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

DIGITAL SERIALIZER DESIGN FOR A SERDES CHIP IN 130NM CMOS TECHNOLOGY

Tesina para obtener el grado de:

ESPECIALISTA EN DISEÑO DE SISTEMAS EN CHIP

Presentan: Christian Aparicio Zuleta

Tutor: Manuel Salim, Victor Avendaño

San Pedro Tlaquepaque, Jalisco. Noviembre de 2017.

Acknowledgments

Foremost, I would like to thank my parents Alma Zuleta and Jorge Aparicio for being a role model for me and for their support on every decision I have made through my entire life. Your example has pushed me to never quit.

Thanks to my brother Erick for acting as my mentor and for demonstrating me how we are able to reach our goals, no matter how ambitious they sound at the beginning.

To my mentors, Dr. Manuel Salim and Dr. Victor Avendaño for sharing their expertise and guidance through this long process.

Thanks to my colleagues Miguel Hoil, Alex Meling, and Oscar Toledo for their dedication and commitment in this project.

My sincere thanks also to Dr. Cuauhtemoc Aguilera for his valuable contributions.

And finally, to CONACYT for promoting the postgraduate studies in our country, and providing me and so many other Mexican students the opportunity of being part of this institution, believing that we are able to stand out in the world.

Abstract

The development of this project is derived from the effort of previous generations from the System on Chip Design Specialty Program at ITESO, who have pioneered the creation of a serializer-deserializer device for high-speed communications in CMOS technology, aiming towards a small and efficient device. The design flow and enhancements implemented within the digital serializer module of the SerDes system, consists of an 8b10b encoder followed by a parallel to serial converter that together reaches a maximum frequency of 239 MHz in a typical cmrf8sf (130 nm) technology manufacturing process, implemented with Cadence tools. The rtl and testbench were taken from the work of Efrain Arrambide, adding a register to store the current disparity value, and thus, enhance the code by adding primitive blocks to improve the behavior of the serializer module and the validation process, generating a summary for every run. The system on chip flow is followed by choosing the variables that best fit the design and a layout with no design violations is generated during the physical synthesis. The individual module layouts were completed successfully in terms of behavior and violations, while the integration of the mixed signal device showed errors that were not resolved in time for manufacturing.

Resumen

El desarrollo de este proyecto parte del trabajo realizado por las generaciones anteriores de la especialidad de diseño de circuitos integrados del ITESO, quienes fueron pioneros en la creación de un dispositivo para comunicaciones de alta velocidad en tecnología CMOS, con el objetivo de obtener un producto final pequeño y eficiente. El flujo de diseño y mejoras implementadas al módulo serializador digital del sistema SerDes, el cual consiste en un codificador 8b10b seguido de un convertidor de datos de paralelo a serial, alcanzan una frecuencia máxima de 239 MHz al ser fabricado y operado en condiciones típicas con la tecnología cmrf8sf (130 nm), además de ser implementado con las herramientas proveídas por Cadence. El código de descripción de hardware y banco de pruebas fueron tomados originalmente de los entregados por Efrain Arrambide, a lo que se le agregó un registro para almacenar el valor de la disparidad del dato enviado, así como la adición de bloques básicos para mejorar el comportamiento y se simplificó el código Verilog. El proceso de validación fue mejorado de tal manera que se prueban bloques por separado y cada iteración genera un registro de transacciones y un resumen al final con los resultados de manera automática para cada iteración. El flujo del diseño de sistemas en chip fue seguido por completo, eligiendo las variables que mejor se adaptan a la respuesta y especificaciones del sistema, así como buscar que genere ninguna violación en el diseño físico. Los distintos bloques del sistema serializador-deserializador fueron diseñados y verificados con éxito, sin embargo, la integración del sistema de señal mixta no fue completada debido a errores que no se lograron resolver a tiempo para cumplir con la fecha de fabricación.

List of Figures

Figure 1-1. SerDes top level architecture. ……19
Figure 1-2. Serializer expected behavior. 20
Figure 1-3. SGMII Connectivity. 22
Figure 1-4. Digital serializer architecture. 23
Figure 1-5. 8b/10b encoding high level scheme. 25
Figure 1-6: 8b/10b encoder block diagram. 26
Figure 1-7. Serialization behavior. ……28
Figure 2-1. Encoder response for control values and disparity 0. 29
Figure 2-2. Encoder response for control values and disparity 1. 30
Figure 2-3. Encoder control data report. 30
Figure 2-4. Encoder first synthesis. 30
Figure 2-5. Serialization block first synthesis. 31
Figure 2-6. Control unit synthesis. 32
Figure 2-7. Comparator first synthesis. 32
Figure 2-8. Comparator waveform results. 33
Figure 2-9. Serializer simulation report. 33
Figure 2-10. Top module serialization waveforms. 34
Figure 2-11. Parallel to serial waveform. 35
Figure 3-1. Schematic representation for top module grouped synthesis. 38
Figure 3-2. Schematic representation of data input register grouped synthesis. 39
Figure 3-3. Schematic representation of encoder grouped synthesis. 39
Figure 3-4. Schematic representation of serialization top module grouped synthesis. 40
Figure 3-5. Schematic representation of PISO control unit grouped synthesis. 40
Figure 3-6. Schematic representation of comparator against 8 grouped synthesis. 40
Figure 3-7. Schematic representation of comparator against 9 grouped synthesis. 41
Figure 3-8. Schematic representation of comparator against 10 grouped synthesis. 41
Figure 3-9. Schematic representation of ungrouped synthesis. 42
Figure 3-10. Static timing analysis histogram results. 44
Figure 3-11. Worst slack timing path. 45
Figure 3-12. Positive-edge FFD with active-low reset schematic representation. 46
Figure 3-13. Schematic representation for AOI3 standard cell. 46
Figure 3-14. RTL against intermediate LEC. 48
Figure 3-15. Serializer gln and rtl waveform comparison 49
Figure 4-1. Encounter import design. 52
Figure 4-2. Serializer die initial area. 52
Figure 4-3. Serializer floorplan area. 53
Figure 4-4. Serializer power rings. 54
Figure 4-5. Serializer horizontal stripes. 55
Figure 4-6. Serializer power grid. 56
Figure 4-7. Serializer standard cells placement. 57
Figure 4-8. Serializer clock tree synthesis. 58
Figure 4-9. Serializer connectivity and antenna verifications 59

Figure 4-10. Serializer physical synthesis layout with timing optimization. 61
Figure 4-11. Serializer floorplan view. 62
Figure 4-12. Serializer standard cells and filler cells. 62
Figure 5-1. SerDes top level architecture. 65
Figure 5-2. Virtuoso final schematic with violations 67

List of Tables

Table 1-1. Special character encoding. 25
Table 1-2. Description of serialization signals 27
Table 3-1. Typical synthesis results. 422
Table 3-2. Results of worst case synthesis. 433
Table 4-1. Place and Route timing optimization 600

List of Acronyms and Abbreviations

8b10b:	Eight bit, Ten bit
Baud:	Amount of bits per second in a communication system.
BIST:	Build-In-Self-Test
Byte:	8-bit arrange with valid values from 0 to 255
CMOS:	Complementary metal-oxide semiconductor
Core:	The central area of the silicon, where the standard cells are placed.
Die:	Total area of the silicon, it includes the I/O pads.
DIP40:	Dual in-line package with 40 pins
Disparity:	Difference in amount of ones and zeros in an array.
ESD:	Electrostatic discharge
FFD:	Flip-Flop D
GLN:	Gate Level Netlist
HDL:	Hardware Description Language
Hz:	Hertz
IC:	Integrated Circuit
IO:	Input-Output
LEC:	Logic Equivalence Check
PISO:	Parallel Input Serial Output
RTL:	Register Transfer Level
SoC:	System on Chip
STA:	Static Time Analysis
TB:	Testbench
VDC:	Voltage direct current
Vih:	Minimum input voltage level to be considered a high.
VLSI:	Very large scale integration

Content

ACKNOWLEDGMENTS ..... **III**
ABSTRACT ..... **V**
RESUMEN ..... **VI**
LIST OF FIGURES ..... **VII**
LIST OF TABLES ..... **IX**
LIST OF ACRONYMS AND ABBREVIATIONS ..... **X**
INTRODUCTION **13**
BACKGROUND **15**
CHAPTER 1. SERDES SYSTEM **19**
1.1 SGMII PROTOCOL 21
1.2 DIGITAL SERIALIZER 22
1.3 ENCODER 8B/10B 23
1.4 SERIALIZATION 27
CHAPTER 2. RTL DESIGN **29**
2.1 ENCODER 8B10B 29
2.2 SERIALIZATION 31
CHAPTER 3. LOGIC SYNTHESIS **37**
3.1 RTL COMPILER TYPICAL SYNTHESIS 38
3.2 WORST CASE SYNTHESIS 43
3.3 STATIC TIMING ANALYSIS (STA) 44
3.4 LOGIC EQUIVALENT CHECKING (LEC) 47
3.5 GATE LEVEL NETLIST SIMULATION 49
CHAPTER 4. PHYSICAL SYNTHESIS **51**
4.1 IMPORT DESIGN 51
4.2 FLOORPLAN DEFINITION 53
4.3 POWER GRID 54
4.3.1 Power Rings *54*
4.3.2 Horizontal Power Stripes *55*
4.3.3 Vertical Power Stripes *56*
4.4 PLACEMENT OF STANDARD CELLS 57
4.5 CLOCK TREE SYNTHESIS 58
4.6 DESIGN ROUTING 59
4.7 TIMING OPTIMIZATION 60
4.8 FILLER CELLS 62
4.9 PLACE AND ROUTE COMPILATION 63
CHAPTER 5. SERDES INTEGRATION **65**
CONCLUSION **69**
APPENDIX A ENCODER RTL. **71**
APPENDIX B ENCODER TESTBENCH. **73**
APPENDIX C SERIALIZATION MODULE RTL. **75**
APPENDIX D CONTROL UNIT RTL. **77**
APPENDIX E COMPARATOR RTL. **78**
APPENDIX F COMPARATOR TESTBENCH. **79**
APPENDIX G TOP MODULE TESTBENCH. **80**
APPENDIX H SERIALIZER_PERFORMANCE_SYNTHESIS.TCL **82**
APPENDIX I SERIALIZER_OPT.SDC **86**
APPENDIX J PHYSICAL SYNTHESIS ENCOUNTER SCRIPT **88**
REFERENCES **91**

Introduction

Throughout the years, electronic systems have been continuously evolving and as time goes by, technological companies keep enhancing designs to improve performance and lower manufacturing costs by considering a smaller area inside an integrated circuit. The idea of implementing high speed Serializers/Deserializers (SerDes) systems is one of several key facts that have allowed the development of technology in terms of performance and capabilities. This has endorsed the creation of electronic devices, such as computers, smartphones, tablets, servers, gadgets and even cars, providing valuable tools for daily life activities.

Nowadays, SerDes systems are commonly implemented at different levels of communication in electronic systems. For instance, in interconnections between two Systems on Chip (SoC) or even inside an Integrated Circuit (IC) for the data transfer between dies or modules, which are continuously sending a great amount of data represented by bits. Considering that digital technology is ruled by binary codes (ones and zeros), a bit is the basic unit, defined as a logical "1" or "0" depending on its analog voltage in reference to a threshold region voltage. For example, when a device has a supply voltage of 5 Voltage Direct Current (VDC), the mentioned region will embrace from 2.4 to 2.6 VDC, in such a way that every input higher than 2.6 VDC (Vih) shall be considered as 1. So, by having a signal continuously toggling from 0 to 1 and vice versa, the product can be smart enough to detect changes meaning that the analog design is also needed for the development of a SerDes [1].

