bit0 = 00, the digital transmitter output with bit1-bit0 = 01, and test_in with bit1-bit0 = 10. Therefore, according to the 3-bits mode signal, the selectors should be one of the three values listed.

- Analog receiver output (00) → modes: 000, 001, 110, 111.
- Digital transmitter output (01) → modes: 010, 100.
- Signal *test_in* (10) → modes: 011, 101.

With this information, Karnaugh maps can be created in order to obtain the logic equations for each multiplexor selector bit. Table 3.11 displays the maps to help define each selector bit.

Table 3.11. Truth tables for each multiplexor selector for the digital receiver input.

bit0					bit1				
m2 m1→ m0↓	00	01	11	10	m2 m1→ m0↓	00	01	11	10
0	0	1	0	1	0	0	0	0	0
1	0	0	0	0	1	0	1	0	1

With each Karnaugh map, the next Boolean equations are created for each selector bit:

- bit0 = !m0 * (m1 XOR m2).
- bit1 = m0 * (m1 XOR m2)

3.3.2 Digital transmitter input multiplexor

The digital transmitter input multiplexor structure resembles the digital receiver input multiplexor due to the three signals that are multiplexed in each operating mode: the digital transmitter data input, the digital receiver output, and the LFSR output, where each is a 9-bit bus. Figure 3.8 displays the 3 to 1 multiplexor arrangement.

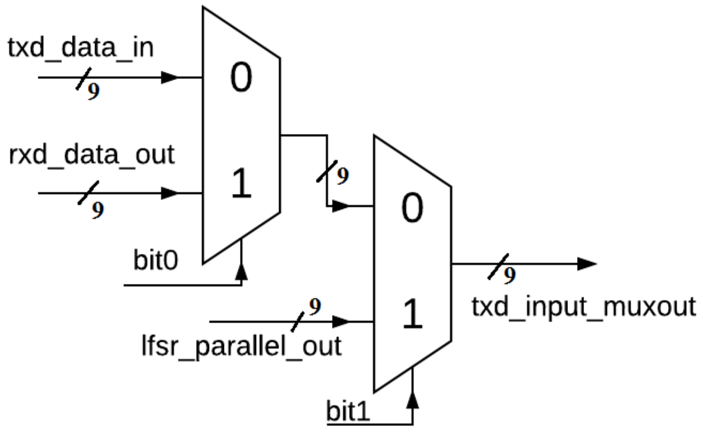


Figure 3.8. Structure of the digital transmitter input multiplexor.

Below the combinations for bit1 and bit0 for all the operating modes are listed and in Table 3.12 where the Karnaugh maps for each bit are shown.

- Digital transmitter data input (00) → modes: 000, 010, 011, 111.
- Digital receiver output (01) → modes: 001, 101.
- LFSR pseudo-random data output (10) → modes: 100, 110.

Table 3.12 Truth tables for the digital transmitter input multiplexor´s selector bits.

bit0					bit1				
m2 m1→ m0↓	00	01	11	10	m2 m1→ m0↓	00	01	11	10
0	0	0	0	0	0	0	0	1	1
1	1	0	0	1	1	0	0	0	0

- bit0 = m0 * !m1.
- bit1 = !m0 * m2.

3.3.3 Analog transmitter input multiplexor

Analog transmitter input multiplexor design is easier than any multiplexor because only two signals are multiplexed, the digital transmitter output and the analog receiver output, and the analog receiver output is selected in only one operating mode; hence, the Boolean equation comes from inspecting the bits for this mode. The 2 to 1 multiplexor arrangement is shown in Figure 3.9.

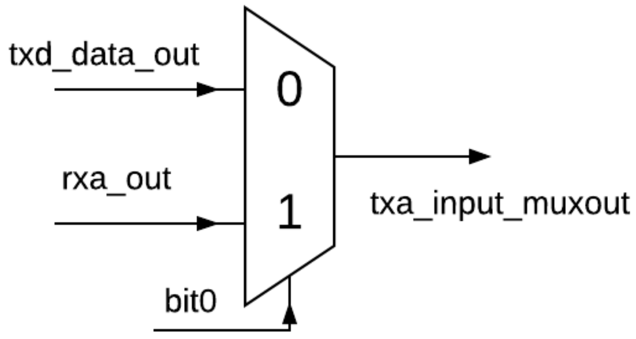


Figure 3.9. Structure of the analog transmitter input multiplexor.

The analog receiver output is selected when the operating mode is 3’b111 and consequently, the Boolean equation is:

- bit0 = m2 * m1 * m0.

3.3.4 Test_out multiplexor

The *test_out* output signal requires a 2 to 1 multiplexor to select between the analog receiver output and the *bist_end* signal. The multiplexor structure is shown in Figure 3.10.

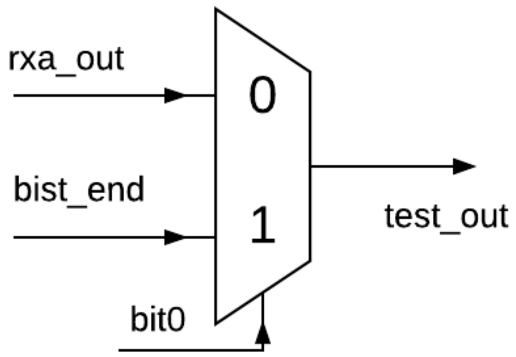


Figure 3.10. Test_out multiplexor structure.

The operating modes and the signals selected are listed next and spread out in Table 3.13.

- Analog receiver output (0) → modes: 000, 001, 010, 011, 101, 111.
- Signal *bist_end* (1) → modes: 100, 110.

Table 3.13.Truth table for the test_out multiplexor selector.

bit0				
m2 m1→ / m0↓	00	01	11	10
0	0	0	1	1
1	0	0	0	0

- bit0 = !m0 * m2.

3.3.5 Digital receiver output multiplexor

This multiplexor is responsible for switching the expected digital receiver output between the 9-bit deserializer output and a 9-bit bus compressing the signals *disp_err*, *code_err*, *dispout* and the 6-bit *num_errors* signal, which is seen in Figure 3.11.

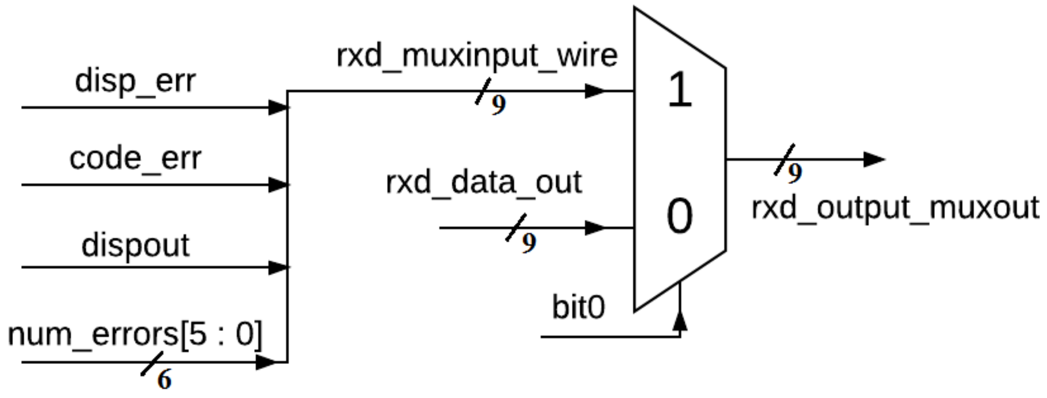


Figure 3.11. Structure of the digital receiver output multiplexor.

Next, the selector value in each operating mode is listed below:

- Digital receiver output (0) → modes: 000, 001, 010, 011, 101, 111.
- rxd_muxinput_wire (1) → modes: 100, 110.

The signal selection for each operating mode is equal to the test_out multiplexor selector, hence, the selector Boolean equation is the same. Nevertheless, this switch is enabled by *errors_en* (*config_in[4]*) signal, therefore, the selector needs to include the signal *errors_en* in order to show the number of errors.

- bit0 = errors_en * !m0 * m2.

3.4 Top module integration

The top module is composed of the LFSR, the comparator, and the signal driver modules. In order to turn on the LFSR and the comparator when the test operative mode includes all the BIST modules, a signal was implemented employing combinational logic. This situation is equal to the multiplexor selector design for the digital receiver output, but instead of the *errors_en* signal, the input pin test_en is applied. This new signal is called bist_on_wire and is used to mask the lfsr_en and c_data_valid inputs which, trigger the transitions in the store data and comparison data modules.

- bist_on_wire = test_en * m2 * !m0.
- txd_frame_start_wire = lfsr_en * bist_on_wire.
- rxd_data_ready_wire = c_data_valid * bist_on_wire.

Another addition to the top module is the two registers to delay the *lsfr_en* input signal going into the LFSR module which, generates a new pseudo-random test pattern. This delay gives three clock cycles to the memory block to store the test pattern since the store data module starts the storing process when the lfsr_en signal is set high. Aside from these delay registers the top module has D flip-flops for each analog signal.

The former digital receiver module implemented two registers for the analog receiver output in order to accomplish the load balancing requirement for the digitalized input signal arriving from the analog differential amplifier and equalizer [23]. The test module implements each register since the top module gathers all SerDes signals and handles them employing the signal driver module.

3.5 Logic synthesis

The logic synthesis employs the RTL Compiler (RC) tool, a TCL script to run the RC process, a constraints file, and takes as input all the RTL code containing the HDL hierarchy for the top module design. As a result, a file is generated with all the logic gates mapped with standard cells from the 130nm IBM´s cmrf8sf Design Kit library. Between the reports generated, there are the timing and power reports, which are important to ensure timing compliance and power consumption for the layout core.

3.6 Test modules physical design

3.6.1 Power grid

The power grid comprises the power ring, the horizontal stripes and the vertical stripes in the core. The power ring is composed by the VDD and VSS stripes around the core and provides the power source for the circuit. The horizontal stripes in the grid, inside the core, are stripes for VDD and VSS, the rows between them are where the standard cells are placed. The farthest the standard cell is from the power ring, the voltage drop is greater, therefore, the vertical stripes comprised each one of VDD and VSS are placed in order to provide a constant source for all the standard cells. Figure 3.12 shows the power grid design without pads for the test modules.

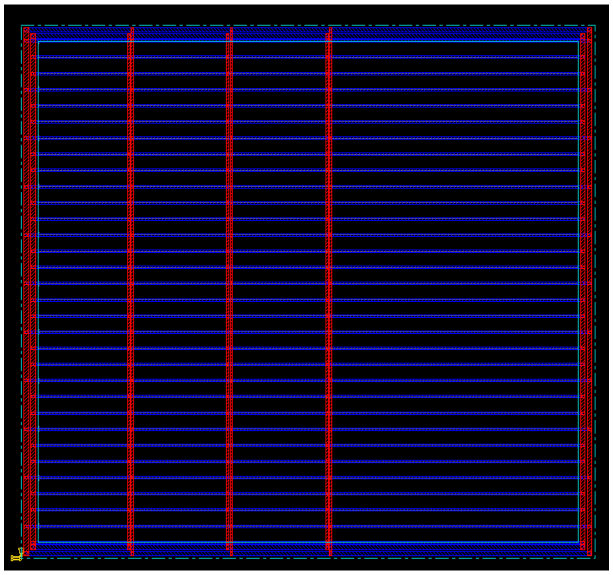


Figure 3.12.Test modules power grid.

3.6.2 Placement

Figure 3.13 illustrates how the standard cells are placed between the horizontal stripes in the core every two rows. This arrangement is possible due to the horizontal stripes comprised of two of VDD after one VSS. It is important to mention that this placement is temporal because the optimization step can reorder the standard cells in order to accomplish timing compliance.

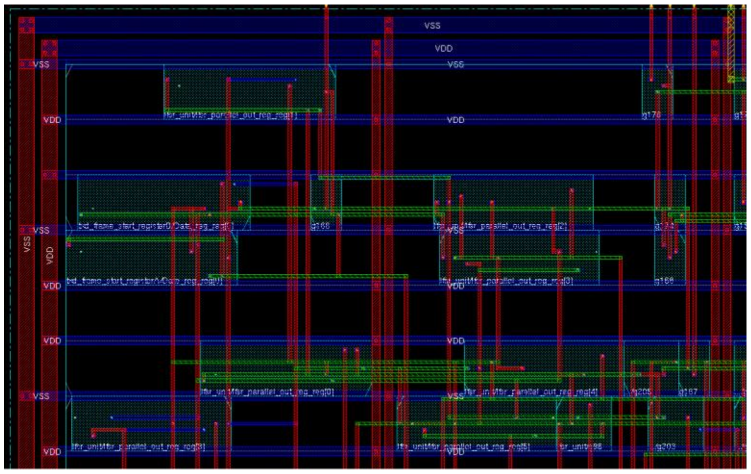


Figure 3.13. Standard cells placed between the horizontal power stripes.

3.6.3 Final layout

Figure 3.14 represents the test modules final layout after the clock three synthesis takes place and a series of optimization are implemented before and after routing the standard cells.

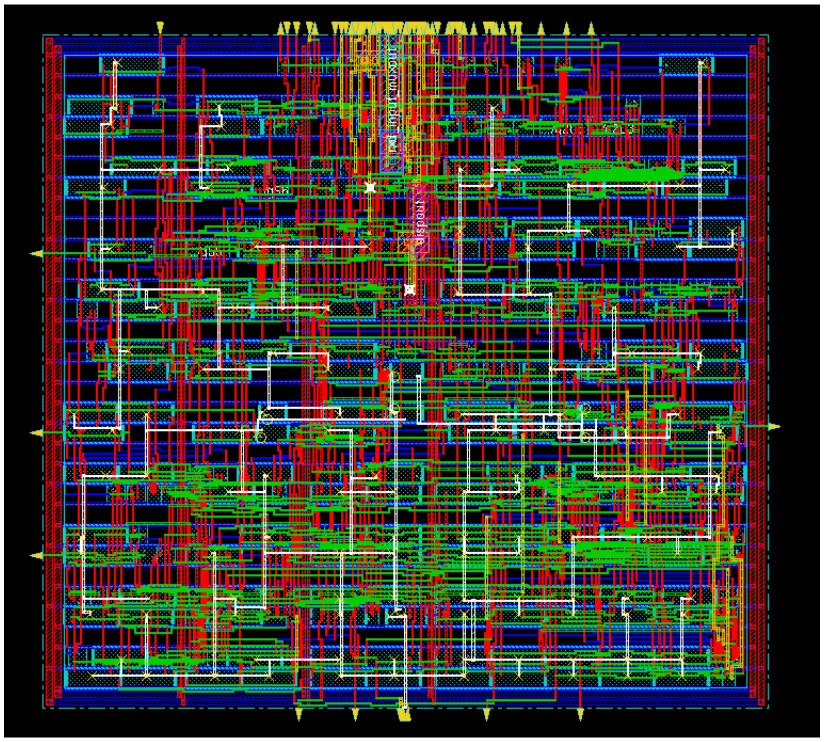


Figure 3.14 Test modules final layout.

3.7 SerDes integration

The SerDes comprises digital modules and analog modules, each type is designed differently, hence, the overall design flow is different. The digital modules are created employing an HDL code which, is translated to standard cells. The analog modules are designed through schematics employing analog components. Later a layout is created taking the schematic as a start point. Therefore, the SerDes design requires integrating these two different types of modules into one distinct flow design.

3.7.1 Chip input and output pins

The SerDes chip has 40 pins, 4 of them are for power and 36 are divided for input and output signals. Before the integration 37 pins had been considered, consequently, one pin was removed. It was decided that the *test_out* signal can be removed and multiplex it within the signal driver module. Table 3.14 shows the final SerDes pins without *test_out*.

Table 3.14. SerDes pins.

Signal name	Pins	Type
rst	1	input
clk	1	input
rxa_in_p	1	input
rxa_in_n	1	input
config_in	8	input
txd_data_in	9	input
test_en	1	input
digital_out	9	output
txa_data_out_p	1	output
txa_data_out_n	1	output
txd_data_out	1	output
tx_frame_start	1	output
c_data_valid	1	output
VDD	1	power
VSS	1	power
DVDD	1	power
DVSS	1	power

3.7.2 Signal driver modification

Without the *test_out* output pin, the multiplexor in Figure 3.10, is useless. The solution was to modify the digital receiver output multiplexor in order to output the signals *rxa_out* and *bist_end*. The final multiplexor structure is illustrated in Figure 3.15 where *num_errors[5:4]* is multiplexed with *{rxa_out, bist_end}* through the selector *config_in[4]*, which continues to show all the bits of the *num_erros* signal. The main multiplexor is controlled with *config_in[5]* and it can choose between the digital receiver output and the *rxd_muxinput_wire* signal. This design enables the data switching independently of the test operative mode but first *test_en* has to be set high.

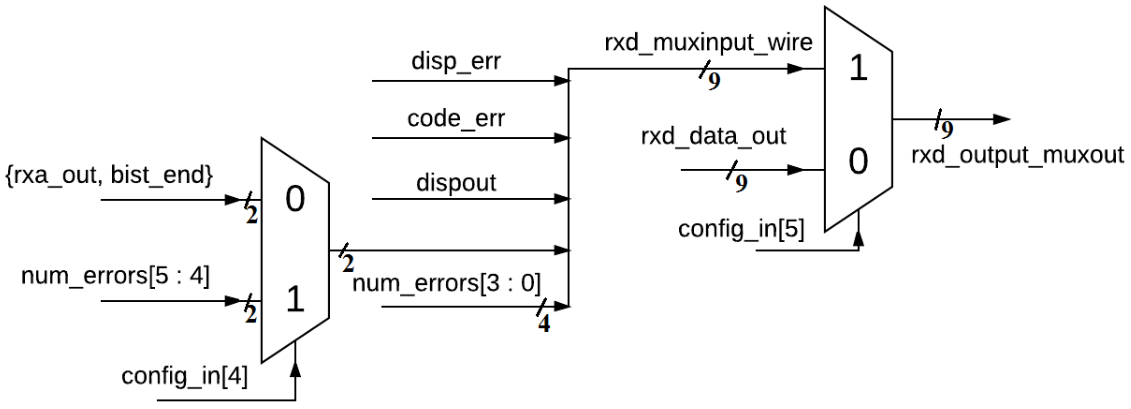
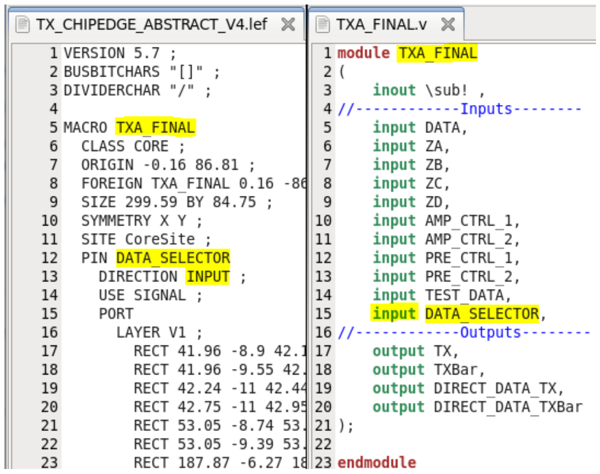


Figure 3.15. New digital receiver output multiplexor.

3.7.3 Logic synthesis

In the usual logic synthesis, a gate level netlist employing standard cells is mapped after the design's RTL coded in some HDL. However, for the SerDes analog modules, LEF files for each one are necessary in order to integrate them with the digital modules in the logic synthesis. Each analog designer has to create the LEF file based on an abstract view and add the file at the end of the list of LEF libraries used in the logic synthesis.

The logic synthesis (appendix B) takes as input an HDL file list, thus implies create a file for each analog module indicating only the input and output ports, that is defining it as a black box module. The black box module has to be named equal to the macro in the analog module LEF file in order to relate them in the logic synthesis, appendices A1 and A2 show the black box modules definitions. Figure 3.16 exemplifies the association between the analog transmitter LEF file and its HDL Verilog code as well as how the ports are declared. After the logic synthesis ends, a way to ensure that the Verilog file was linked to the LEF file is checking the log file and finding a snippet like the one in Figure 3.17. The other method is viewing the analog module in the schematic from the logic synthesis graphical interface. Placing the mouse over the module will indicate the libcell name equals the macro name in the LEF file.



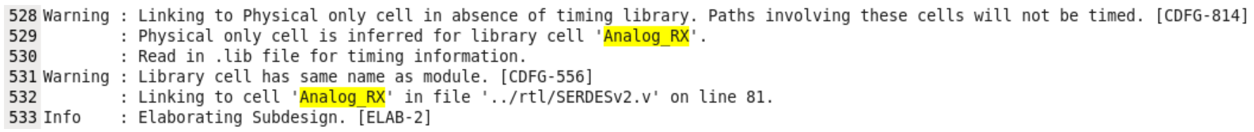
```
TX_CHIPEDGE_ABSTRACT_V4.lef
VERSION 5.7 ;
BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;

MACRO TXA_FINAL
  CLASS CORE ;
  ORIGIN -0.16 86.81 ;
  FOREIGN TXA_FINAL 0.16 -86
  SIZE 299.59 BY 84.75 ;
  SYMMETRY X Y ;
  SITE CoreSite ;
  PIN DATA_SELECTOR
    DIRECTION INPUT ;
    USE SIGNAL ;
    PORT
      LAYER V1 ;
        RECT 41.96 -8.9 42.1
        RECT 41.96 -9.55 42.
        RECT 42.24 -11 42.44
        RECT 42.75 -11 42.95
        RECT 53.05 -8.74 53.
        RECT 53.05 -9.39 53.
        RECT 187.87 -6.27 18
```

```
TXA_FINAL.v
module TXA_FINAL
(
    inout \sub! ,
//------------Inputs--------
    input DATA,
    input ZA,
    input ZB,
    input ZC,
    input ZD,
    input AMP_CTRL_1,
    input AMP_CTRL_2,
    input PRE_CTRL_1,
    input PRE_CTRL_2,
    input TEST_DATA,
    input DATA_SELECTOR,
//-----------Outputs--------
    output TX,
    output TXBar,
    output DIRECT_DATA_TX,
    output DIRECT_DATA_TXBar
);

endmodule
```

Figure 3.16. Analog transmitter LEF file and Verilog file linking.



```
Warning : Linking to Physical only cell in absence of timing library. Paths involving these cells will not be timed. [CDFG-814]
        : Physical only cell is inferred for library cell 'Analog_RX'.
        : Read in .lib file for timing information.
Warning : Library cell has same name as module. [CDFG-556]
        : Linking to cell 'Analog_RX' in file '../rtl/SERDESv2.v' on line 81.
Info    : Elaborating Subdesign. [ELAB-2]
```

Figure 3.17. Logic synthesis log snippet.

In order to continue with the physical design, an HDL file is created indicating the input/output pads. This file replaces the top module and it instantiates the former top module. It is important to notice that the design requires four extra pins, two for the core power and ground, and, two for the pads power and ground. With the HDL file for pads in the logic synthesis, the gate level netlist also includes the pads instances in order to consider them in the physical design.

3.7.4 Physical design

The physical design is made in the Encounter Digital Implementation (EDI) tool, it has a Graphical User Interface (GUI) that eases the layout design and at the same time can run commands through the OS shell. The EDI tool logs all commands via shell and most of the editions made within the GUI. Such log file can be reviewed in order to generate a script and automate the flow. Since the design flow is an iterative process, a script for each physical design stage was created, therefore, each step can be modified by editing the script. Appendices from D to F4 are used in the EDI tool, they can be reviewed to follow the design flow.

3.7.4.1 File input

To start the physical design, certain files are required to configure the tool:

- A gate level netlist with the design mapped with standard cells in a particular transistor technology.
- A file defining the input/output pads positions around the core (ioc extension).
- An analysis view file in order to run tests into the layout design.
- Constraints files generated by the logic synthesis tool after applying different timing libraries in order to define the corners used in the view file.
- Technology LEF libraries for the standard cells and pads.
- A file to import the design comprising all the previously mentioned files.

After loading the design, the layout has the die area with the pads arranged as the file with ioc extension defined. Figure 3.18 shows the SerDes pads and the core after importing the design, it can be compared to the appendix D to appreciate the pads organization.

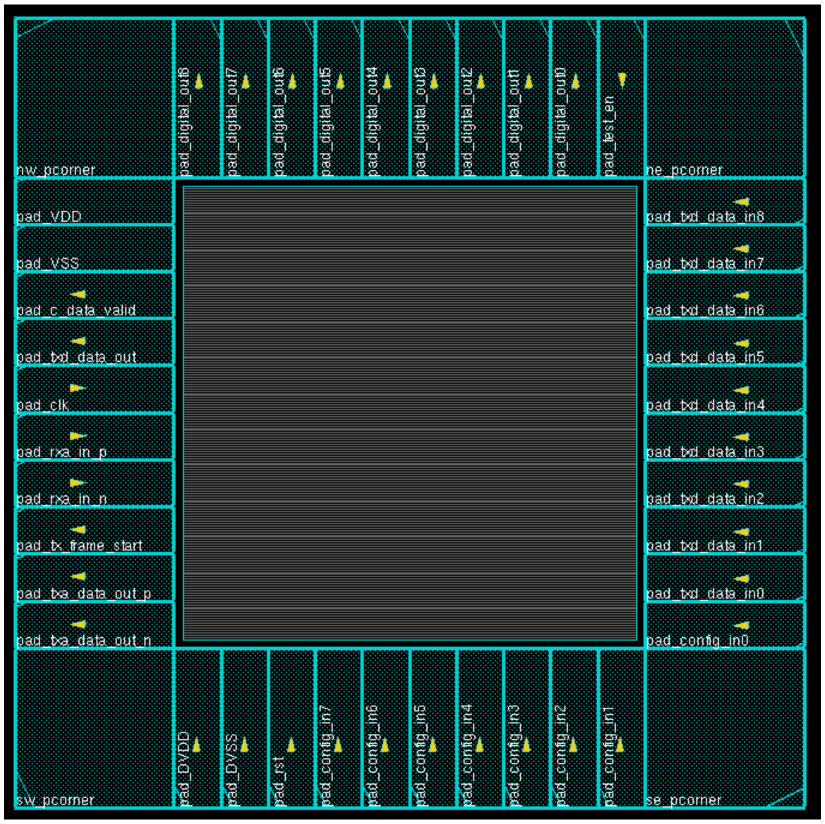


Figure 3.18. SerDes pads and core after importing the design.

3.7.4.2 Power grid

Before defining the power grid, the core size is specified to leave the space to place the power ring. In Figure 3.19, the rules indicate the space delimited by the pads as well as the core width and height.

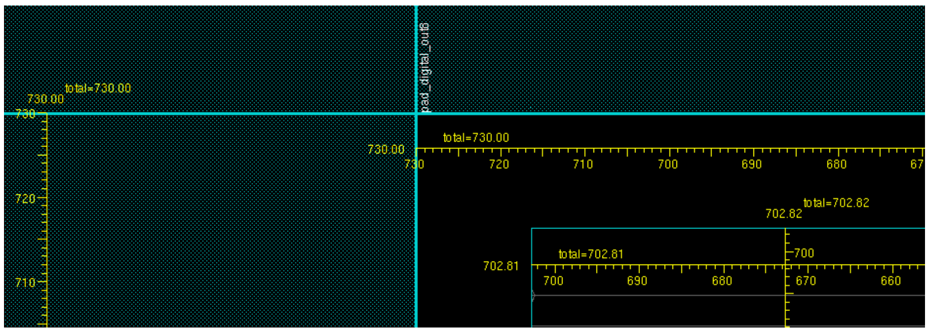


Figure 3.19. SerDes core size.

Each LEF file specifies the metal layers employed in the analog module layout, however, in order to prevent the overlap with the power grid metals, an area is created to place the analog modules. Figure 3.20 illustrates four zones where the horizontal stripes were allocated in two different ways, for the A zone both horizontal stripes were placed. For B zones, VDD and VSS were placed separately since the area with the horizontal stripes covers from the core border to the last vertical stripe of the same type of stripes being created. Appendix F1 covers all the commands applied for making the power grid.

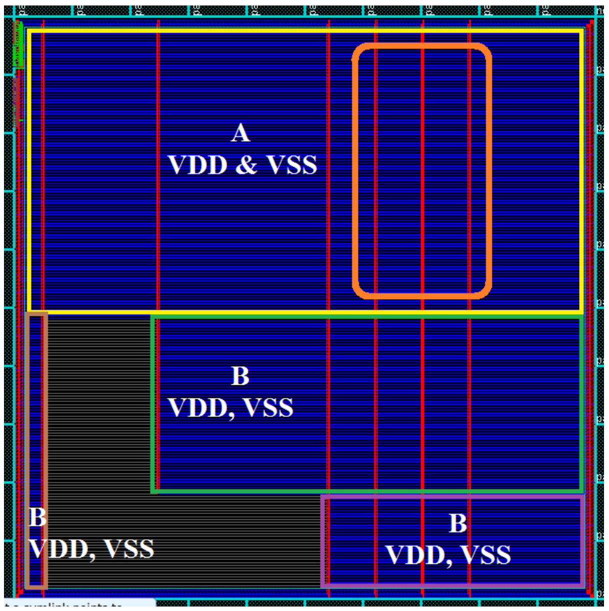
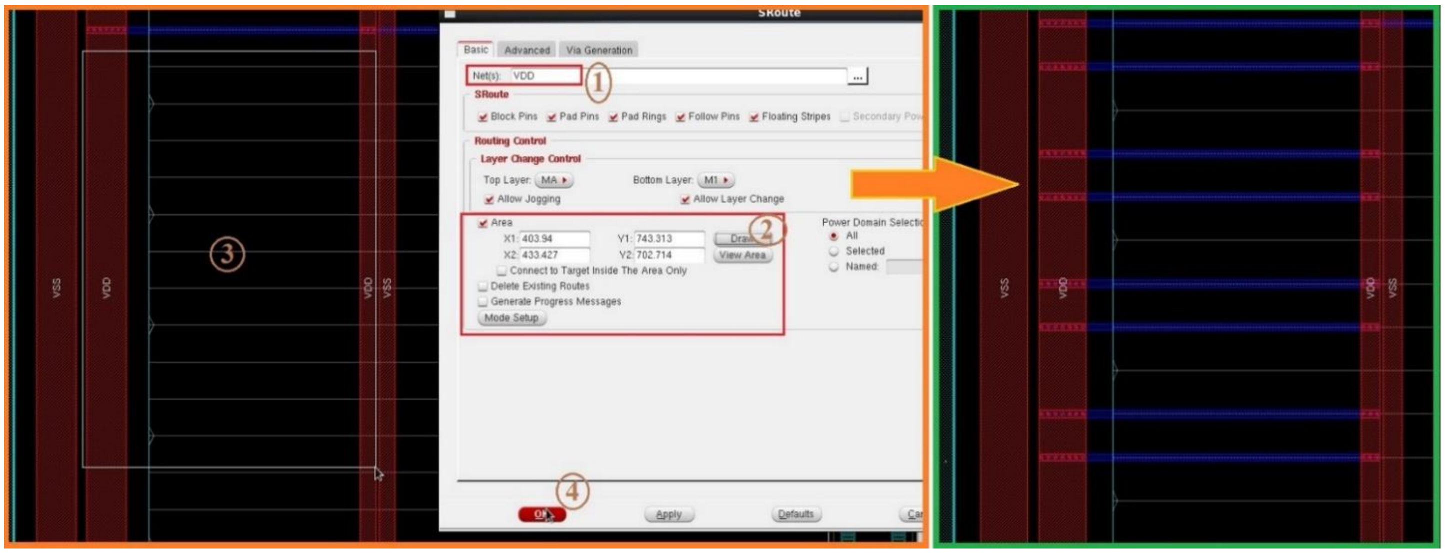


Figure 3.20. Horizontal stripes creation by zones.

Figure 3.21 depicts the process to create the VDD horizontal stripes in B zones. Placing VSS horizontal stripes follows the same method but with VSS instead of VDD in the Net(s) textbox, indicated as 1 in the figure.



Net(s): VDD

SRoute

[✓ Block Pins] [✓ Pad Pins] [✓ Pad Rings] [✓ Follow Pins] [✓ Floating Stripes] Secondary Pow

Routing Control

Layer Change Control

Top Layer: MA Bottom Layer: M1

[✓ Allow Jogging] [✓ Allow Layer Change]

[✓ Area]

X1: 403.94 Y1: 743.313

X2: 433.427 Y2: 702.714

Connect to Target Inside The Area Only

Delete Existing Routes

Generate Progress Messages

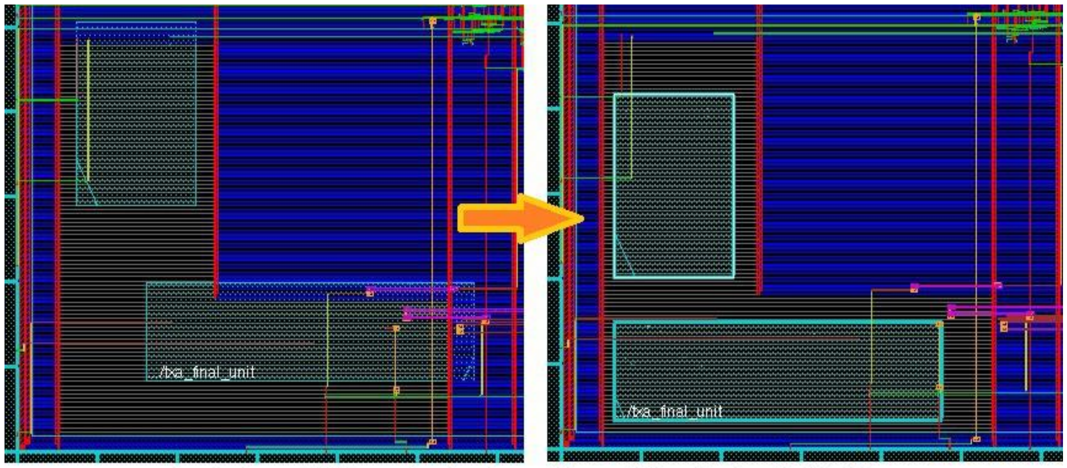
Power Domain Selectio

[✓ All] Selected Named:

Figure 3.21. Horizontal stripes placing process.

3.7.4.3 Placement

After executing the commands to place the standard cells and the macros from the analog modules, the macros are placed out of the area designed for them as Figure 3.22 shows in the left image. To move them, the command *placeInstance* had to be executed, this command requires indicating the module’s instance name and the x and y coordinates. Figure 3.22 shows the commands introduced and the image in the right depicts the results.



```
placeInstance SERDES_core/analog_transmitter_unit/txa_final_unit 445 423
placeInstance SERDES_core/analog_receiver_unit 445 545
```

Figure 3.22. Analog modules before and after running the commands to move them.

3.7.4.4 CTS and routing

The last steps in the physical design only require running the script for the CTS, optimization, and routing in order to create the SerDes final layout in the EDI tool. After these steps, verifying the layout and check timing compliance will indicate if some step requires being changed and iterated again in order to have a free error layout and constraints compliance.

4 Results

4.1 LFSR

Figure 4.1 shows a segment of the LFSR simulation where the count of the different patterns generated is indicated by the *counter* signal. This part of the simulation demonstrates when the pattern generation is restarted since *counter* goes until 62 and then to zero. The waveforms in the simulation are divided into inputs, outputs, and internal. The internal divider has the 10-bit LFSR output and the seed value to compare. As explained before, if the 10-bits output is equal to the seed, then the top module output has the ninth value set to one. This outcome is represented by the signal *LFSR_Kbit,* which is the ninth bit, and signal *LFSR_8bits* is the transmitted data, both in conjunction are the *LFSR_OUT(9bits)* signal.

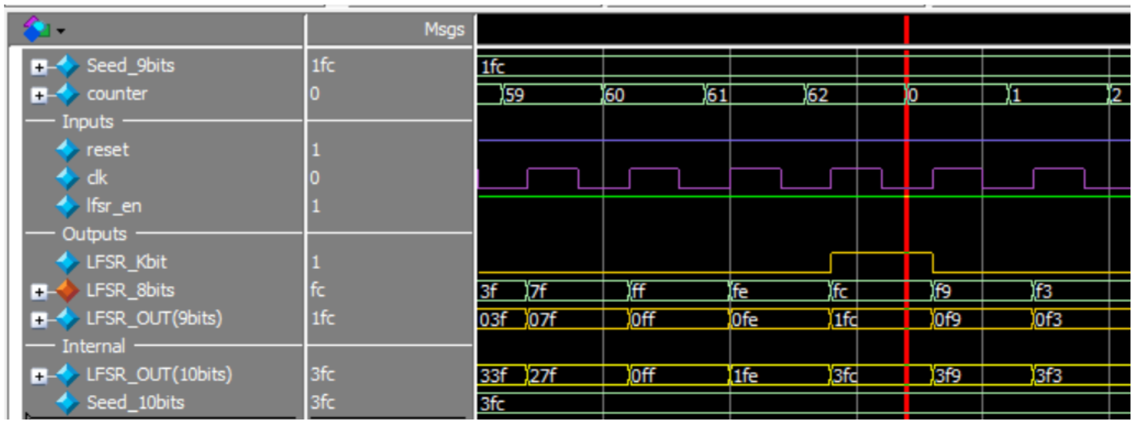


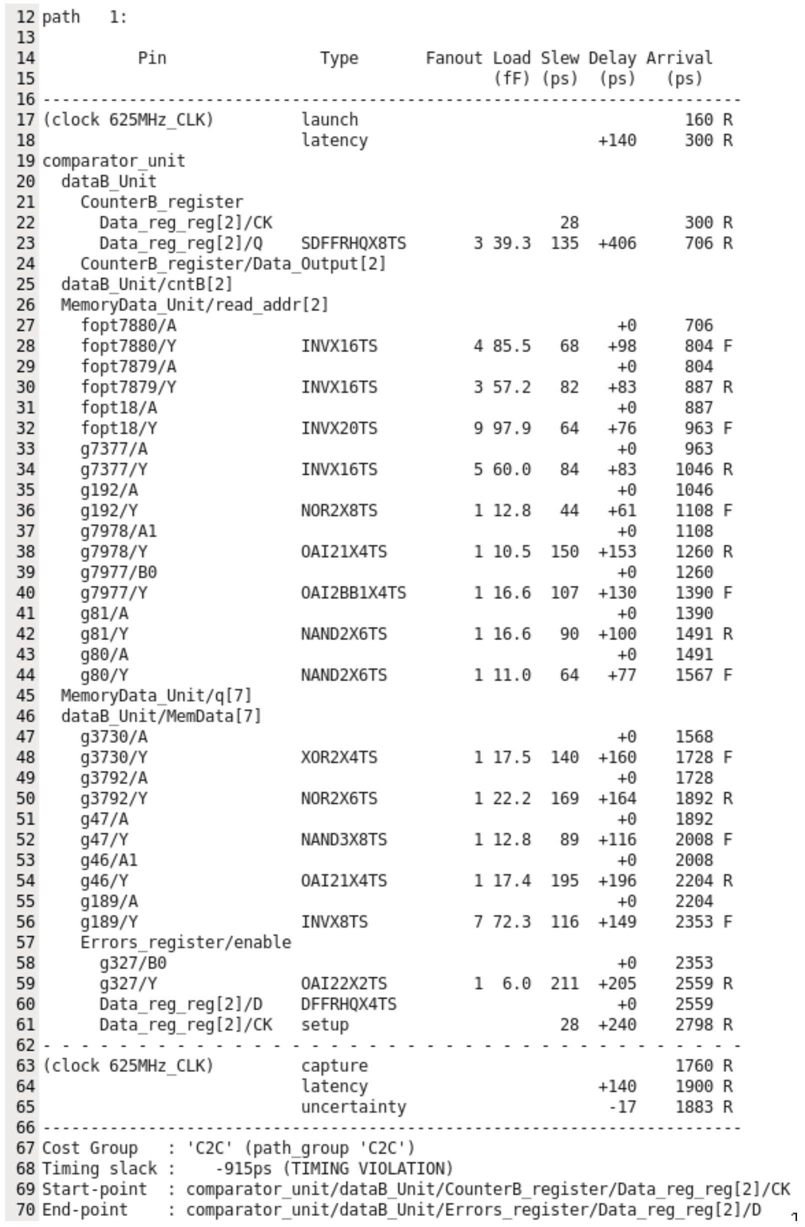
Figure 4.1. LFSR Simulation

4.2 Comparator module

Now, the timing reports from the logic synthesis are presented for each modification made to both modules, dataA_save and dataB_compare. The results begin with both modules designs with a finite state machine, the next modification with combinational logic and finally, only the dataB_compare module with the D flip-flops stages. For each timing report, only the first path is displayed since it has the worst slack.

4.2.1 Design with behavioral finite state machines

With a behavioral finite state machine, the worst slack is -915ps. In Figure 4.2, the route starts in the Most Significant Bit (MSB) from the memory read pointer in dataB_compare module, and continues to the read_addr MSB a memory input port. From here the path continues from the memory output MSB and returns to the dataB_compare module through the MemData MSB input and ends in the enable signal from the Errors_register module.



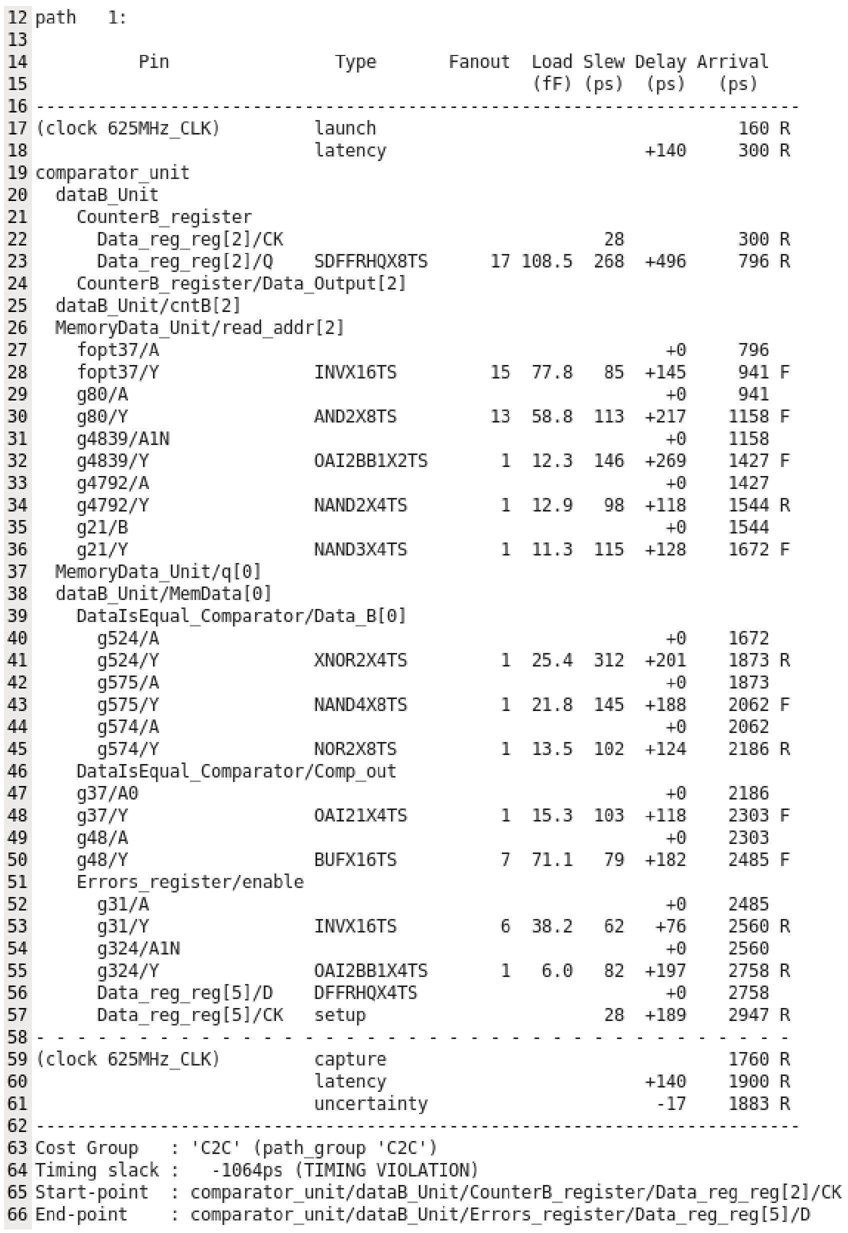
```
path   1:

          Pin                Type       Fanout Load Slew Delay Arrival
                                               (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock 625MHz_CLK)         launch                                  160 R
                           latency                        +140     300 R
comparator_unit
  dataB_Unit
    CounterB_register
      Data_reg_reg[2]/CK                                28            300 R
      Data_reg_reg[2]/Q    SDFFRHQX8TS       3 39.3  135  +406     706 R
    CounterB_register/Data_Output[2]
  dataB_Unit/cntB[2]
  MemoryData_Unit/read_addr[2]
    fopt7880/A                                              +0     706
    fopt7880/Y             INVX16TS          4 85.5   68   +98     804 F
    fopt7879/A                                              +0     804
    fopt7879/Y             INVX16TS          3 57.2   82   +83     887 R
    fopt18/A                                                +0     887
    fopt18/Y               INVX20TS          9 97.9   64   +76     963 F
    g7377/A                                                 +0     963
    g7377/Y                INVX16TS          5 60.0   84   +83    1046 R
    g192/A                                                  +0    1046
    g192/Y                 NOR2X8TS          1 12.8   44   +61    1108 F
    g7978/A1                                                +0    1108
    g7978/Y                OAI21X4TS         1 10.5  150  +153    1260 R
    g7977/B0                                                +0    1260
    g7977/Y                OAI2BB1X4TS       1 16.6  107  +130    1390 F
    g81/A                                                   +0    1390
    g81/Y                  NAND2X6TS         1 16.6   90  +100    1491 R
    g80/A                                                   +0    1491
    g80/Y                  NAND2X6TS         1 11.0   64   +77    1567 F
  MemoryData_Unit/q[7]
  dataB_Unit/MemData[7]
    g3730/A                                                 +0    1568
    g3730/Y                XOR2X4TS          1 17.5  140  +160    1728 F
    g3792/A                                                 +0    1728
    g3792/Y                NOR2X6TS          1 22.2  169  +164    1892 R
    g47/A                                                   +0    1892
    g47/Y                  NAND3X8TS         1 12.8   89  +116    2008 F
    g46/A1                                                  +0    2008
    g46/Y                  OAI21X4TS         1 17.4  195  +196    2204 R
    g189/A                                                  +0    2204
    g189/Y                 INVX8TS           7 72.3  116  +149    2353 F
    Errors_register/enable
      g327/B0                                               +0    2353
      g327/Y               OAI22X2TS         1  6.0  211  +205    2559 R
      Data_reg_reg[2]/D    DFFRHQX4TS                       +0    2559
      Data_reg_reg[2]/CK   setup                        28  +240    2798 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 625MHz_CLK)         capture                                1760 R
                           latency                        +140    1900 R
                           uncertainty                     -17    1883 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :    -915ps (TIMING VIOLATION)
Start-point  : comparator_unit/dataB_Unit/CounterB_register/Data_reg_reg[2]/CK
End-point    : comparator_unit/dataB_Unit/Errors_register/Data_reg_reg[2]/D
```

Figure 4.2 Test module worst slack path from the timing report.

4.2.2 Finite state machine described with combinational logic

Figure 4.3 displays the worst slack path for the test module after modifying the finite state machines with combinational logic and implementing an equality comparator. The path is equal as the one shown in Figure 4.2, except the paths from the DataIsEqual_Comparator instance. As a result, the slack is negative, -1064ps, 140ps or 16% picoseconds more from the last result.



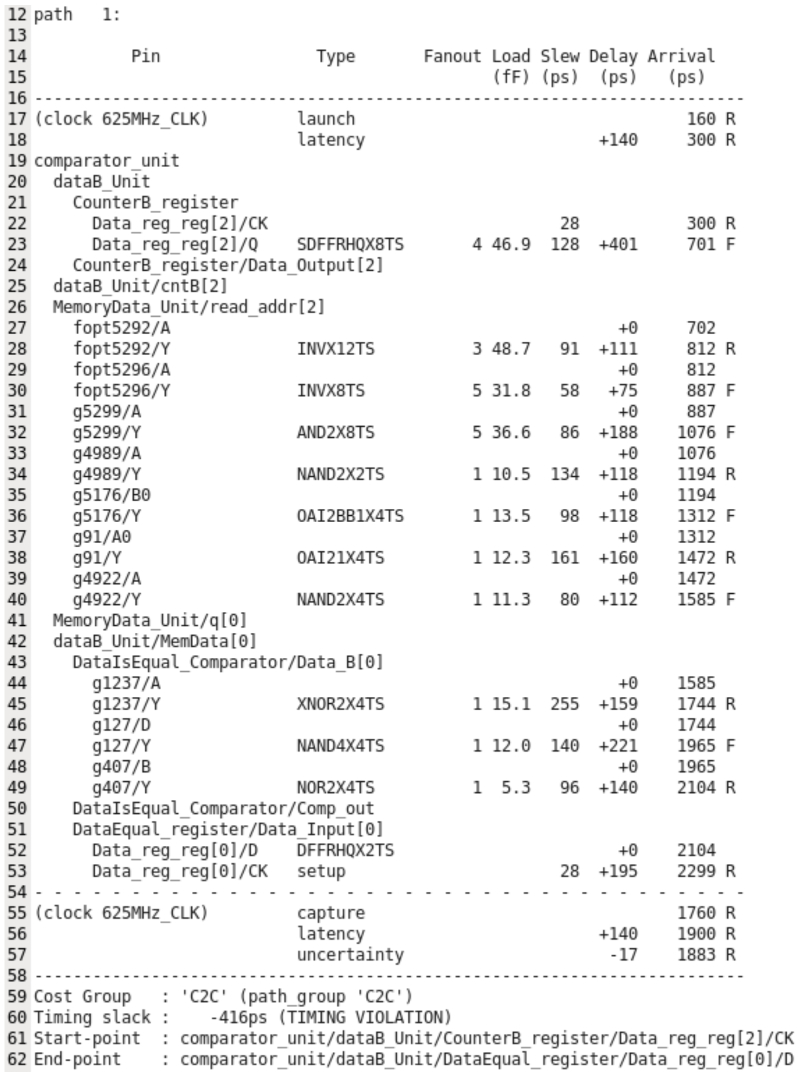
```
path   1:

          Pin                Type       Fanout  Load Slew Delay Arrival
                                                (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock 625MHz_CLK)          launch                                 160 R
                            latency                         +140   300 R
comparator_unit
  dataB_Unit
    CounterB_register
      Data_reg_reg[2]/CK                                28         300 R
      Data_reg_reg[2]/Q    SDFFRHQX8TS      17 108.5  268  +496    796 R
    CounterB_register/Data_Output[2]
  dataB_Unit/cntB[2]
  MemoryData_Unit/read_addr[2]
    fopt37/A                                                +0     796
    fopt37/Y               INVX16TS         15  77.8   85  +145    941 F
    g80/A                                                   +0     941
    g80/Y                  AND2X8TS         13  58.8  113  +217   1158 F
    g4839/A1N                                               +0    1158
    g4839/Y                OAI2BB1X2TS       1  12.3  146  +269   1427 F
    g4792/A                                                 +0    1427
    g4792/Y                NAND2X4TS         1  12.9   98  +118   1544 R
    g21/B                                                   +0    1544
    g21/Y                  NAND3X4TS         1  11.3  115  +128   1672 F
  MemoryData_Unit/q[0]
  dataB_Unit/MemData[0]
    DataIsEqual_Comparator/Data_B[0]
      g524/A                                                +0    1672
      g524/Y               XNOR2X4TS         1  25.4  312  +201   1873 R
      g575/A                                                +0    1873
      g575/Y               NAND4X8TS         1  21.8  145  +188   2062 F
      g574/A                                                +0    2062
      g574/Y               NOR2X8TS          1  13.5  102  +124   2186 R
    DataIsEqual_Comparator/Comp_out
    g37/A0                                                  +0    2186
    g37/Y                  OAI21X4TS         1  15.3  103  +118   2303 F
    g48/A                                                   +0    2303
    g48/Y                  BUFX16TS          7  71.1   79  +182   2485 F
    Errors_register/enable
      g31/A                                                 +0    2485
      g31/Y                INVX16TS          6  38.2   62   +76   2560 R
      g324/A1N                                              +0    2560
      g324/Y               OAI2BB1X4TS       1   6.0   82  +197   2758 R
      Data_reg_reg[5]/D    DFFRHQX4TS                       +0    2758
      Data_reg_reg[5]/CK   setup                        28  +189   2947 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 625MHz_CLK)          capture                               1760 R
                            latency                         +140  1900 R
                            uncertainty                      -17  1883 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :   -1064ps (TIMING VIOLATION)
Start-point  : comparator_unit/dataB_Unit/CounterB_register/Data_reg_reg[2]/CK
End-point    : comparator_unit/dataB_Unit/Errors_register/Data_reg_reg[5]/D
```

Figure 4.3 Worst slack path taken from the timing report for the test module with combinational logic modification.

4.2.3 Comparison data module implemented with registers between the combinational logic

Analyzing the path in Figure 4.4, the starting point is from the dataB_Unit module and continues until the register for the DataIsEqual_Comparator module output, the DataEqual_register D flip-flop. Different from Figure 4.3, where the path continues until the Errors_register module, the structure with D flip-flops between the combinational logic provides a negative slack of -416ps, a difference of 648ps and representing an improvement of 61% picoseconds in timing with respect to the last structure.



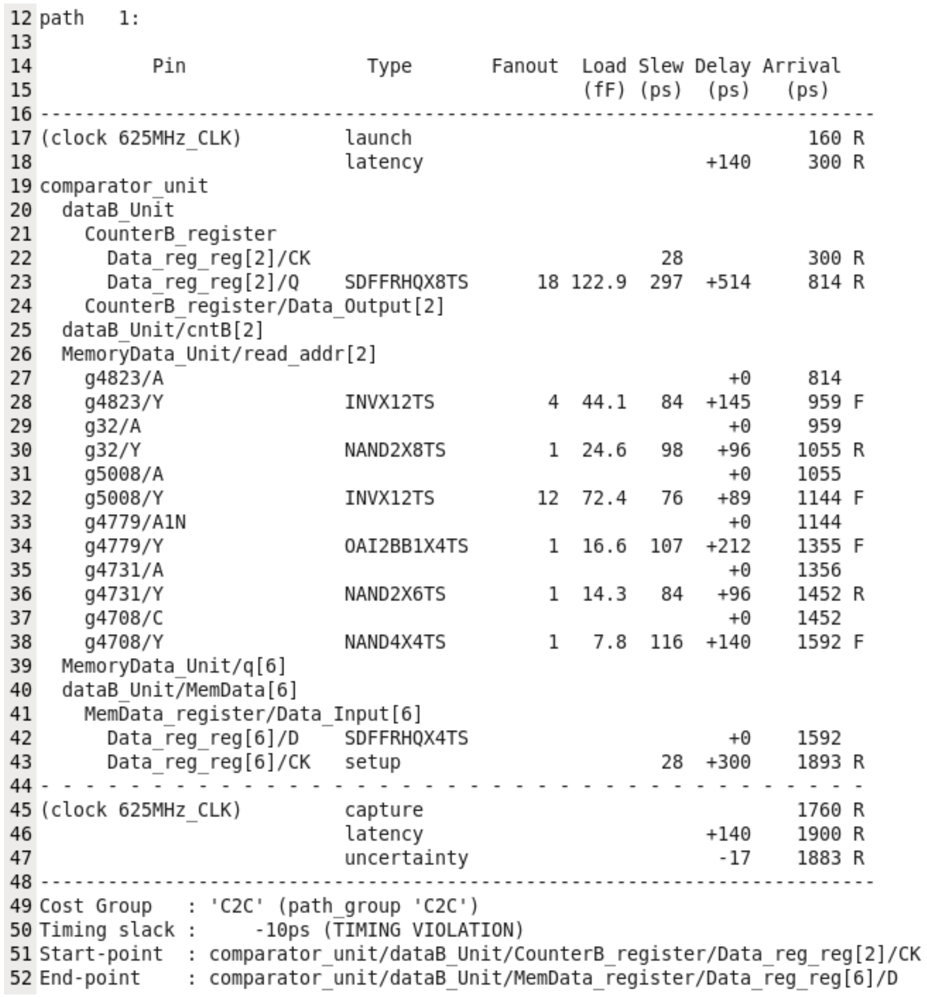
```
path   1:

          Pin                Type       Fanout Load Slew Delay Arrival
                                               (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock 625MHz_CLK)         launch                                 160 R
                           latency                        +140    300 R
comparator_unit
  dataB_Unit
    CounterB_register
      Data_reg_reg[2]/CK                              28           300 R
      Data_reg_reg[2]/Q    SDFFRHQX8TS       4 46.9  128  +401     701 F
    CounterB_register/Data_Output[2]
  dataB_Unit/cntB[2]
  MemoryData_Unit/read_addr[2]
    fopt5292/A                                              +0     702
    fopt5292/Y             INVX12TS          3 48.7   91  +111     812 R
    fopt5296/A                                              +0     812
    fopt5296/Y             INVX8TS           5 31.8   58   +75     887 F
    g5299/A                                                 +0     887
    g5299/Y                AND2X8TS          5 36.6   86  +188    1076 F
    g4989/A                                                 +0    1076
    g4989/Y                NAND2X2TS         1 10.5  134  +118    1194 R
    g5176/B0                                                +0    1194
    g5176/Y                OAI2BB1X4TS       1 13.5   98  +118    1312 F
    g91/A0                                                  +0    1312
    g91/Y                  OAI21X4TS         1 12.3  161  +160    1472 R
    g4922/A                                                 +0    1472
    g4922/Y                NAND2X4TS         1 11.3   80  +112    1585 F
  MemoryData_Unit/q[0]
  dataB_Unit/MemData[0]
    DataIsEqual_Comparator/Data_B[0]
      g1237/A                                               +0    1585
      g1237/Y              XNOR2X4TS         1 15.1  255  +159    1744 R
      g127/D                                                +0    1744
      g127/Y               NAND4X4TS         1 12.0  140  +221    1965 F
      g407/B                                                +0    1965
      g407/Y               NOR2X4TS          1  5.3   96  +140    2104 R
    DataIsEqual_Comparator/Comp_out
    DataEqual_register/Data_Input[0]
      Data_reg_reg[0]/D    DFFRHQX2TS                       +0    2104
      Data_reg_reg[0]/CK   setup                      28  +195    2299 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 625MHz_CLK)         capture                               1760 R
                           latency                        +140   1900 R
                           uncertainty                     -17   1883 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :    -416ps (TIMING VIOLATION)
Start-point  : comparator_unit/dataB_Unit/CounterB_register/Data_reg_reg[2]/CK
End-point    : comparator_unit/dataB_Unit/DataEqual_register/Data_reg_reg[0]/D
```

Figure 4.4 Worst slack for the test module where the comparison data module has registers between its combinational logic.

4.2.4 Store data and comparison data modules with registers for input data

With the last modifications implemented in both modules, the negative slack is almost reduced to zero. The worst slack is generated by the route from the CounterB_register to the MSB memory input register in the comparison data module. The negative slack is -10ps, as seen in Figure 4.5, a difference of 406ps or 97% picoseconds improvement compared to the previous modification.



```
path   1:

          Pin               Type       Fanout  Load Slew Delay Arrival
                                               (fF) (ps)  (ps)   (ps)
--------------------------------------------------------------------------
(clock 625MHz_CLK)         launch                                  160 R
                           latency                        +140     300 R
comparator_unit
  dataB_Unit
    CounterB_register
      Data_reg_reg[2]/CK                               28           300 R
      Data_reg_reg[2]/Q    SDFFRHQX8TS      18 122.9  297  +514     814 R
    CounterB_register/Data_Output[2]
  dataB_Unit/cntB[2]
  MemoryData_Unit/read_addr[2]
    g4823/A                                                 +0      814
    g4823/Y                INVX12TS          4  44.1   84  +145     959 F
    g32/A                                                   +0      959
    g32/Y                  NAND2X8TS         1  24.6   98   +96    1055 R
    g5008/A                                                 +0     1055
    g5008/Y                INVX12TS         12  72.4   76   +89    1144 F
    g4779/A1N                                               +0     1144
    g4779/Y                OAI2BB1X4TS       1  16.6  107  +212    1355 F
    g4731/A                                                 +0     1356
    g4731/Y                NAND2X6TS         1  14.3   84   +96    1452 R
    g4708/C                                                 +0     1452
    g4708/Y                NAND4X4TS         1   7.8  116  +140    1592 F
  MemoryData_Unit/q[6]
  dataB_Unit/MemData[6]
    MemData_register/Data_Input[6]
      Data_reg_reg[6]/D    SDFFRHQX4TS                      +0     1592
      Data_reg_reg[6]/CK   setup                       28  +300    1893 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 625MHz_CLK)         capture                                 1760 R
                           latency                        +140     1900 R
                           uncertainty                     -17     1883 R
--------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :     -10ps (TIMING VIOLATION)
Start-point  : comparator_unit/dataB_Unit/CounterB_register/Data_reg_reg[2]/CK
End-point    : comparator_unit/dataB_Unit/MemData_register/Data_reg_reg[6]/D
```

Figure 4.5 Worst slack for the save data and comparison data modules with input data registers.

4.2.5 Simulation

4.2.5.1 Comparison data module

Figure 4.6 illustrates the comparison data module simulation where the dataB input gets five decoded words, the second and fifth are decoded commas pointing the start and end of the transaction. From transitions in Figure 3.3, the states are defined as 0 for IDLE, 1 for CLEAR, 2 for WAIT, and 3 for COMPARE. In the simulation, the *State* signal shows the transitions starting the process when a decoded comma is received, it stays at the WAIT state until receiving new data, then, changes to the COMPARE state.

In the COMPARE state, *MemData* signal and *dataB* signal, are compared. For the next state *cntB* increases in order to read the next slot in memory, therefore, the *MemData* signal acquires a new value for the next comparison. When in the COMPARE state, a second decoded comma is received, a transition to the end of the comparison process is made and the *bist_end* signal is set to a high logic state in order to mark the end of the communication.

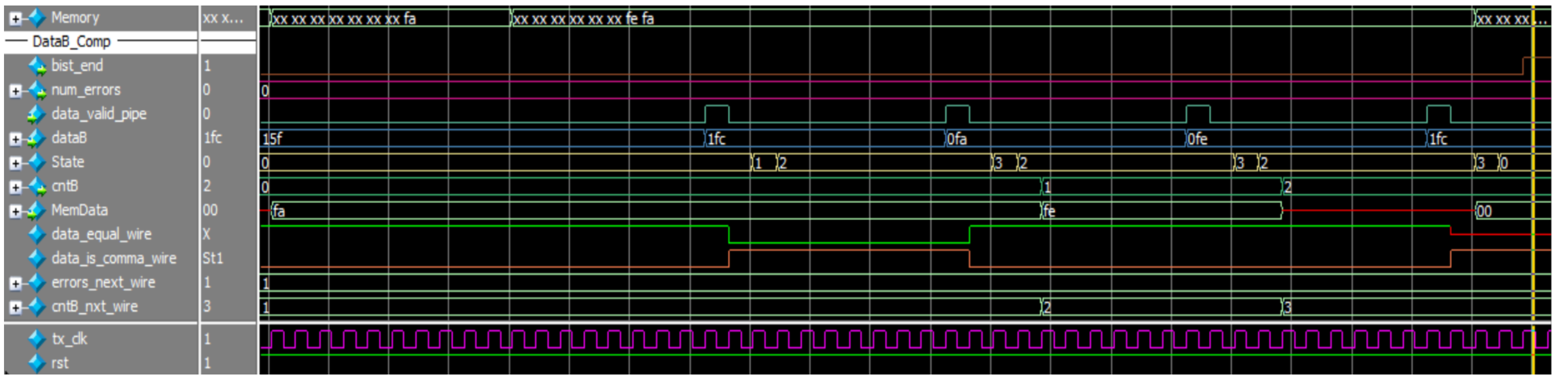


Figure 4.6 Comparison data module simulation for data transmission.

The results in the comparison process made by implementing registers for input data and after the finite state machine combinational logic, can be seen in Figure 4.7. When new data goes into the dataB input port, the state transition is delayed two clock cycles. Once the transition is made, the outcome as a result of the current state is visible after two clock cycles. In total, four clock cycles are required in order to have a response, one for the input data registers, one for the state transition, another for the registers after the combinational logic and one for the output signal´s register.

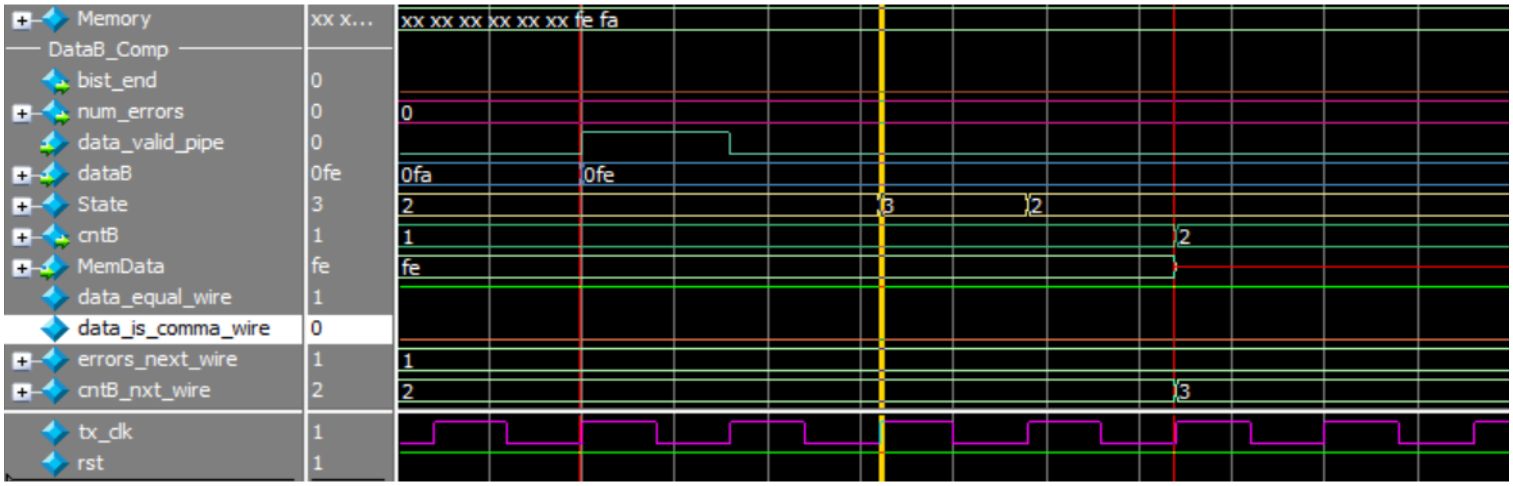


Figure 4.7 Delays generated by the registers in the comparison data module structure.

4.2.5.2 Store data module

The data store module saves the transmitted data between two decoded commas in a transmission. The simulation in Figure 4.8 depicts the storing process after the first decoded comma is received. In the *State* signal, 1 stands for the WAIT state and 2 for the SAVE state, Figure 3.2 can be reviewed for all the transition details. From the figure below, the effects of the input data registers are corroborated, when *dataA* signal changes, the D flip-flop takes one clock cycle to register the input and one cycle for the state transition. In the SAVE state, the *we* signal enables the data memory writing and the *cntA* signal indicates the slot memory. After one clock cycle, *cntA* increases and the data in memory is registered.

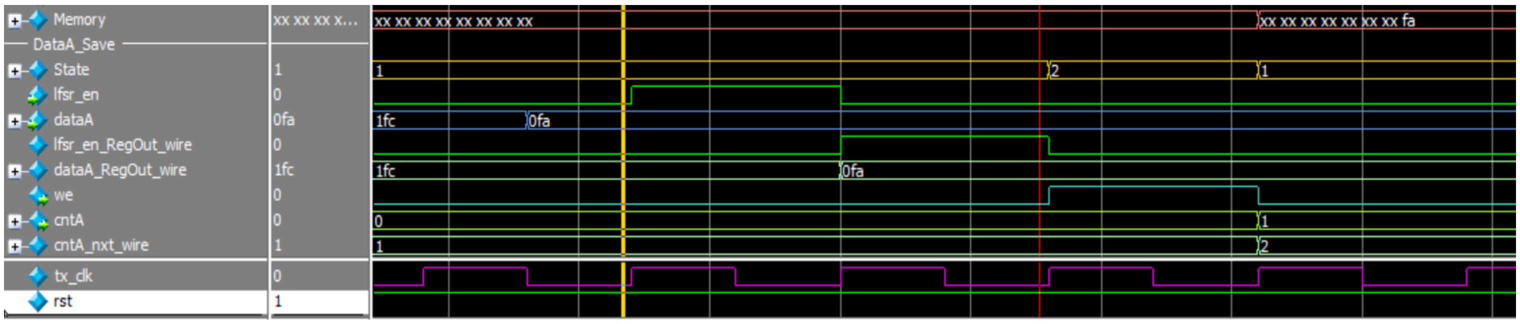


Figure 4.8 Store data module simulation.

4.3 Signal driver

In the next subsections, each multiplexor designed in section 3.3, is simulated in order to show its performance. One input is chosen by the multiplexor for each test operating mode, hence, one simulation for each multiplexor input is shown.

4.3.1 Digital receiver input multiplexor

The digital receiver multiplexor has as input the analog receiver output (*rxa_out*), the digital transmitter output (*txd_data_out*) and the *test_in* signal. In Figure 4.9 are three test operative modes where each signal is selected, *rxa_out* for 3b001, *txd_data_out* for 3b010 and *test_in* for 3b011.

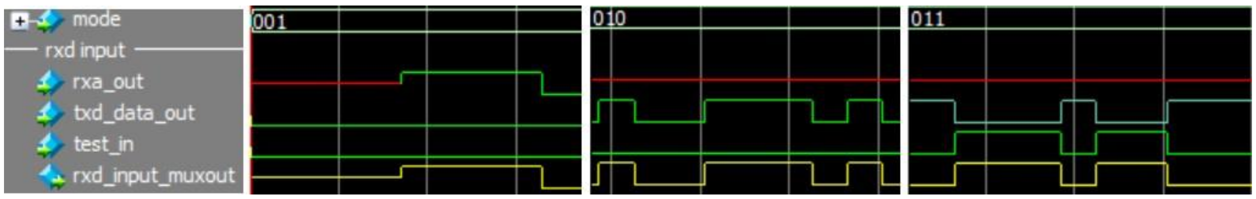


Figure 4.9 Digital receiver input multiplexor simulation for three test operative modes.

4.3.2 Digital transmitter input multiplexor

Figure 4.10 shows the digital transmitter input multiplexor simulation where *txd_data_in* is the digital transmitter input data, *rxd_data_out* is the digital receiver output, and *lfsr_parallel_out* is the LFSR output. The multiplexor output (*txd_input_muxout*) is a 9-bit bus and from the image the output changes with the *mode* signal value, 3b010 outputs *txd_data_in*, 3b001 outputs *rxd_data_out* and 3b110 outputs *lfsr_parallel_out*.