SerDes system is mainly composed of two modules: a serial to parallel converter and a parallel to serial converter, granting the module the ability of embedding a clock signal by adding an 8b10b encoder, which is responsible for translating a byte (8-bit arrange) into a new code of 10 bits with at least three transitions between 1 and 0 [2]. By avoiding parallel links between dies during integration, electronic designers and architects can considerably reduce area, noise, capacitive elements, power consumption, data integrity, and manufacturing costs [3] [4] [5].

The full SerDes IC design is formed by 5 different stages: analog receiver, digital receiver (deserialization), analog transmitter, digital transmitter (Serializer) and Built-In-Self-Test (BIST). These stages can be designed independently following desired specifications. Once all 5 functional blocks are together inside a SoC, a full-duplex communication channel between devices is possible, where both ends can send and receive information simultaneously, replacing parallel data buses and filling the gap that remains by a single ended signal bus, using a CMOS 130nm technology integrated in a DIP40 package and working with a supply voltage of 1.2 VDC.

Likewise, the digital transmission module or serializer is a parallel input serial output (PISO) block, which is the first stage of the transfer of data. It is subdivided into two smaller components that allow a byte to be sent through a single wire. The first stage of the serializer block is known as an 8b10b encoder while the second, delivers the output of the encoder to a single pin at a given frequency.

Despite having available a CMOS technology of 130nm, which was first used at the beginning of the century, the main purpose of this development is to embrace the complete design process of an IC, starting from the research phase all the way through the manufacturing of a SerDes chip. This system is an adequate choice as it needs both, analog and digital design, followed by its integration. Meanwhile, the approach of this project is to make the digital engineering of a digital transmission block and its proper pre-silicon verification, obeying the design rules for the selected technology within the base libraries provided by IBM.

The process starts with the creation of a Register Transfer Level (RTL), which is a high-level description of signals and blocks of the behavior of the hardware, done with Verilog hardware description language (HDL). Cadence tools are used for validation (Simvision), synthesis (ncverilog), static time analysis (STA), place and route for layout design with the corresponding verification and finally, the integration with the rest of the modules. The final design is intended to be manufactured by MOSIS in California.

Background

Fifty two years ago, and five years after the IC invention, Gordon Moore established that the future of electronics would be dictated by the evolution of integrated circuits where he introduced Moore's Law, which states that the number of components (transistors) within a single chip would increase at a rate of roughly a factor of 2 per year [6]. Ten years later, he realized that his hypothesis was partially wrong in terms of rate, declaring that computer performance was being doubled every 18 months, so he reformulated the law to double the components inside a single IC within a 2-year time frame [7]. Moore's assumption turned out to be true by at least 50 years since transistors are slowly reaching their lowest size limits [8].

The easiest way of transferring data between electrical components will always be to directly connect each output lane in a chip with the corresponding input lane of the next. Unfortunately, the density of a pin has not evolved as fast as the components themselves, making the reduction of area per transistor useless because the area used by their connector was larger [9]. As a result, the duty of decreasing the number of pins between components first appeared and serial communications ruled by various protocols were introduced in multiple designs, providing systems with the capabilities of increasing the data transfer frequency, and the clock skew reduction that refers to the difference in arrival time of a clock edge into any pair of flip-flops.

Later, a new disadvantage of using parallel communication systems came to life restricting the maximum clock frequency. As both chips work at the same clock signal leading to a synchronous device, the data inputs of the receiver device must meet setup and holding times of the signal, in addition in order to allow different delays of the clock distribution in both ends of the communication [9]. In other words, by meeting these timing factors, flip-flops are able to operate at high speeds considering that input data must arrive at a given setup time before the clock rising edge and be held more than a hold time so that it can be safely stored while testing with the worst case scenario parameters [10].

From the beginning, serial communication systems have been adopted by several companies and researchers, creating digital solutions with two main purposes: reduce manufacturing costs and enhance performance (speed).

This is the third time that a SerDes system has been selected as a thesis for the degree of System on Chip specialist at ITESO, so the intention is to retake both previous projects and enhance them with the proposal of new features and improvements in terms of performance. Previous jobs are focused on the employment of the PCI protocol and this time the communication interface is SGMII. In the case of a serializer module, the main difference between both protocols is the operating frequency as SGMII protocol works at 625 MHz [11].

The major changes done to the designs by Efrain Arrambide and Graciela González are related mainly to enhance validation, disparity storage, and code structure. The testing first approves the correct behavioral functionality of the 8b10b encoder, followed by the response of the serializer, then the two modules together and finally, the full design considering the disparity of the last and current character with a specific test bench for each validation stage.

A relevant feature that has been omitted by previous works, is the need to store the current disparity of the data during the transition of two encoding data flows as it has a dependency on the coming character, this for, a simple Flip-Flop D (FFD) is added at the loopback of the encoder, making the disparity value available when the following data arrives at the encoder. Furthermore, the purpose of the design remains the same, but the code is simplified to ease the debug when simulation errors appear.

Serializer modules by themselves tend to be vulnerable to failures while working at high-speed, as a minimum change in the environment can cause an unwanted behavior of the system. The well-suited solution for high-speed serial communications issues is to send each character immersed in a 10-bit packet, first proposed by A. X. Widmer and P. A. Franaszek in September 1983 [12]. This proposal from IBM researchers is commonly implemented as with combinational logic at the input of the serializer module.

Open source Verilog code of the 8b10b encoder as well as the decoder and its testbench has been developed by Chuck Benz and used for the development of the Serializer module of this thesis. A remaining task to execute in order to improve performance for an IC, which involves this large number of combinational logic, is to split with registers the longer paths of the architecture into shorter ones, making it possible to overcome speed limitations [13]. This technique is known as pipeline and is commonly used in digital design as the first alternative when gate level simulation returns a timing analysis violation, also known as negative slack.

Previous efforts on the topic have been done and mainly minor changes are proposed, hence, an exhaustive analysis of the RTL (Register Transfer Level) coding in order to detect logic that can decrease efficiency during synthesis has been performed, followed by the design stages needed in order to reflect the design in a working silicon.

Chapter 1. SerDes System

The design of this system can be adapted into several different serial communication interfaces as all share the need to transmit data through a single channel. This time, the design is done with the corresponding specifications of the SGMII protocol, which can transmit an 8-bit wide word at 625 MHz.

As seen in Figure 1-1, the full chip design is covered by 5 different stages:

- Analog receiver
- Digital receiver (Deserializer)
- Analog transmitter
- Digital transmitter (Serializer)
- BIST: Built in Self-Test

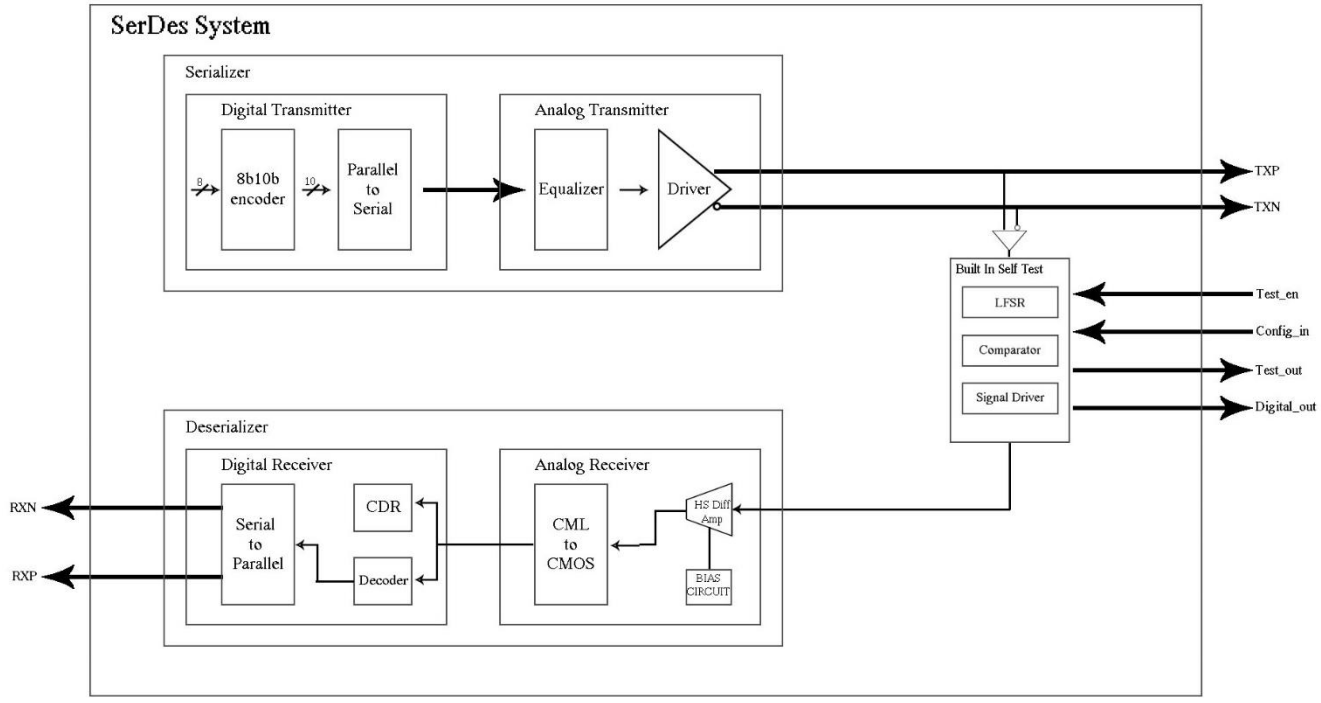

Figure 1-1. SerDes top level architecture.

The communication between digital integrated circuits involves the transference of several amounts of data represented by bits, which are considered as a logical “1” or “0” depending on its analog voltage in reference to a threshold voltage, in such a way that analog design is also needed for the development of any digital circuit.

To enable the emission of data, it is possible to utilize large amounts of communication lines, also known as parallel buses. Despite being able to transmit several bits through a parallel data bus, several significant features are affected, such as the integrity of transmitted data, area used by multiple communication lines, and power consumption. The outcome of exposing those three features is a deterioration of the final performance of the involved devices. The serializer deserializer system replaces parallel data buses and fills the gap between them by using a single-ended signal bus within a differential signal, as represented in Figure 1-2.