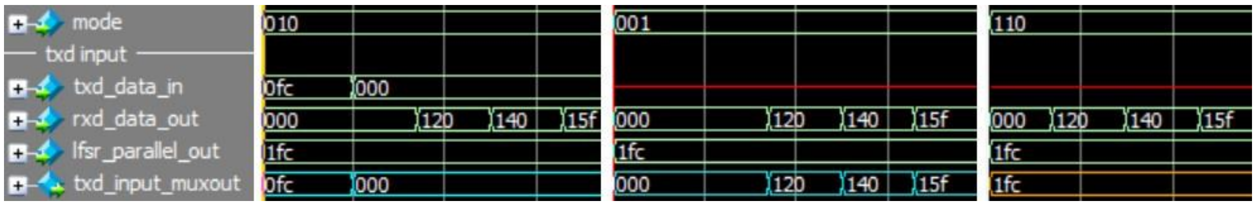


Figure 4.10 Digital transmitter input multiplexor simulation.

4.3.3 Analog transmitter input multiplexor

A two to one multiplexor handles the analog transmitter input and enables to switch between the digital transmitter output (*txd_data_out*) and the analog receiver output (*rxa_out*). Figure 4.11 depicts two test operating mode simulations; mode 3b111 is the only one where the *rxa_out* signal is selected, and for the remaining modes, *txd_data_out* is chosen.

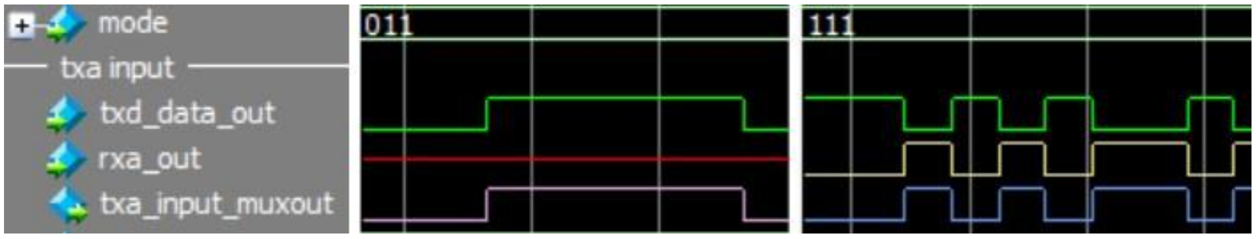


Figure 4.11 Analog transmitter input multiplexor for two operating modes.

4.3.4 Test_out multiplexor

Figure 4.12 illustrates a simulation for the test_out pin possible outputs. This switch is the multiplexor performance with the analog receiver output (*rxa_out*) and *bist_end* signal as inputs. In the image, the last square to the right has the *mode* 3b100, one of the two test operating modes with the BIST modules, therefore, for the other modes, *test_out* outputs the *rxa_out* signal.

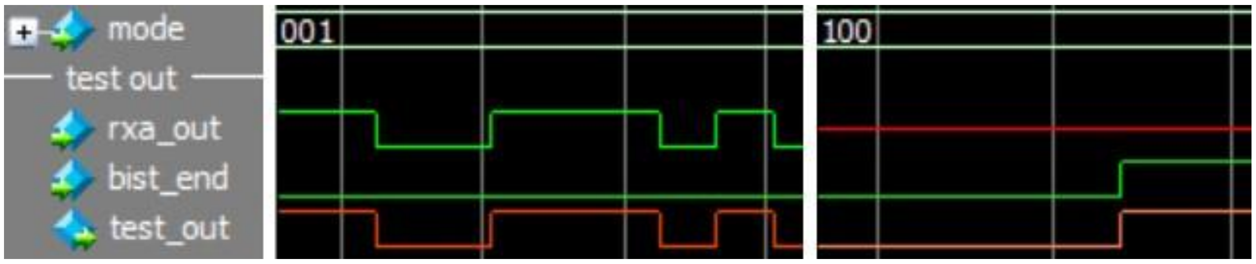


Figure 4.12 Simulation of the test_out pin outputs.

4.3.5 Digital receiver output multiplexor

The SerDes has nine output pins for the digital receiver output but these pins output can be switched by employing a multiplexor. When the test operative mode involves the BIST modules (3b100 and 3b110), the fourth bit from config_in is used to switch the digital receiver output (*rxd_data_out*) with the signals displayed in Figure 4.13, which include the number of errors generated by the BIST process. The image below has a yellow marker to point out when *config_in[4]* will be set high and *rxd_output_muxout* switches to the *rxd_muxinput1_wire* signal.

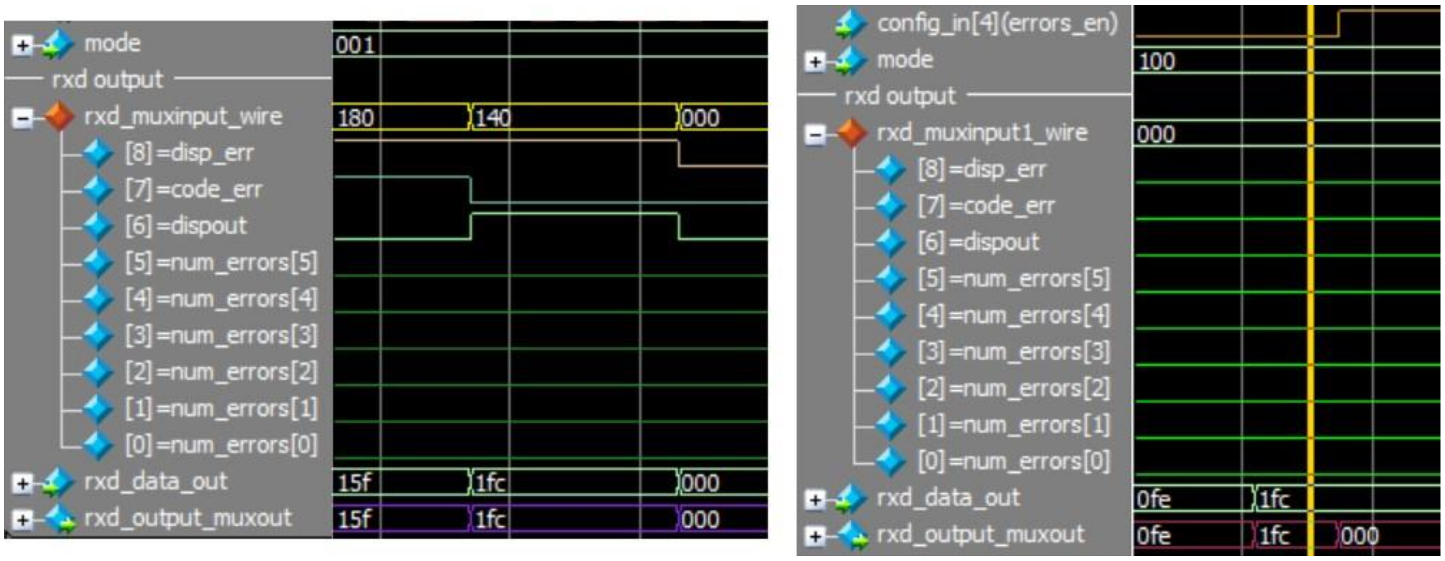


Figure 4.13 Digital receiver output multiplexor simulation for the modes 3b001 and 3b100.

4.4 Top module

Figure 4.14 shows how the store data and the comparison data modules are enabled when the *bist_on_wire* signal is set to a logical high level. The simulation on the left side sets to zero the *txd_frame_star_wire* and *rxd_data_ready_wire* signals when the *test_en* is 1, *config_in[2]* is 1, and *config_in[0]* is 0, therefore, the *State* signal under the dataB_compare and dataA_save divisors stays at the IDLE state. The simulation on the right side has set *bist_on_wire* signal to one, enabling the state transition for *State* signal, hence, the storing and comparing processes are performed.

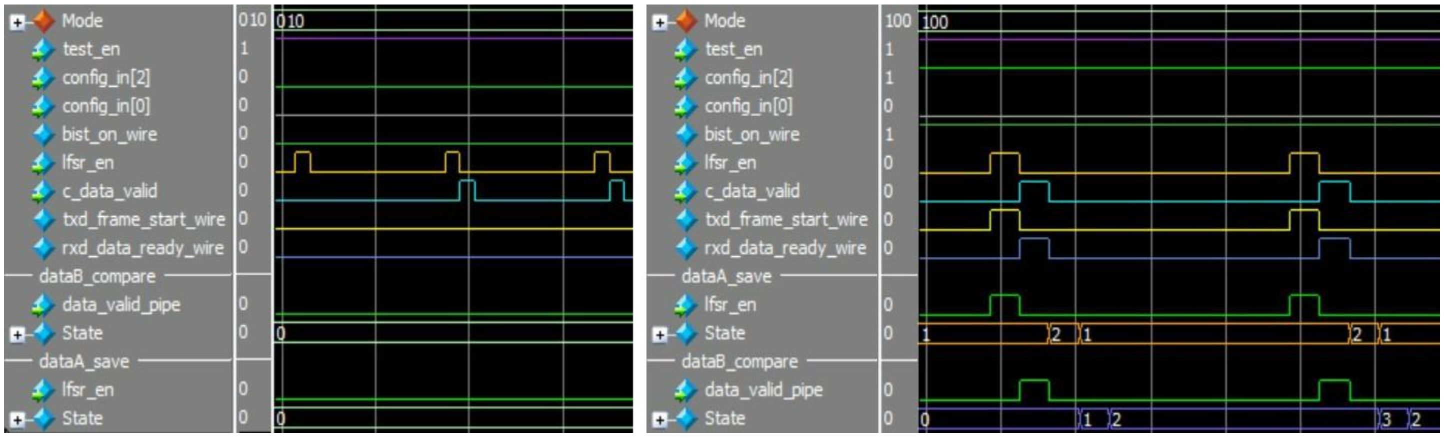


Figure 4.14 BIST modules are enabled with the test operating mode 3b100.

Whenever the LFSR is enabled in certain test operating modes, the data generated is delayed three cycles. When the *lfsr_en* signal is set high, it is registered with two D flip-flops connected serially and when the transition from low to high reaches the LFSR, the data takes one clock cycle to be produced by the LFSR registers, therefore, is a three clock cycle process. This process is illustrated in Figure 4.15 where *txd_frame_start_regOutA_wire* and *txd_frame_start_regOutB_wire* signals are the D flip-flops outputs, and the *lfsr_parallel_out* signal is the LFSR data generated. For modes 3b100 and 3b110, the transmitter data input (*txd_input_muxout*) is the LFSR output (*lfsr_parallel_out*), which is also the dataA signal in the store data module.

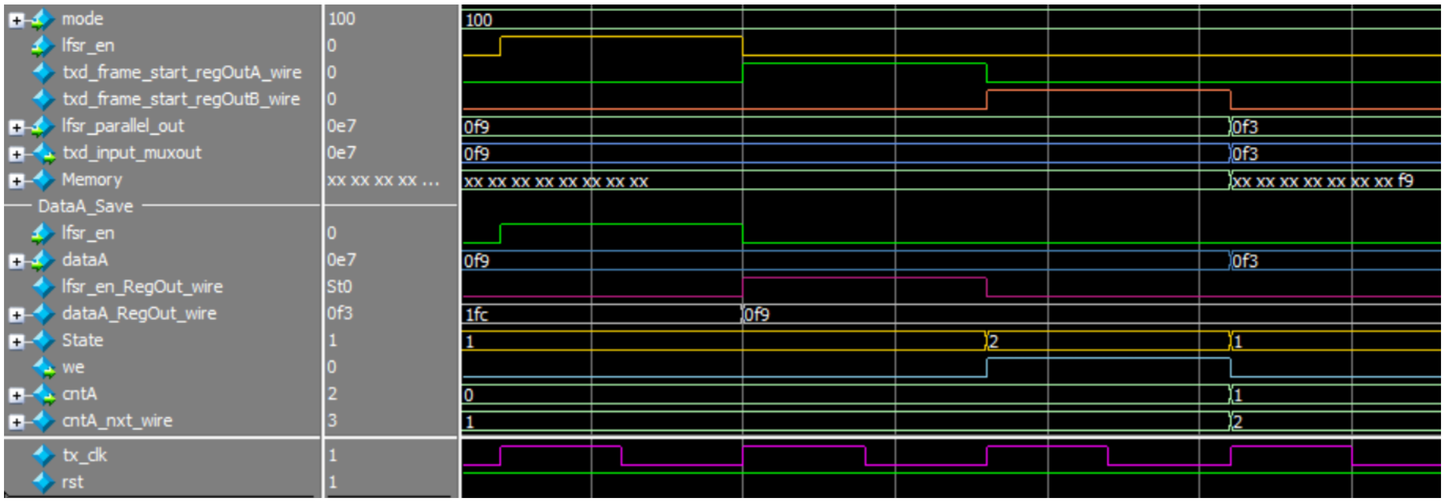


Figure 4.15 Simulation of the LSFR data generation after the enabling signa is delayed two clock cycles.

In the simulation shown in Figure 4.15, the storing process is represented by the signals under the DataA_Save divisor. The module receives the *lfsr_en* signal without delays and the state transition takes two clock cycles due to the input registers, whose output are the *lfsr_en_RegOut_wire* and *dataA_RegOut_wire* signals. Therefore, the data is stored with one clock cycle left before the new data is generated.

4.5 Gate level simulation

The gate level simulation is the process where the netlist produced by the logic synthesis is simulated in order to verify the correct performance. The RC tool generated a file named test_modulse_m.v after mapping with standard cells the design in the file test_modules.v. For the next subsections, each test operating mode is simulated applying the gate level netlist in the test_module_m.v file.

4.5.1 Operating mode 1 (3’b001) – Parallel loopback

The simulation in Figure 4.16 shows how the analog receiver is fed with the data in the *data_transmit* signal, which is transmitted serially by the test bench. The analog receiver output is connected to the digital receiver, the data is deserialized and the output byte serves as the digital transmitter input whose output goes into the analog transmitter. In the simulation, the *txd_input_muxout_wire* signal, and the *rxd_output_muxout_wire* signal, have the same value since this mode has a parallel connection. All these connections are made by the signal driver according to the current mode.

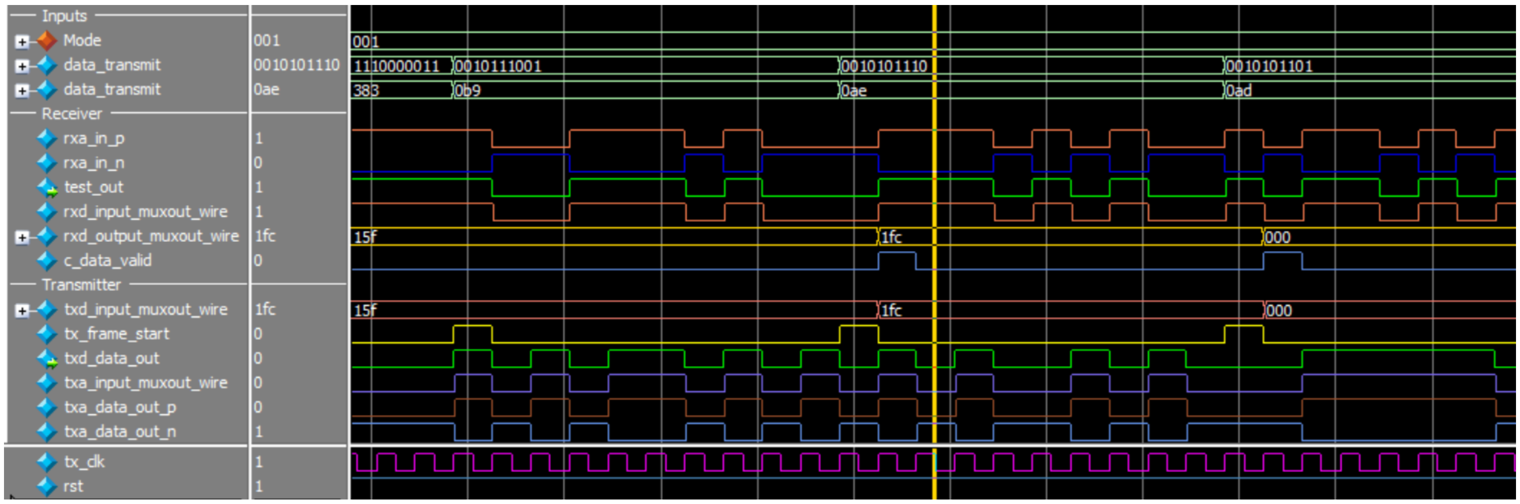


Figure 4.16. Test operating mode 1, gate level simulation.

4.5.2 Operating mode 2 (3’b010) – Serial loopback

This test operating mode enables the connection between the digital transmitter output and the digital receiver input while the analog receiver is isolated. Figure 4.17 depicts the *transmitter_input* signal, which is the data input to encode and transmit by the digital transmitter. The serialization process starts with the *txd_input_muxout_wire* signal and the serial output is sent to the analog transmitter and the digital receiver with the signals *txa_input_muxout_wire* and *rxd_input_muxout_wire*.

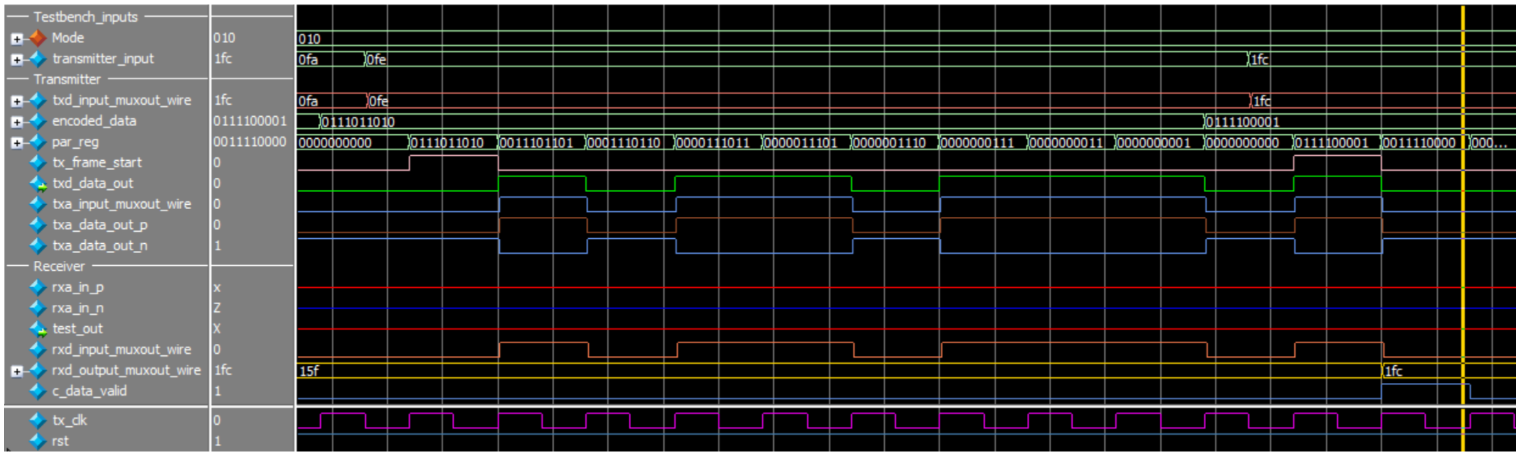


Figure 4.17. Test operating mode 2, gate level simulation.

As a result of the serial loopback, the transmitter input data is also produced by the digital receiver´s *rxd_output_muxout_wire* output signal. In the simulation 9’h1FC, 9’h0FA, 9’0FE, and 9’1FC are transmitted and while the last data is sent, the digital receiver obtains the first data.

4.5.3 Operating mode 3 (3’b011) – RXA bypass

The test operating mode 3, RXA bypass, isolates the analog receiver and the digital receiver input comes from the pin config_in[3] (*test_in*). The simulation illustrated in Figure 4.18 shows *rxa_in_p*, *rxa_in_n* and *test_out* signals with unknown values as a result of being isolated. The test bench uses the complement from output pin *txd_data_out* as the input for the *test_in* signal, therefore, in Figure 4.18 shows that the *txa_data_out_n* signal is equal to the *test_in* signal and is also equal to the *rxd_input_muxout_wire* signal.

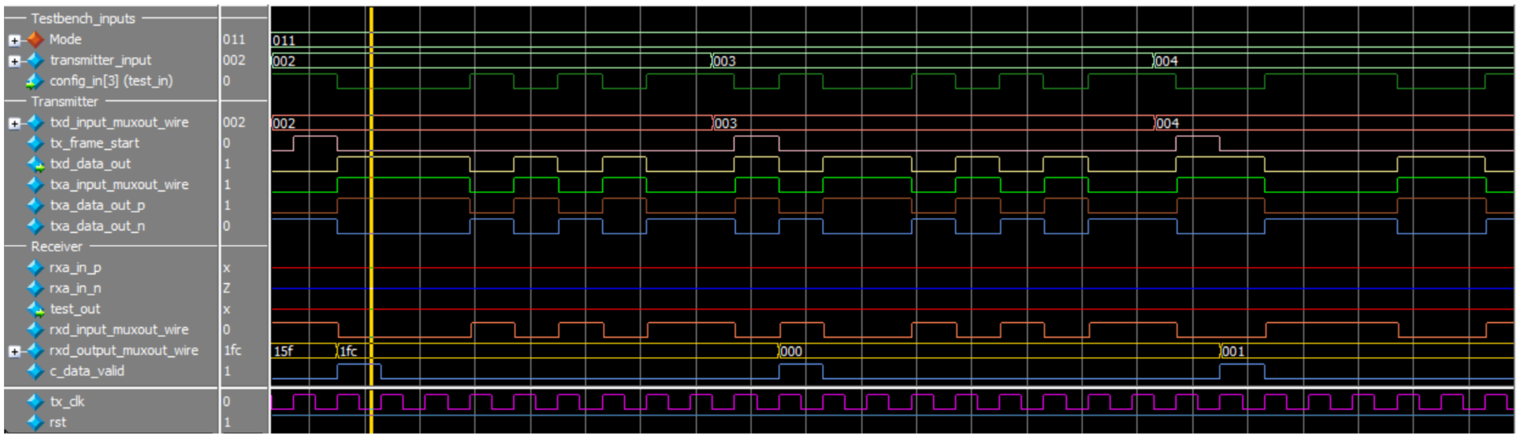


Figure 4.18. Test operating mode 3, gate level simulation.

4.5.4 Operating mode 4 (3'b100) – BIST with serial loopback

This test operating mode enables all the BIST modules. The LFSR output feds the digital transmitter input, the signal driver makes a serial connection between the digital transmitter output and the digital receiver input data and, the comparator module makes the comparison. In Figure 4.19 the *lfsr_parallel_out* signal is the LSFR pseudo-random data and serves as input for the *txd_input_muxout_wire* signal. After the data is transmitted, the receiver deserializes the data in the third reception as seen in the *rxd_output_muxout_wire* signal.

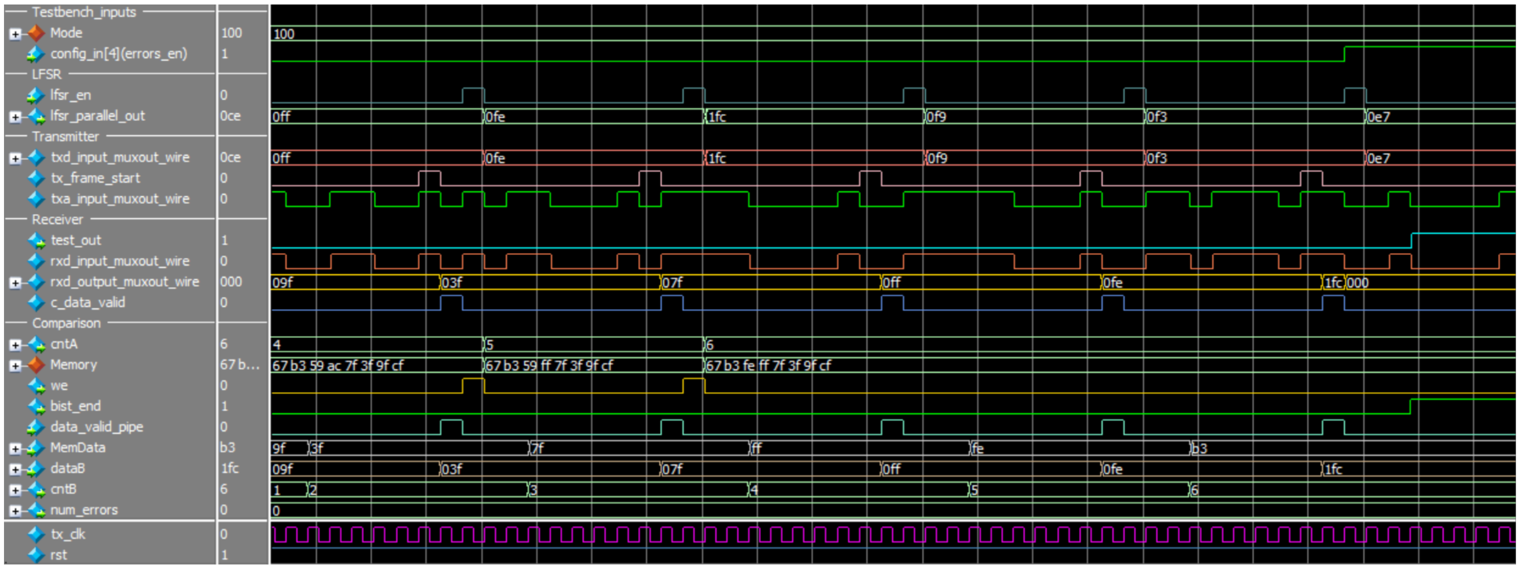


Figure 4.19. Test operating mode 4, gate level simulation.

In the simulation, under the comparison divider, the wave signals from the comparison process are displayed. When the *we* signal is set high and after two clock cycles, the digital transmitter input data is recorded in the slot from *Memory* signal pointed by *cntA* signal. When the *data_valid_pipe* signal is set high and after two clock cycles, a comparison between *dataB* and *MemData* takes place. The comparison result is visible after two clock cycles when, if there is a mismatch, it is registered in the *num_erros* signal and *cntB* increases in order to read a new value for *MemData* for the next comparison while *dataB* is taken from the digital receiver output.

4.5.5 Operating mode 5 (3’b101) – RXA bypass with parallel loopback

This mode is a combination of the analog receiver bypass and the parallel connection between the digital receiver output and the digital transmitter input. The config_in[3] pin is used as input for the digital receiver as Figure 4.20 shows, where it is equal to the *rxd_input_muxout_wire* signal. The parallel loopback connection is demonstrated since the *rxd_output_muxout_wire* signal and *txd_input_muxout_wire signal* have the same value.

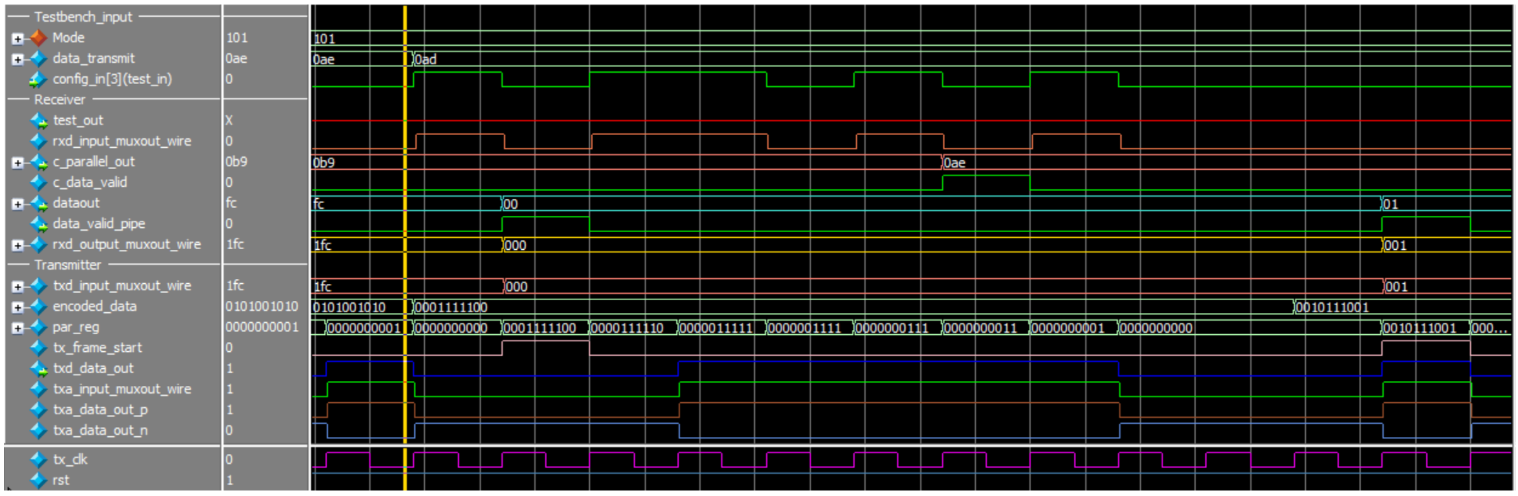


Figure 4.20. Test operating mode 5, gate level simulation.

4.5.6 Operating mode 6 (3’b110) – Open BIST

The test operating mode 6 resembles the mode 4 but without the analog receiver bypass and the parallel loopback, and the analog receiver output is wired to the digital receiver. With these internal connections, the test bench links the analog transmitter differential output with the analog receiver differential input. This configuration is displayed in Figure 4.21, *txa_data_out_p* is equal to *rxa_in_p* and *txa_data_out_n* is equal to *rxa_in_n*. Another characteristic in the simulation is the config_in[4] pin set to a logical high level, hence, the *rxd_ouput_muxout_wire* switches from the digital receiver output to the number of errors registered in the comparison process.

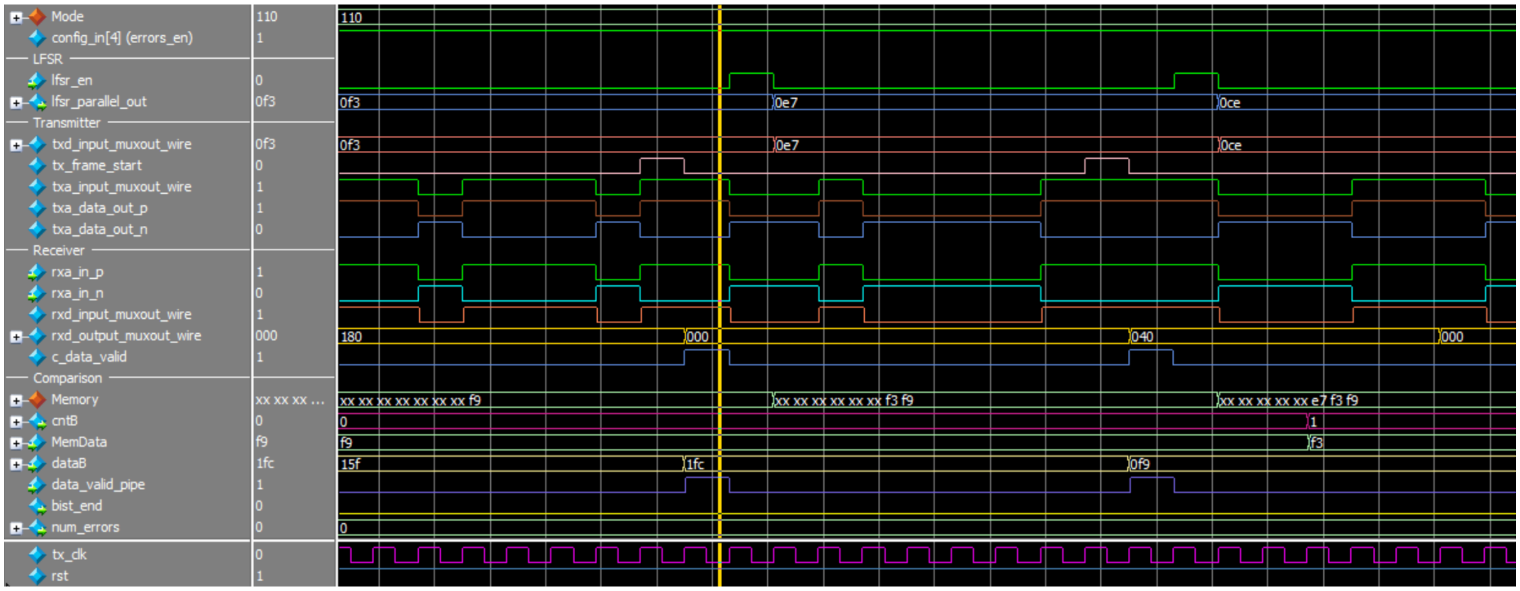


Figure 4.21. Test operating mode 6, gate level simulation.

4.5.7 Operating mode 7 (3'b111) – RXA output with analog loopback

The test bench for the test operating mode 7 applies the *txd_data_out* signal in order to produce the *rxa_in_p* and *rxa_in_n* signals for the analog receiver. The signals that demonstrate the analog loopback are shown in Figure 4.22, *test_out* is the signal coming from the analog receiver output and it is propagated through the digital receiver input with the *rxd_input_muxout_wire* signal and in the analog transmitter input with the *txa_input_muxout_wire* signal. All of these signals are equal because the analog loopback links the analog receiver output and the analog transmitter input.

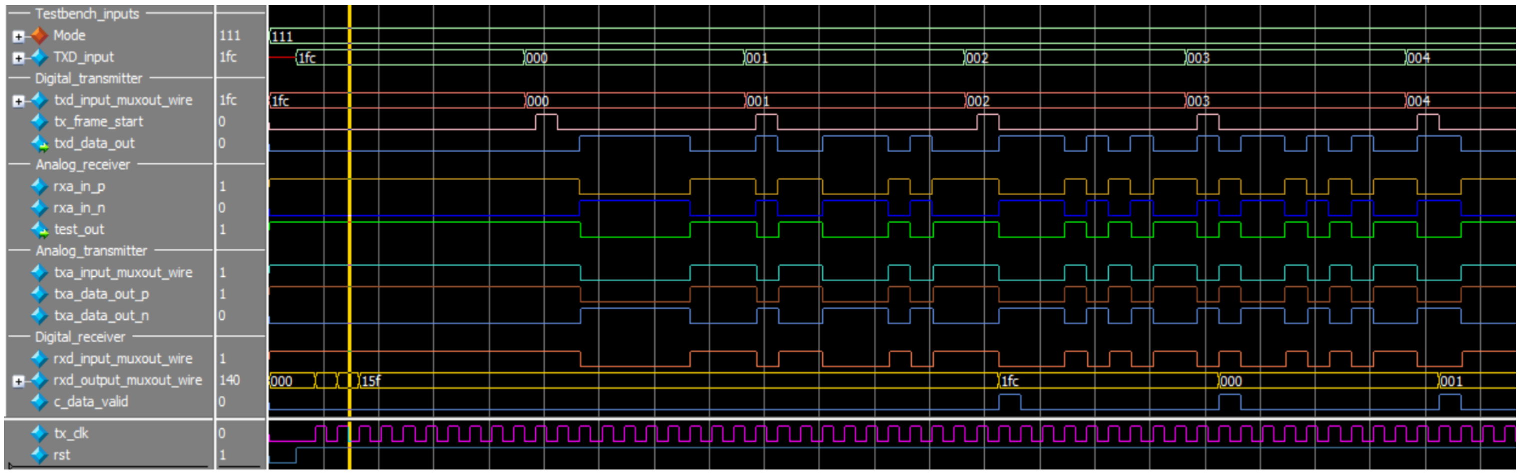
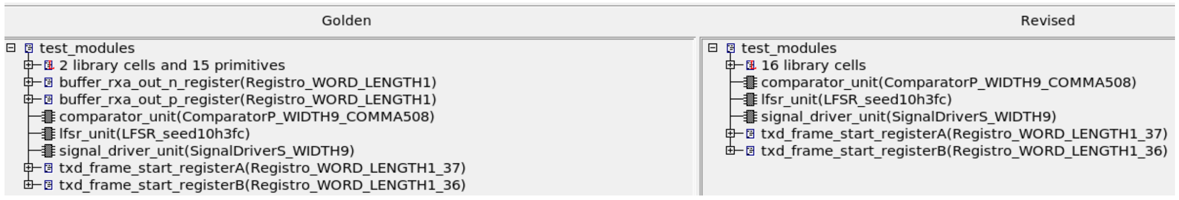


Figure 4.22. Test operating mode 7, gate level simulation.

4.6 Logical Equivalence Checker

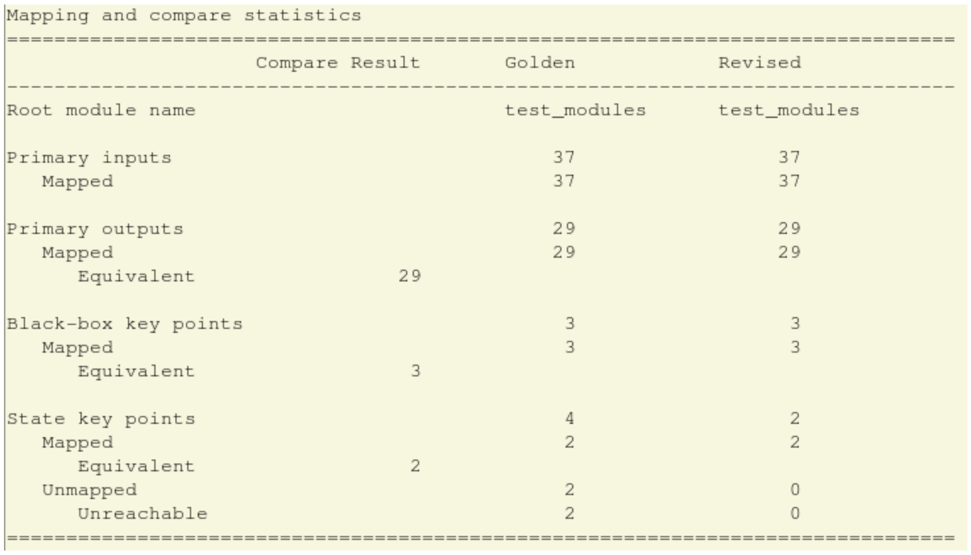
The Logical Equivalence Checker (LEC) tool compares the original design and the gate level netlist in order to ensure the same functionality. In order to perform this comparison, a file with commands for the LEC tool is generated by the logic synthesis. Figure 4.23 shows the components of each design being compared, although both are different, the tool defines compare points in order to check the logic equivalence. Figure 4.24 illustrates a statistics report from the comparison and all the points are equivalent, therefore, both designs are equivalent.



```
Golden
test_modules
  2 library cells and 15 primitives
  buffer_rxa_out_n_register(Registro_WORD_LENGTH1)
  buffer_rxa_out_p_register(Registro_WORD_LENGTH1)
  comparator_unit(ComparatorP_WIDTH9_COMMA508)
  lfsr_unit(LFSR_seed10h3fc)
  signal_driver_unit(SignalDriverS_WIDTH9)
  txd_frame_start_registerA(Registro_WORD_LENGTH1_37)
  txd_frame_start_registerB(Registro_WORD_LENGTH1_36)

Revised
test_modules
  16 library cells
  comparator_unit(ComparatorP_WIDTH9_COMMA508)
  lfsr_unit(LFSR_seed10h3fc)
  signal_driver_unit(SignalDriverS_WIDTH9)
  txd_frame_start_registerA(Registro_WORD_LENGTH1_37)
  txd_frame_start_registerB(Registro_WORD_LENGTH1_36)
```

Figure 4.23 Components of each design.



```
Mapping and compare statistics
================================================================================
                     Compare Result      Golden            Revised
--------------------------------------------------------------------------------
Root module name                         test_modules      test_modules

Primary inputs                                37                37
   Mapped                                     37                37

Primary outputs                               29                29
   Mapped                                     29                29
      Equivalent               29

Black-box key points                           3                 3
   Mapped                                      3                 3
      Equivalent                3

State key points                               4                 2
   Mapped                                      2                 2
      Equivalent                2
   Unmapped                                    2                 0
      Unreachable                              2                 0
================================================================================
```

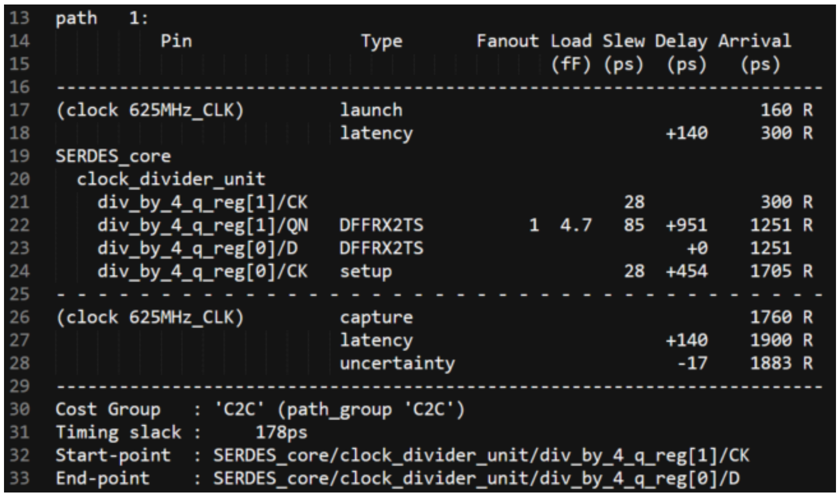
Figure 4.24. Statistics results of the comparison.

4.7 SerDes integration

4.7.1 Logic synthesis

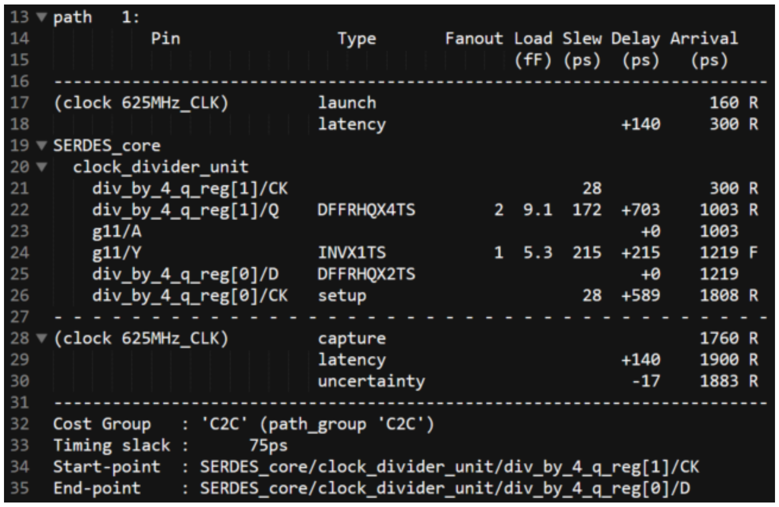
The timing reports generated by the logic synthesis of the SerDes, including the pads, are shown in the next two images. Figure 4.25 displays the results of the typical process while Figure

4.26 displays the worst-case result. The worst-case has a 75ps positive slack and as expected, is by 103ps worse than the 178ps positive slack in the typical process. In both cases, the slacks are positive, therefore, the design is timing compliant.



```
path   1:
               Pin                 Type       Fanout Load Slew Delay Arrival
                                                      (fF) (ps)  (ps)   (ps)
----------------------------------------------------------------------------
(clock 625MHz_CLK)             launch                                160 R
                               latency                      +140     300 R
SERDES_core
  clock_divider_unit
    div_by_4_q_reg[1]/CK                                28           300 R
    div_by_4_q_reg[1]/QN       DFFRX2TS          1  4.7  85  +951   1251 R
    div_by_4_q_reg[0]/D        DFFRX2TS                       +0    1251
    div_by_4_q_reg[0]/CK       setup                    28  +454    1705 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 625MHz_CLK)             capture                              1760 R
                               latency                      +140    1900 R
                               uncertainty                   -17    1883 R
----------------------------------------------------------------------------
Cost Group    : 'C2C' (path_group 'C2C')
Timing slack  :     178ps
Start-point   : SERDES_core/clock_divider_unit/div_by_4_q_reg[1]/CK
End-point     : SERDES_core/clock_divider_unit/div_by_4_q_reg[0]/D
```

Figure 4.25. Worst slack path for the typical process.

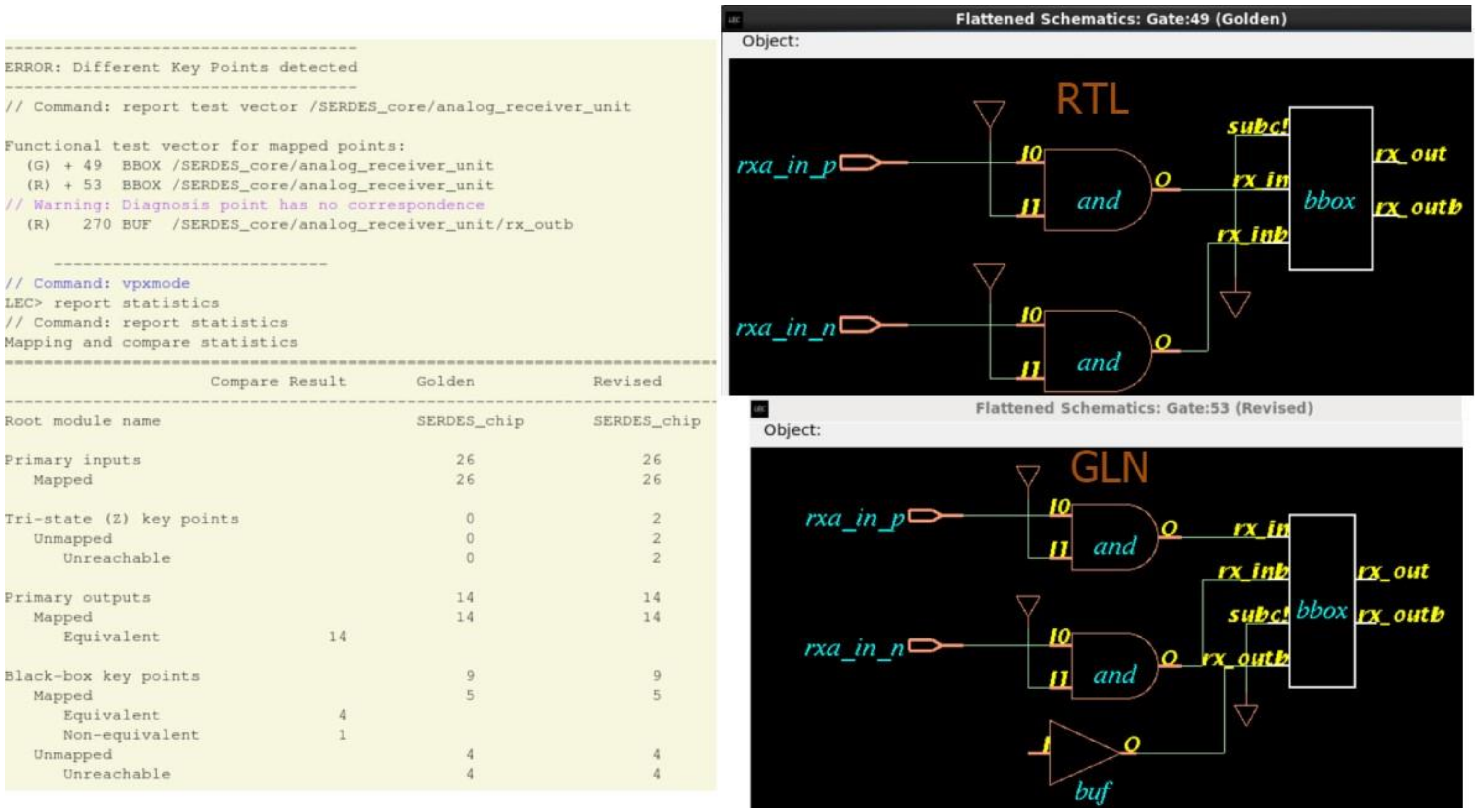


```
path   1:
               Pin                 Type       Fanout Load Slew Delay Arrival
                                                      (fF) (ps)  (ps)   (ps)
----------------------------------------------------------------------------
(clock 625MHz_CLK)             launch                                160 R
                               latency                      +140     300 R
SERDES_core
  clock_divider_unit
    div_by_4_q_reg[1]/CK                                28           300 R
    div_by_4_q_reg[1]/Q        DFFRHQX4TS        2  9.1 172  +703   1003 R
    g11/A                                                     +0    1003
    g11/Y                      INVX1TS           1  5.3 215  +215   1219 F
    div_by_4_q_reg[0]/D        DFFRHQX2TS                     +0    1219
    div_by_4_q_reg[0]/CK       setup                    28  +589    1808 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 625MHz_CLK)             capture                              1760 R
                               latency                      +140    1900 R
                               uncertainty                   -17    1883 R
----------------------------------------------------------------------------
Cost Group    : 'C2C' (path_group 'C2C')
Timing slack  :      75ps
Start-point   : SERDES_core/clock_divider_unit/div_by_4_q_reg[1]/CK
End-point     : SERDES_core/clock_divider_unit/div_by_4_q_reg[0]/D
```

Figure 4.26. Worst slack path for the worst-case process.

4.7.2 LEC

The LEC tool is executed with the three files generated by the RC tool, each file has commands to compare two designs. The first file is the RTL against an intermediate gate level netlist, the second is the intermediate gate-level netlist against the final gate-level netlist, and the final file is the RTL against the final gate-level netlist. For the first comparison, one difference is present in the analog receiver, the second comparison does not have differences, but the final comparison finds different analog receiver structures, as Figure 4.27 shows. Therefore, the intermediate gate-level netlist and the final version are equivalent but the RTL design is not, thus the logic synthesis generates a mapping mismatch. This irregularity should be analyzed, hence, it is proposed as a future work.



```
------------------------------------
ERROR: Different Key Points detected
------------------------------------
// Command: report test vector /SERDES_core/analog_receiver_unit

Functional test vector for mapped points:
  (G) + 49  BBOX /SERDES_core/analog_receiver_unit
  (R) + 53  BBOX /SERDES_core/analog_receiver_unit
// Warning: Diagnosis point has no correspondence
  (R)   270 BUF  /SERDES_core/analog_receiver_unit/rx_outb

     ----------------------------
// Command: vpxmode
LEC> report statistics
// Command: report statistics
Mapping and compare statistics
=========================================================================
                     Compare Result       Golden           Revised
-------------------------------------------------------------------------
Root module name                          SERDES_chip      SERDES_chip

Primary inputs                               26                26
   Mapped                                    26                26

Tri-state (Z) key points                      0                 2
   Unmapped                                   0                 2
      Unreachable                             0                 2

Primary outputs                              14                14
   Mapped                                    14                14
      Equivalent                   14

Black-box key points                          9                 9
   Mapped                                     5                 5
      Equivalent                    4
      Non-equivalent                1
   Unmapped                                   4                 4
      Unreachable                             4                 4
```

Figure 4.27. RTL to final gate level netlist comparison results.

4.7.3 Physical design

4.7.3.1 Power ring

After defining the core size, the power grid is placed. However, the EDI tool produces two errors due to the wrong connection between metals. Figure 4.28 illustrates these errors and how they should be fixed. Though the errors are corrected, other errors arise due to the lack of vias for metal 1 - metal 2. Figure 4.29 shows the location of these via errors and how they are corrected by copying a via for metal 1 - metal 2 and pasting it into the place of each error.

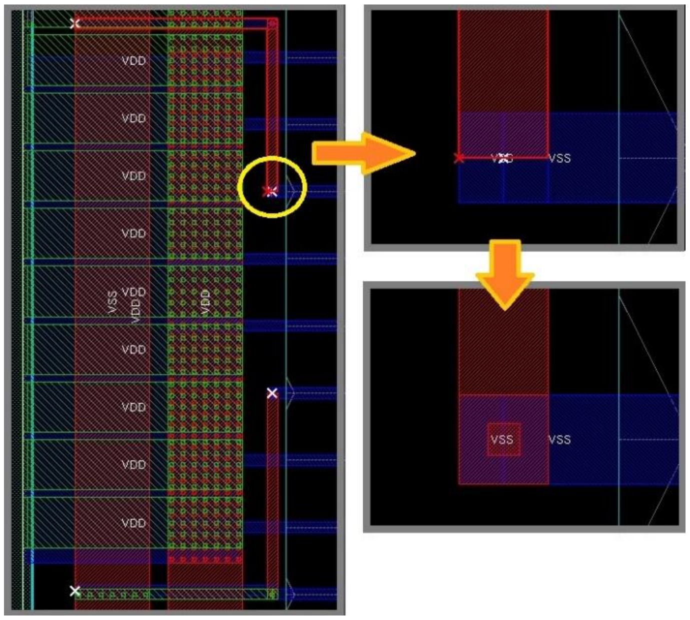


Figure 4.28. Connection error generated by the EDI tool.

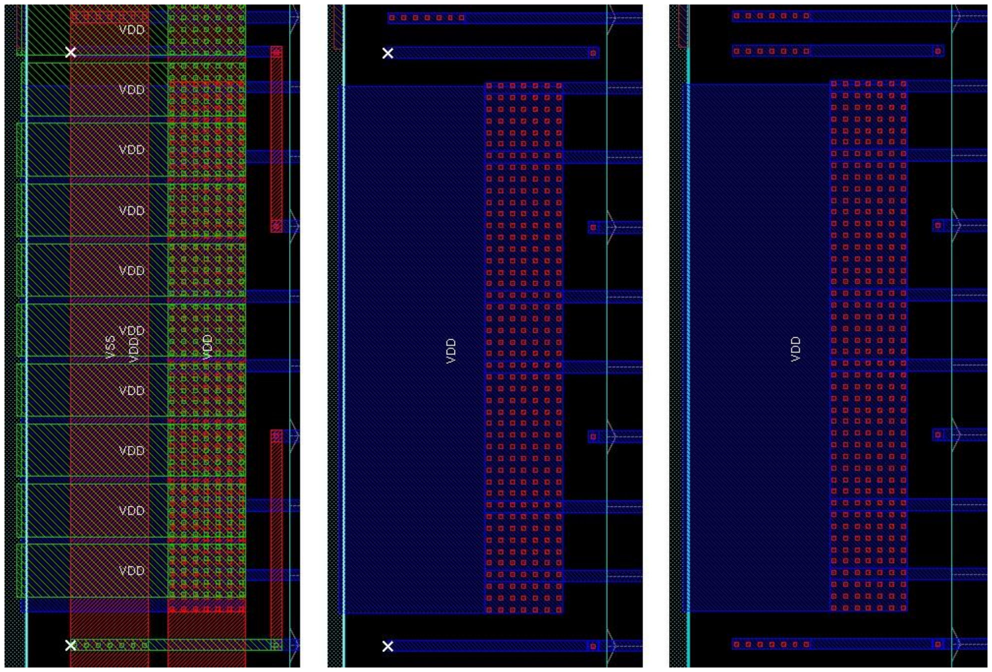
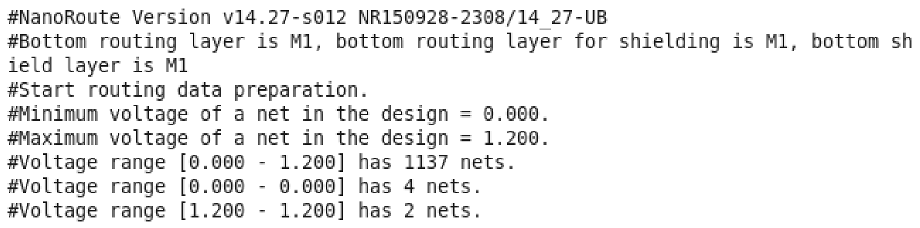


Figure 4.29. Via errors generated and the correction.

4.7.3.2 CTS

The only issue with the CTS process was that the GUI freezes and stops responding. Figure 4.30 shows the lines where the EDI stops working and the command prompt does not change.

Upon letting the tool enough time, it was discovered that with the log file, the CTS takes half hour to end.



```
#NanoRoute Version v14.27-s012 NR150928-2308/14_27-UB
#Bottom routing layer is M1, bottom routing layer for shielding is M1, bottom sh
ield layer is M1
#Start routing data preparation.
#Minimum voltage of a net in the design = 0.000.
#Maximum voltage of a net in the design = 1.200.
#Voltage range [0.000 - 1.200] has 1137 nets.
#Voltage range [0.000 - 0.000] has 4 nets.
#Voltage range [1.200 - 1.200] has 2 nets.
```

Figure 4.30. Last lines where the toll seems to not work.

4.7.3.3 Final layout

After all the physical design steps, the final layout, including the fillers, is generated as Figure 4.31 shows. Clearly, the digital modules are separated into opposite corners from the analog modules, which are placed near their corresponding pads. The distribution of the standard cells for each SerDes digital module is illustrated in Figure 4.32, where the test modules and the digital receiver fill most of the area within their standard cells.

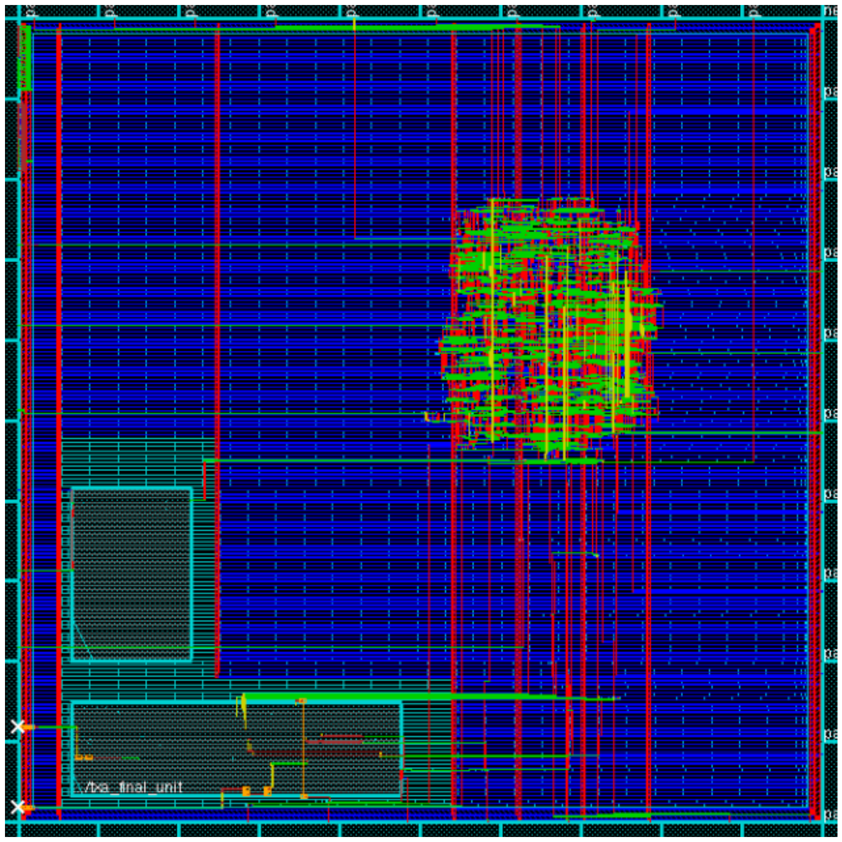
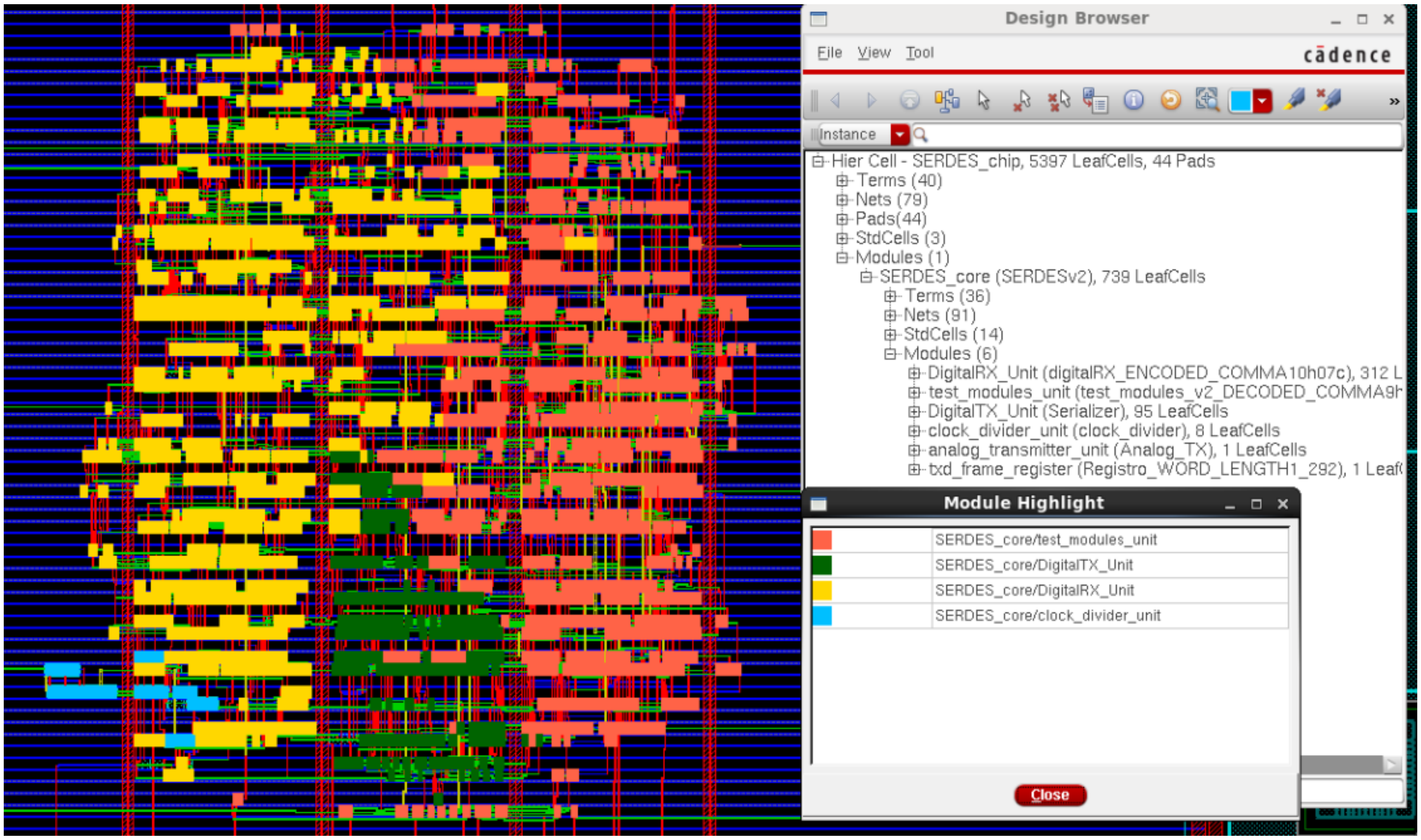


Figure 4.31. SerDes final layout.



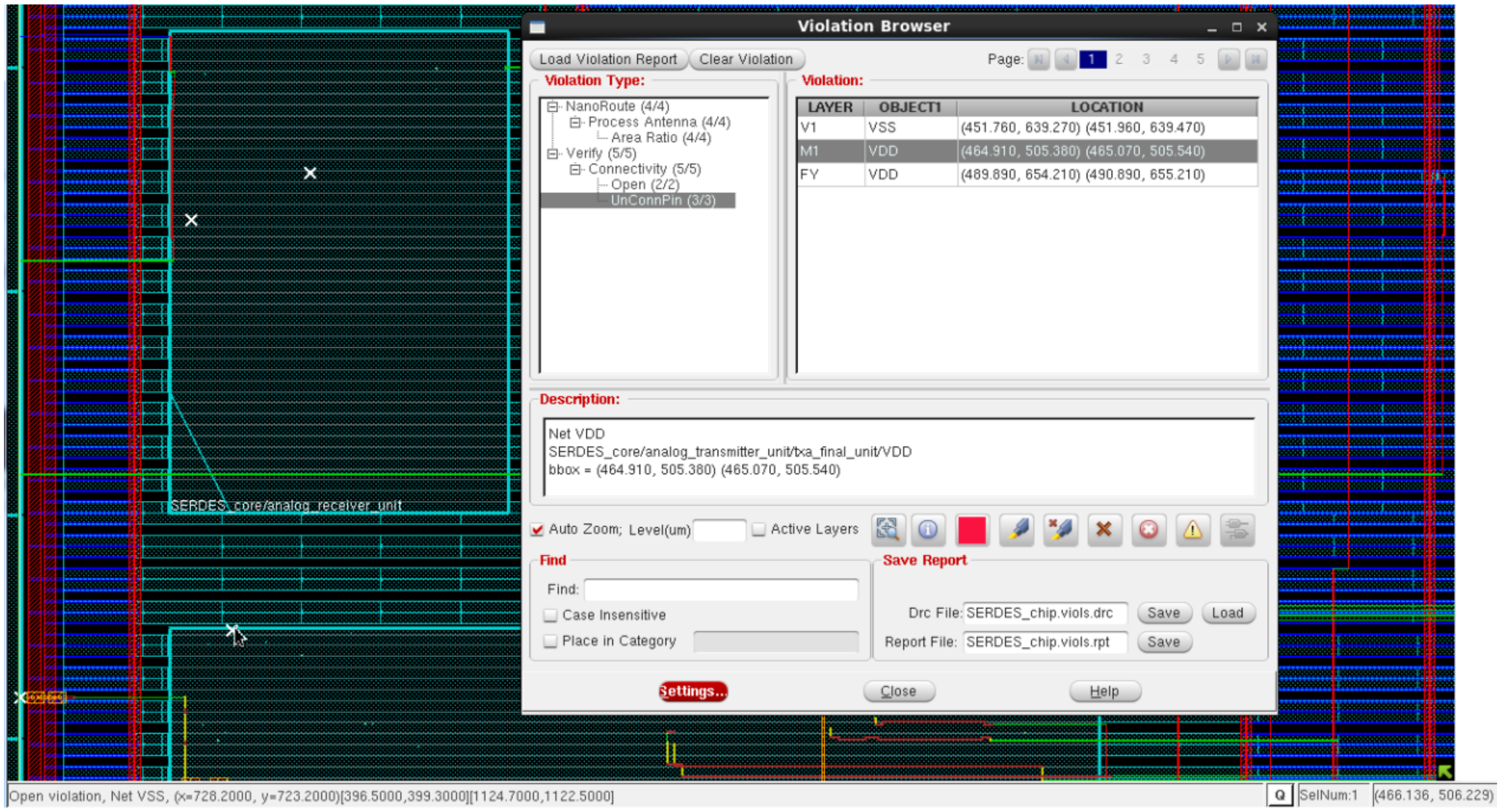
```
⊟ Hier Cell - SERDES_chip, 5397 LeafCells, 44 Pads
  ⊞ Terms (40)
  ⊞ Nets (79)
  ⊞ Pads(44)
  ⊞ StdCells (3)
  ⊟ Modules (1)
    ⊟ SERDES_core (SERDESv2), 739 LeafCells
      ⊞ Terms (36)
      ⊞ Nets (91)
      ⊞ StdCells (14)
      ⊟ Modules (6)
        ⊞ DigitalRX_Unit (digitalRX_ENCODED_COMMA10h07c), 312 L
        ⊞ test_modules_unit (test_modules_v2_DECODED_COMMA9h
        ⊞ DigitalTX_Unit (Serializer), 95 LeafCells
        ⊞ clock_divider_unit (clock_divider), 8 LeafCells
        ⊞ analog_transmitter_unit (Analog_TX), 1 LeafCells
        ⊞ txd_frame_register (Registro_WORD_LENGTH1_292), 1 Leaf
```

🟥	SERDES_core/test_modules_unit
🟩	SERDES_core/DigitalTX_Unit
🟨	SERDES_core/DigitalRX_Unit
🟦	SERDES_core/clock_divider_unit

Figure 4.32. SerDes digital modules distribution in the layout.

4.7.3.4 Design verifications

Without running any verification, the final layout indicates two errors as seen in Figure 4.31. In order to check all the errors, Design Rules Check (DRC), connectivity, and geometry verifications are applied into the design. The verification results are classified and displayed in the *violation browser* window. From Figure 4.33, the layout has 9 errors, 4 routing errors, and 5 connectivity errors. The figure also shows the unconnected pin errors, all of them are from the VDD and VSS ports of the analog modules.



LAYER	OBJECTI	LOCATION
V1	VSS	(451.760, 639.270) (451.960, 639.470)
M1	VDD	(464.910, 505.380) (465.070, 505.540)
FY	VDD	(489.890, 654.210) (490.890, 655.210)

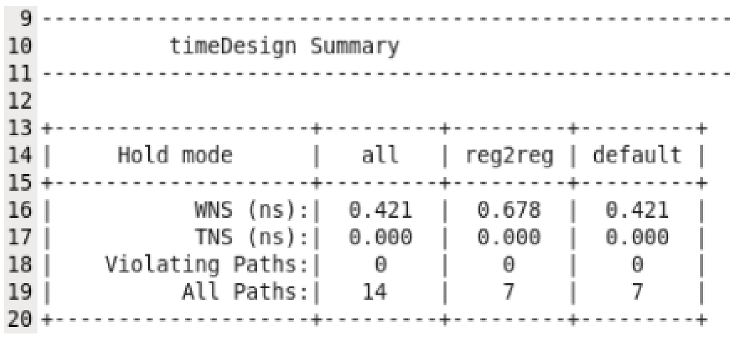
```
Net VDD
SERDES_core/analog_transmitter_unit/txa_final_unit/VDD
bbox = (464.910, 505.380) (465.070, 505.540)
```

```
Open violation, Net VSS, (x=728.2000, y=723.2000)[396.5000,399.3000][1124.7000,1122.5000]
```

Figure 4.33. Violation browser window and unconnected pin errors displayed.

4.7.3.5 Hold and setup timing results summary

Figure 4.34 shows the timing analysis summary for the hold time. It is created with the typical process and as a result, the slacks are positive, 421ps is the worst slack while 678ps is the worst slack for a register to register path.

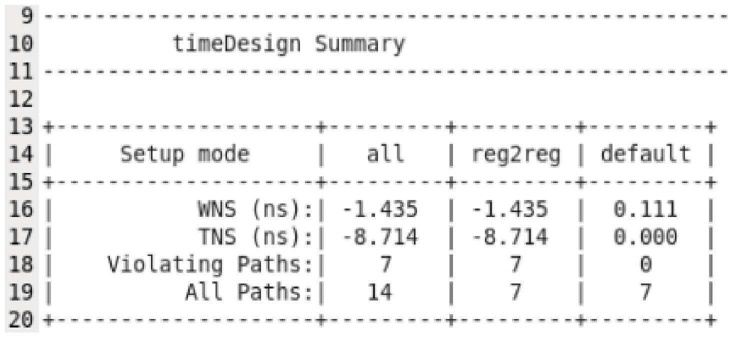


```
------------------------------------------------------
          timeDesign Summary
------------------------------------------------------

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.421  |  0.678  |  0.421  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   14    |    7    |    7    |
+--------------------+---------+---------+---------+
```

Figure 4.34. Hold timing analysis summary.

The timing analysis summary for the setup time is illustrated in Figure 4.35. Since the analysis is created with the worst-case process, the design proves to not work under these conditions because the worst slack is a negative -1.435ns. The setup summary also indicates that there are 14 violating paths with negative slacks.


```
-----------------------------------------------------
          timeDesign Summary
-----------------------------------------------------

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):| -1.435  | -1.435  |  0.111  |
|           TNS (ns):| -8.714  | -8.714  |  0.000  |
|    Violating Paths:|    7    |    7    |    0    |
|          All Paths:|   14    |    7    |    7    |
+--------------------+---------+---------+---------+
```

Figure 4.35. Setup timing analysis summary.

It is important to mention the vertical stripes inside the orange rectangle in Figure 3.20 because modifying the distance between them can affect timing results. Since the digital modules are placed in this region, these stripes provide the power supply and at the same time, represent an obstacle for routing the design.