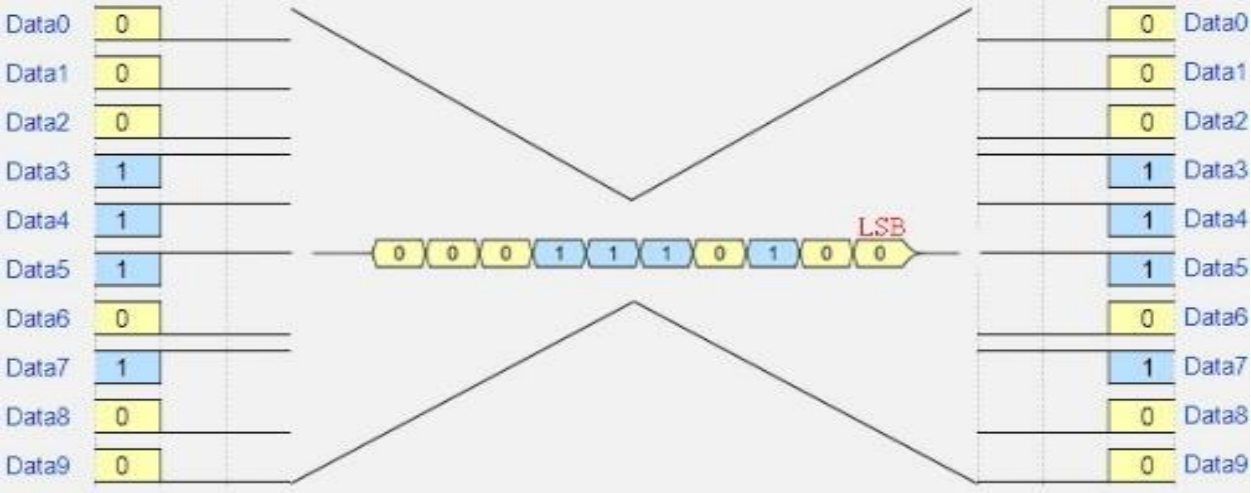

Figure 1-2. Serializer expected behavior.

The main goals of the system are:

- Set a high speed and effective channel.
- Reduce considerably the chip area.
- Enable a “Full-Duplex” communication channel.

As the BIST block has been added to the design, the working system can swipe from functional into different test modes. In such a way that each module has the option to be set as a “bypass” and they can also enable an internal connection from the serializer into the deserializer in order to create a loop, which converts the data back again from serial to parallel for verification purposes.

The device is designed using a CMOS 130nm technology, a supply voltage labeled as VDD of 1.2VDC and ground or VSS of 0V DC and will be integrated in a DIP40 packaging.

1.1 SGMII Protocol

The serial Gigabit Media Independent Interface (SGMII) is known as a digital interface that can support an 8-bit and 4-bit wide data and work either as a full-duplex or a half-duplex configuration. It was originally designed as a connection bus between Ethernet MAC and PHY abstraction layers by Cisco Systems.

A high level working diagram of the protocol is shown in , where the proper specifications of the interface are met at the hardware level by the implementation of four single ended data buses: Rx, RxClock, Tx, and TxClock powering it up to achieve operating points up to 1.25 Gbaud (Billion bits of data per second) and a clock frequency of 625MHz [11].

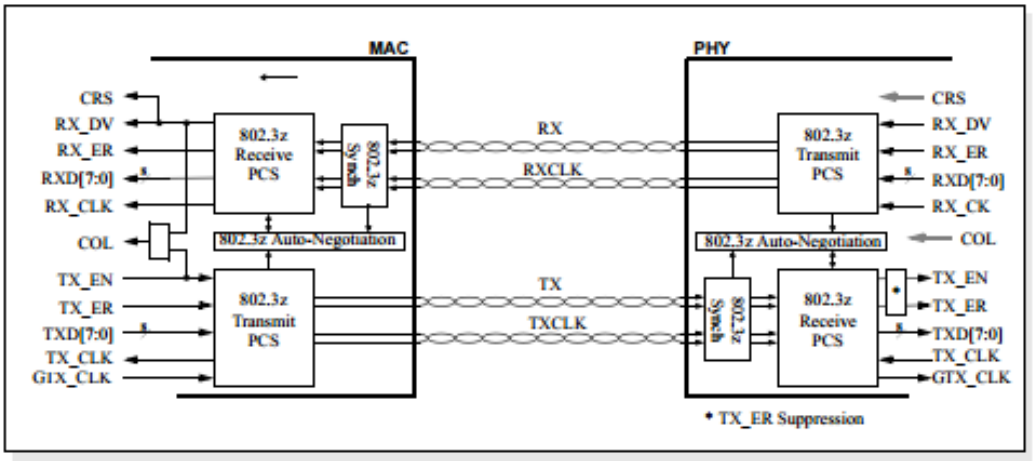

Figure 1-3. SGMII Connectivity.

For an effective digital transmission module, SGMII protocol specifications shall be used for the design of the SerDes ASIC (Application Specific Integrated Circuit), along with the 8b/10b encoding, explained further in this document.

1.2 Digital Serializer

The purpose of the digital serializer block is to transmit data at a desired frequency through a single ended bus, an n-bit character. This kind of system is also known as PISO (Parallel Input Serial Output), which consists in reading the information from an outside boundary through more than one pin, storing it during a short period of time and periodically send it through a single pin.

shows the high level architecture of the digital serializer of the proposed design, where an eight-bit-ten-bit encoder is placed as the first step of the module in order to translate the two nibbles (4-bit data) into a specific equivalent 10-bit data that is less sensitive to errors during transmission. Once the information is encoded, it feeds the parallel data of the serialization interface and it is delivered to the analog transmission block, which is responsible for adapting the signal in voltage levels that can be interpreted outside the integrated circuit.

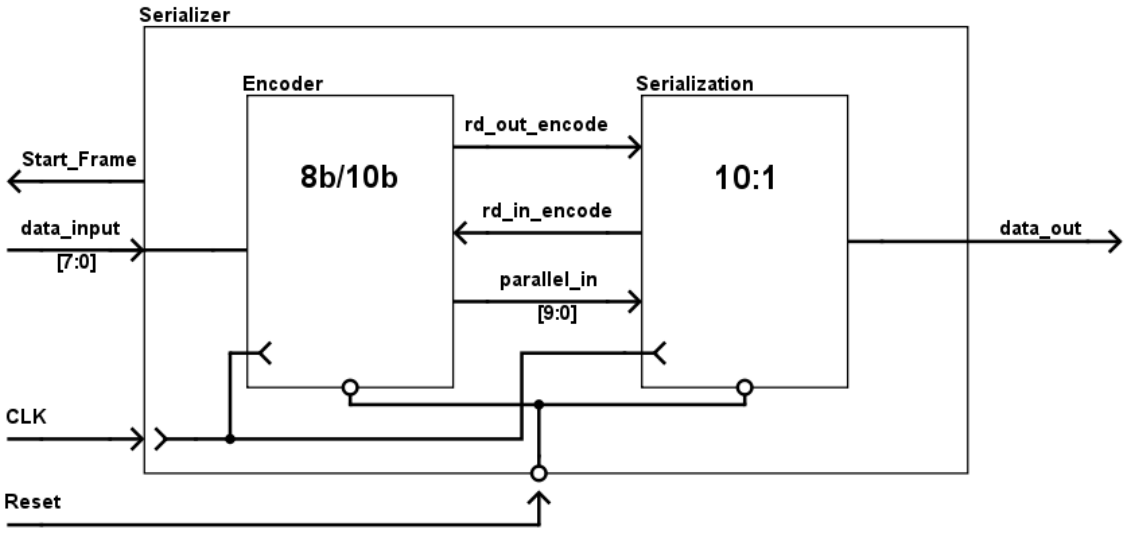

Figure 1-4. Digital serializer architecture.

The top level of the serializer block takes as inputs:

- 8-bit data that comes directly from the outside of the chip.
- CLK, which is internally wired to the Rx Clock pin.
- Reset, as most of the digital systems, it has a common active-high reset signal for the entire device.

The signals generated by the serializer block or outputs are:

- Data out: single ended data bus delivered to the analog transmitter.
- Frame: active when the last bit is sent, meaning that it is ready to take another input.
-

1.3 Encoder 8b/10b

The combinational logic of the 8b/10b encoder, first developed in 1983 by IBM, is meant to be implemented in the design of this module as it mainly helps achieve a DC-Balance during high-speed transmissions, enhancing the performance of the entire design.

By implementing this technique, it is possible to carry out three important milestones that help reduce errors in transmission and instrumentation costs:

- *Embedding a clock into the data.* By setting boundaries of how many identical contiguous symbols (run length) can be given at the output, the encoder ensures enough transitions for timing recovery and almost suppresses dc spectral components. So, a clock lane would not be needed and it could reduce the probability of having difficulties while routing or during electro-magnetic interferences.

- *Maintaining DC balance.* Encoding allows to have different common-mode voltage in the transmitter and receiver, so both devices will not need to have the same reference voltage, making the design easier. This is achieved by the encoder when it tracks the disparity of the last packet sent so that the output depends on whether the last transmission had more ones or zeros.

- *Enhance error detection*. Unless there are 1024 possible patterns given by the 10-bit resolution, only 512 codes are needed (2 per character) where the packets with neutral disparity have the same code for every case. So, there are less valid unique codes that can be detected at the output, and when they do not have a match, an error signal can be triggered.

Considering that the encoder is expected to provide an output depending on its inputs, it is time to take a look at what is inside and comprehend the logic that determines the signals the encoder delivers. The encoder is a combinational module (), which receives an 8-bit data, current disparity, and a control bit to change between data and control characters.

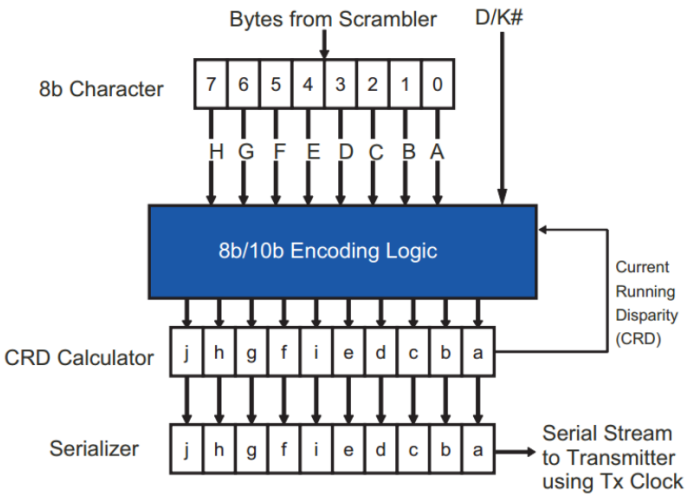

Figure 1-5. 8b/10b encoding high level scheme.

The control bit allows the system to define special characters beyond the 256 valid possibilities that can be encoded using some of the remaining schemes from the 1024 possibilities provided by the 10-bit output bus. This can be used to establish byte synchronization, indicate the start of a transmission and the end of any packet or functions, such as abort, reset, shut-off or idle. There are a total of 12 special characters implemented in this design that can be seen in Table 1-1. Special character encoding-1.

Table 1-1. Special character encoding.

Input-Special Symbols				RD = -1	RD = +1	Information
K-Codes	DEC	HEX	HGF_EDCBA	abcdei_fghj	abcdei_fghj	OOB
K28.0	28	1C	000_11100	001111_0100	110000_1011	SKP
K28.1	60	3C	001_11100	001111_1001	110000_0110	FTS
K28.2	92	5C	010_11100	001111_0101	110000_1010	SDP
K28.3	124	7C	011_11100	001111_0011	110000_1100	IDL
K28.4	156	9C	100_11100	001111_0010	110000_1101	Electric IDLE
K28.5	188	BC	101_11100	001111_1010	110000_0101	COM
K28.6	220	DC	110_11100	001111_0110	110000_1001	Receiver Detect
K28.7	252	FC	111_11100	001111_1000	110000_0111	Beacon
K23.7	247	F7	111_10111	111010_1000	000101_0111	PAD
K27.7	251	FB	111_11011	110110_1000	110110_1000	STP
K29.7	253	FD	111_11101	101110_1000	101110_1000	END
K30.7	254	FE	111_11110	011110_1000	011110_1000	EDB

Furthermore, it can be observed that the current disparity takes a crucial role in the encoding, as it has been mentioned before, the encoded data depends on the disparity of the last sent data. The encoder has been designed to have at most 6 bits with the same value, having up to 5 consecutive identical symbols. This way, the disparity will always be between -1 and +1, having 0 as a legal value.