4.7.4 Virtuoso importation

Despite the errors pointed by the EDI, the process to import a GDS stream into Virtuoso was done. The result displayed in Figure 4.36 shows the lack of the analog modules due to the empty spaces where they were placed in the physical design. To fix this issue, each analog module layout was instanced and manually placed where it belongs after applying the respective library created by the analog designer. Figure 4.37 shows the final layout with every analog module instance placed within their respective location.

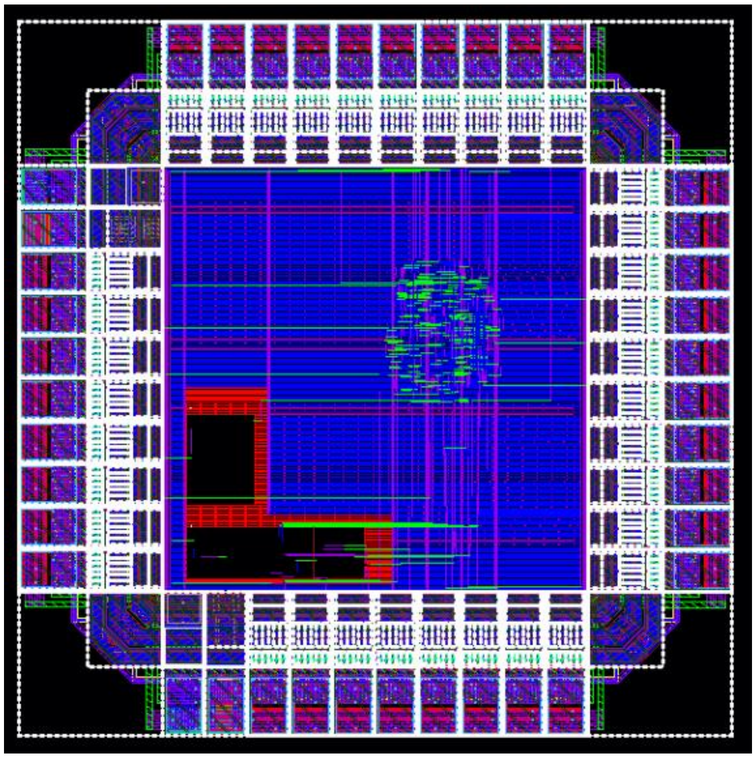


Figure 4.36. Layout in Virtuoso after the importation.

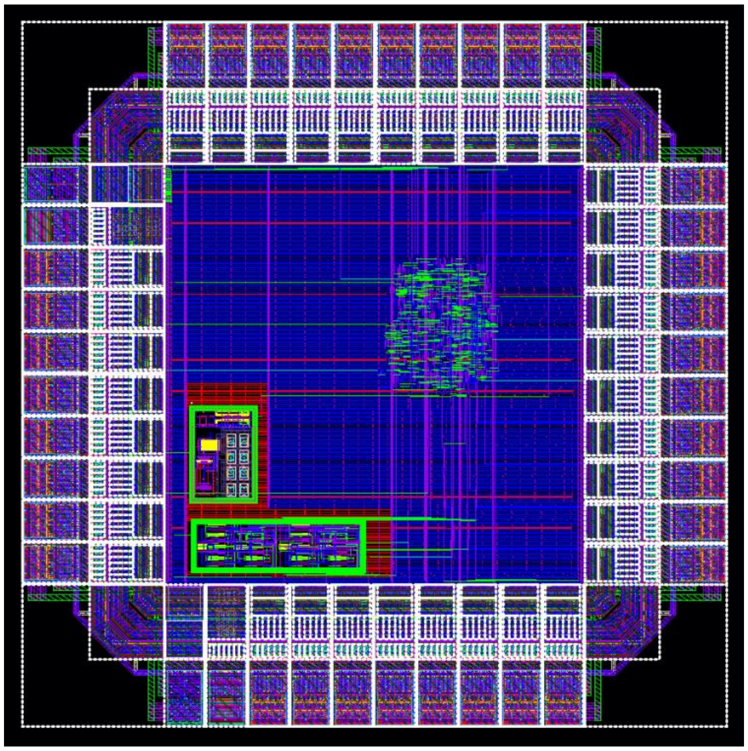


Figure 4.37. Final layout after instance the analog modules.

5 Discussion

5.1 LFSR

The first pseudo-random test pattern generated by the LFSR is the 10-bit comma 10'b11_1111_1100 and after a comparison, a 9-bit test pattern is produced. Table 5.1 shows the 12 control characters and the 10-bit comma value is not listed, nevertheless, the 8-bit K28.7 with the ninth bit set to 1 is the 9-bit binary value 9'b1_1111_1100 and is similar to the 10-bit comma used in the comparison to produce the 9-bit test pattern.

Table 5.1. Control characters for 8-bit/10-bit codification (H-MSB, j-MSB).

Code	9th bit	8-bit decoded data	10-bit encoded data	
	K	HGFE DCBA	RD- ab cdei fghj	RD+ ab cdei fghj
K28.0	1	0001 1100	00 1111 0100	11 0000 1011
K28.1	1	0011 1100	00 1111 1001	11 0000 0110
K28.2	1	0101 1100	00 1111 0101	11 0000 1010
K28.3	1	0111 1100	00 1111 0011	11 0000 1100
K28.4	1	1001 1100	00 1111 0010	11 0000 1101
K28.5	1	1011 1100	00 1111 1010	11 0000 0101
K28.6	1	1101 1100	00 1111 0110	11 0000 1001
K28.7	1	1111 1100	00 1111 1000	11 0000 0111
K23.7	1	1111 0111	11 1010 1000	00 0101 0111
K27.7	1	1111 1011	11 0110 1000	00 1001 0111
K29.7	1	1111 1101	10 1110 1000	01 0001 0111
K30.7	1	1111 1110	01 1110 1000	10 0001 0111

In the test operating modes 3'b100 and 3'b110, the 9-bit digital transmitted input is fed by the LFSR output. Since the digital transmitter receives decoded data, the LFSR should produce 9-bits test patterns, which are not encoded, and it should not implement 10 registers. Hence, the LFSR structure can be optimized and use fewer registers. A recommendation is to have eight registers with the binary value 8'b1111_1100 as the seed, thus, whenever the data produced is equal to the seed value a ninth bit is set to one and concatenated to the data in order to indicate the comma.

5.2 Comparator module design

5.2.1 Former code in the 2016 thesis

The original comparator's RTL code has a finite state machine with five states, the transitions are illustrated in Figure 5.1. When the transition reaches the comparison state, COMP, a behavioral description code is executed and a comparison between the data transmitted previously and the data received is made. It is important to point out that *dataA_in* is deserialized first in the digital transmitter, it is stored in a memory block and later it is compared to the received data when the same data is captured by the digital receiver. This is the general scheme for the comparison process.

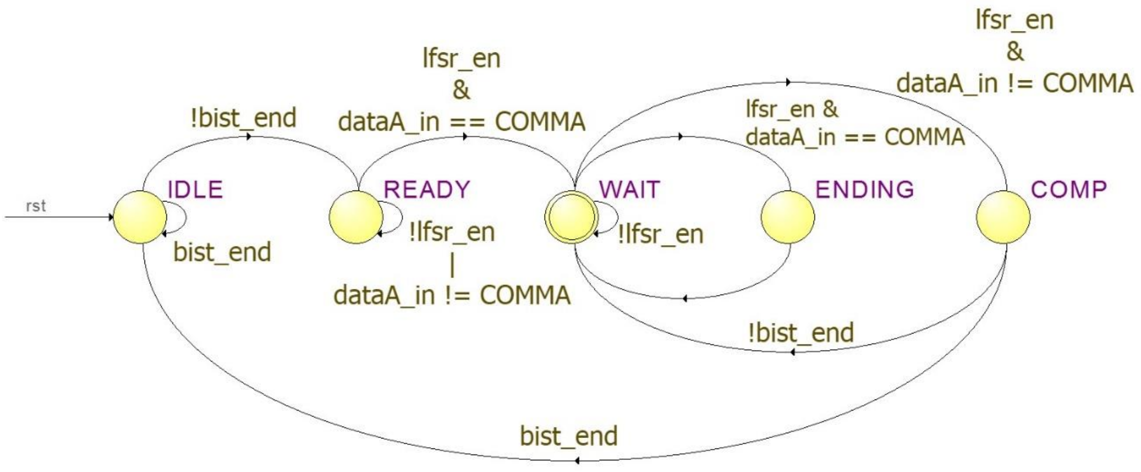


Figure 5.1. Finite state machine transitions of the 2016 thesis.

The first observation is the loop in the IDLE state, which is met when the comparator has received two commas. In this situation, the *bist_end* signal has been modified in the COMP state where is set high, hence, there is a transition to IDLE and the finite state machine stays permanently in the IDLE state. Until this point, the changes have been done by reading the transmitted data *dataA_in*.

The overall state transition process does not check the digital receiver data when it is captured since almost every transition has the signal *dataA_in*. The transition from the WAIT state toward the COMP state is triggered by the *lfsr_en* signal and the comparison of *dataA_in*, the data to transmit, with the value of the comma. Therefore, the comparison is done when the data is transmitted, although the receiver could not have captured the data to compare.

Every time when the finite state machine reaches the COMP state a behavioral description HDL code is executed, the data flow is illustrated in the Figure 5.2. The registers cntA and cntB are used as pointers, cntA is the pointer to store the data to transmit and cntB serves as a pointer to read the data in memory to compare with the data received. The data flow always starts storing dataA_in, next, matchFlag is evaluated and since it is initially zero, the no path is taken. In this context, matchFlag is a flag indicating if the data read in RegMem is equal to the data received dataB_in.

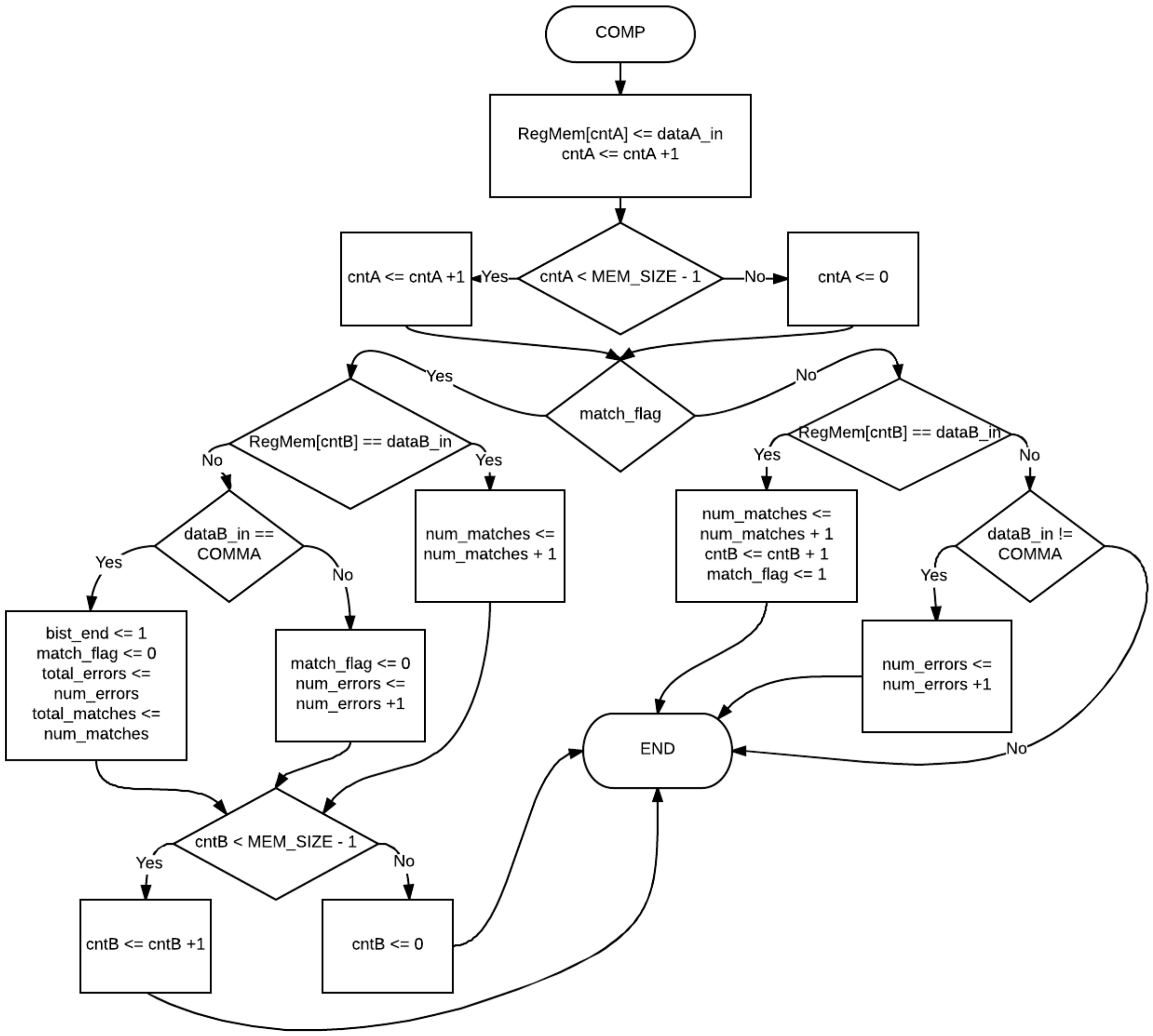


Figure 5.2. Data flow in the COMP state.

The main problem in the data flow is at the start when the first time at the COMP state, the data does not match or when there are two consecutive mismatches consequently, cntB will not increase and the same data will be compared. Another significant problem is the bist_end assignment due to the condition that a match between dataB_in and RegMem[cntB] must occur before the second comma, otherwise, there can be two commas captured and the process will continue.

To summarize both descriptions, the finite state machine, and the comparison process executed in the COMP state have issues. The finite state machine with the state transition triggering signals and the HDL code in the COMP state with the process of accomplishing the comparison. The state transition should focus on the start of the data transmission, at the moment when the receiver has the parallel data ready as well as when the data captured is a comma. On the other hand, the comparison should check if there is a match when the received data is available instead of checking if the previous comparison had a match to check the new data.

5.2.2 New comparator HDL code conception

After the analysis of the former comparator code, it was clear that:

- The overall process starts when the digital transmitter receives the first comma.
- When the digital transmitter receives a byte to transmit, it is stored in the memory block.
- The comparison step takes place when the digital receiver captures all the serial data and outputs a parallel data, therefore, this data should be the same as the stored in memory.
- Since the process starts at the digital transmitter and continues at the digital receiver, there is always data stored to be compared and there is a delay between the data to compare.

With all this information, the new idea for the comparator was to create two independent modules: one controlling the data to store and one reading the data from the memory and making the comparison. In both cases, a finite state machine was defined in order to track the data between two commas and perform the storing or comparison process. Although each module was designed to work independently, the integration requires that the data transmitted should be stored, next, the receiver should catch the same data and compare it against the stored data.

5.2.3 Logic synthesis's timing reports results

Table 5.2 summarizes the worst slack results for each modification made in the test modules. The first three rows are the comparison data module modifications while the remaining rows also include modifications in the store data module. Although the behavioral HDL finite state machine has a better slack (149ps less) than the one made with combinational logic, this last structure eased the addition of registers in order to accomplish a slack of -416ps. The last structure involved adding registers for the inputs in both submodules of the comparator module and the first stage of registers enclosing the combinational logic were removed.

Table 5.2. Worst slack path for each design modification.

Design modification	Clock (MHz)	Worst slack (ps)
Finite state machines with behavioral HDL.	625	-915
Finite state machine made with combinational logic	625	-1064
Registers between the combinational logic (Figure 3.5).	625	-416
Registers for the input data (Figure 3.6).	625	-10
Registers for the input data – Typical process.	156.25	2550
Registers for the input data – Worst-case process.	156.25	20

The last modification with registers for the input data reported the best result by running a 625MHz clock. At this point, the clock frequency was discussed because the clock first passes through a clock divider, which outputs four clock signals in different phases and with the fourth part of frequency. Therefore, a 156.25MHz clock feds all the circuits, thus the timing constraint for the test modules were changed and two corners were tested, a typical process (25°C, 1.2V) and a worst-case process (125°C, 1.08V). The timing reports in both cases were positive indicating timing constraint compliance.

5.2.4 Simulation

The registers implemented in the comparator module increase the timing performance, however, they introduce delays in the functionality. From section 4.2.2, the comparison data module takes four clock cycles to process the data and make a comparison while the store data module requires three clock cycles in order to save the data in the memory block. Therefore, before the data transmission of a new data, the store module takes three clock cycles in order to have a new data to compare stored in memory and the comparison data module takes four clock cycles after the receiver has deserialized the transmitted data. Nevertheless, these delays are acceptable since the transmitter has sent two encoded data when the receiver has caught the first one, hence, there are enough clock cycles to make the comparison. The worst-case scenario would be if the transmitter is faster than the receiver so the block memory overwrites the data to compare.

5.3 Signal driver

5.3.1 The former code

The 2016 signal driver HDL code had issues with its functionality that do not correspond to the test operating modes, they are listed below:

- When the test mode is enabled, the digital receiver output can be multiplexed at any time, therefore, it is independent of the operation mode.
- The *test_out* signal purpose is to have a way to check the analog receiver output and multiplex it with the *bist_end* signal when the BIST mode is enabled, nevertheless, *test_out* is also multiplexed with the *c_data_valid* signal, which is an alias for the *data_valid_pipe* signal, and it has its own output pin.
- Since a case statement block code is implemented, the default block code should be the mode 0, the normal function mode, however, instead of the analog receiver output, there is the *c_data_valid* signal and *bist_end signal*.
- Although the BIST modules, the comparator module, and the LFSR module, are used only in two operational modes, they are always working regardless of the current operating mode.

5.3.2 The improvements

The code was changed from a behavioral description to a structural description, hence, the module is composed of multiplexors and combinational logic designed for each multiplexor selector in order to output the right signal. Previous fixes, in other words, were:

- The digital receiver output is multiplexed only when the operating mode requires the BIST modules.
- The *test_out* signal is multiplexed only with the analog receiver output and with the *bist_end* signal.
- The code was changed to a structural description thus there is not the need for a default case.
- The LFSR and the comparator module are enabled in the top module only when the test mode is enabled, however, they must restart whenever they are used in order to reset the finite state machines.

5.4 Test modules top level

The test modules top level is in the test_modules.v file comprising the LFSR module, the comparator module, and the signal driver module. There are not many changes from the 2016 thesis except the new comparator HDL code and the renaming of the signals in the signal driver. The most important modifications are three new signals, explained in section A, that control the pseudo-random data generation and the start of the comparison.

Although these new signals enable the LFSR and the comparator module, it is important to notice that this will not reset the comparator module's finite state machine neither will it restart the pseudo-random data generation. Hence, the test has to start with a comma and end with a comma, if the test mode is turned off or the operating mode is changed, the LFSR and the comparator module will stay paused unless an asynchronous reset is triggered.

5.5 SerDes integration

5.5.1 Logic synthesis

5.5.1.1 Analog modules' LEF files

The main problem with the integration in the logic synthesis step was the generation of the LEF files. Initially, the analog modules designers produced LEF files including layers definitions, which were already defined in the technology LEF file. In particular, the layer CA was not defined so it was removed to avoid errors. However, this file edition resulted in problems with the analog transmitter LEF file. Therefore, the LEF file had to be created in a different way.

The LEF structure of the analog receiver was manually modified with the right format. As a result, the logic synthesis ran with only some warnings. Although this modification worked, the analog transmitter could not pass the logic synthesis due to the definition of the layer CA. Again, the LEF definition was not correct in order to be considered in the design.

To continue with the design flow, a new method to generate the LEF file was applied. First, the abstract generator tool is executed to create an abstract view, next, the LEF is generated selecting the abstract view created and indicating the option to not use the technology. Applying this method, the analog designers created LEF files, which passed the logic synthesis, though some warnings were indicated.

5.5.1.2 Timing report

Although the LEF files are linked to each analog module, there is no lib file attached to consider in the timing reports. Hence, the timing results could not be accurate since the analog modules do not have a timing library.

Another problem with the timing reports could be the paths marked as unconstrained, which can be the result of the global clock not being the one used for the clock divider. The global clock for all the SerDes modules is one of the clocks produced by the clock divider. This situation should be further analyzed to consider if it is correct.

5.5.2 LEC

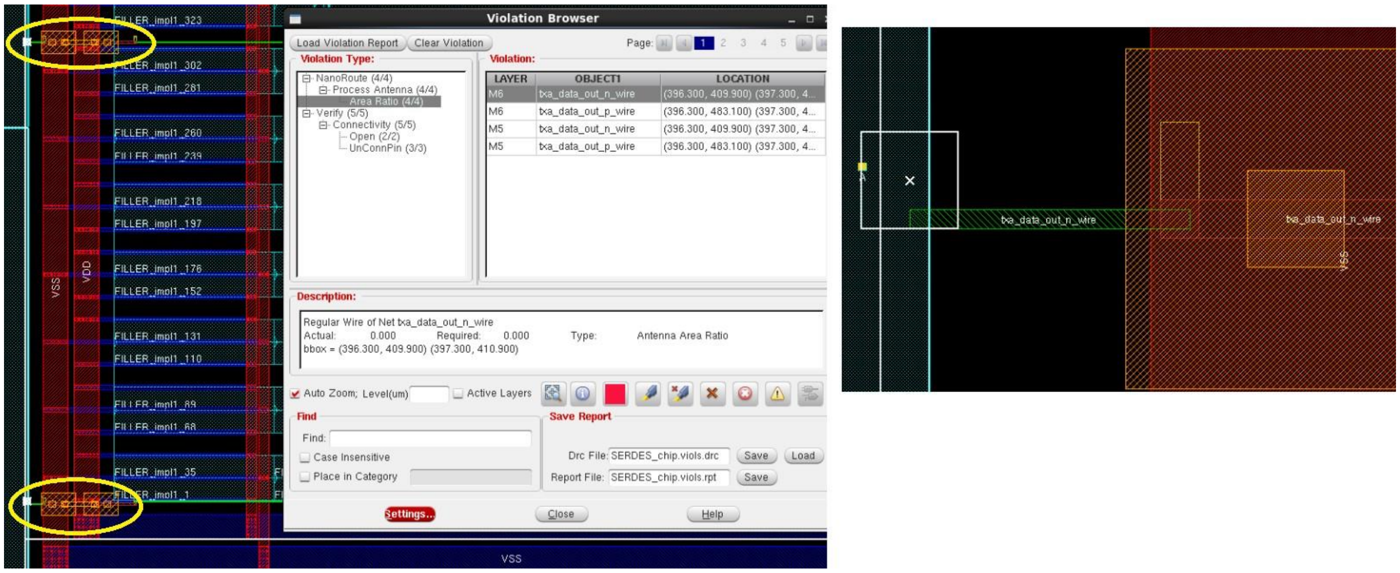
The LEC results indicate that the RTL to final gate-level netlist comparison presents differences and the reason could be an attribute that is set in the logic synthesis script (appendix B). The line "*set_attribute hdl_allow_inout_const_port_connect true*" allows to have inout type ports and it is used due to the analog modules port declaration that has one port of this type. Therefore, looking at Figure 4.27, this logic synthesis setting could generate two rx_outb ports in the analog receiver module, one as input and one as output. This issue is proposed as future research in order to ensure the logical equivalence of the RTL design with the gate level netlist.

5.5.3 Physical design

After the SerDes modules placement, the analog modules overlap with the power grid, so the floorplan should consider a special area for the analog modules and, after the routing, there should not be any route over the analog modules to prevent any metal overlap. Although analog designers modified their layout to have the input and output ports in the borders of the layout, as Figure 3.22 shows, the EDI makes routes over the module. Therefore, the way to prevent this routing is a pending issue to resolve.

Regarding the setup time's negative slack reported in section 4.7.3.5, although it is a timing violation and it is not tolerated in the semiconductor industry, it is acceptable in the academic degree. As an academic research, this worst-case result can be avoided applying in the chip a higher voltage or a lower frequency.

Another problem generated in the final layout is the connection between the transmitter outputs and their respective output pads. This issue is illustrated in Figure 5.3, this can be a tool error but, since the error is only from an analog module it could be a problem generated by the LEF file. In any case, it is a problem that requires further analysis.



Violation Type:

- NanoRoute (4/4)
  - Process Antenna (4/4)
    - Area Ratio (4/4)
- Verify (5/5)
  - Connectivity (5/5)
    - Open (2/2)
    - UnConnPin (3/3)

Violation:

LAYER	OBJECT1	LOCATION
M6	txa_data_out_n_wire	(396.300, 409.900) (397.300, 4...
M6	txa_data_out_p_wire	(396.300, 483.100) (397.300, 4...
M5	txa_data_out_n_wire	(396.300, 409.900) (397.300, 4...
M5	txa_data_out_p_wire	(396.300, 483.100) (397.300, 4...

Description:

```
Regular Wire of Net txa_data_out_n_wire
Actual:       0.000        Required:     0.000        Type:        Antenna Area Ratio
bbox = (396.300, 409.900) (397.300, 410.900)
```

[✓] Auto Zoom; Level(um) ☐ Active Layers

Find
Find:
☐ Case Insensitive
☐ Place in Category

Save Report
Drc File: SERDES_chip.viols.drc Save Load
Report File: SERDES_chip.viols.rpt Save

Figure 5.3. Error between the pads and the analog transmitter outputs.

All errors, in order to be resolved, require the cooperation between the physical designer and the analog designer in order to have constant feedback of the results in each side, the analog layout and the results of the LEF in the physical design.

5.5.4 Virtuoso importation

Regardless the errors presented in the EDI final layout, it was exported to Virtuoso with the idea of manually fixing the errors. After the importation, the absence of the analog modules was expected, hence, each layout was instantiated from the library created by the analog designer. Nevertheless, after each module was instantiated, it was found that placing the module in its position and connecting it to the appropriate route, requires a significant effort. This outcome is due to the lack of a schematic with each element linked to the respective component layout. An effort to integrate the SerDes into a schematic was done without results, it is highly recommended in further work to create it.

Conclusion

The test modules were designed to be manufactured in the 130nm transistor technology of IBM´s cmrf8sf Design Kit library. The top module comprises the LFSR module, the comparator module, and the signal driver module, which constitute a basic BIST structure. The BIST modules perform an offline test in order to check the SerDes functionality. This test is enabled through the signal driver module as one of the 8 test operating modes. Therefore, the test modules activate different test operating modes to test each SerDes functionality as well as implement BIST techniques.

The timing results of the logic synthesis for the typical and worst-case process have a positive worst slack when the frequency is 156.25MHz (the fourth part of 625Mhz). This result shows that the design is timing compliant. The results from the gate level simulation and the LEC ensure that the modules perform its function correctly. Consequently, the test modules are compliant in timing and functionality.

The design has some points for enhancement in the LFSR and comparator modules. The LFSR can be optimized implementing fewer registers to produce the pseudo-random patterns but first, it is necessary to ensure if the modification does not affect the standard LFSR performance. Although the new comparator module structure improved the former comparator timing, future enhancements could include a solution for the mismatches counter overflow, the implementation of an online BIST or change the structure for timing improvement. In general, the comparator module has been always the main issue in the test modules, hence, any improvement is a great contribution.

The SerDes integration required all the designers to work together in order to collaborate in the integration of the digital and analog modules. The digital designers created RTL code in the Verilog HDL and they verified the functionality with test benches applied in the simulation of each digital module. The analog designers made layouts based on schematics and they applied DRC and Layout Versus Schematic (LVS) checks in order to ensure a final layout was correctly done.

Conclusion

When the final chip layout has digital modules, standard cells are used, and the design flow is made almost automatically; nevertheless, analog modules integration requires LEF files created with each analog module layout.

The LEF file contains information about the layout so the method to generate the LEF file is very important. First, an abstract view should be generated and when the LEF is going to be generated the abstract view should be selected. Also, a black box module should be created based on the LEF ports in order to integrate the analog module to the design flow. However, it is recommended to further investigate how to create a lib file of an analog module in order to produce a more accurate timing report.

When the LEF file and the black box module for each analog module are added as part of the logic synthesis input files, the log and the schematic interpreting the gate level netlist indicate the integration of the analog modules. The relevance of the LEF file definition arises in the physical design due to the routing to the analog module ports. The objective is to prevent any routing over the analog module placing the ports in the borders of the module. This modification is made directly in the analog module layout, hence, a constant interaction between the analog designer and the physical designer is required in order to get a direct feedback between the routing results using the LEF file and the actual ports in the analog module layout.

Although the analog designers made a layout without any DRC and LVS error, the final issue is how the LEF file generated is handled in the physical design. Nevertheless, if the final layout errors are ignored they could be manually fixed after importing the design into the Virtuoso layout tool. The main limitation of correcting the errors manually, is the lack of a schematic linked to the final chip layout. This restriction complicates the identification of the design layout in order to fix the errors and it also makes it impossible to run an LVS check. As future work, the integration of the analog modules and the gate level netlist in a schematic should be researched in order to provide the possibility to run a LVS check and to fix errors manually in the final layout.

In conclusion, the integration requires concentrating efforts in generating a well-defined LEF file with the analog designers' collaboration. It is also suggested to research layout techniques on the integration of analog and digital modules since facts like the placement and routing of both kinds of modules also affect the circuit performance.

Appendices

A1. CLOCK DIVIDER RTL

```
/*************************************************************************
*Name:
*    clock_divider.v
* This block takes a clock reference and divides it by 8.
* It will generate 8 clocks at equidistant phases.
/************************************************************************/
module clock_divider(
      input rst,
      input clk,
      output reg [3 : 0] clks_out
);

      reg [1 : 0] div_by_4_q;
      reg [1 : 0] div_by_4_reg;
      wire clk_div_by_4;

      assign clk_div_by_4 = div_by_4_reg[1];

      always@(posedge clk or negedge rst) begin
            if(rst == 1'b0) begin
                  div_by_4_q <= 2'b0 ;
                  div_by_4_reg <= 2'b0 ;
            end else begin
                  div_by_4_q <= {div_by_4_q[0], ~div_by_4_q[1]} ;
                  div_by_4_reg <= div_by_4_q ;
            end
      end

      always@(posedge clk or negedge rst) begin
            if(rst == 1'b0) begin
                  clks_out <= 4'd0;
            end else begin
                  clks_out <= {clks_out[2 : 0], clk_div_by_4};
            end
      end

endmodule
```

A2. ANALOG RECEIVER RTL

```
/*****************************************************************************
*Name:
*    Analog_RX.v
*Description:
*    This block is a behavioural model for the analog
* front end of the Deserializer. The analog front end transforms
* a weak differential signal into a CMOS single ended signal.
*Editor:
*    Miguel Mihail Hoil Loria.
*Changes:
*    Ports named after the LEF file ports and declared as a black box.
/****************************************************************************/

module Analog_RX
(
    inout \subc! ,
//------------Inputs--------
      input rx_in,
      input rx_inb,
//------------Outputs--------
      output rx_out,
      output rx_outb
 );

//Uncomment for tests with verilog
//assign rxa_outb = rxa_inb;
//assign rxa_out = rxa_in;

endmodule
```

A3. ANALOG TRANSMITTER RTL

```
/***************************************************************************
*Name:
*    TXA_FINAL.v
*Description:
*    This module is the black box for the MACRO with the same name in the LEF
file.
*Version:
*    1.0
*Author:
*    Miguel Mihail Hoil Loria.
*Date:
*    07/11/2017
***************************************************************************/

module TXA_FINAL
(
    inout \sub! ,
//------------Inputs--------
    input DATA,
    input ZA,
    input ZB,
    input ZC,
    input ZD,
    input AMP_CTRL_1,
    input AMP_CTRL_2,
    input PRE_CTRL_1,
    input PRE_CTRL_2,
      input TEST_DATA,
      input DATA_SELECTOR,
//------------Outputs--------
      output TX,
      output TXBar,
    output DIRECT_DATA_TX,
    output DIRECT_DATA_TXBar
);

endmodule
```

A4. ANALOG TRANSMITTER WRAPPER RTL

```
/**************************************************************************
*Name:
*    Analog_TX.v
*Description:
*    This block is a wrapper for the analog transmitter.
*Editor:
*    Miguel Mihail Hoil Loria.
*Changes:
*    Instances the analog transmitter black box.
**************************************************************************/

module Analog_TX
(
//------------Inputs--------
      input txa_data_in,
      input [7 : 0] tx_config,
//------------Outputs--------
      output  txa_data_out_p,
      output txa_data_out_n
);

TXA_FINAL
txa_final_unit (
    .\sub! (1'b0) ,
//------------Inputs--------
    .DATA(txa_data_in),
    .ZA(tx_config[0]),
    .ZB(tx_config[1]),
    .ZC(tx_config[2]),
    .ZD(tx_config[3]),
    .AMP_CTRL_1(tx_config[4]),
    .AMP_CTRL_2(tx_config[5]),
    .PRE_CTRL_1(tx_config[6]),
    .PRE_CTRL_2(tx_config[7]),
    .TEST_DATA(1'b0),
    .DATA_SELECTOR(1'b0),
//------------Outputs--------
    .TX(txa_data_out_p),
    .TXBar(txa_data_out_n),
    .DIRECT_DATA_TX(),
    .DIRECT_DATA_TXBar()
);

endmodule
```

A5. LFSR RTL

```
/***************************************************************************
* Name:
*    LFSR.v
* Author:
*    Cesar Limones 2016
*Description:
*    This module is a linear feedback shift
*  register that will generate parallel data to be sent by
*  the transmitter when BIST mode is enabled. It has the
*  initial value fixed to a comma symbol.
/**************************************************************************/
module LFSR #(    parameter seed = 10'b11_1111_1100   )
(
//------------Inputs--------
      input rst, clk, lfsr_en,
//------------Outputs--------
      output [8 : 0] lfsr_parallel_out
);

//------------Internal Variables--------

wire linear_feedback, linear_feedback_i;
reg [8 : 0] lfsr_data_out_reg;
reg [9 : 0] lfsr_parallel_out_reg;

//-------------Code Starts Here-------

assign linear_feedback_i = lfsr_parallel_out_reg[5] ^
lfsr_parallel_out_reg[9];
assign linear_feedback = lfsr_parallel_out_reg[4] ^ linear_feedback_i;
assign lfsr_parallel_out = lfsr_data_out_reg;

always@(posedge clk or negedge rst) begin
      if(rst == 1'b0)   // active low reset
            lfsr_parallel_out_reg <= seed;
      else if (lfsr_en) begin
            lfsr_parallel_out_reg <= {lfsr_parallel_out_reg[8 : 0],
linear_feedback};
      end
end

always@(lfsr_parallel_out_reg) begin
 if (lfsr_parallel_out_reg[9 : 0] == seed)
      lfsr_data_out_reg = lfsr_parallel_out_reg[8 : 0];
 else //If there is no comma on the data send a zero on the k bit.
      lfsr_data_out_reg = {1'b0, lfsr_parallel_out_reg[7 : 0]};
end

endmodule
```

A6. MEMORY RTL

```
// Quartus II Verilog Template
// Simple Dual Port RAM with separate read/write addresses and
// single read/write clock

/*
*    30/05/17
*        -The read is now asynchronous
*/

module simple_dual_port_ram_single_clock
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=6)
(
      input [(DATA_WIDTH-1) : 0] data,
      input [(ADDR_WIDTH-1) : 0] read_addr, write_addr,
      input we, clk,
      output [(DATA_WIDTH-1) : 0] q
);

      // Declare the RAM variable
      reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];

      always@(posedge clk)
      begin
            // Write
            if (we)
                  ram[write_addr] <= data;

      end

      assign q = ram[read_addr];

endmodule
```

A7. STORE DATA MODULE RTL

```
/*****************************************************************************
*Name: dataA_save.v
*Author: Miguel Mihail Hoil Loria.
*Date: 31/05/2017
*Description:
*    This module controls memory recording for data transmitted.
*Inputs:
*    clk: The input for the global clock.
*    reset: The global reset signal.
*    dataA: the data being transmitted.
*    lfsr_en: indicates when the transmitter starts sending new data
*Outputs:
*    cntA: write memory pointer
*    we: enables memory writing
*Version: 1.0
*-Changes:
*     02/09/2017
*           -Finite state machine was changed for combinational logic.
*     20/10/2017
*           -Registers for input data were added.
*****************************************************************************/
module dataA_save #(
      parameter WIDTH = 9,
      parameter [8 : 0] DECODED_COMMA = 9'b1_1111_1100,
      parameter MEM_SIZE = 8
)
(
//Inputs
      input clk, reset, lfsr_en,
      input [ WIDTH -1 : 0 ]  dataA,
//Outputs
      output [$clog2(MEM_SIZE) - 1 : 0] cntA,
      output we
);

localparam CNTR_SIZE = $clog2(MEM_SIZE);
localparam IDLE = 2'd0;
localparam WAIT = 2'd1;
localparam SAVE = 2'd2;

reg [1 : 0] State;
wire lfsr_en_RegOut_wire;
wire [WIDTH -1 : 0] dataA_RegOut_wire;
wire [1 : 0] nxtSte_wire;
wire notSt1_St0_wire, enable_register_wire, dataA_is_dcdComma_wire;
wire [CNTR_SIZE - 1 : 0] cntA_wire, cntA_nxt_wire;
// - * - * - Code Starts - * - * -
// - - - Register data Input - - -
Registro #(1)
LFSRen_register (
      .clk(clk),
      .reset(reset),
      .enable(1'b1),
      .Data_Input(lfsr_en),
      .Data_Output(lfsr_en_RegOut_wire)
);
Registro #(WIDTH)
DataA_register (
      .clk(clk),
      .reset(reset),
      .enable(lfsr_en),
      .Data_Input(dataA),
      .Data_Output(dataA_RegOut_wire)
);
// - * - Sequential state assignment - * -
always@(posedge clk or negedge reset) begin
      if(reset == 1'b0)
            State <= IDLE;
      else
            State <= nxtSte_wire;
end
// - * - Combinational control signals assignment - * -
assign notSt1_St0_wire = (!State[1]) & State[0];
assign enable_register_wire = State[1] & (!State[0]);
assign nxtSte_wire[1] = notSt1_St0_wire & lfsr_en_RegOut_wire &
(!dataA_is_dcdComma_wire);
assign nxtSte_wire[0] = enable_register_wire | (notSt1_St0_wire &
(!lfsr_en_RegOut_wire))
 | ((!State[0]) & lfsr_en_RegOut_wire & dataA_is_dcdComma_wire);
// - * - Structural signals assignment - * -
Registro #(CNTR_SIZE)
CounterA_register
(
      .clk(clk),
      .reset(reset),
      .enable(enable_register_wire),
      .Data_Input(cntA_nxt_wire),
      .Data_Output(cntA_wire)
);
assign dataA_is_dcdComma_wire = dataA_RegOut_wire == DECODED_COMMA;
assign cntA_nxt_wire = cntA_wire + 1'b1;
assign cntA = cntA_wire;
assign we = enable_register_wire;

endmodule
```

A8. REGISTER WITH SYNCHRONOUS RESET RTL

```
/******************************************************************************
*Name:
*    RegistroSyncRst.v
*Description:
*    This module is the modified register from Moodle taught in class.
*Inputs:
*    clk: The input for the global clock.
*    reset: The global reset signal.
*    sync_reset: Reset the register on a clock transition.
*    enable: Disable o enable the register.
*sync_reset
*    Data_Input: The data input.
*Outputs:
*    Data_Output: The register data output.
*Version:
*    1.0
*Author:
*    Miguel Mihail Hoil Loria.
*Date:
*    28/02/2017
******************************************************************************/
module RegistroSyncRst #(
      parameter WORD_LENGTH = 8
)
(
      // Input Ports
      input clk,
      input reset,
      input sync_reset,
      input enable,
      input [WORD_LENGTH-1 : 0] Data_Input,
      // Output Ports
      output [WORD_LENGTH-1 : 0] Data_Output
);

reg [WORD_LENGTH-1 : 0] Data_reg;

always@(posedge clk or negedge reset) begin
      if(reset == 1'b0)
            Data_reg <= 0;
      else if(enable == 1'b1)
      begin
            if(sync_reset == 1'b1)
                  Data_reg <= 0;
            else
                  Data_reg <= Data_Input;
      end
end
assign Data_Output = Data_reg;

endmodule
```

A9. COMPARISON DATA MODULE RTL

```
/***************************************************************************
*Name: dataB_compare.v
*Description: This module makes the comparison reading the memory data and
comparing it with the received data.
*Inputs:
*    clk: The input for the global clock.
*    reset: The global reset signal.
*    data_valid_pipe: indicates when the digital receiver output is valid.
*    dataB: received data.
*    MemData: memory data read.
*Outputs:
*    cntB: read memory pointer
*    bist_end: indicates when the data comparison received between two commas
has ended.
*    num_errors: number of mismatches.
*Version: 1.0
*Author: Miguel Mihail Hoil Loria.
*Date: 31/05/2017
*-Changes:
*     02/09/2017
*           -Finite state machine was changed for combinational logic.
*     12/09/2017
*           -D flip-flops stages added
*     20/10/2017
*           -Registers for input data were added. The registers surrounding
*    the combinational logic were reduced to one stage.
***************************************************************************/
module dataB_compare #( parameter WIDTH = 9, parameter [8 : 0] DECODED_COMMA
= 9'b1_1111_1100, parameter MEM_SIZE = 8, parameter ERRORS_WIDTH = 6 )
(
      input clk, reset, data_valid_pipe,
      input [WIDTH-1 : 0]  dataB,
      input [WIDTH-2 : 0]  MemData,
      output [$clog2(MEM_SIZE) - 1 : 0] cntB,
      output bist_end,
      output [ERRORS_WIDTH-1 : 0 ] num_errors
);
// * * * * Variables * * * *
localparam CNTR_SIZE = $clog2(MEM_SIZE), IDLE = 2'd0, CLEAR = 2'd1;
localparam WAIT = 2'd2, COMPARE = 2'd3;
reg [1 : 0] State;
wire [1 : 0] nxtSte_wire;
wire [WIDTH-1 : 0]  dataB_RegOut_wire;
wire [WIDTH-2 : 0]  MemData_RegOut_wire;
wire data_valid_pipe_RegOut_wire, flag_end_wire, syncRst_registers_wire;
wire enable_cntrB_register_wire, S1_notS0_wire, data_is_comma_wire;
wire data_equal_wire;
wire [ERRORS_WIDTH-1 : 0 ] errors_next_wire, errors_wire;
wire [CNTR_SIZE - 1 : 0] cntB_wire, cntB_nxt_wire;
wire bist_end_wire, enable_errors_register_wire;
wire enable_BistEnd_register_wire;
wire enable_cntrB_register_pipeB_wire, syncRst_registers_pipeB_wire;
wire data_is_comma_pipeB_wire;
wire enable_BistEnd_register_pipeB_wire, enable_errors_register_pipeB_wire;
// - * - * - Code Starts - * - * -
// - - - Register input data - - -
Registro #(1) DataValidPipeIn_register (
      .clk(clk), .reset(reset), .enable(1'b1), .Data_Input(data_valid_pipe),
      .Data_Output(data_valid_pipe_RegOut_wire)
);
Registro #(WIDTH-1) MemData_register (
      .clk(clk),.reset(reset),.enable(data_valid_pipe),.Data_Input(MemData),
      .Data_Output(MemData_RegOut_wire)
);
Registro #(WIDTH) DataB_register (
      .clk(clk),.reset(reset),.enable(data_valid_pipe),.Data_Input(dataB),
      .Data_Output(dataB_RegOut_wire)
);
// - - - Equality comparators - - -
Comparator #(.WORD_LENGTH(WIDTH) ) DataIsComma_Comparator (
        .Data_A(dataB_RegOut_wire), .Data_B(DECODED_COMMA),
        .Comp_out(data_is_comma_wire)
);
Comparator #(.WORD_LENGTH(WIDTH-1)) DataIsEqual_Comparator (
        .Data_A(dataB_RegOut_wire[WIDTH-2 : 0]), Data_B(MemData_RegOut_wire),
        .Comp_out(data_equal_wire)
);
// - * - Combinational control signals assignment - * -
assign S1_notS0_wire = State[1] & (!State[0]);
assign flag_end_wire = State[1] & State[0];
assign syncRst_registers_wire = (!State[1]) & State[0];
assign enable_cntrB_register_wire = flag_end_wire & (!data_is_comma_wire);
assign nxtSte_wire[1] = syncRst_registers_wire | enable_cntrB_register_wire |
S1_notS0_wire;
assign nxtSte_wire[0] = ((!State[0]) & data_valid_pipe_RegOut_wire &
data_is_comma_wire)
      | (S1_notS0_wire & data_valid_pipe_RegOut_wire);
assign enable_errors_register_wire = enable_cntrB_register_wire &
!data_equal_wire | syncRst_registers_wire;
assign enable_BistEnd_register_wire = flag_end_wire & data_is_comma_wire |
syncRst_registers_wire;
// - - - Flops Stage B - - -
Registro #(1) EnableRegister_register (
      .clk(clk), .reset(reset),
      .enable(1'b1), .Data_Input(enable_cntrB_register_wire),
      .Data_Output(enable_cntrB_register_pipeB_wire)
);
Registro #(1) SyncRst_register (
      .clk(clk), .reset(reset), .enable(1'b1),
      .Data_Input(syncRst_registers_wire),
      .Data_Output(syncRst_registers_pipeB_wire)
);
Registro #(1) DataIsComma_PipeB_register (
      .clk(clk), .reset(reset), .enable(1'b1),
      .Data_Input(data_is_comma_wire),
      .Data_Output(data_is_comma_pipeB_wire)
);
Registro #(1) Enable_BistEnd_register (
      .clk(clk), .reset(reset), .enable(1'b1),
      .Data_Input(enable_BistEnd_register_wire),
      .Data_Output(enable_BistEnd_register_pipeB_wire)
);
Registro #(1) Enable_Errors_register (
      .clk(clk), .reset(reset), .enable(1'b1),
      .Data_Input(enable_errors_register_wire),
      .Data_Output(enable_errors_register_pipeB_wire)
);
// - * - Sequential state assignment - * -
always@(posedge clk or negedge reset) begin
      if(reset == 1'b0)
            State <= IDLE;
      else
            State <= nxtSte_wire;
end

// - * - Structural signals assignment - * -
Registro #(CNTR_SIZE) CounterB_register (
      .clk(clk), .reset(reset), .enable(enable_cntrB_register_pipeB_wire),
      .Data_Input(cntB_nxt_wire),
      .Data_Output(cntB_wire)
);
RegistroSyncRst #(6) Errors_register (
      .clk(clk), .reset(reset), .sync_reset(syncRst_registers_pipeB_wire),
      .enable(enable_errors_register_pipeB_wire),
      .Data_Input(errors_next_wire),
      .Data_Output(errors_wire)
);
RegistroSyncRst #(1) Bist_end_register (
      .clk(clk), .reset(reset), .sync_reset(syncRst_registers_pipeB_wire),
      .enable(enable_BistEnd_register_pipeB_wire),
      .Data_Input(data_is_comma_pipeB_wire),
      .Data_Output(bist_end_wire)
);
//assign data_is_comma_wire = dataB_RegOut_wire == DECODED_COMMA;
//assign data_equal_wire = dataB_RegOut_wire[WIDTH-2 : 0] ==
MemData_RegOut_wire;
assign errors_next_wire = errors_wire + 1'b1;
assign cntB_nxt_wire = cntB_wire + 1'b1;
// - - Outputs assignment - -
assign cntB = cntB_wire;
assign num_errors = errors_wire;
assign bist_end = bist_end_wire;
endmodule
```

A10. COMPARATOR TOP MODULE RTL

```
/***********************************************************************
*Name: ComparatorP.v
*Description: This module compares the data transmitted with the received
one. Is composed of 3 submodules, one controls the transmitted data
recording, other module makes reads the memory data and compares it with the
received data. The last submodule is a memory to saving all the transmitted
data due to the difference in time with the transmission and recepcion.
*Inputs:
*    clk: The input for the global clock.
*    rst: The global reset signal.
*    dataA_in: the data being transmitted.
*    dataB_in: received data.
*    lfsr_en: indicates when the transmitter starts sending new data.
*    data_valid_pipe: indicates when the digital receiver output is valid.
*Outputs:
*    bist_end: indicates when the data comparison received between two commas
has ended.
*    num_errors: number of mismatches.
*Version: 1.0
*Author: Miguel Mihail Hoil Loria.
*Date: 08/10/2017
***********************************************************************/
module ComparatorP #(
      parameter WIDTH = 9,
      parameter MEM_SIZE = 8,
      parameter [8 : 0] COMMA = 9'b1_1111_1100,
      parameter ERRORS_WIDTH = 6
)
(
      //------------Inputs--------
      input clk,
      input rst,
      input [ WIDTH -1 : 0 ] dataA_in,
      input [ WIDTH -1 : 0 ] dataB_in,
      input lfsr_en,
      input data_valid_pipe,
      //------------Outputs--------
      output bist_end,
      output [ERRORS_WIDTH-1 : 0 ] num_errors
);

localparam CNTR_SIZE = $clog2(MEM_SIZE);

wire [CNTR_SIZE - 1 : 0] cntA_wire, cntB_wire;
wire [ERRORS_WIDTH-1 : 0 ] num_errors_wire;
wire bist_end_wire, write_enable_wire;
wire [WIDTH-2 : 0] MemData_wire;   //0 -> WIDTH-2 = 8 data bits without Ko

dataA_save
#(
      .WIDTH(WIDTH),
      .DECODED_COMMA(COMMA),
      .MEM_SIZE(MEM_SIZE)
)
dataA_Unit (
//Inputs
      .clk(clk),
      .reset(rst),
      .lfsr_en(lfsr_en),
      .dataA(dataA_in),
//Outputs
      .cntA(cntA_wire),
      .we(write_enable_wire)
);
simple_dual_port_ram_single_clock #(
      .DATA_WIDTH(WIDTH-1),
      .ADDR_WIDTH(CNTR_SIZE)
)
MemoryData_Unit (
//INPUTS
      .clk(clk),
      .data(dataA_in[WIDTH-2 : 0]),
      .write_addr(cntA_wire),
      .we(write_enable_wire),
      .read_addr(cntB_wire),
//OUTPUTS
      .q(MemData_wire)
);
dataB_compare #(
      .WIDTH(WIDTH),
      .DECODED_COMMA(COMMA),
      .MEM_SIZE(MEM_SIZE),
      .ERRORS_WIDTH(ERRORS_WIDTH)
)
dataB_Unit (
//Inputs
      .clk(clk),
      .reset(rst),
      .data_valid_pipe(data_valid_pipe),
      .dataB(dataB_in),
      .MemData(MemData_wire),

//Outputs
      .cntB(cntB_wire),
      .bist_end(bist_end_wire),
      .num_errors(num_errors_wire)
);

assign bist_end = bist_end_wire;
assign num_errors = num_errors_wire;

endmodule
```

A11. SIGNAL DRIVER RTL

```
///////////////////////////////////////////////////////////////
// Module: signal_driver
// Author: Cesar Limones 2016
// Description: This module functionality is to gather all the signals from
the other modules and drive them to their respective counterpart depending on
the test mode that is being executed.
///////////////////////////////////////////////////////////////
/**********************************************************************
*Name: SignalDriverS2.v
*Editor:  Miguel Mihail Hoil Loria.
*Changes:
*24/05/2017
*-Output port names changed:
*     rxd_in_i > rxd_input_muxout
*     txd_data_in_i > txd_input_muxout
*     txa_data_in_i > txa_input_muxout
*     digital_out > rxd_output_muxout
*-test_out is always rxa_out except for the BIST modes.
*-The default mode and the functional mode are the same.
*-The input c_data_valid is not required due to the output pin that works the
same.
*09/06/2017
*-behavioral description changed to structural description
*05/11/2017
*-bist_end and rxa_out are selected with two multiplexors, the first chooses
between them and the two
*num_errors MSBs, this output is the input for the multiplexor that selects
the digital receiver output.
*In both cases the selectors do not have any combinational logic.
**********************************************************************/
module SignalDriverS2 #( parameter WIDTH = 9)
(
//------------Inputs--------
      input [2 : 0] mode,           //config_in[2 : 0]
      input test_in,                //config_in[3]
      input errors_en,              //config_in[4]
      input test_out_muxsel,        //config_in[5]
/* RXA OUT  */
      input rxa_out,
/* RXD OUT  */
      input [WIDTH - 1 : 0] rxd_data_out,
      input code_err,
      input disp_err,
      input dispout,
/*          TXD   */
      input txd_data_out,
      input [ WIDTH - 1 : 0 ] txd_data_in,
/*    LFSR OUT    */
      input [ WIDTH - 1 : 0 ] lfsr_parallel_out,
/*          Comparator  OUT   */
      input bist_end,
      input  [5 : 0] num_errors,
//------------Outputs--------
output rxd_input_muxout, txa_input_muxout
output [WIDTH - 1 : 0] txd_input_muxout,
output [WIDTH - 1 : 0] rxd_output_muxout
);
//------------Internal Variables--------
wire rxd_input_muxout_wire, txa_input_muxout_wire;
wire [WIDTH-1 : 0] txd_input_muxout_wire;
wire [1 : 0] test_out_wire;
wire [WIDTH-1 : 0] rxd_output_muxout_wire;
wire rxd_input_bitZero_muxsel_wire, rxd_input_bitOne_muxsel_wire;
wire rxd_muxout_A_wire, txd_input_bitZero_muxsel_wire;
wire txd_input_bitOne_muxsel_wire, txa_input_muxsel_wire;
wire [WIDTH-1 : 0] txd_muxout_A_wire;
wire [WIDTH-1 : 0] rxd_muxinput_wire;
wire mZero_inv, mOne_Xor_mTwo, mZeroInv_And_mTwo;
//-------------Code Starts Here-------
/* * * * rxd_input_mux * * * */
Mux2to1 #( .WORD_LENGTH(1) ) RXD_input_A_MUX (
//Input Ports
      .Selector(rxd_input_bitZero_muxsel_wire),
      .Data_0(rxa_out), .Data_1(txd_data_out),
//Output Ports
      .Mux_Output(rxd_muxout_A_wire) );
Mux2to1 #( .WORD_LENGTH(1) ) RXD_input_B_MUX (
//Input Ports
      .Selector(rxd_input_bitOne_muxsel_wire),
      .Data_0(rxd_muxout_A_wire), .Data_1(test_in),
//Output Ports
      .Mux_Output(rxd_input_muxout_wire) );
/* * * * txd_input_mux * * * */
Mux2to1 #( .WORD_LENGTH(WIDTH) ) TXD_input_A_MUX (
//Input Ports
      .Selector(txd_input_bitZero_muxsel_wire),
      .Data_0(txd_data_in), .Data_1(rxd_data_out),
//Output Ports
      .Mux_Output(txd_muxout_A_wire) );
Mux2to1 #( .WORD_LENGTH(WIDTH) ) TXD_input_B_MUX (
//Input Ports
      .Selector(txd_input_bitOne_muxsel_wire),
      .Data_0(txd_muxout_A_wire), .Data_1(lfsr_parallel_out),
//Output Ports
      .Mux_Output(txd_input_muxout_wire) );
/* * * * txa_input_mux * * * */
Mux2to1 #( .WORD_LENGTH(1) ) TXA_input_MUX (
//Input Ports
      .Selector(txa_input_muxsel_wire),
      .Data_0(txd_data_out), .Data_1(rxa_out),
//Output Ports
      .Mux_Output(txa_input_muxout_wire) );
/* * * * test_out_mux * * * */
Mux2to1 #( .WORD_LENGTH(2) ) Test_out_MUX (
//Input Ports
      .Selector(errors_en),
      .Data_0({rxa_out, bist_end}), .Data_1(num_errors[5 : 4]),
//Output Ports
```

```
      .Mux_Output(test_out_wire) );
/* * * * rxd_output_mux * * * */
Mux2to1 #( .WORD_LENGTH(WIDTH) ) RXD_output_MUX (
//Input Ports
      .Selector(test_out_muxsel),
      .Data_0(rxd_data_out), .Data_1(rxd_muxinput_wire),
//Output Ports
      .Mux_Output(rxd_output_muxout_wire) );
//-------------Wire assignations-------
assign rxd_muxinput_wire = {disp_err, code_err, dispout, test_out_wire,
num_errors[3 : 0]};
assign mZero_inv = !mode[0];
assign mOne_Xor_mTwo = mode[1] ^ mode[2];
assign mZeroInv_And_mTwo = mZero_inv & mode[2];
//    rxd input muxtiplexor 3-1 selectors
assign rxd_input_bitZero_muxsel_wire = mZero_inv & mOne_Xor_mTwo;
assign rxd_input_bitOne_muxsel_wire = mode[0] & mOne_Xor_mTwo;
//    txd input muxtiplexor 3-1 selectors
assign txd_input_bitZero_muxsel_wire = mode[0] & !mode[1];
assign txd_input_bitOne_muxsel_wire = mZeroInv_And_mTwo;
//    txa input muxtiplexor 2-1 selector
assign txa_input_muxsel_wire = &mode;
// Assign output ports
assign rxd_output_muxout = rxd_output_muxout_wire;
assign rxd_input_muxout = rxd_input_muxout_wire;
assign txd_input_muxout = txd_input_muxout_wire;
assign txa_input_muxout = txa_input_muxout_wire;
endmodule
```

A12. TEST MODULES RTL

```
/////////////////////////////////////////////////////////////////
// Module: test_modules
// Author: Cesar Limones 2016
// Description: This block contains all the testing modules
// for a SerDes. It contains the LFSR, a comparator
// and a signal driver.
/////////////////////////////////////////////////////////////////
/****************************************************************************
*Name: test_modules_v2.v
*Editor: Miguel Mihail Hoil Loria.
*>Inputs:
*    ser_clk: The input for the global clock.
*    rst: The global reset signal.
*       test_en: enables the test operating modes
*       lfsr_en: enables the lfsr, it should be the frame output from digital
transmitter
*       config_in: analog transmitter configuration, in test mode the 5 less
significant bits
*     are used for test configuration
*       rxa_out: analog receiver positive output (RXA OUT)
*       rxa_out_n: analog receiver negative output (RXA OUT)
*       rxd_data_out: digital receiver output (RXD OUT)
*       c_data_valid: indicates when the digital receiver output is valid (RXD
OUT)
*       code_err: indicates an invalid 10 bit data received (RXD OUT)
*    disp_err: indicates a running disparity error (RXD OUT)
*       dispout: running disparity (RXD OUT)
*       txd_data_in: digital transmitter input
*       txd_data_out: digital transmitter output
*<Outputs:
*       rxd_input_muxout: digital receiver real input
*       txd_input_muxout: digital transmitter real input
*       txa_input_muxout: analog transmitter real input
*       rxd_output_muxout: SerDes output, can be the digital receiver output
or the comparator errors
*       test_out: SerDes output for the analog receiver output (RXA OUT)
*       tx_config: final analog transmitter configuration
*-Changes:
*     24/05/2017
*          -Output port names changed:
*                rxd_in_i > rxd_input_muxout
*                txd_data_in_i > txd_input_muxout
*                txa_data_in_i > txa_input_muxout
*                digital_out > rxd_output_muxout
*     08/10/2017
*          -Two registers were added to delay the LFSR data generation
allowing enough time for register
*     the actual output from the LFSR
*     05/11/2017
*          -test_out output removed because now is multiplexed in the signal
driver in order to save one pin.
****************************************************************************/
```

```
module test_modules_v2 #( parameter WIDTH = 9, parameter DECODED_COMMA =
9'h1FC )
(
//------------Inputs--------
       input rst, input ser_clk, input test_en, input lfsr_en,
       input [ 7 : 0 ] config_in,
/* RXA OUT  */
       input  rxa_out, input  rxa_out_n,
/* RXD OUT  */
       input [WIDTH - 1 : 0] rxd_data_out,
       input c_data_valid, input code_err, input disp_err, input dispout,
/*          TXD   */
       input [ WIDTH - 1 : 0 ] txd_data_in,
       input txd_data_out,
//------------Outputs--------
       output rxd_input_muxout, txa_input_muxout,
       output [ WIDTH - 1 : 0 ] txd_input_muxout,
       output [ WIDTH - 1 : 0 ] rxd_output_muxout, //RXD OUTPUT
       output [ 7 : 0 ] tx_config );
//------------Internal Variables--------
wire bist_on_wire, txd_frame_start_wire, txd_frame_start_regOutA_wire;
wire txd_frame_start_regOutB_wire, rxd_data_ready_wire, bist_end_wire;
wire [ WIDTH - 1 : 0 ] lfsr_parallel_out_wire;
wire [ 5 : 0 ] num_errors_wire;
reg [2 : 0] mode_reg;
reg test_in_reg, errors_en_reg, test_out_sel_reg;
reg [7 : 0] tx_config_reg;
 //-------------Code Starts Here-------
always@(test_en or config_in) begin
      if (test_en) begin
            mode_reg = config_in[2:0];
            test_in_reg = config_in[3];
            errors_en_reg = config_in[4];
            test_out_sel_reg = config_in[5];
            tx_config_reg = 8'b0;
      end
      else begin
            mode_reg = 3'b000;
            test_in_reg = 1'b0;
            errors_en_reg = 1'b0;
            test_out_sel_reg = 1'b0;
            tx_config_reg = config_in;
      end
end
//------------- Module instantiation -------
//Buffers?
Registro #(1) buffer_rxa_out_p_register (
      .clk(ser_clk), .reset(rst), .enable(1'b1), .Data_Input(rxa_out),
      .Data_Output() );
Registro #(1) buffer_rxa_out_n_register (
      .clk(ser_clk), .reset(rst), .enable(1'b1), .Data_Input(rxa_out_n),
      .Data_Output() );
//Signal driver
SignalDriverS2 #(9)
signal_driver_unit (
// INPUTS
      .mode(mode_reg), .test_in(test_in_reg), .errors_en(errors_en_reg),
      .test_out_muxsel(test_out_sel_reg), .rxa_out(rxa_out),
      .rxd_data_out(rxd_data_out), .code_err(code_err), .disp_err(disp_err),
      .dispout(dispout), .txd_data_in(txd_data_in),
      .txd_data_out(txd_data_out), .lfsr_parallel_out(lfsr_parallel_out_wire),
      .bist_end(bist_end_wire), .num_errors(num_errors_wire),
// OUTPUTS
      .rxd_input_muxout(rxd_input_muxout),.txd_input_muxout(txd_input_muxout),
      .txa_input_muxout(txa_input_muxout),
      .rxd_output_muxout(rxd_output_muxout) );
// Register for delay the data generation and allow current data recording.
Registro #(1) txd_frame_start_registerA (
      .clk(ser_clk), .reset(rst), .enable(bist_on_wire),
      .Data_Input(txd_frame_start_wire),
      .Data_Output(txd_frame_start_regOutA_wire) );
Registro #(1) txd_frame_start_registerB (
      .clk(ser_clk), .reset(rst), .enable(bist_on_wire),
      .Data_Input(txd_frame_start_regOutA_wire),
      .Data_Output(txd_frame_start_regOutB_wire) );
//Pseudo Alleatory pattern Generator LFSR
LFSR #( .seed({1'b1,DECODED_COMMA}) )
lfsr_unit (
      .rst(rst), .clk(ser_clk), .lfsr_en(txd_frame_start_regOutB_wire),
      .lfsr_parallel_out(lfsr_parallel_out_wire) );
//Comparator
ComparatorP #( .WIDTH(WIDTH), .COMMA(DECODED_COMMA) )
comparator_unit (
//    INPUTS
      .clk(ser_clk),
      .rst(rst),
      .dataA_in(txd_input_muxout),
      .dataB_in(rxd_data_out),
      .lfsr_en(txd_frame_start_wire),
      .data_valid_pipe(rxd_data_ready_wire),
// OUTPUTS
      .bist_end(bist_end_wire),
      .num_errors(num_errors_wire)
);

// The next two signals are important for recording and comparison function
in the comparator
assign txd_frame_start_wire = lfsr_en & bist_on_wire;      //This signal
permits check and store the incoming txd data.
assign rxd_data_ready_wire = c_data_valid & bist_on_wire;  //This signal
allows the comparation between the store data and actual rxd data.
assign bist_on_wire = test_en & mode_reg[2] & !mode_reg[0];
assign tx_config = tx_config_reg;

endmodule
```

A13. FLIP FLOP T RTL

```
/*****************************************************************************
*Name:
*    FlipFlopT.v
*Description:
*    This module is the behavioral description for a T flip flop.
*Inputs:
*    clk: The input for the global clock.
*    reset: The global reset signal.
*    enable: Disable o enable the register.
*    Data_Input: The data input.
*Outputs:
*    Data_Output: The register data output.
*Version:
*    1.0
*Author:
*    Miguel Mihail Hoil Loria.
*Date:
*    18/02/2017
*****************************************************************************/

module FlipFlopT
#(
      parameter WORD_LENGTH = 8
)

(
      // Input Ports
      input clk,
      input reset,
      input enable,
      input [WORD_LENGTH-1 : 0] Data_Input,
      // Output Ports
      output [WORD_LENGTH-1 : 0] Data_Output
);

reg [WORD_LENGTH-1 : 0] Data_reg;

always@(posedge clk or negedge reset) begin
      if(reset == 1'b0)
            Data_reg <= 0;
      else if(enable == 1'b1)
            Data_reg <= Data_Input ^ Data_Output; //El xor actua como inversor
si una entrada es uno
end

assign Data_Output = Data_reg;

endmodule
```

A14. DESERIALIZER RTL

```
// deserializer.v
// This block samples an asynchronous signal. and transforms it into
// a source synchronous parallel bus.
// It uses up-sampling data recovery and a shift register to transform the
// serial input stream into a parallel bus.
// This block does not decode or encode any of the inputs.
/***************************************************************************
*Editor:
*    Miguel Mihail Hoil Loria.
/**************************************************************************/
module deserializer #( parameter ENCODED_COMMA = 10'h07C )
(rst, clks_in, a_rx, c_parallel_out, clk_out, disparity_d, disparity_q,
c_data_valid);
  input rst, a_rx, disparity_d;
  input [3:0] clks_in;
  output reg [9:0] c_parallel_out;
  output reg clk_out;
  output reg disparity_q;
  output c_data_valid;
  wire clk, c_rx, comma_detected, restart_counter_wire;
  reg comma_detected_reg;
  reg [9:0] shift_2reg;
  reg [3:0] cycle_count;
  wire samp_test, comma_detected_regOut_wire, c_data_valid_wire;
// Use the phase 0 clk as the system clk
  assign clk = clks_in[0];
CDR CDR1(
      .rst(rst), .clks_in(clks_in), .a_rx(a_rx), .samp_test(samp_test) );
// Use a flip flop to remember the running disparity in the decoder
always @(posedge clk or negedge rst) begin
    if (rst == 1'b0)
        disparity_q <= 1'b0;
    else
        disparity_q <= disparity_d;
  end
  // 10 bit shift register
  always@(posedge clk or negedge rst) begin
    if (rst == 1'b0) begin
        shift_2reg <= 10'h000;
    end
    /*This code clears the shift registers except the MSB which registers the
sampled bit. Thus, there won't be an incorrect check of the new data, this is
very important to catch the comma value.*/
    else if(comma_detected | restart_counter_wire) begin
            shift_2reg <= {samp_test, 9'b0};
    end
    else begin
            shift_2reg <= {samp_test, shift_2reg[9:1]};
    end
  end
  // Look for comma symbol
  assign comma_detected = shift_2reg == ENCODED_COMMA || shift_2reg ==
~ENCODED_COMMA;
```

```
// Use a 10 cycle counter to generate a data_valid signal.
// rst the counter if a special sync character, such as a comma is identified
  always @(posedge clk or negedge rst) begin
      if (rst == 1'b0) begin
          cycle_count <= 4'd9;
          clk_out <= 1'b0;
      end
      else begin
                        if(comma_detected_regOut_wire) begin //starts the
count when a comma is catched (ENABLE the counter)
                              if(restart_counter_wire)
                                     cycle_count <= 4'd9;// Restart
                              else
                                     cycle_count <= cycle_count - 1'b1;//Count
down
                end
/* * * clk_out duty cycle * * */
        // Assert clk_out one cycle after data_valid
        // De-assert clk_out in cycle 5
        if(cycle_count == 4'd5)
            clk_out <= 1'b0;
        else if (cycle_count == 4'd0 && (!comma_detected))
            clk_out <= 1'b1;
      end
  end
/* * * Data is valid when the countdown expires. * * */
      Registro #(1) C_data_valid_register (
            .clk(clk), .reset(rst), .enable(1'b1),
            .Data_Input(comma_detected | restart_counter_wire),
            .Data_Output(c_data_valid_wire) );
/* * * * comma detected register * * * */
/*With a flip-flop T is possible to assert the register with one comma and
with a second one revert it to 0*/
      FlipFlopT #(1) CommaDetected_registerFFT (
            .clk(clk), .reset(rst), .enable(comma_detected),
            .Data_Input(comma_detected),
            .Data_Output(comma_detected_regOut_wire) );
/* * * * data out register * * * */
  always @(posedge clk or negedge rst) begin
      if(rst == 1'b0) begin
          c_parallel_out <= 10'd0;
      end
      else begin
          if (comma_detected | restart_counter_wire) begin
              c_parallel_out <= shift_2reg;
                end
          end
  end
assign restart_counter_wire = (cycle_count == 4'd0);
assign c_data_valid = c_data_valid_wire;
endmodule
```

A15. DIGITAL RECEIVER RTL

```
module digitalRX
#(
      parameter ENCODED_COMMA = 10'h07C
)
(
      input       rst,
      input       a_rx,
      input       disparity_d,
      output      [7:0] dataout,
      output      dispout,
      output      code_err,
      output      disp_err,
      output      data_valid_pipe,
      output      clk_4f,
      output      ko,

      input [3:0]clks_in
);

      wire dispin ;
      wire [9:0] c_parallel_out;
      wire clk_out;
      wire c_data_valid;
      wire disparity_q;
      wire a_rx_buff;
      wire a_rx_diff_buff;

      assign clk_4f = clks_in[0];

      deserializer #(
            .ENCODED_COMMA(ENCODED_COMMA)
      )
      deserializer1(rst, clks_in, a_rx, c_parallel_out, clk_out, disparity_d,
disparity_q, c_data_valid);

      decodePipe decode1(rst, clks_in[0], c_data_valid, c_parallel_out,
disparity_q, dataout, data_valid_pipe, dispout, disp_err, code_err, ko);

endmodule
```