- Negative disparity (-1): where in the encoded data are 4 ones and 6 zeros.
- Positive disparity (+1): where in the encoded data are 6 ones and 4 zeros.
- Neutral disparity (0): where in the encoded data are 5 ones and 5 zeros

The 8b/10b encoder block output is given by the concatenation of two separate encoders, the first one takes the 5 less significant bits of data and generates a 6 bit output, while the second encoder takes the remaining 3 data bits and generates a 4-bit output, resulting in a 10-bit output as shown in Figure 1-6.

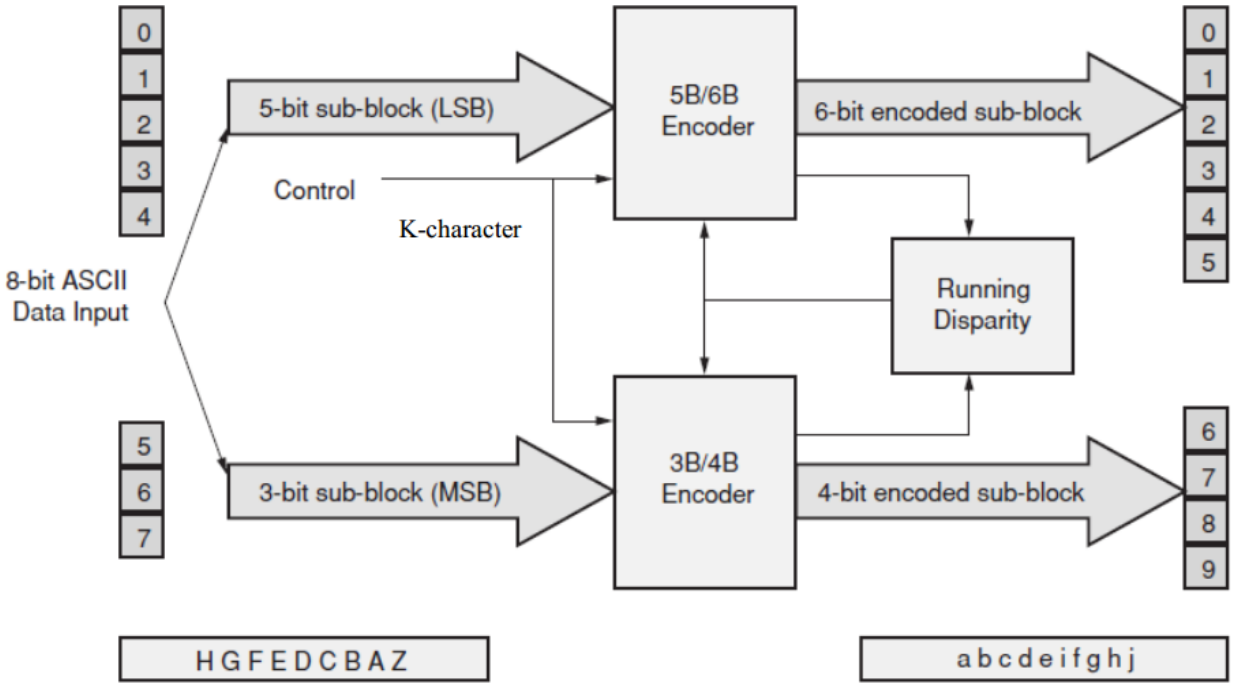

Figure 1-6. 8b/10b encoder block diagram.

1.4 Serialization

The serialization block is the most important instance of the digital serializer module, as it is responsible for translating data from parallel into serial and sending it sequentially to the analog transmission block.

This module implements a 0 to 10 counter, where 0 is only reached when reset is in a low state and works at the given frequency at the CLK input signal while the data is sent continuously from count 1 through 10 synchronized with the start frame signal. When the counter is equal to ten, the start frame signal is set to a high state and then is set to low during the transition from ten to 0, where its negative edge matches the moment when the transmission begins.

The internal and IO (Input-Output) signals of the serialization module are described in *Table* 1-2. Description of s*erialization signal*s.

Table 1-2. Description of serialization signals

Name	Size	Direction	Description
Data_in	10	Input	Coming from the encoder, is the information that must be read by the receiver.
clk	1	Input	Clock signal, determines the operation frequency of the module.
reset	1	Input	Synchronous active low signal, it means that the registers will go to 0 if a low state is present at the positive edge of the clock.
rd_in	1	Input	The running disparity of the data is bypassed through the serialization block and back to the register at the loopback of the encoder.
rd_out	1	Output	Delivers the running disparity outside the module.
serial_out	1	Output	Transmission link, goes out from the serializer to the analog tranmitter.
start_frame	1	Output	Indicates with a negative edge when the character starts its transmission
nine	1	Output	Active when the count is nine, which acts as enable signal for the register that stores the data and disparity.

A basic example of the serialization behavior is shown in **Error! Reference source not found.**, where the counter does not start its job unless a positive edge of a clock matches a logic high for reset. Once the counter gets to 9, the start frame transitions to high and immediately low in the next positive edge clock signal. Finally, the data is transmitted starting from the less significant bit (LSB) when counter is at one.

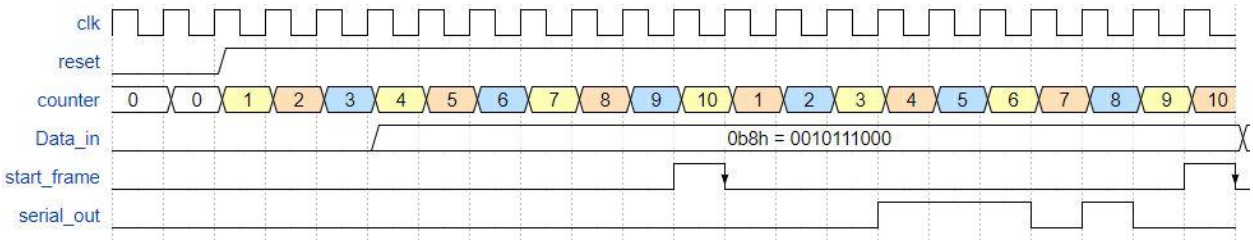

Figure 1-7. Serialization behavior.

Chapter 2. RTL Design

In digital hardware design, the first step after defining the specifications of the device, is to propose a RTL (Register Transfer Level) of the system, imitating its behavior with high level description languages, such as VHDL or Verilog. These tools provide the designer with the opportunity of representing the conduct of the design into a code that is mapped in logic gates and registers further in the design flow.

For this project, the design is implemented as a structural design using Verilog, this allows to write specific codes for the design represented in and then, merge them to obtain the final product.

2.1 Encoder 8b10b

As mentioned before, the code of the encoder available as Appendix A, was originally created by Chuck Benz and adopted for this project. The testbench (Appendix B) is in charge of stressing the rtl with known values in such a way that expected values are met. The injected values from the testbench into the design are read in an external .mem file as well as its golden/expected data. The hardware first executes a run with control values, followed by a run of certain values, both with disparity 0 and 1.

The points of interest of the response that the encoder provides when control values are injected to it are seen in Figure 2-1 and Figure 2-2 respectively, depending on the dispin (disparity in) value, satisfying the expected values described in Table 1-1. Special character encoding. The displayed results are seen in Figure 2-*3*.

Figure 2-1. Encoder response for control values and disparity 0.

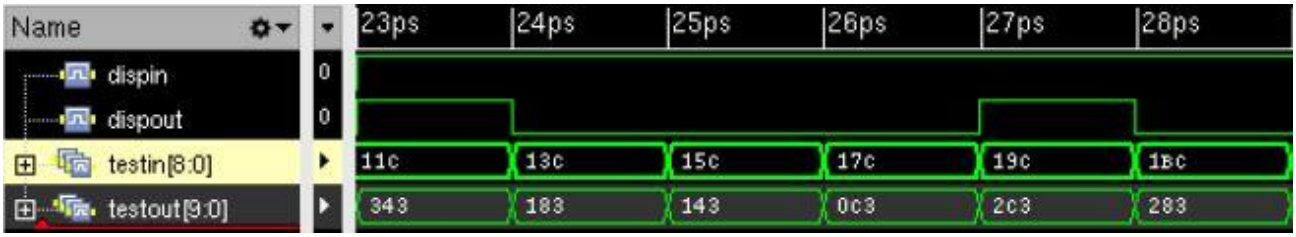

Figure 2-2. Encoder response for control values and disparity 1.

```
Control data encode with negative disparity has been succesfully achieved
Control data encode with positive disparity has been succesfully achieved
```

Figure 2-3. Encoder control data report.

The RTL has been developed and compiled with the IDE named Quartus II, which delivers a first synthesis into the logic gates reported as Figure 2-4. Here, the system does not include any flip flops as well as a clock signal, making it a fully combinational block that has as inputs the 9 bit data bus (8 bits for data and 1 bit for control) and the disparity in the signal, and once integrated, it represents the disparity of the last sent data.

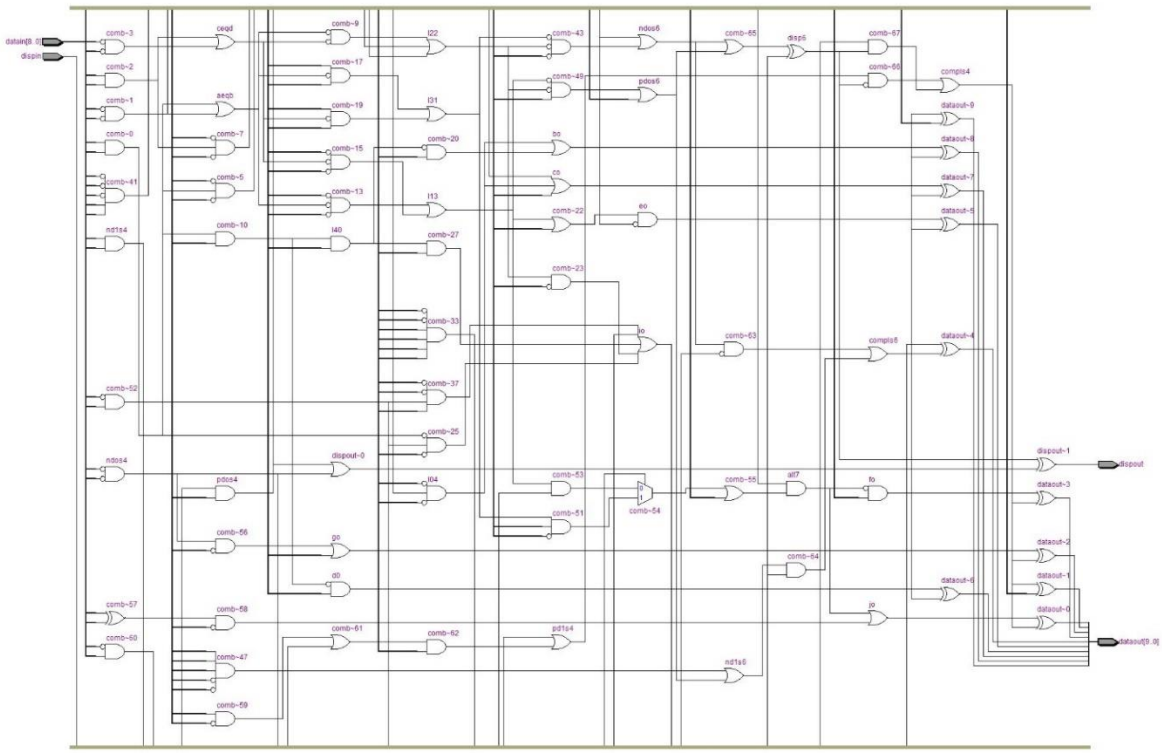

Figure 2-4. Encoder first synthesis.

2.2 Serialization

The main purpose of the serialization block is to store the 10-bit character provided by the encoder and send it through a single lane within 10 clock cycles. It is composed by 3 registers, each one with a multiplexor at the input and the control unit, based on three comparators that enable or disable the registers at counts of eight, nine, or ten, respectively.

The Verilog code for the serialization module is available as Appendix C and it describes the behavior when the count is equal to eight and the value on the register of disparity is updated with the input rd_in; on the other hand, when it the count reaches nine, the register at the input in the encoder stores the value of the data to be sent and finally, when the count is equal to ten, the counter restarts and the start frame signal makes a transition from 0 to 1.

The first synthesis of this block described in Verilog code in Appendix D and E with Quartus II, standard cells, and no enhancements, is shown in Figure 2-5, while the results of the synthesis for the control block can be seen in Figure 2-6 and Figure 2-7, where the comparator has been implemented using an array of XOR gates and a bitwise OR.

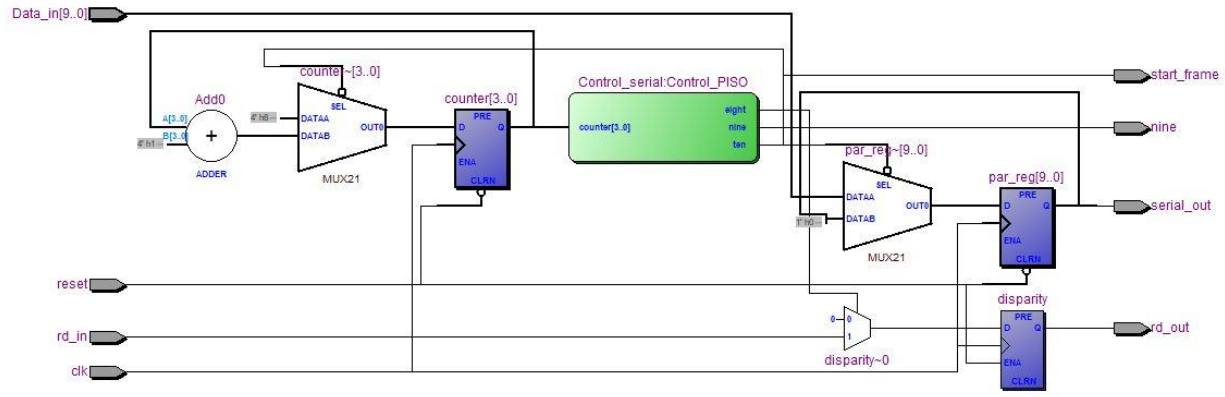

Figure 2-5. Serialization block first synthesis.

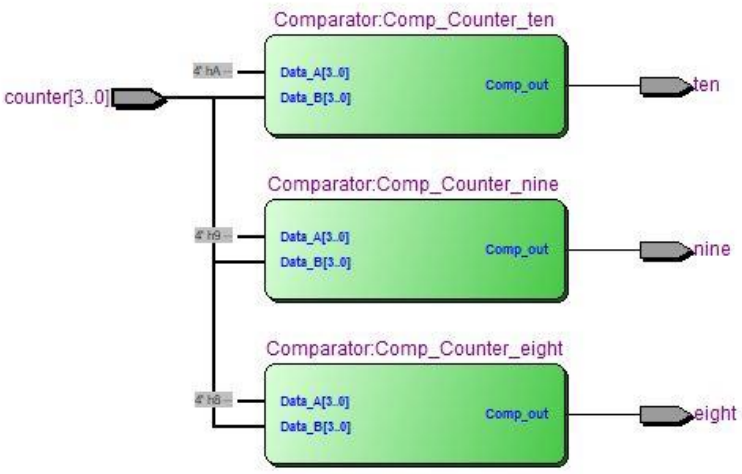

Figure 2-6. Control unit synthesis.

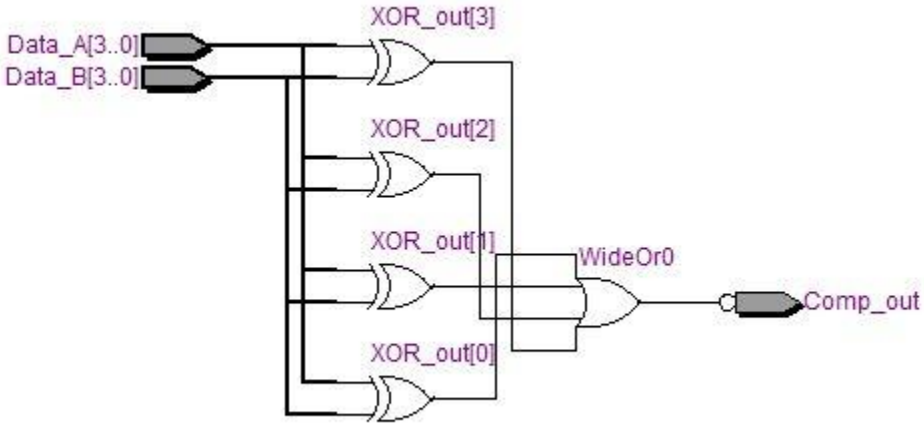

Figure 2-7. Comparator first synthesis.

Every block within this module has been successfully validated by implementing a specific testbench for each of them. In the case of the comparator, the testbench (Appendix F) alternates the two input values in such a way that a logic zero is expected when the inputs are different and a logic one if they are equivalent. The waveform results (Figure 2-8), demonstrate that in the instant that both inputs are not equivalent, the output is set to low.

Figure 2-8. Comparator waveform results.

The serialization module behavior is validated with all the blocks of the digital transmission module with the testbench, available as Appendix G. The test consists in an implementation of an exhaustive validation process as it writes to the module the 255 possible inputs and also reports the result every time the data is sent either correct or incorrectly. Lastly, it writes a summary with the amount of success transmissions as well as the amount of error transmissions, this is shown in Figure 2-9.

```
 -Info- Match Generate data = 248,  Expected data encoded= 1e8, Run disparity 1

 -Info- Match Generate data = 249,  Expected data encoded= 1cc, Run disparity 1

 -Info- Match Generate data = 250,  Expected data encoded= 219, Run disparity 0

 -Info- Match Generate data = 251,  Expected data encoded= 1da, Run disparity 1

 -Info- Match Generate data = 252,  Expected data encoded= 1e4, Run disparity 1

 -Info- Match Generate data = 253,  Expected data encoded= 21c, Run disparity 0

 -Info- Match Generate data = 254,  Expected data encoded= 21d, Run disparity 0


------Simulation Report---------
Number of iterations:  253
Number of errors:    0
Number of matching data:  253
```

Figure 2-9. Serializer simulation report.

The first column in the report represents the data that the TB has injected into the design, while the second column refers to the one read at the output of the encoder, which matches the output of the serializer, once the 10 clock cycles are completed. Finally, the third column is the current disparity of the data that is being sent. All this information is a brief interpretation of the

behavior of the design, as a detailed verification is the visual inspection of the waveforms (Figure 2-10).

Figure 2-10. Top module serialization waveforms.

The right functionality of the design can be confirmed by the interpretation of the signals shown in Figure 2-100. First, the testbench stresses the design with the data values seen in the signal bus called “data” and stores bit by bit of the serial output in the signal bus “compare_data”; every time the counter reaches ten, it makes a comparison between both buses as long as the compare signal (yellow) is active. Every time a comparison is done, the variable “iterations” increase by one as well as “errors” and “matches”, in each case.

The top serializer signal group represents the signals coming in and out from the design that can be reached by the adjacent modules, the “data_in” represents the parallel input and

according to specifications, the negative edge of the start_frame signal indicates the start of a new transmission block, checked by zooming in and adding the “counter” bus in Figure 2-11.

Figure 2-11. Parallel to serial waveform.

.

In this case, the input data is “4” and the previous disparity is equal to “0”, so the encoder delivers a hexadecimal value of “363”, which is updated as golden data when the count reaches 9 and its binary value is equal to “1101100011”. The transmission is performed correctly, starting with the least significant bit at count “1” and matches the golden data at count “10”.

The following signal group in Figure 2-100, is the representation of the register that stores the previous run disparity as LSB, followed by the data. It can be observed how the value is updated at the clock positive edge that follows the high value at enable, as the data is mapped from bits one through 9, the value can be characterized by the following equation:

$$Data_{input} = 2(data\_in) \ + rd$$

As the encoder has been already verified, this time the focus of the validation is how it stores the current disparity and reads it for the following data value. The signal “dispin” is updated by perceiving the pulse of disparity signal of the serialization module at count equal to nine, which is done only if the current disparity is one. This is what makes possible to visualize the same values in both signals, but displaced 10 clock cycles.

Chapter 3. Logic Synthesis

The logic synthesis is the step that follows the behavioral description in the IC design process, and once the rtl is completed, a logic synthesis must be executed. This step is responsible for exporting the high level description of the design into logic gates. The logic gates are previously designed and characterized in such a way that they comply with the technology requirements.

The 130nm technology cmrf8sf by IBM is employed for purposes of this design, while the standard cells have been created by ARM, based on the proper specifications by the technology.

There is no straight path or recipe to follow in order to execute the logic synthesis. The designer is free of changing the parameters that can impact on the result of the synthesis. These parameters are also known as constraints and by these specifications, the synthesizer selects from a group of standards cells. Every logic block has more than one standard cell, all with the same height but different width. The tool instantiates the one that fits the best in the design and the constraints.

Some of the parameters that need to be considered before executing the logic synthesis are:

- *Clock:* Periodic signal with a fixed frequency used to sequence the operation of the circuit. For the synthesis, the period in femtoseconds or picoseconds must be defined and the duty cycle is optional.

- *Slew:* The time it takes for the waveforms to go from zero to one or from one to zero, considering one, depending on the technology specifications, every value greater than a specified percentage of the voltage source close to 80% and a zero every value below a fixed point close to 20% of the voltage source.

- *Latency:* Is the time that takes to a clock to arrive from the source to an endpoint.

- *Skew:* is the difference between the higher and the lower latencies of a clock distribution tree.

- *Slack:* is the difference of time between the clock period and the times it takes to a signal to be propagated through the combinational logic from a sequential block to the nearest datapath.

3.1 RTL Compiler typical synthesis

The logic synthesis is performed by the RTL compiler tool from cadence and considering typical, worst case, and best case operating conditions in an iterative way, in order to adjust the constraints closest to reality and best suited for the project.

The tool first executes a logic synthesis with generic gates, then optimizes it to an intermediate synthesis and finally, an incremental synthesis, which is the result of optimizing the design and constraints. Everything is done by executing a single tcl script, since the tool also provides templates of the script, depending on what is meant to be optimized, and thus, there is a tradeoff. For example, if the designer wants to improve the operating frequency, the design will consume a greater amount of power. The performance template is used for the synthesis of the digital serializer module, specifying a clock frequency of 625/4 MHz or a clock period of 6400 ps.