A16. SERDES CORE MODULE RTL

```
//////////////////////////////////////////////////////////////////
// Module: SerDes
// Description: This block is the top level wrapper of the
// SerDes system, it contains the digital and analog transmitter,
// the digital and analog receiver, the test modules and the
// clock divider.
//////////////////////////////////////////////////////////////////
module SERDESv2 #(
parameter WIDTH = 9,
parameter DECODED_COMMA = 9'h1FC,
parameter ENCODED_COMMA = 10'h07C )
(
//------------Inputs--------
      input reset, //reset
      input clk, //main system clock
      input rxa_in_p,//external Positive input for Analog Receiver
      input rxa_in_n, //external Negative input for Analog Receiver
      input [7 : 0] config_in,      //Analog transmitter configuration
      input [WIDTH - 1 : 0] txd_data_in, //external digital transmission input
      input test_en,
//------------Outputs--------
      output [WIDTH - 1 : 0] digital_out,
      output txa_data_out_p,
      output txa_data_out_n,
      output txd_data_out,
      output tx_frame_start,
      output c_data_valid
 );

//------------Internal Variables--------
wire rxa_out_p_wire;
wire rxa_out_n_wire;
wire txa_input_muxout_wire;
wire [7 : 0] txa_config_wire;
wire txa_data_out_p_wire;
wire txa_data_out_n_wire;
wire [3 : 0] clks_in_wire;
wire [WIDTH - 1 : 0] txd_input_muxout_wire;
wire txd_data_out_wire;
wire frame_wire, frame_regOut_wire;
wire rxd_input_muxout_wire;
wire [7:0] rxd_data_out_wire;
wire Ko_wire;
wire code_err_wire;
wire disp_err_wire;
wire dispout_wire;
wire data_valid_wire;
wire [WIDTH - 1 : 0] rxd_output_muxout_wire;


//------------Module instantiation --------
//  Analog modules   //
```

```
//Analog receiver
Analog_RX analog_receiver_unit(
    .\subc! (1'b0) ,
//    INPUTS
      .rx_in(rxa_in_p), .rx_inb(rxa_in_n),
//    OUTPUTS
      .rx_out(rxa_out_p_wire), .rx_outb(rxa_out_n_wire) );
//Analog Transmitter
Analog_TX analog_transmitter_unit(
//    INPUTS
      .txa_data_in(txa_input_muxout_wire), .tx_config(txa_config_wire),
//    OUTPUTS
      .txa_data_out_p(txa_data_out_p_wire),
      .txa_data_out_n(txa_data_out_n_wire) );
//  Test modules  //
test_modules_v2 #( .DECODED_COMMA(DECODED_COMMA) )
test_modules_unit (
//------------INPUTS--------
       .rst(reset),
       .ser_clk(clks_in_wire[0]),
       .test_en(test_en),
       .lfsr_en(frame_regOut_wire),
       .config_in(config_in),
/* RXA OUT  */
       .rxa_out(rxa_out_p_wire),
       .rxa_out_n(rxa_out_n_wire),
/* RXD OUT  */
       .rxd_data_out({Ko_wire, rxd_data_out_wire}),
       .c_data_valid(data_valid_wire),
       .code_err(code_err_wire),
       .disp_err(disp_err_wire),
       .dispout(dispout_wire),
/*          TXD   */
       //-INPUT-
       .txd_data_in(txd_data_in),
       //-OUTPUT-
       .txd_data_out(txd_data_out_wire),
//------------OUTPUTS--------
       .rxd_input_muxout(rxd_input_muxout_wire),
       .txd_input_muxout(txd_input_muxout_wire),
       .txa_input_muxout(txa_input_muxout_wire),
       .rxd_output_muxout(rxd_output_muxout_wire),
       .tx_config(txa_config_wire)
);
//     Digital modules  //
clock_divider clock_divider_unit(reset, clk, clks_in_wire);
//Digital receiver
digitalRX #( .ENCODED_COMMA(ENCODED_COMMA) ) DigitalRX_Unit (
//Inputs
   .rst(reset), .a_rx(rxd_input_muxout_wire), .disparity_d(1'b0),
//Outputs
   .dataout(rxd_data_out_wire), .dispout(dispout_wire),
   .code_err(code_err_wire), .disp_err(disp_err_wire),
   .data_valid_pipe(data_valid_wire), .clk_4f(), .ko(Ko_wire),
   .clks_in(clks_in_wire) );
```

```
//Digital Transmitter
Serializer DigitalTX_Unit (
//Inputs
  .clk(clks_in_wire[0]), .rst(reset), .data_in(txd_input_muxout_wire),
 //Outputs
  .data_out(txd_data_out_wire), .frame(frame_wire) );
// Signal frame_wire is early
Registro #(1) txd_frame_register (
      .clk(clks_in_wire[0]), .reset(reset), .enable(1'b1),
      .Data_Input(frame_wire),
      .Data_Output(frame_regOut_wire) );

assign txa_data_out_p = txa_data_out_p_wire;
assign txa_data_out_n = txa_data_out_n_wire;
assign digital_out = rxd_output_muxout_wire;
assign c_data_valid = data_valid_wire;
assign txd_data_out = txd_data_out_wire;
assign tx_frame_start = frame_regOut_wire;

endmodule
```

A17. SERDES CORE WITH INPUT/OUTPUT PADS RTL

```
/****************************************************************************
*Description:
*    This module instatiates the SerDes core and connect the ports to their
*  correspondig pad.
*Version:
*    1.0
*Author:
*    Miguel Mihail Hoil Loria.
*Date:
*    07/11/2017
****************************************************************************/
module SERDES_chip
(
//------------Inputs--------
      input VDD, VSS,
      input DVDD, DVSS,
      input rst,
      input clk,

      input rxa_in_p,
      input rxa_in_n,

      input [7 : 0] config_in,
      input [8 : 0] txd_data_in,

      input test_en,

//------------Outputs--------
      output [8 : 0] digital_out,

      output txa_data_out_p,
      output txa_data_out_n,

      output txd_data_out,
      output tx_frame_start,
      output c_data_valid
);

 wire VDD_wire, VSS_wire, DVDD_wire, DVSS_wire;

 wire rst_wire, clk_wire;
 wire rxa_in_p_wire, rxa_in_n_wire;
 wire txa_data_out_p_wire, txa_data_out_n_wire;
 wire test_en_wire, txd_data_out_wire, tx_frame_start_wire,
c_data_valid_wire;
 wire [7 : 0] config_in_wire;
 wire [8 : 0] txd_data_in_wire, digital_out_wire;


 SERDESv2
 SERDES_core
(
      //------------Inputs--------
      .reset(rst_wire),
      .clk(clk_wire),
      .rxa_in_p(rxa_in_p_wire),
      .rxa_in_n(rxa_in_n_wire),
      .config_in(config_in_wire),
      .txd_data_in(txd_data_in_wire),
      .test_en(test_en_wire),

      //------------Outputs--------
      .digital_out(digital_out_wire),
      .txa_data_out_p(txa_data_out_p_wire),
      .txa_data_out_n(txa_data_out_n_wire),
      .txd_data_out(txd_data_out_wire),
      .tx_frame_start(tx_frame_start_wire),
      .c_data_valid(c_data_valid_wire)
);

assign VDD_wire = VDD;
assign VSS_wire = VSS;
assign DVDD_wire = DVDD;
assign DVSS_wire = DVSS;

//------------ POWER ------------

// core power ring
PVDD pad_VDD (.VDD(VDD_wire));
PVSS pad_VSS (.VSS(VSS_wire));

// IO power ring
PDVDD pad_DVDD (.DVDD(DVDD_wire));
PDVSS pad_DVSS (.DVSS(DVSS_wire));

//------------ Input PADS ------------

PICSD pad_rst(.P(rst), .IE(1'b1), .PD(1'b1), .Y(rst_wire));
PIC pad_clk(.P(clk), .IE(1'b1), .Y(clk_wire) );

PIC pad_rxa_in_p(.P(rxa_in_p), .IE(1'b1), .Y(rxa_in_p_wire) );
PIC pad_rxa_in_n(.P(rxa_in_n), .IE(1'b1), .Y(rxa_in_n_wire) );

PIC pad_config_in0(.P(config_in[0]), .IE(1'b1), .Y(config_in_wire[0]));
PIC pad_config_in1(.P(config_in[1]), .IE(1'b1), .Y(config_in_wire[1]));
PIC pad_config_in2(.P(config_in[2]), .IE(1'b1), .Y(config_in_wire[2]));
PIC pad_config_in3(.P(config_in[3]), .IE(1'b1), .Y(config_in_wire[3]));
PIC pad_config_in4(.P(config_in[4]), .IE(1'b1), .Y(config_in_wire[4]));
PIC pad_config_in5(.P(config_in[5]), .IE(1'b1), .Y(config_in_wire[5]));
PIC pad_config_in6(.P(config_in[6]), .IE(1'b1), .Y(config_in_wire[6]));
PIC pad_config_in7(.P(config_in[7]), .IE(1'b1), .Y(config_in_wire[7]));

PIC pad_txd_data_in0(.P(txd_data_in[0]), .IE(1'b1), .Y(txd_data_in_wire[0]));
PIC pad_txd_data_in1(.P(txd_data_in[1]), .IE(1'b1), .Y(txd_data_in_wire[1]));
PIC pad_txd_data_in2(.P(txd_data_in[2]), .IE(1'b1), .Y(txd_data_in_wire[2]));
PIC pad_txd_data_in3(.P(txd_data_in[3]), .IE(1'b1), .Y(txd_data_in_wire[3]));
PIC pad_txd_data_in4(.P(txd_data_in[4]), .IE(1'b1), .Y(txd_data_in_wire[4]));
PIC pad_txd_data_in5(.P(txd_data_in[5]), .IE(1'b1), .Y(txd_data_in_wire[5]));
PIC pad_txd_data_in6(.P(txd_data_in[6]), .IE(1'b1), .Y(txd_data_in_wire[6]));
PIC pad_txd_data_in7(.P(txd_data_in[7]), .IE(1'b1), .Y(txd_data_in_wire[7]));
PIC pad_txd_data_in8(.P(txd_data_in[8]), .IE(1'b1), .Y(txd_data_in_wire[8]));

PIC pad_test_en(.P(test_en), .IE(1'b1), .Y(test_en_wire) );

//------------ Output PADS ------------
POC4C pad_txa_data_out_p(.A(txa_data_out_p_wire), .P(txa_data_out_p) );
POC4C pad_txa_data_out_n(.A(txa_data_out_n_wire), .P(txa_data_out_n) );

POC4C pad_txd_data_out(.A(txd_data_out_wire), .P(txd_data_out) );

POC4C pad_tx_frame_start(.A(tx_frame_start_wire), .P(tx_frame_start) );

POC4C pad_c_data_valid (.A(c_data_valid_wire), .P(c_data_valid) );

POC4C pad_digital_out0(.A(digital_out_wire[0]), .P(digital_out[0]));
POC4C pad_digital_out1(.A(digital_out_wire[1]), .P(digital_out[1]));
POC4C pad_digital_out2(.A(digital_out_wire[2]), .P(digital_out[2]));
POC4C pad_digital_out3(.A(digital_out_wire[3]), .P(digital_out[3]));
POC4C pad_digital_out4(.A(digital_out_wire[4]), .P(digital_out[4]));
POC4C pad_digital_out5(.A(digital_out_wire[5]), .P(digital_out[5]));
POC4C pad_digital_out6(.A(digital_out_wire[6]), .P(digital_out[6]));
POC4C pad_digital_out7(.A(digital_out_wire[7]), .P(digital_out[7]));
POC4C pad_digital_out8(.A(digital_out_wire[8]), .P(digital_out[8]));

//------------ CORNERS ------------

PCORNER sw_pcorner ();
PCORNER se_pcorner ();
PCORNER nw_pcorner ();
PCORNER ne_pcorner ();

endmodule
```

B. LOGIC SYNTHESIS SCRIPT

```
#Name: run_rc_serdes_chip_mixed.tcl
if {[file exists /proc/cpuinfo]} {
  sh grep "model name" /proc/cpuinfo
  sh grep "cpu MHz"    /proc/cpuinfo
}
puts "Hostname : [info hostname]"
## Preset global variables and attributes
set DESIGN SERDES_chip
set SYN_EFF high
set MAP_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH outputs_${DATE}
set _REPORTS_PATH reports_${DATE}
set _LOG_PATH logs_${DATE}
set run_worst_case 0
if {$run_worst_case} {
  ##< WORST CASE >###
  set lib_rary {scx3_cmos8rf_lpvt_ss_1p08v_125c.lib
iogpil_cmrf8sf_rvt_ss_1p08v_2p3v_125c.lib}
  set SDC_OUT_SUFFIX _WC
} else {
  ##< Typical >###
  set lib_rary {scx3_cmos8rf_lpvt_tt_1p2v_25c.lib
iogpil_cmrf8sf_rvt_tt_1p2v_2p5v_25c.lib}
  set SDC_OUT_SUFFIX _Typ
}
set lef_library { /opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-
x/lef/ibm13rflpvt_macros.lef /opt/libs/ARM/IB03LB501-FB-00000-r0p0-
00rel0/aci/sc-x/lef/ibm13_8lm_2thick_3rf_tech.lef /opt/libs/ARM/IB03IG502-FB-
00000-r0p0-00rel0/aci/io/lef/iogpil_cmrf8sf_rvt_M2_3_3.lef
/home/mhoil/Proyecto/SerDes/lef/RXA_backup/oto_analog_receiver_LEF_20171109.l
ef /home/mhoil/Proyecto/SerDes/lef/TXA_backup/TX_CHIPEDGE_ABSTRACT_V4.lef}
set_attribute lib_search_path {/opt/libs/ARM/IB03LB501-FB-00000-r0p0-
00rel0/aci/sc-x/synopsys /opt/libs/ARM/IB03IG502-FB-00000-r0p0-
00rel0/aci/io/synopsys} /
set_attribute script_search_path {.. } /
set_attribute hdl_search_path {../rtl} /
set_attribute information_level 9 /
##allow inout subc! connection to constant 1'b0, ¡IT ALSO ALLOWS THE INPUT TO
OUTPUT PORTS!
set_attribute hdl_allow_inout_const_port_connect true /
##When set to true, RTL Compiler binds instances of design M to component M
from the technology library
#set_attribute hdl_use_techelt_first true /
## Library setup
set_attribute library $lib_rary
## PLE
set_attribute lef_library $lef_library /
set_attribute auto_ungroup none /
## Load Design
read_hdl -v2001 {
Comparator.v
```

```
Registro.v
Register.v
clock_divider.v
Analog_RX.v
TXA_FINAL.v
Analog_TX.v
LFSR.v
simple_dual_port_ram_single_clock.v
dataA_save.v
RegistroSyncRst.v
dataB_compare.v
ComparatorP.v
Mux2to1.v
SignalDriverS2.v
test_modules_v2.v
FlipFlopT.v
CDR.v
decodePipe.v
deserializer.v
digitalRX.v
encode.v
Control_serial.v
Serialization.v
Serializer.v
SERDESv2.v
SERDES_chip.v
}
elaborate $DESIGN
#set_attr preserve true [find /designs/$DESIGN -subdesign Analog_RX]
#set_attr preserve true [find /designs/$DESIGN -subdesign TXA]
#set_attribute preserve true [find /designs/$DESIGN -net rxa_out_*]
#set_attribute preserve true [find /designs/$DESIGN -net txa_data_out_*]
#set_attribute preserve true [find /designs/$DESIGN -net config_in*]
#set_attribute preserve true [find /designs/$DESIGN -net tx_config*]
#set_attribute preserve true [find /designs/$DESIGN -net txa_config_wire*]
puts "Runtime & Memory after 'read_hdl'"
timestat Elaboration
## Constraints Setup
read_sdc SerDes_Cuau.sdc
puts "The number of exceptions is [llength [find /designs/$DESIGN -exception
*]]"
if {![file exists ${_LOG_PATH}]} {
  file mkdir ${_LOG_PATH}
  puts "Creating directory ${_LOG_PATH}"
}
if {![file exists ${_OUTPUTS_PATH}]} {
  file mkdir ${_OUTPUTS_PATH}
  puts "Creating directory ${_OUTPUTS_PATH}"
}
if {![file exists ${_REPORTS_PATH}]} {
  file mkdir ${_REPORTS_PATH}
  puts "Creating directory ${_REPORTS_PATH}"
}
report timing -lint
## Define cost groups (clock-clock, clock-output, input-clock, input-output)
if {[llength [all::all_seqs]] > 0} {
  define_cost_group -name I2C -design $DESIGN
  define_cost_group -name C2O -design $DESIGN
  define_cost_group -name C2C -design $DESIGN
  path_group -from [all::all_seqs] -to [all::all_seqs] -group C2C -name C2C
  path_group -from [all::all_seqs] -to [all::all_outs] -group C2O -name C2O
  path_group -from [all::all_inps]  -to [all::all_seqs] -group I2C -name I2C
}
define_cost_group -name I2O -design $DESIGN
path_group -from [all::all_inps]  -to [all::all_outs] -group I2O -name I2O
foreach cg [find / -cost_group *] {
  report timing -cost_group [list $cg] >> $_REPORTS_PATH/${DESIGN}_pretim.rpt
}

## Synthesizing to generic
synthesize -to_generic -eff $SYN_EFF
write_hdl > ${DESIGN}_generic.v
puts "Runtime & Memory after 'synthesize -to_generic'"
timestat GENERIC
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_generic.rpt
report power > $_REPORTS_PATH/generic_power.rpt
generate_reports -outdir $_REPORTS_PATH -tag generic
summary_table -outdir $_REPORTS_PATH
## Synthesizing to gates
synthesize -to_mapped -eff $MAP_EFF -no_incr
puts "Runtime & Memory after 'synthesize -to_map -no_incr'"
timestat MAPPED
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_map.rpt
report power > $_REPORTS_PATH/map_power.rpt
foreach cg [find / -cost_group *] {
  report timing -cost_group [list $cg] > $_REPORTS_PATH/${DESIGN}_[basename
$cg]_post_map.rpt
}
generate_reports -outdir $_REPORTS_PATH -tag map
summary_table -outdir $_REPORTS_PATH
##Intermediate netlist for LEC verification..
write_hdl -lec > ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v
write_do_lec -revised_design ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v -
logfile ${_LOG_PATH}/rtl2intermediate.lec.log >
${_OUTPUTS_PATH}/rtl2intermediate.lec.do
## Incremental Synthesis
synthesize -to_mapped -eff $MAP_EFF -incr
report power > $_REPORTS_PATH/incremental_power.rpt
generate_reports -outdir $_REPORTS_PATH -tag incremental
summary_table -outdir $_REPORTS_PATH
puts "Runtime & Memory after incremental synthesis"
timestat INCREMENTAL
foreach cg [find / -cost_group *] {
  report timing -cost_group [list $cg] > $_REPORTS_PATH/${DESIGN}_[basename
$cg]_post_incr.rpt
}
## write Encounter file set (verilog, SDC, config, etc.)
report qor > $_REPORTS_PATH/${DESIGN}_qor.rpt
report power > $_REPORTS_PATH/${DESIGN}_power.rpt
report timing     > $_REPORTS_PATH/${DESIGN}_timing.rpt
report area > $_REPORTS_PATH/${DESIGN}_area.rpt
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_incr.rpt
report messages > $_REPORTS_PATH/${DESIGN}_messages.rpt
report gates > $_REPORTS_PATH/${DESIGN}_gates.rpt
write_design -basename ${_OUTPUTS_PATH}/${DESIGN}_m
write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_m${SDC_OUT_SUFFIX}.sdc
### write_do_lec
write_do_lec -verbose -no_exit -golden_design
${_OUTPUTS_PATH}/${DESIGN}_intermediate.v -revised_design
${_OUTPUTS_PATH}/${DESIGN}_m.v -logfile
${_LOG_PATH}/intermediate2final.lec.log >
${_OUTPUTS_PATH}/intermediate2final.lec.do
write_do_lec -verbose -no_exit -revised_design ${_OUTPUTS_PATH}/${DESIGN}_m.v
-logfile ${_LOG_PATH}/rtl2final.lec.log > ${_OUTPUTS_PATH}/rtl2final.lec.do
puts "Final Runtime & Memory."
timestat FINAL
report power
file copy [get_attr stdout_log /] ${_LOG_PATH}/.
check_design -unresolved
```

C. LOGIC SYNTHESIS CONSTRAINTS FILE

```
# ITESO University
# Cuauhtemoc Aguilera
# User Constraint File
# Mon 09 Oct 2017 02:00:45 PM CDT, Miguel Hoil
# Clock constraints were modified

set_time_unit -picoseconds
set_load_unit -femtofarads

# Clock definition
define_clock -name 625MHz_CLK -period 1600 -rise 10 -fall 90 [get_ports clk]

# slew rate definitions (min rise, min fall, max rise, max fall).
set_attribute slew { 28 28 28 28 } 625MHz_CLK

# network clock latency
set_attribute clock_network_late_latency 90    625MHz_CLK
set_attribute clock_network_early_latency 70   625MHz_CLK

# source clock latency
set_attribute clock_source_late_latency 50     625MHz_CLK
set_attribute clock_source_early_latency 40    625MHz_CLK

# clock skew
set_attribute clock_setup_uncertainty {17 10}  625MHz_CLK
set_attribute clock_hold_uncertainty {14 8}    625MHz_CLK

# We are considering around six times the clock slew rate.
set_attribute max_transition 150 /designs/Adders
```

D. PADS POSITIONS FILE

```
( globals
   version = 3
   space = 0
# Name: SERDES_arm.ioc
# io_order: clockwise | counterclockwise | default: vertical bottom to top;
horizontal left to right
   io_order = default
)

( row_margin
  ( top
  ( io_row ring_number = 1 margin = 150)
 )
  ( bottom
  ( io_row ring_number = 1 margin = 150)
 )
  ( left
  ( io_row ring_number = 1 margin = 150)
)
  ( right
  ( io_row ring_number = 1 margin = 150)
)
  )

#Pad: ne_corner NE1

( iopad
  ( topleft
    (locals ring_number = 1)
    ( inst name= "nw_pcorner")
  )

  ( left
    ( locals ring_number = 1)
    ( inst name= "pad_txa_data_out_n" )
    ( inst name= "pad_txa_data_out_p" )
    ( inst name= "pad_tx_frame_start" )
    ( inst name= "pad_rxa_in_n" )
    ( inst name= "pad_rxa_in_p" )
    ( inst name= "pad_clk" )
    ( inst name= "pad_txd_data_out" )
    ( inst name= "pad_c_data_valid" )
    ( inst name= "pad_VSS" )
    ( inst name= "pad_VDD" )
  )

  ( bottomleft
     (locals ring_number = 1)
     ( inst name= "sw_pcorner")
  )

  ( bottom
    ( locals ring_number = 1 )
    ( inst name= "pad_DVDD" )
    ( inst name= "pad_DVSS" )
    ( inst name= "pad_rst" )
    ( inst name= "pad_config_in7" )
    ( inst name= "pad_config_in6" )
    ( inst name= "pad_config_in5" )
    ( inst name= "pad_config_in4" )
    ( inst name= "pad_config_in3" )
    ( inst name= "pad_config_in2" )
    ( inst name= "pad_config_in1" )
  )


  ( bottomright
    ( locals ring_number = 1)
    ( inst name= "se_pcorner" )
  )

  ( right
    ( locals ring_number = 1    )
    ( inst name= "pad_config_in0" )
    ( inst name= "pad_txd_data_in0" )
    ( inst name= "pad_txd_data_in1" )
    ( inst name= "pad_txd_data_in2" )
    ( inst name= "pad_txd_data_in3" )
    ( inst name= "pad_txd_data_in4" )
    ( inst name= "pad_txd_data_in5" )
    ( inst name= "pad_txd_data_in6" )
    ( inst name= "pad_txd_data_in7" )
    ( inst name= "pad_txd_data_in8" )
  )

  ( topright
    (locals ring_number = 1)
    ( inst name= "ne_pcorner")
  )

  ( top
    ( locals ring_number = 1    )
    ( inst name= "pad_digital_out8" )
    ( inst name= "pad_digital_out7" )
    ( inst name= "pad_digital_out6" )
    ( inst name= "pad_digital_out5" )
    ( inst name= "pad_digital_out4" )
    ( inst name= "pad_digital_out3" )
    ( inst name= "pad_digital_out2" )
    ( inst name= "pad_digital_out1" )
    ( inst name= "pad_digital_out0" )
    ( inst name= "pad_test_en" )
  )
)
```

E. VIEW DEFINITION FILE

```
# Version:1.0 MMMC View Definition File
# Do Not Remove Above Line
# Name: SERDES_chip_analysis_Typ_WC.view
create_rc_corner -name RC_Corner_Typ -T {25} -preRoute_res {1.0} -
preRoute_cap {1.0} -preRoute_clkres {0.0} -preRoute_clkcap {0.0} -
postRoute_res {1.0} -postRoute_cap {1.0} -postRoute_xcap {1.0} -
postRoute_clkres {0.0} -postRoute_clkcap {0.0}
create_rc_corner -name RC_Corner_WC -T {125} -preRoute_res {1.0} -
preRoute_cap {1.0} -preRoute_clkres {0.0} -preRoute_clkcap {0.0} -
postRoute_res {1.0} -postRoute_cap {1.0} -postRoute_xcap {1.0} -
postRoute_clkres {0.0} -postRoute_clkcap {0.0}
create_library_set -name SERDES_chip_Typ_lib -timing
{/opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-
x/synopsys/scx3_cmos8rf_lpvt_tt_1p2v_25c.lib /opt/libs/ARM/IB03IG502-FB-
00000-r0p0-00rel0/aci/io/synopsys/iogpil_cmrf8sf_rvt_tt_1p2v_2p5v_25c.lib}
create_library_set -name SERDES_chip_WC_lib -timing {/opt/libs/ARM/IB03LB501-
FB-00000-r0p0-00rel0/aci/sc-x/synopsys/scx3_cmos8rf_lpvt_ss_1p08v_125c.lib
/opt/libs/ARM/IB03IG502-FB-00000-r0p0-
00rel0/aci/io/synopsys/iogpil_cmrf8sf_rvt_ss_1p08v_2p3v_125c.lib}
create_constraint_mode -name SERDES_chip_Typ_constraints -sdc_files
{../SERDES_chip_m_Typ.sdc}
create_constraint_mode -name SERDES_chip_WC_constraints -sdc_files
{../SERDES_chip_m_WC.sdc}
create_delay_corner -name Delay_Corner_Typ -library_set {SERDES_chip_Typ_lib}
-rc_corner {RC_Corner_Typ}
create_delay_corner -name Delay_Corner_WC -library_set {SERDES_chip_WC_lib} -
rc_corner {RC_Corner_WC}
create_analysis_view -name SERDES_chip_view_Typ -constraint_mode
{SERDES_chip_Typ_constraints} -delay_corner {Delay_Corner_Typ}
create_analysis_view -name SERDES_chip_View_WC -constraint_mode
{SERDES_chip_WC_constraints} -delay_corner {Delay_Corner_WC}
set_analysis_view -setup {SERDES_chip_View_WC} -hold {SERDES_chip_view_Typ}
```

F1. SETUP AND POWER GRID SCRIPT

```
# Name:
#     edi_script_SERDES_s1_pwr_grid.tcl
# Author:
#     Christian Aparicio Zuleta
###< SETUP >###
set_global    _enable_mmmc_by_default_flow    $CTE::mmmc_default
suppressMessage ENCEXT-2799
win
set ::TimeLib::tsgMarkCellLatchConstructFlag 1
set conf_qxconf_file NULL
set conf_qxlib_file NULL
set defHierChar /
set distributed_client_message_echo 1
set gpsPrivate::dpgNewAddBufsDBUpdate 1
set gpsPrivate::lsgEnableNewDbApiInRestruct 1
set init_gnd_net {VSS DVSS}
set init_io_file ../SERDES_arm.ioc
set init_lef_file {/opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-
x/lef/ibm13_8lm_2thick_3rf_tech.lef /opt/libs/ARM/IB03LB501-FB-00000-r0p0-
00rel0/aci/sc-x/lef/ibm13rflpvt_macros.lef /opt/libs/ARM/IB03IG502-FB-00000-
r0p0-00rel0/aci/io/lef/iogpil_cmrf8sf_rvt_M2_3_3.lef
/home/mhoil/Proyecto/SerDes/lef/RXA_backup/oto_analog_receiver_LEF_20171109.l
ef /home/mhoil/Proyecto/SerDes/lef/TXA_backup/TX_CHIPEDGE_ABSTRACT_V4.lef}
set init_mmmc_file ../SERDES_chip_analysis_Typ_WC.view
set init_pwr_net {VDD DVDD}
set init_top_cell SERDES_chip
set init_verilog ../SERDES_chip_m.v
set lsgOCPGainMult 1.000000
set pegDefaultResScaleFactor 1.000000
set pegDetailResScaleFactor 1.000000
set timing_library_float_precision_tol 0.000010
set timing_library_load_pin_cap_indices {}
set tso_post_client_restore_command {update_timing ; write_eco_opt_db ;}
init_design

setDesignMode -process 130

###< FLOORPLAN >###
getIoFlowFlag
setFPlanRowSpacingAndType 3.6 2
setIoFlowFlag 0
floorPlan -dieSizeByIoHeight max -site IBM13SITE -s 703.0 703.0 13.5 13.5
13.5 13.5
uiSetTool select
getIoFlowFlag

###< LINK PWR SOURCES >###
clearGlobalNets
globalNetConnect VDD -type pgpin -pin VDD -inst * -verbose
globalNetConnect VSS -type pgpin -pin VSS -inst * -verbose
globalNetConnect VDD -type tiehi -pin VDD -inst * -verbose
globalNetConnect VSS -type tielo -pin VSS -inst * -verbose
```

```
###< POWER RING >###
set sprCreateIeRingNets {}
set sprCreateIeRingLayers {}
set sprCreateIeRingWidth 1.0
set sprCreateIeRingSpacing 1.0
set sprCreateIeRingOffset 1.0
set sprCreateIeRingThreshold 1.0
set sprCreateIeRingJogDistance 1.0
addRing -skip_via_on_wire_shape Noshape -skip_via_on_pin Standardcell -center
1 -stacked_via_top_layer MA -type core_rings -jog_distance 0.2 -threshold 0.2
-nets {VDD VSS} -follow core -stacked_via_bottom_layer M1 -layer {bottom M1
top M1 right M2 left M2} -width 4 -spacing 1 -offset 0.2

###< POWER GRID >###

#HORIZONTAL STRIPES, VDD & VSS - TOP
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { M1 MA } -blockPinTarget { nearestTarget } -
padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -
corePinTarget { firstAfterRowEnd } -floatingStripeTarget { blockring padring
ring stripe ringpin blockpin followpin } -allowJogging 1 -
crossoverViaLayerRange { M1 MA } -area { 393.51 1130.773 1139.834 747.188 } -
nets { VDD VSS } -allowLayerChange 1 -blockPin useLef -targetViaLayerRange {
M1 MA }

#HORIZONTAL STRIPES, VDD - MIDDLE RIGHT
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { M1 MA } -blockPinTarget { nearestTarget } -
padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -
corePinTarget { firstAfterRowEnd } -floatingStripeTarget { blockring padring
ring stripe ringpin blockpin followpin } -allowJogging 1 -
crossoverViaLayerRange { M1 MA } -area { 576.12 747.61 1126.321 528.105 } -
nets { VDD } -allowLayerChange 1 -blockPin useLef -targetViaLayerRange { M1
MA }

#HORIZONTAL STRIPES, VSS - MIDDLE RIGHT
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { M1 MA } -blockPinTarget { nearestTarget } -
padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -
corePinTarget { firstAfterRowEnd } -floatingStripeTarget { blockring padring
ring stripe ringpin blockpin followpin } -allowJogging 1 -
crossoverViaLayerRange { M1 MA } -area { 578.02 749.456 1126.099 524.083 } -
allowLayerChange 1 -nets { VSS } -blockPin useLef -targetViaLayerRange { M1
MA }

#HORIZONTAL STRIPES, VDD - SE, BOTTOM RIGHT
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { M1 MA } -blockPinTarget { nearestTarget } -
padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -
corePinTarget { firstAfterRowEnd } -floatingStripeTarget { blockring padring
ring stripe ringpin blockpin followpin } -allowJogging 1 -
crossoverViaLayerRange { M1 MA } -area { 790.01 528.488 1120.189 408.829 } -
allowLayerChange 1 -nets { VDD } -blockPin useLef -targetViaLayerRange { M1
MA }
```

```
#HORIZONTAL STRIPES, VSS - SE, BOTTOM RIGHT
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { M1 MA } -blockPinTarget { nearestTarget } -
padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -
corePinTarget { firstAfterRowEnd } -floatingStripeTarget { blockring padring
ring stripe ringpin blockpin followpin } -allowJogging 1 -
crossoverViaLayerRange { M1 MA } -area { 791.91 523.863 1125.423 408.975 } -
allowLayerChange 1 -nets { VSS } -blockPin useLef -targetViaLayerRange { M1
MA }

#HORIZONTAL STRIPES, VDD - SE, LEFT MARGIN
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { M1 MA } -blockPinTarget { nearestTarget } -
padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -
corePinTarget { firstAfterRowEnd } -floatingStripeTarget { blockring padring
ring stripe ringpin blockpin followpin } -allowJogging 1 -
crossoverViaLayerRange { M1 MA } -area { 399.3 748.254 433.431 404.301 } -
allowLayerChange 1 -nets { VDD } -blockPin useLef -targetViaLayerRange { M1
MA }

#HORIZONTAL STRIPES, VSS - SE, LEFT MARGIN
sroute -connect { blockPin padPin padRing corePin floatingStripe } -
layerChangeRange { M1 MA } -blockPinTarget { nearestTarget } -
padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -
corePinTarget { firstAfterRowEnd } -floatingStripeTarget { blockring padring
ring stripe ringpin blockpin followpin } -allowJogging 1 -
crossoverViaLayerRange { M1 MA } -area { 399.3 740.964 435.332 399.299 } -
allowLayerChange 1 -nets { VSS } -blockPin useLef -targetViaLayerRange { M1
MA }

#1st VERTICAL STRIPE
addStripe -skip_via_on_wire_shape Noshape -block_ring_top_layer_limit M3 -
max_same_layer_jog_length 8 -padcore_ring_bottom_layer_limit M1 -
number_of_sets 1 -skip_via_on_pin Standardcell -stacked_via_top_layer MA -
padcore_ring_top_layer_limit M3 -spacing .3 -merge_stripes_value 0.2 -layer
M2 -block_ring_bottom_layer_limit M1 -stop_x {1200} -width 1.6 -nets {VDD
VSS} -start_x 431.83 -stacked_via_bottom_layer M1

#2nd VERTICAL STRIPE
addStripe -skip_via_on_wire_shape Noshape -block_ring_top_layer_limit M3 -
max_same_layer_jog_length 8 -padcore_ring_bottom_layer_limit M1 -
number_of_sets 1 -skip_via_on_pin Standardcell -stacked_via_top_layer MA -
padcore_ring_top_layer_limit M3 -spacing .3 -merge_stripes_value 0.2 -layer
M2 -block_ring_bottom_layer_limit M1 -stop_x {1200} -width 1.6 -nets {VDD
VSS} -start_x 576.116 -stacked_via_bottom_layer M1

#3rd VERTICAL STRIPE
addStripe -skip_via_on_wire_shape Noshape -block_ring_top_layer_limit M3 -
max_same_layer_jog_length 8 -padcore_ring_bottom_layer_limit M1 -
number_of_sets 1 -skip_via_on_pin Standardcell -stacked_via_top_layer MA -
padcore_ring_top_layer_limit M3 -spacing .3 -merge_stripes_value 0.2 -layer
M2 -block_ring_bottom_layer_limit M1 -stop_x {1200} -width 1.6 -nets {VDD
VSS} -start_x 790.004 -stacked_via_bottom_layer M1

#DIGITAL MODULES' VERTICAL STRIPES
```

```
set sprCreateIeStripeNets {}
set sprCreateIeStripeLayers {}
set sprCreateIeStripeWidth 10.0
set sprCreateIeStripeSpacing 2.0
set sprCreateIeStripeThreshold 1.0
#original
addStripe -skip_via_on_wire_shape Noshape -block_ring_top_layer_limit M3 -
max_same_layer_jog_length 8 -padcore_ring_bottom_layer_limit M1 -
number_of_sets 3 -skip_via_on_pin Standardcell -stacked_via_top_layer MA -
padcore_ring_top_layer_limit M3 -spacing 0.3 -xleft_offset 438.51 -
xright_offset 142.8 -merge_stripes_value 0.2 -layer M2 -
block_ring_bottom_layer_limit M1 -width 1.6 -nets {VDD VSS} -
stacked_via_bottom_layer M1

#Recortar excedente de stripes
editCutWire -x1 576.119 -y1 529.119 -x2 577.72 -y2 529.119
editCutWire -x1 578.02 -y1 532.72 -x2 579.621 -y2 532.72
selectWire 576.1200 404.3000 577.7200 529.1200 2 VDD
selectWire 578.0200 399.3000 579.6200 532.7200 2 VSS
deleteSelectedFromFPlan
```