The synthesis is done by executing the tcl script available as Appendix H and the constraints (sdc) file available as Appendix I. The result provides a Verilog Gate Level Netlist that is shown as schematic Figures 3-1 through 3-8.

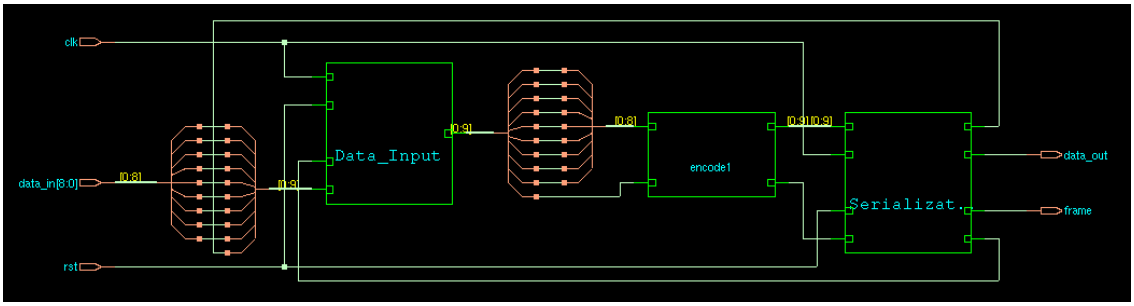

Figure 3-1. Schematic representation for top module grouped synthesis.

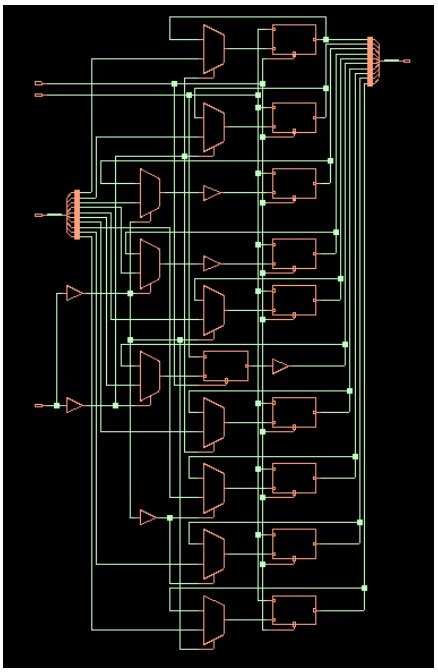

Figure 3-2. Schematic representation of data input register grouped synthesis.

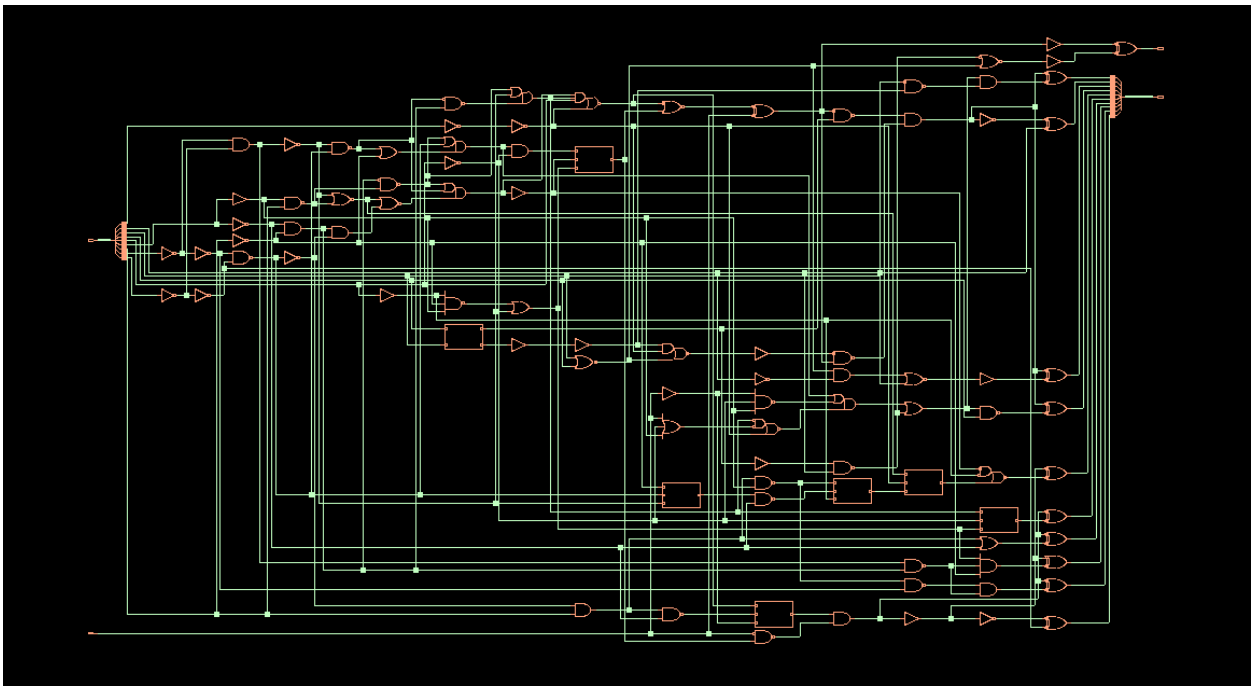

Figure 3-3. Schematic representation of encoder grouped synthesis.

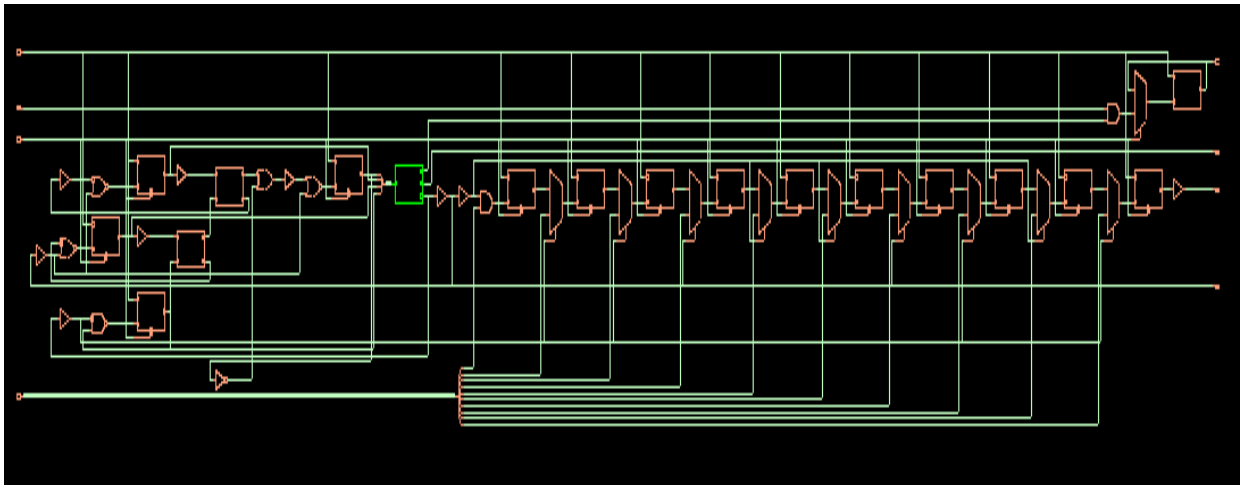

Figure 3-4. Schematic representation of serialization top module grouped synthesis.

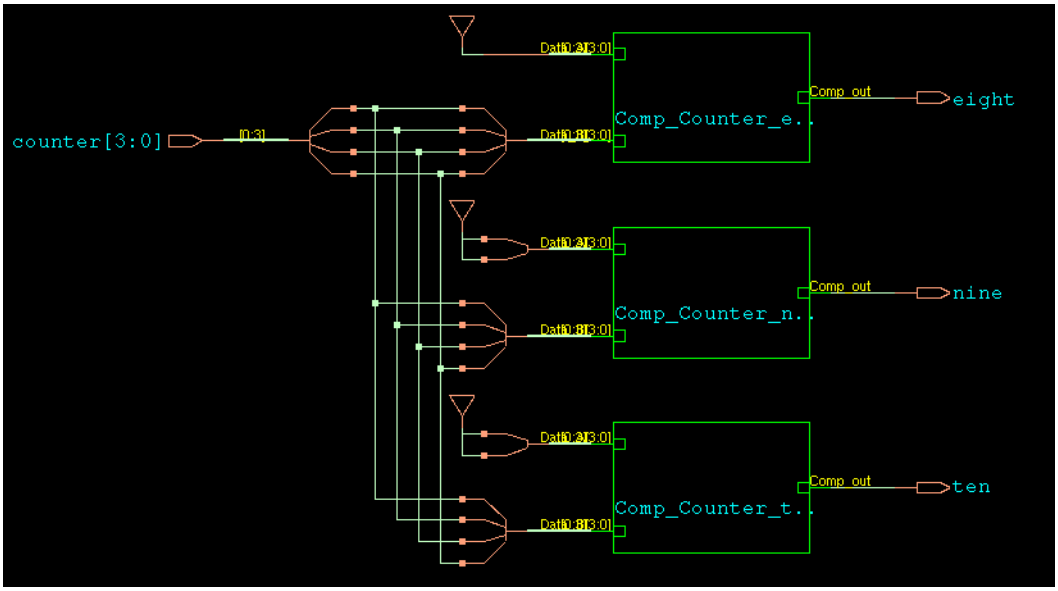

Figure 3-5. Schematic representation of PISO control unit grouped synthesis.

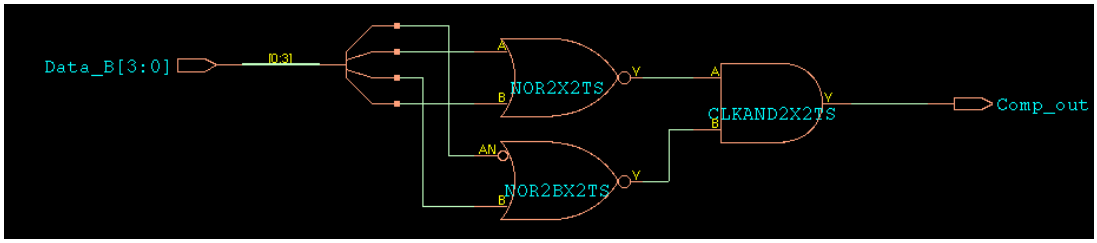

Figure 3-6. Schematic representation of comparator against 8 grouped synthesis.

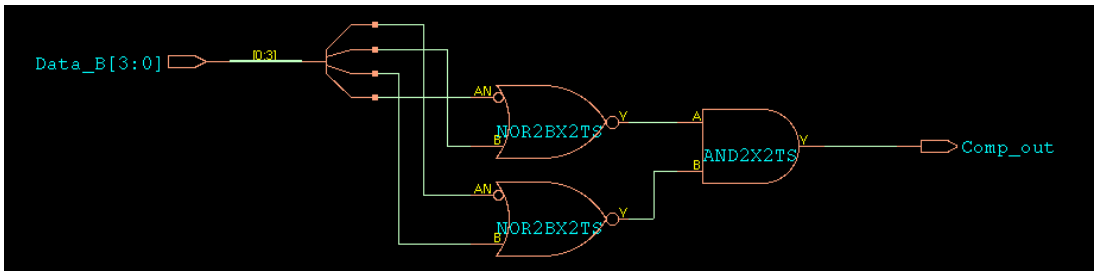

Figure 3-7. Schematic representation of comparator against 9 grouped synthesis.