F2. PLACE STANDARD CELLS AND MODULES SCRIPT

```
# Name: edi_script_SERDES_place_cells.tcl

setEndCapMode -reset
setEndCapMode -boundary_tap false
setPlaceMode -reset
setPlaceMode -congEffort auto -timingDriven 1 -modulePlan 1 -clkGateAware 1 -
powerDriven 0 -ignoreScan 0 -reorderScan 0 -ignoreSpare 0 -placeIOPins 1 -
moduleAwareSpare 0 -preserveRouting 0 -rmAffectedRouting 0 -checkRoute 0 -
swapEEQ 0
setPlaceMode -fp false
placeDesign

placeInstance SERDES_core/analog_transmitter_unit/txa_final_unit 445 423
placeInstance SERDES_core/analog_receiver_unit 445 545
```

F3 CTS AND OPTIMIZATION

```
# Name: edi_script_SERDES_cts.tcl

setCTSMode -engine ck -opt true -optLatency true
createClockTreeSpec -bufferList {CLKBUFX12TS CLKBUFX16TS CLKBUFX20TS
CLKBUFX2TS CLKBUFX3TS CLKBUFX4TS CLKBUFX6TS CLKBUFX8TS CLKINVX12TS
CLKINVX16TS CLKINVX1TS CLKINVX20TS CLKINVX2TS CLKINVX3TS CLKINVX4TS
CLKINVX6TS CLKINVX8TS DLY1X1TS DLY1X4TS DLY2X1TS DLY2X4TS DLY3X1TS DLY3X4TS
DLY4X1TS DLY4X4TS} -file serdes_rxa_pads_clk.ctstch

# ITESO University
# Cuauhtémoc Aguilera

Puts "Timing the design before CTS"

# Calculates the delays for paths based on max. opertating conditions (op)
and min. op.
setAnalysisMode -analysisType onChipVariation

timeDesign -preCTS -prefix preCTS_setup
timeDesign -preCTS -prefix preCTS_hold  -hold

Puts "Running CTS"
dbDeleteTrialRoute
#El comando tarda ~30 minutos
clockDesign -specFile serdes_rxa_pads_clk.ctstch -outDir clock_report -
fixedInstBeforeCTS
Puts "Finished running CTS"

Puts "Timing the design after CTS"
timeDesign -postCTS -prefix postCTS_setup
timeDesign -postCTS -prefix postCTS_hold  -hold

#DRV

Puts "Setting Optimizaiton Mode Options for DRV fixes"
setOptMode -fixFanoutLoad true -fixDRC true -addInst true
setOptMode -addInstancePrefix postCTSdrv

Puts "Optimizing for DRV"
optDesign -postCTS -drv

Puts "Timing the design after DRV fixes"
timeDesign -postCTS -prefix postCTS_setup_DRVfix
timeDesign -postCTS -prefix postCTS_hold_DRVfix -hold

#postCTS SETUP

Puts "Setting Optimization Mode Options for Setup fixes"
setOptMode -effort high -addInstancePrefix postCTSsetup

Puts "Optimizing for Setup"
optDesign -postCTS -incr
```

```
Puts  "Timing the design after Setup fixes"
timeDesign -postCTS -prefix postCTS_setup_Setupfix
timeDesign -postCTS -prefix postCTS_hold_Setupfix  -hold

#postCTS HOLD

setOptMode -allEndPoints true -reclaimArea true -addInstancePrefix
postCTShold
optDesign -postCTS -hold
Puts  "Timing the design after Hold fixes"
timeDesign -postCTS -prefix postCTS_setup_Holdfix
timeDesign -postCTS -prefix postCTS_hold_Holdfix  -hold
```

F4. ROUTING AND FILLERS

```
# Name: edi_script_SERDES_route.tcl

####< ROUTING >####

Puts "Routing the Design"
setNanoRouteMode -quiet -timingEngine {}
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven false
setNanoRouteMode -quiet -routeWithSiDriven false
routeDesign -globalDetail

####< POST ROUTE OPT>####

setOptMode -postRouteSetupRecovery true -postRouteAreaReclaim setupAware -
addInstancePrefix postRoute_setupFx
optDesign -postRoute -incr

setOptMode -holdFixingEffort high -addInstancePrefix postRoute_holdFx
optDesign -postRoute -hold

###

Puts "Timing the design after Route"
timeDesign -postRoute -prefix postRoute_setup
timeDesign -postRoute -prefix postRoute_hold  -hold

#Add fillers
addFiller -cell FILL8TS FILL64TS FILL4TS FILL32TS FILL2TS FILL1TS FILL16TS -
prefix FILLER

#power report
check_power_library /opt/libs/ARM/IB03LB501-FX-00001-r0p0-00rel0/aci/sc-
x/voltagestorm/ibm13rflpvt_8lm_2thick_3rf.cl/
set_power_analysis_mode -reset
set_power_analysis_mode -method static -corner max -create_binary_db true -
write_static_currents true -honor_negative_energy true -
ignore_control_signals true
report_power -outfile SerDesPwr -sort { total }
```

G1. TEST OPERATING MODE 1 TESTBENCH

```
// Name: SerDes_m1_tb.v
`timescale 1ns/1ps
`define DELAY 10
module SerDes_m1_tb;
localparam DATA_WIDTH = 9;
//Inputs
reg rst, clk, rxa_in_p, test_en;
reg [7 : 0] config_in;
reg [DATA_WIDTH-1 : 0] txd_data_in;
wire rxa_in_n;
//Outputs
wire [DATA_WIDTH-1 : 0] digital_out;
wire txa_data_out_p, txa_data_out_n, txd_data_out, test_out;
wire tx_frame_start, c_data_valid;
//Other
wire [5:0]  num_errors;
wire [2:0] mode;//Bist Configuration pins
wire tx_clk;
integer i;
//Module instantiation.
SERDES #(.WIDTH(9)) serdes0 (
//INPUTS
      .reset(rst), .clk(clk), .rxa_in_p(rxa_in_p), .rxa_in_n(rxa_in_n),
      .config_in(config_in), .txd_data_in(txd_data_in), .test_en(test_en),
//OUTPUTS
      .digital_out(digital_out), .txa_data_out_p(txa_data_out_p),
      .txa_data_out_n(txa_data_out_n), .txd_data_out(txd_data_out),
      .test_out(test_out), .tx_frame_start(tx_frame_start),
      .c_data_valid(c_data_valid) );
/**********************************************************/
initial // Clock generator
  begin
    #0 clk = 0;
    forever #10 clk = !clk;
  end
/**********************************************************/
assign tx_clk = serdes0.clks_in_wire[0];
assign mode = config_in[2 : 0];
assign num_errors = digital_out[5 : 0];
task serial (
      input [DATA_WIDTH : 0] data_transmit );
begin
  for (i = 0; i <= DATA_WIDTH ; i = i + 1)   begin
      rxa_in_p=data_transmit[i];//serial data
      @(posedge tx_clk);
  end
end
endtask
// input generation
initial
begin
  #0
```

```
      config_in=8'b0000_0001;
      test_en = 1'b1;
// reset
      rst = 1'b0;
      #(`DELAY*10) rst = 1'b1;
      @(posedge tx_clk);
//Mode 01
//Send Comma
      serial(10'b11_1000_0011);
      serial(10'h0b9);
      serial(10'h0ae);
      serial(10'h0ad);
      #(`DELAY*1000);
      $stop();
//END MODE 1
end

assign rxa_in_n = ~rxa_in_p;

endmodule
```

G2. TEST OPERATING MODE 2 TESTBENCH

```
// Name: SerDes_m2_tb.v
`timescale 1ns/1ps
`define DELAY 10
module SerDes_m2_tb;
localparam DATA_WIDTH = 9;
//Inputs
reg rst, clk, rxa_in_p;
wire rxa_in_n;
reg [7 : 0] config_in;
reg [DATA_WIDTH-1 : 0] txd_data_in;
reg test_en;
//Outputs
wire [DATA_WIDTH-1 : 0] digital_out;
wire txa_data_out_p, txa_data_out_n, txd_data_out;
wire test_out, tx_frame_start, c_data_valid;
//Other
wire [5:0]  num_errors;
wire [2:0] mode;//Bist Configuration pins
wire tx_clk;
integer i;
//Module instantiation.
SERDES #(.WIDTH(9)) serdes0 (
//INPUTS
      .reset(rst), .clk(clk), .rxa_in_p(rxa_in_p), .rxa_in_n(rxa_in_n),
      .config_in(config_in), .txd_data_in(txd_data_in), .test_en(test_en),
//OUTPUTS
      .digital_out(digital_out), .txa_data_out_p(txa_data_out_p),
      .txa_data_out_n(txa_data_out_n), .txd_data_out(txd_data_out),
      .test_out(test_out), .tx_frame_start(tx_frame_start),
      .c_data_valid(c_data_valid) );
/*********************************************************/
initial // Clock generator
  begin
    #0 clk = 0;
    forever #10 clk = !clk;
  end
/*********************************************************/
assign tx_clk = serdes0.clks_in_wire[0];
assign mode = config_in[2 : 0];
assign num_errors = digital_out[5 : 0];
task parallel (
      input[DATA_WIDTH-1 : 0] data );
begin
            txd_data_in = data;
            repeat(DATA_WIDTH+1)
      @(negedge tx_clk);
end
endtask

// input generation
initial
begin
```

```
  #0
  txd_data_in = 9'b0_1111_1100;
//Mode 02
      config_in=8'b0000_0010;
      test_en = 1'b1;
// reset
      rst = 1'b0;
      #(`DELAY*10) rst = 1'b1;
//Send Comma
// tx_data input generation
   parallel(9'd0); //Wait the 0 transmission
      parallel(9'b1_1111_1100); //Sending the first comma to start
      parallel(9'd250);
      //parallel(9'd252);
      parallel(9'd254);
      parallel(9'b1_1111_1100); //Sending a second comma to end
  parallel(9'b0_0000_0111);
      parallel(9'b1_1111_1100); //Sending the first comma to start
      for (i = 0; i < 2; i = i + 1)
            parallel(i[DATA_WIDTH-1 : 0]); //Send messages from 0 to 61
      parallel(9'b1_1111_1100); //Sending another comma to end
      for (i = 0; i < 62; i = i + 1)
            parallel(i[DATA_WIDTH-1 : 0]); //Send messages from 0 to 61
      #(`DELAY*1000);
      $stop();
//END MODE 1
end
endmodule
```

G3. TEST OPERATING MODE 3 TESTBENCH

```
// Name: SerDes_m3_tb.v
`timescale 1ns/1ps
`define DELAY 10

module SerDes_m3_tb;

localparam DATA_WIDTH = 9;

//Inputs
reg rst;
reg clk;
reg rxa_in_p;
wire rxa_in_n;
wire [7 : 0] config_in;
reg [DATA_WIDTH-1 : 0] txd_data_in;
reg test_en;
//Outputs
wire [DATA_WIDTH-1 : 0] digital_out;
wire txa_data_out_p;
wire txa_data_out_n;
wire txd_data_out;
wire test_out;
wire tx_frame_start;
wire c_data_valid;

//Other
wire [5:0]  num_errors;
wire [2:0] mode;//Bist Configuration pins
wire tx_clk;

integer i;

//Module instantiation.
SERDES #(.WIDTH(9))
serdes0
(
//INPUTS
      .reset(rst),
      .clk(clk),
      .rxa_in_p(rxa_in_p),
      .rxa_in_n(rxa_in_n),
      .config_in(config_in),
      .txd_data_in(txd_data_in),
      .test_en(test_en),
//OUTPUTS
      .digital_out(digital_out),
      .txa_data_out_p(txa_data_out_p),
      .txa_data_out_n(txa_data_out_n),
      .txd_data_out(txd_data_out),
      .test_out(test_out),
      .tx_frame_start(tx_frame_start),
      .c_data_valid(c_data_valid)
);

/*******************************************************/
initial // Clock generator
  begin
    #0 clk = 0;
    forever #10 clk = !clk;
  end
/*******************************************************/

assign tx_clk = serdes0.clks_in_wire[0];
assign mode = config_in[2 : 0];
assign num_errors = digital_out[5 : 0];

task parallel
(
      input[DATA_WIDTH-1 : 0] data
);
begin
            txd_data_in = data;
            repeat(DATA_WIDTH+1)
      @(negedge tx_clk);
end
endtask

// input generation
initial
begin
  #0 txd_data_in = 9'h0;
//Mode 02
      test_en = 1'b1;

// reset
      rst = 1'b0;
      #(`DELAY*10) rst = 1'b1;
      //@(posedge tx_clk);

//Send Comma
// tx_data input generation
      parallel(9'b1_1111_1100); //Sending the first comma to start
      for (i = 0; i < 62; i = i + 1)
            parallel(i[DATA_WIDTH-1 : 0]); //Send messages from 0 to 61
      parallel(9'b1_1111_1100); //Sending another comma to end
      for (i = 0; i < 62; i = i + 1)
            parallel(i[DATA_WIDTH-1 : 0]); //Send messages from 0 to 61
      #(`DELAY*1000);
      $stop();
//END MODE 1
end

assign config_in = { 4'd1, !txd_data_out, 3'd3};

endmodule
```

G4. TEST OPERATING MODE 4 TESTBENCH

```
// Name: SerDes_m4_tb.v
`timescale 1ns/1ps
`define DELAY 10

module SerDes_m4_tb;

localparam DATA_WIDTH = 9;
integer flag = 1;

//Inputs
reg rst;
reg clk;
reg rxa_in_p;
wire rxa_in_n;
reg [7 : 0] config_in;
reg [DATA_WIDTH-1 : 0] txd_data_in;
reg test_en;
//Outputs
wire [DATA_WIDTH-1 : 0] digital_out;
wire txa_data_out_p;
wire txa_data_out_n;
wire txd_data_out;
wire test_out;
wire tx_frame_start;
wire c_data_valid;

//Other
wire [5:0]  num_errors;
wire [2:0] mode;//Bist Configuration pins
wire tx_clk;

integer i;

//Module instantiation.
SERDES #(.WIDTH(9))
serdes0
(
//INPUTS
      .reset(rst),
      .clk(clk),
      .rxa_in_p(rxa_in_p),
      .rxa_in_n(rxa_in_n),
      .config_in(config_in),
      .txd_data_in(txd_data_in),
      .test_en(test_en),
//OUTPUTS
      .digital_out(digital_out),
      .txa_data_out_p(txa_data_out_p),
      .txa_data_out_n(txa_data_out_n),
      .txd_data_out(txd_data_out),
      .test_out(test_out),
      .tx_frame_start(tx_frame_start),
      .c_data_valid(c_data_valid)
);

/**********************************************************/
initial // Clock generator
  begin
    #0 clk = 0;
    forever #10 clk = !clk;
  end
/**********************************************************/

assign tx_clk = serdes0.clks_in_wire[0];
assign mode = config_in[2 : 0];
assign num_errors = digital_out[5 : 0];

initial
begin
  #0
      //Mode 4
      config_in=8'b0000_0100; //config_in[4] (errors_en)
      test_en = 1'b1;
      // reset
      rst = 1'b0;
      #(`DELAY*10) rst = 1'b1;
      //Send Comma
      @(posedge tx_clk);
      // tx_data input generation
      for (i = 0; i < 62; i = i + 1)
            @(posedge tx_clk)
      #(`DELAY*1000);
      $stop();
end

initial
begin
    forever
    begin
      @(negedge c_data_valid)
      if(!config_in[4] && digital_out == 9'h1fc) begin
        if(flag == 2) begin
          config_in[4] = 1;
          flag = 1;
          $stop();
        end
        else
          flag = 2;
      end
      else if(config_in[4] && flag == 1) begin
        config_in[4] = 0;
        flag = 2;
      end
    end
end
endmodule
```

G5. TEST OPERATING MODE 5 TESTBENCH

```
// Name: SerDes_m5_tb.v
`timescale 1ns/1ps
`define DELAY 10

module SerDes_m5_tb;

localparam DATA_WIDTH = 9;

//Inputs
reg rst;
reg clk;
reg rxa_in_p;
wire rxa_in_n;
reg [7 : 0] config_in;
reg [DATA_WIDTH-1 : 0] txd_data_in;
reg test_en;
//Outputs
wire [DATA_WIDTH-1 : 0] digital_out;
wire txa_data_out_p;
wire txa_data_out_n;
wire txd_data_out;
wire test_out;
wire tx_frame_start;
wire c_data_valid;

//Other
wire [5:0]  num_errors;
wire [2:0] mode;//Bist Configuration pins
wire tx_clk;

integer i;

//Module instantiation.
SERDES #(.WIDTH(9))
serdes0
(
//INPUTS
      .reset(rst),
      .clk(clk),
      .rxa_in_p(rxa_in_p),
      .rxa_in_n(rxa_in_n),
      .config_in(config_in),
      .txd_data_in(txd_data_in),
      .test_en(test_en),
//OUTPUTS
      .digital_out(digital_out),
      .txa_data_out_p(txa_data_out_p),
      .txa_data_out_n(txa_data_out_n),
      .txd_data_out(txd_data_out),
      .test_out(test_out),
      .tx_frame_start(tx_frame_start),
      .c_data_valid(c_data_valid)
);

/*********************************************************/
initial // Clock generator
  begin
    #0 clk = 0;
    forever #10 clk = !clk;
  end
/*********************************************************/

assign tx_clk = serdes0.clks_in_wire[0];
assign mode = config_in[2 : 0];
assign num_errors = digital_out[5 : 0];

task serial
(
      input [DATA_WIDTH : 0] transmitted_data
);
begin
  for (i = 0; i <= DATA_WIDTH ; i = i + 1)
  begin
      config_in[3] = transmitted_data[i];//serial data
      @(posedge tx_clk);
  end
end
endtask

// input generation
initial
begin
  #0
//Mode 5
  config_in[2 : 0]= 3'd5;
  config_in[7 : 4] = 4'd0;
  test_en = 1'b1;
// reset
      rst = 1'b0;
      #`DELAY rst = 1'b1;
//Send Comma
      serial(10'b00_0111_1100);
      serial(10'h0b9);
      serial(10'h0ae);
      serial(10'h0ad);
      #(`DELAY*1000);
      $stop();
end

endmodule
```

G6. TEST OPERATING MODE 6 TESTBENCH

```
// Name: SerDes_m6_tb.v
`timescale 1ns/1ps
`define DELAY 10

module SerDes_m6_tb;

localparam DATA_WIDTH = 9;

//Inputs
reg rst;
reg clk;
reg [7 : 0] config_in;
reg [DATA_WIDTH-1 : 0] txd_data_in;
reg test_en;
//Outputs
wire [DATA_WIDTH-1 : 0] digital_out;
wire txa_data_out_p;
wire txa_data_out_n;
wire txd_data_out;
wire test_out;
wire tx_frame_start;
wire c_data_valid;

//Other
wire [5:0]  num_errors;
wire [2:0] mode;//Bist Configuration pins
wire tx_clk;

integer i;

//Module instantiation.
SERDES #(.WIDTH(9))
serdes0
(
//INPUTS
      .reset(rst),
      .clk(clk),
      .rxa_in_p(txa_data_out_p),
      .rxa_in_n(txa_data_out_n),
      .config_in(config_in),
      .txd_data_in(txd_data_in),
      .test_en(test_en),
//OUTPUTS
      .digital_out(digital_out),
      .txa_data_out_p(txa_data_out_p),
      .txa_data_out_n(txa_data_out_n),
      .txd_data_out(txd_data_out),
      .test_out(test_out),
      .tx_frame_start(tx_frame_start),
      .c_data_valid(c_data_valid)
);
```

```
/*******************************************************/
initial // Clock generator
  begin
    #0 clk = 0;
    forever #10 clk = !clk;
  end
/*******************************************************/

assign tx_clk = serdes0.clks_in_wire[0];
assign mode = config_in[2 : 0];
assign num_errors = digital_out[5 : 0];

initial
begin
  #0
      //Mode 4
      config_in=8'b0001_0110;//config_in[4] (errors_en)
      test_en = 1'b1;
      // reset
  rst = 1'b0;
      #`DELAY rst = 1'b1;
      //Send Comma
      @(posedge tx_clk);
      // tx_data input generation
      for (i = 0; i < 62; i = i + 1)
            @(posedge tx_clk)
      #(`DELAY*1000);
      $stop();
end

endmodule
```

G7. TEST OPERATING MODE 7 TESTBENCH

```
// Name: SerDes_m7_tb.v
`timescale 1ns/1ps
`define DELAY 10

module SerDes_m7_tb;

localparam DATA_WIDTH = 9;

//Inputs
reg rst;
reg clk;
wire rxa_in_p;
wire rxa_in_n;
reg [7 : 0] config_in;
reg [DATA_WIDTH-1 : 0] txd_data_in;
reg test_en;
//Outputs
wire [DATA_WIDTH-1 : 0] digital_out;
wire txa_data_out_p;
wire txa_data_out_n;
wire txd_data_out;
wire test_out;
wire tx_frame_start;
wire c_data_valid;

//Other
wire [5:0]  num_errors;
wire [2:0] mode;//Bist Configuration pins
wire tx_clk;

integer i;

//Module instantiation.
SERDES #(.WIDTH(9))
serdes0 (
//INPUTS
      .reset(rst),
      .clk(clk),
      .rxa_in_p(rxa_in_p),
      .rxa_in_n(rxa_in_n),
      .config_in(config_in),
      .txd_data_in(txd_data_in),
      .test_en(test_en),
//OUTPUTS
      .digital_out(digital_out),
      .txa_data_out_p(txa_data_out_p),
      .txa_data_out_n(txa_data_out_n),
      .txd_data_out(txd_data_out),
      .test_out(test_out),
      .tx_frame_start(tx_frame_start),
      .c_data_valid(c_data_valid)
);
```

```
/*********************************************************/
initial // Clock generator
  begin
    #0 clk = 0;
    forever #10 clk = !clk;
  end
/*********************************************************/

assign tx_clk = serdes0.clks_in_wire[0];
assign mode = config_in[2 : 0];
assign num_errors = digital_out[5 : 0];

task parallel
(
      input[DATA_WIDTH-1 : 0] transmit_data
);
begin
            txd_data_in = transmit_data;
            repeat(DATA_WIDTH+1)
      @(negedge tx_clk);
end
endtask

// input generation
initial
begin
  #0
  txd_data_in = 9'b1_1111_1100;
//Mode 07
      config_in=8'd7;
      test_en = 1'b1;
// reset
      rst = 1'b0;
      #(`DELAY*10) rst = 1'b1;
      //@(posedge tx_clk);
//Send Comma
// tx_data input generation
      parallel(9'b1_1111_1100); //Sending the first comma to start
      for (i = 0; i < 62; i = i + 1)
            parallel(i[DATA_WIDTH-1 : 0]); //Send messages from 0 to 61
      parallel(9'b1_1111_1100); //Sending another comma to end

      for (i = 0; i < 62; i = i + 1)
            parallel(i[DATA_WIDTH-1 : 0]); //Send messages from 0 to 61
      #(`DELAY*1000);

      $stop();
end

assign rxa_in_n = txd_data_out;
assign rxa_in_p = !txd_data_out;

endmodule
```

H1. SIGNAL DRIVER VERSION 1 RTL

```
/////////////////////////////////////////////////////////////
// Module: signal_driver
// Author: Cesar Limones 2016
/////////////////////////////////////////////////////////////
/***************************************************************************
*Name: SignalDriverS.v
*Editor: Miguel Mihail Hoil Loria.
**************************************************************************/
module SignalDriverS # ( parameter WIDTH = 9) (
input [2 : 0] mode,
input test_in, errors_en, rxa_out,
input [WIDTH - 1 : 0] rxd_data_out,
input code_err, input disp_err, input dispout,txd_data_out,
input [ WIDTH - 1 : 0 ] txd_data_in, lfsr_parallel_out,
input bist_end,
input [5 : 0] num_errors,
output rxd_input_muxout, txa_input_muxout, test_out
output [WIDTH - 1 : 0] txd_input_muxout, rxd_output_muxout );
wire rxd_input_muxout_wire, txa_input_muxout_wire, test_out_wire;
wire [WIDTH-1 : 0] txd_input_muxout_wire, rxd_output_muxout_wire;
wire rxd_input_bitZero_muxsel_wire, rxd_input_bitOne_muxsel_wire;
wire rxd_muxout_A_wire, txd_input_bitZero_muxsel_wire;
wire txd_input_bitOne_muxsel_wire;
wire [WIDTH-1 : 0] txd_muxout_A_wire, rxd_muxinput_wire;
wire txa_input_muxsel_wire, test_out_muxsel_wire, rxd_output_muxsel_wire;
wire mZero_inv, mOne_Xor_mTwo, mZeroInv_And_mTwo;
//-------------Code Starts Here-------
Mux2to1 #( .WORD_LENGTH(1) ) RXD_input_A_MUX (
.Selector(rxd_input_bitZero_muxsel_wire),
.Data_0(rxa_out), .Data_1(txd_data_out),
.Mux_Output(rxd_muxout_A_wire) );
Mux2to1 #( .WORD_LENGTH(1) ) RXD_input_B_MUX (
.Selector(rxd_input_bitOne_muxsel_wire),
.Data_0(rxd_muxout_A_wire), .Data_1(test_in),
.Mux_Output(rxd_input_muxout_wire) );
Mux2to1 #( .WORD_LENGTH(WIDTH) ) TXD_input_A_MUX (
.Selector(txd_input_bitZero_muxsel_wire),
.Data_0(txd_data_in), .Data_1(rxd_data_out), .Mux_Output(txd_muxout_A_wire));
Mux2to1 #( .WORD_LENGTH(WIDTH) ) TXD_input_B_MUX (
.Selector(txd_input_bitOne_muxsel_wire),
.Data_0(txd_muxout_A_wire), .Data_1(lfsr_parallel_out),
.Mux_Output(txd_input_muxout_wire) );
Mux2to1 #( .WORD_LENGTH(1) ) TXA_input_MUX (
.Selector(txa_input_muxsel_wire),
.Data_0(txd_data_out), .Data_1(rxa_out), .Mux_Output(txa_input_muxout_wire));
Mux2to1 #( .WORD_LENGTH(1) ) Test_out_MUX (
.Selector(test_out_muxsel_wire), .Data_0(rxa_out), .Data_1(bist_end),
.Mux_Output(test_out_wire) );
Mux2to1 #( .WORD_LENGTH(WIDTH) ) RXD_output_MUX (
.Selector(rxd_output_muxsel_wire),
.Data_0(rxd_data_out), .Data_1(rxd_muxinput_wire),
.Mux_Output(rxd_output_muxout_wire) );
```

```
//-------------Wire assignations-------
assign rxd_muxinput_wire = {disp_err, code_err, dispout, num_errors[5:0]};
assign mZero_inv = !mode[0];
assign mOne_Xor_mTwo = mode[1] ^ mode[2];
assign mZeroInv_And_mTwo = mZero_inv & mode[2];
assign rxd_input_bitZero_muxsel_wire = mZero_inv & mOne_Xor_mTwo;
assign rxd_input_bitOne_muxsel_wire = mode[0] & mOne_Xor_mTwo;
assign txd_input_bitZero_muxsel_wire = mode[0] & !mode[1];
assign txd_input_bitOne_muxsel_wire = mZeroInv_And_mTwo;
assign txa_input_muxsel_wire = &mode;
assign test_out_muxsel_wire = mZeroInv_And_mTwo;
assign rxd_output_muxsel_wire = mZeroInv_And_mTwo & errors_en;
assign rxd_output_muxout = rxd_output_muxout_wire;
assign rxd_input_muxout = rxd_input_muxout_wire;
assign txd_input_muxout = txd_input_muxout_wire;
assign txa_input_muxout = txa_input_muxout_wire;
assign test_out = test_out_wire;
endmodule
```

H2. TEST MODULES VERSION 1 RTL

```
////////////////////////////////////////////////////////////////
// Module: test_modules
// Author: Cesar Limones 2016
/////////////////////////////////////////////////////////////////
/***********************************************************************
*Name: test_modules.v
*Editor: Miguel Mihail Hoil Loria.
***********************************************************************/
module test_modules #(parameter WIDTH = 9, parameter DECODED_COMMA = 9'h1FC)
(
input rst, ser_clk, test_en, lfsr_en,
input [ 7 : 0 ] config_in,
input rxa_out, rxa_out_n,
input [WIDTH - 1 : 0] rxd_data_out, txd_data_in,
input c_data_valid, code_err, disp_err, dispout, txd_data_out,
output rxd_input_muxout,
output [ WIDTH - 1 : 0 ] txd_input_muxout, rxd_output_muxout,
output txa_input_muxout, test_out,
output [ 7 : 0 ] tx_config );
wire bist_on_wire, txd_frame_start_wire, txd_frame_start_regOutA_wire;
wire txd_frame_start_regOutB_wire, rxd_data_ready_wire, bist_end_wire;
wire [ WIDTH - 1 : 0 ] lfsr_parallel_out_wire;
wire [ 5 : 0 ] num_errors_wire;
reg [2 : 0] mode_reg;
reg test_in_reg, errors_en_reg;
reg [7 : 0] tx_config_reg;
always@(test_en or config_in)
begin
      if (test_en) begin
            mode_reg = config_in[2:0];
            test_in_reg = config_in[3];
            errors_en_reg = config_in[4];
            tx_config_reg = 8'b0;
      end
      else begin
            mode_reg = 3'b000;
            test_in_reg = 1'b0;
            errors_en_reg = 1'b0;
            tx_config_reg = config_in;
      end
end
Registro #(1) buffer_rxa_out_p_register (
.clk(ser_clk), .reset(rst), .enable(1'b1), .Data_Input(rxa_out),
.Data_Output() );
Registro #(1) buffer_rxa_out_n_register (
.clk(ser_clk), .reset(rst), .enable(1'b1), .Data_Input(rxa_out_n),
.Data_Output() );
SignalDriverS #(9) signal_driver_unit (
.mode(mode_reg), .test_in(test_in_reg), .errors_en(errors_en_reg),
.rxa_out(rxa_out), .rxd_data_out(rxd_data_out), .code_err(code_err),
.disp_err(disp_err),    .dispout(dispout), .txd_data_in(txd_data_in),
.txd_data_out(txd_data_out), .lfsr_parallel_out(lfsr_parallel_out_wire),
.bist_end(bist_end_wire), .num_errors(num_errors_wire),
.rxd_input_muxout(rxd_input_muxout), .txd_input_muxout(txd_input_muxout),
.txa_input_muxout(txa_input_muxout), .rxd_output_muxout(rxd_output_muxout),
.test_out(test_out) );
Registro #(1) txd_frame_start_registerA (
.clk(ser_clk), .reset(rst), .enable(bist_on_wire),
.Data_Input(txd_frame_start_wire),
.Data_Output(txd_frame_start_regOutA_wire) );
Registro #(1) txd_frame_start_registerB (
.clk(ser_clk), .reset(rst), .enable(bist_on_wire),
.Data_Input(txd_frame_start_regOutA_wire),
.Data_Output(txd_frame_start_regOutB_wire) );
LFSR #( .seed({1'b1,DECODED_COMMA}) ) lfsr_unit (
.rst(rst), .clk(ser_clk), .lfsr_en(txd_frame_start_regOutB_wire),
.lfsr_parallel_out(lfsr_parallel_out_wire) );
ComparatorP #( .WIDTH(WIDTH), .COMMA(DECODED_COMMA)//9 bits Decoded comma )
comparator_unit (
.clk(ser_clk), .rst(rst), .dataA_in(txd_input_muxout),
.dataB_in(rxd_data_out), .lfsr_en(txd_frame_start_wire),
.data_valid_pipe(rxd_data_ready_wire),
.bist_end(bist_end_wire), .num_errors(num_errors_wire) );
assign txd_frame_start_wire = lfsr_en & bist_on_wire;
assign rxd_data_ready_wire = c_data_valid & bist_on_wire;
assign bist_on_wire = test_en & mode_reg[2] & !mode_reg[0];
assign tx_config = tx_config_reg;
endmodule
```

References

[1] L. Carbonell Soto, *Fundamentos de informática*. Universidad de Alicante, 1998.

[2] J. L. Vaquero Antonio, *INFORMATICA: GLOSARIO DE TERMINOS Y SIGLAS*. McGraw-Hill, 1985.

[3] A. Athavale and C. Christensen, *High-Speed Serial I/O Made Simple. A Designer's Guide, with FPGA Applications*, 1st ed. Connectivity Solutions.

[4] A. Patel, "Planet Analog - Articles - The basics of SerDes (serializers/deserializers) for interfacing," 16-Sep-2010. [Online]. Available: http://www.planetanalog.com/document.asp?doc_id=528099. [Accessed: 08-Sep-2017].

[5] R. B. Thompson, *PC Hardware in a Nutshell /*, 3a edición. O'Reilly, 2003.

[6] M. L. Bushnell, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits /*. Kluwer, 2000.

[7] L.-T. Wang, C.-W. Wu, and X. Wen, Eds., *VLSI test principles and architectures: design for testability*. Amsterdam; Boston: Elsevier Morgan Kaufmann Publishers, 2006.

[8] Kenneth William Ferguson, Paul Laprise, and Chris Siu, "Built in self test (BIST) for high-speed serial transceivers.", U.S. Patent No. 7,756,197 B1, July 13, 2010.

[9] G. Jervan, "Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems.", PhD thesis, Linköpings universitet INSTITUTE OF TECHNOLOGY, Linköping, Sweden, 2006.

[10] S. Sunter, A. Roy, and J. F. Cote, "An automated, complete, structural test solution for SERDES," in *2004 International Conferce on Test*, 2004, pp. 95–104.

[11] W.-Z. Chen and G.-S. Huang, "A low power programmable PRBS generator and a clock multiplier unit for 10 Gbps serdes applications," in *2006 IEEE International Symposium on Circuits and Systems*, 2006, p. 4 pp.-.

[12] R. Godínez-Maldonado, "Test Module Design for ITESO TV1 SerDes", Guadalajara, Jalisco, Mexico, Project Report ITESO, December 2015.

[13] C. F. Limones-Mora, "Test Modules Design for a SerDes Chip in 130 nm CMOS technology", Guadalajara, Jalisco, Mexico, Project Report ITESO, July 2016.

[14] Davide Tonietto, John Hogeboom, "SERDES with jitter-based built-in self test (BIST) for adapting FIR filter coefficients.", US Patent No. 8,228,972 B2, July 24, 2012.

[15] Om P. Agrawal, Jock Tomlinson, Kuang Chi, Ji Zhao, Ju Shen, Jinghui Zhu, "SERDES with programmable I/O architecture.", US Patent No. 7,208,975 B1, April 24, 2007

[16] M. Maadi, "*An 8b/10b Encoding Serializer/Deserializer (SerDes) Circuit for High Speed Communication Applications Using a DC Balanced, Partitioned-Block, 8b/10b T*", vol. 3. 2015.

[17] D. Lewis, N. Semiconductor, *"SerDes Architectures and Applications"* in *DesignCon 2004*.

[18] Lattice Semiconductor, “8b10b Encoder Decoder Documentation" January 2015. [Online]. Available: http://www.latticesemi.com/~/media/LatticeSemi/Documents/ReferenceDesigns/1D/8b10bEncoderDecoder-Documentation.pdf?document_id=5653. [Accessed: 27-Sep-2017].

[19] Y. Yu, Z. Yang, X. Peng, and D. Xu, “Efficient concurrent BIST with comparator-based response analyzer,” in *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2013, pp. 1115–1119.

[20] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing & Testable Design*, 1 edition. New York: Wiley-IEEE Press, 1994.

[21] H. Iwata, S. Satonaka, and K. Yamaguchi, “An Efficient Test Pattern Generator-Mersenne Twister.”, Department of Information Engineering, Faculty of Advanced Engineering, Nara National College of Technology, 2013.

[22] I. Ghosh, N. K. Jha, and S. Bhawmik, “A BIST scheme for RTL circuits based on symbolic testability analysis,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 1, pp. 111–128, Jan. 2000.

[23] R. Rivas-Villegas, "*Design, Implementation and Verification of a Deserializer Module for a SerDes Mixed Signal System on Chip in 130 nm CMOS Technology"*, Guadalajara, Jalisco, Mexico, Project Report ITESO, August 2016.