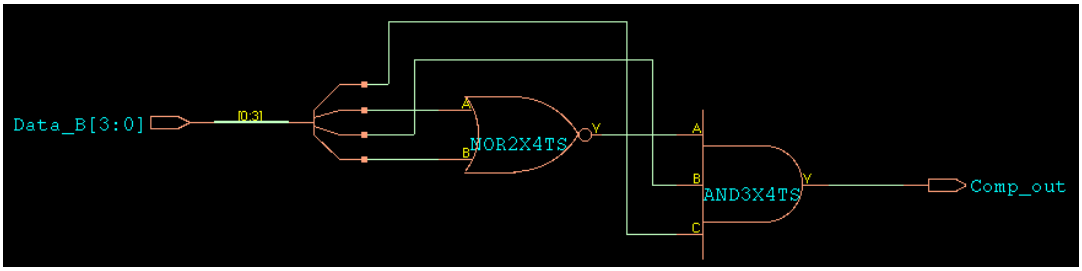

Figure 3-8. Schematic representation of comparator against 10 grouped synthesis.

While having replicated the structural design of the code in the first synthesis, it can be merged in order to have the design in a single structure, allowing the tool to analyze the complete behavior of the module and generate an alternative. As a result, a better level of optimization in performance and area. So, by changing the value of the attribute “auto_ungroup” in the tcl script from “none” to “both”, the schematic representation in Figure 3-99 is obtained.

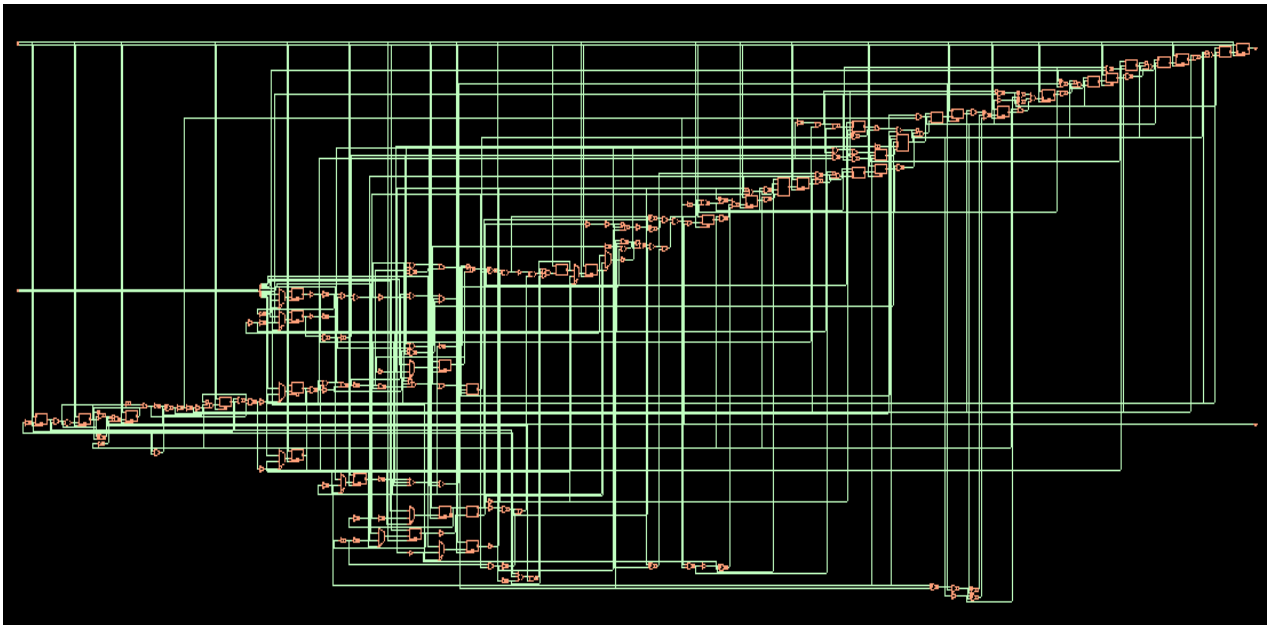

Figure 3-9. Schematic representation of ungrouped synthesis.

Every synthesis at the different levels, implies different factors such as, timing, area, number of gates, and power, in such a way that comparing all of them is needed in order to select the one that fits best in the device. Table 3-1 shows the results of the grouped synthesis in the three stages (generic, map, and incremental) as well as the results of the ungrouped synthesis are at the right side.

Table 3-1. Typical synthesis results.

	Grouped			Ungrouped		
	Generic	Map	Incremental	Generic	Map	Incremental
Worst Slack (156.25 MHz)	3127.8 ps	2072.5 ps	**2209.3 ps**	3127.8 ps	2314.6 ps	1849.8 ps
Max Frequency	305 MHz	231 MHz	**239 MHz**	305 MHz	245 MHz	219.78 MHz
Area	3645	2629	**2523**	3645	2601	2498
Instances	243	192	**171**	243	229	189
Average fanout	1.6	1.9	**2**	1.6	1.9	2
Total Power	536.05 uW	609.85 uW	**599.06 uW**	536.05 uW	529.74 uW	541.64 uW

The results of the first synthesis demonstrate how every feature improvement implies a deterioration of another one, and with the grouped incremental synthesis, less instances are needed and a higher frequency against the incremental ungrouped synthesis, but the price to pay is the power consumption and area used. The synthesis delivers a Gate Level Netlist for each run and the incremental grouped synthesis can be used further for the development of the system on chip.

3.2 Worst case synthesis

The typical synthesis characterizes the response of the system at regular conditions of the manufacturing process, voltage source, and temperature, set to 25 Celsius degrees and a 1.2 VDC respectively, while the worst case synthesis emulates the behavior of the logic gates with a voltage input of 1.08 VDC, at a temperature of 125 Celsius degrees. This synthesis helps the designer ensure that the chip will work at any real conditions that cannot be controlled. The results can be compared in Table .

Table 3-2. Results of worst case synthesis.

	Grouped (Typical)			Grouped (Worst Case)		
	Generic	Map	Incremental	Generic	Map	Incremental
Worst Slack (156.25 MHz)	3127.8 ps	2072.5 ps	2209.3 ps	-474.1 ps	336.6 ps	0.2 ps
Max Frequency	305 MHz	231 MHz	239 MHz	145 MHz	164 MHz	156.25 MHz
Area	3645	2629	2523	3645	5750	6498
Instances	243	192	171	243	409	435
Average fanout	1.6	1.9	2	1.6	1.5	1.5
Total Power	536.05 uW	609.85 uW	599.06 uW	448 uW	1240.5 uW	1403.67 uW

Beyond performing the best optimization possible, the incremental synthesis takes into account facts, such as wiring area and parasitic capacitances, this is why the approach of the results are closer to the physical device. For the worst case synthesis, the incremental results decrease performance and optimization, as the synthesis still tries to comply with the given constraints, originating a greater design and a lesser maximum frequency.

3.3 Static timing analysis (STA)

One of the most important phases of the design of a chip, is to verify that the gate level netlist generated by the logical synthesis is able to work correctly at the established frequency, this means that it is crucial to have a positive slack in every path of the design.

According to the results of both, typical and worst case synthesis, the design is well suited to operate at up 125 degrees without compromising performance, so the timing analysis is done successfully. The complete analysis is the typical incremental rtl is represented as a histogram in Figure 3-10, as most of the paths to follow within the design are closer to a slack of 4600 ps out of a period of 6400 ps, meaning that it takes 1800 ps to go from end to end blocks. As seen previously, the worst slack is around 2300 ps and there are 5 paths reporting a similar timing.

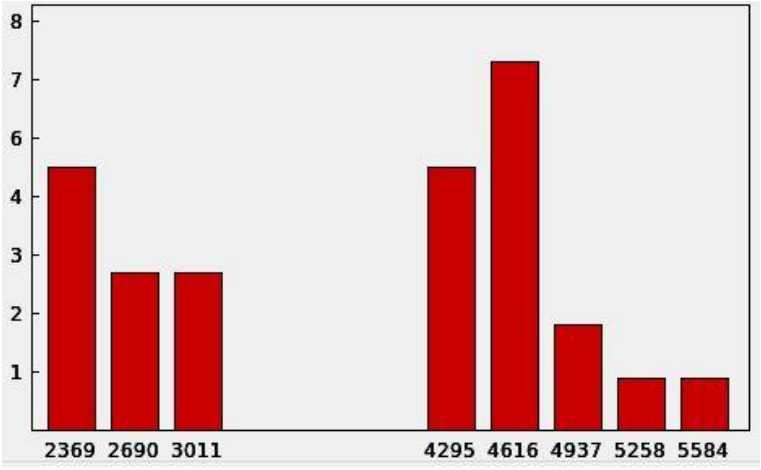

Figure 3-10. Static timing analysis histogram results.

The tool also provides the specific block by block delay for the worst path, which goes from the input of bit 4 of the data input register, to bit 8 of the output of the encoded data, passing through the combinational logic of the encoder, as observed in the schematic representation in Figure 3-11.

The two blocks with a considerable delay above average, are marked in yellow. The first cell is a Flip-Flop D (FFD) from the register contributing with a delay of 332.1 picoseconds and the combinational gate named AOI31X4TS, delaying the response to 518.7 picoseconds.

Airlines	Endpoint	Slack (ps)	Rise Slew (ps)	Fall Slew
	Serialization1/par_reg_reg[6]/D	2209.30	141.80	

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)
(clock 156MHz_CLK)	launch					640.00
(clock 156MHz_CLK)	latency				140.00	780.00
Data_reg_reg[4]/CK				28.00		780.00
Data_reg_reg[4]/Q	DFFRHQX2TS	1	6.80	93.60	332.10	1112.10
drc/A					0.00	1112.10
drc/Y	BUFX4TS	3	20.60	88.60	189.80	1301.90
Data_Input/Data_Output[4]						
encode1/datain[3]						
g1641/A					0.00	1301.90
g1641/Y	INVX4TS	4	23.50	132.20	120.90	1422.80
g1636/A					0.00	1422.80
g1636/Y	AND2X4TS	4	23.30	131.50	236.50	1659.30
g1628/A					0.00	1659.30
g1628/Y	CLKAND2X2TS	1	8.20	125.50	296.90	1956.20
g1616/B					0.00	1956.20
g1616/Y	NOR2X2TS	1	5.90	62.40	96.10	2052.30
g1883/B0					0.00	2052.30
g1883/Y	OA21X2TS	2	9.30	136.80	264.30	2316.60
g1880/A0					0.00	2316.60
g1880/Y	AOI31X4TS	2	10.70	81.50	518.70	2835.30

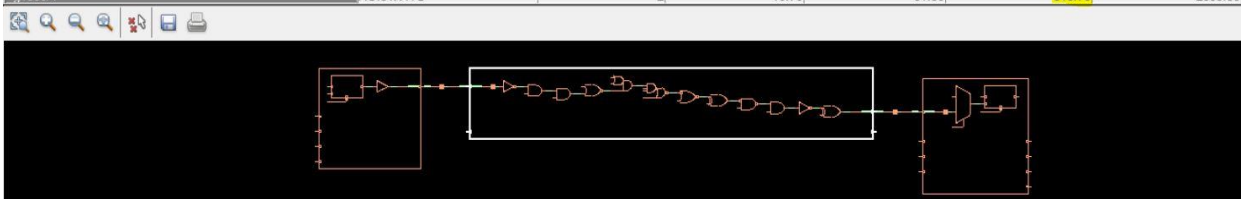

Figure 3-11. Worst slack timing path.

Every block from the library has more than two equivalent logic designs, but the synthesis selects the one that best fits the given constraints. In the case of the positive-edge triggered FFD with asynchronous active-low reset, the standard cell library data book states that the selected cell is the best option as it comprises less area than the other equivalent cells and adds the lowest intrinsic delay. The schematic representation of the cell shown in Figure 3-12.

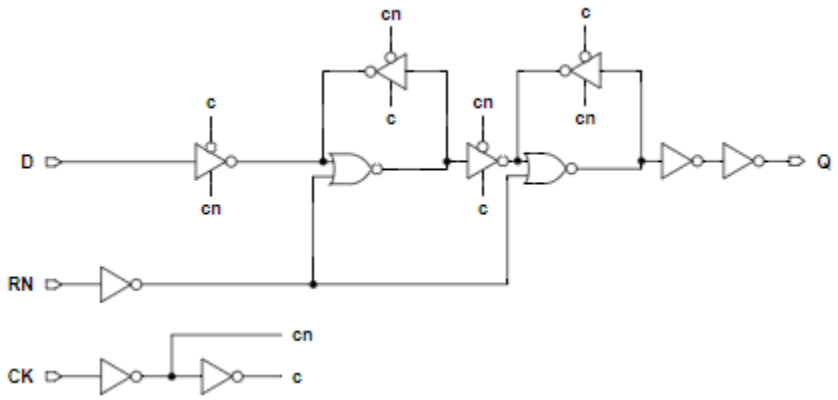

Figure 3-12. Positive-edge FFD with active-low reset schematic representation.

The logic gate AOI3X4TS standard cell gives at the output the logical inverted OR of one AND group and an additional input is represented in the following Boole Equation:

$$Y = \overline{(A0 \cdot A1 \cdot A2) + B0}$$

And expressed in the block diagram in Figure 3-133.

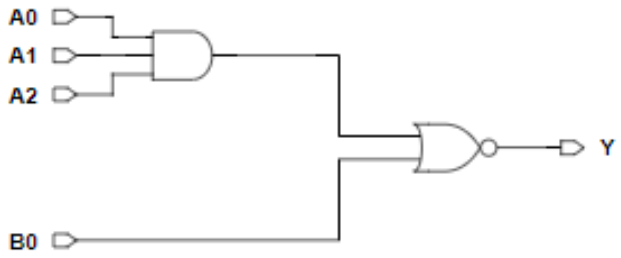

Figure 3-13. Schematic representation for AOI3 standard cell.

The slowest change of the cell is done when a change from the input A1 generates a transition at the output Y. This cell can be substituted in order to obtain a better performance, resulting in an increase of the maximum frequency of the design.

3.4 Logic Equivalent Checking (LEC)

After each iteration of the logical synthesis, testing vectors for the each model are generated. These vectors can be used as a post synthesis verification with the "Conformal" extension of RTL compiler from cadence, where a comparison of the current output is compared against either to the previous netlist generated by the synthesis itself or against the rtl. As the synthesis performs 3 iterations, labeled as generic, map, and incremental, the comparisons are performed as follows:

- RTL vs Map
- Map vs Incremental
- RTL vs Incremental

LEC is considered a fast process that can cover the full-design and reduce the risk of missing an error in gate-level simulations, apart from pointing the node that generates the bug.

The logic synthesis outcomes a .lec.do file that acts as input of the conformal verification by adding the following command to the tcl script:

"write_do_lec –verbose –no_exit –golden_design {design} -revised_design {design} > {name}.lec.do"

Once performed, the logic synthesis delivers one logic equivalent do file for every behavior to be compared, the first verification is performed taking as golden design the rtl and the intermediate, also known as mapped synthesis, is subject to equivalence check. For this run, the equivalence is declared in Figure 3-14. The remaining equivalence checks are also done successfully but images are omitted as they are similar to the one presented.

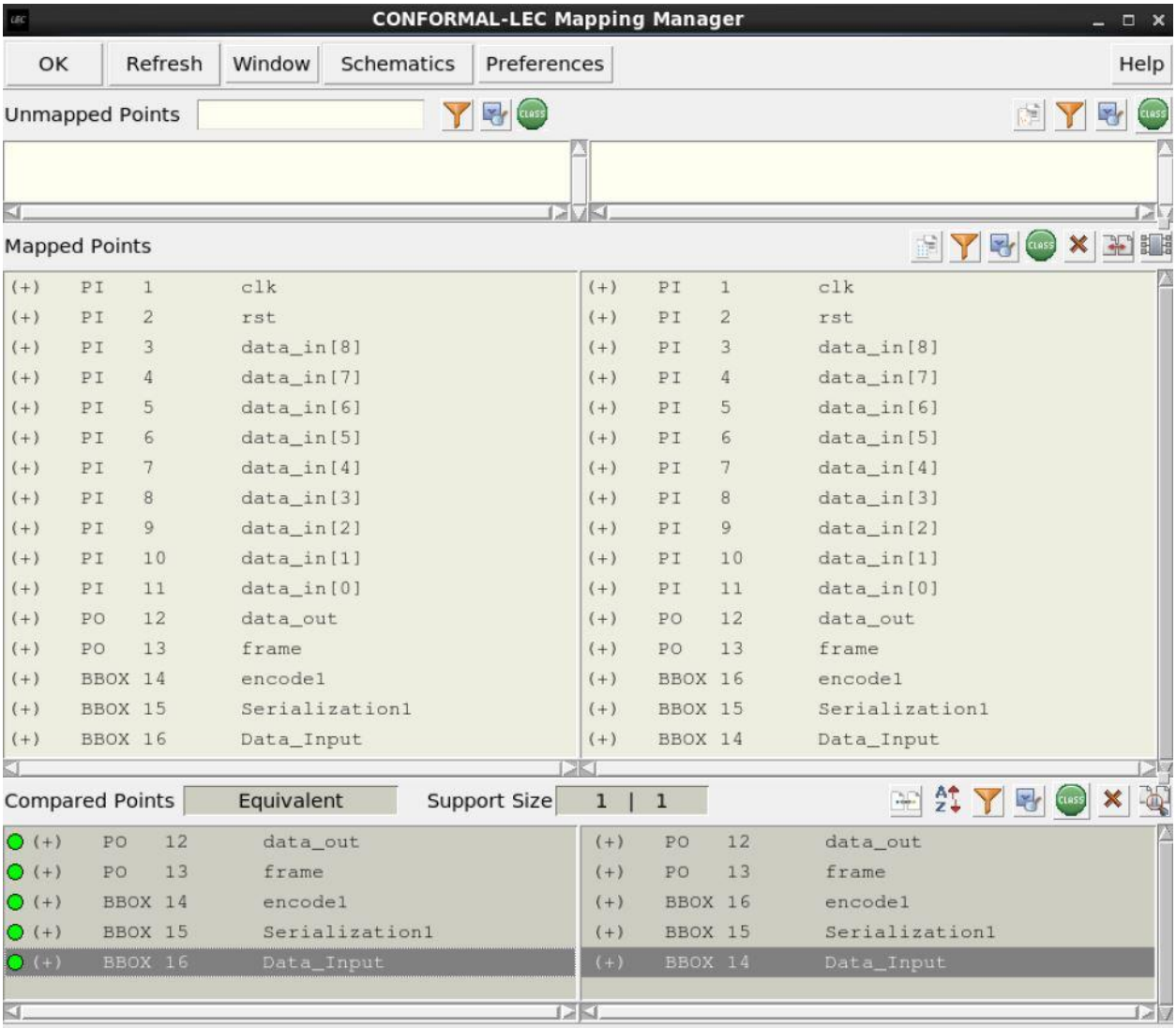

Figure 3-14. RTL against intermediate LEC.

The LEC report shows interactively every node that has been compared and report with a green dot when the node is equivalent in the two models is and a red dot when it is not. At this point, it can be confirmed that the gate level netlist behaves equal to the high level design.

3.5 Gate level netlist simulation

Even though the logic equivalence check has been successfully performed, an addition to the validation process of the logic synthesis, is to have a visual comparison of the rtl and the gate level netlist (gln) by performing a simulation with the same testbench and analyzing the waveforms.

An instance of the gate level netlist is added to the testbench attached as Appendix G and pointed to the corresponding standard cells verilog characterization while compiling with ncverilog. After the simulation is finished, the waveforms are dumped into simvision, and the output signals of both models are compared.

The results reported by the simulation log file does not return any mismatch during simulation and the summary is the same obtained when the simulation only ran with the rtl (Figure 2-9). Also, the waveforms of the outputs of both, gln and rtl, to common inputs are equivalent, see the input values coming from the testbench in green, rtl response in yellow and gln response in red in Figure 3-15.

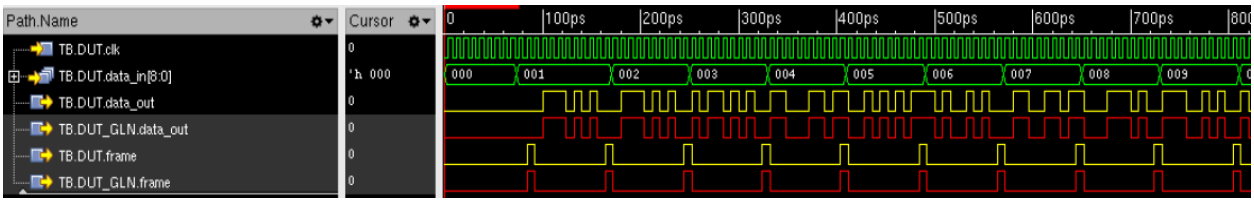

Figure 3-15. Serializer gln and rtl waveform comparison

Until this point, the logic synthesis results have been verified and confirmed to match the design and these comply with the timing collaterals of the technology. Then, the gate level netlist can be exported to a layout using the 130 nm standard cells provided by ARM.

Chapter 4. Physical Synthesis

The physical synthesis is the process where the output standard cells from the logic synthesis are placed and routed inside the layout. All the standard cells have the same height and width, allowing a more efficient area distribution. They have been developed and verified by ARM, incorporating up to 8 metal layers and following the design rules of the manufacturing process.

The best run of the logic synthesis, as discussed before, is the grouped synthesis with typical conditions of operation, where the design was translated into 171 instances of standard cells, so they must be placed in the layout. To avoid inserting one by one and reduce the error probability, the tool named "Encounter" assists during the procedure and helps make an automation of the physical synthesis.

This is usually an iterative process, so the first time it must be performed with the graphic user interface and then, the instruction set is merged into one or more tcl scripts.

4.1 Import Design

Encounter is able to convert a verilog file into a file that stores all the physical design split into metals, polysilicon, and vias. The first step is to import the gate level netlist with all the attribute files of the technology and the design, such as constraints and libraries (.lib and .lef), specifying the conditions of operation. This step is performed as shown in Figure 4-1, once the design is loaded into the tool, the die area is delimited in the layout, providing a rectangle of 57.6 µm x 62.58 µm that results in an area of 3604 μm^2, which is greater than the area needed for the logic synthesis, 2523 μm^2 (**Error! Reference source not found.**).

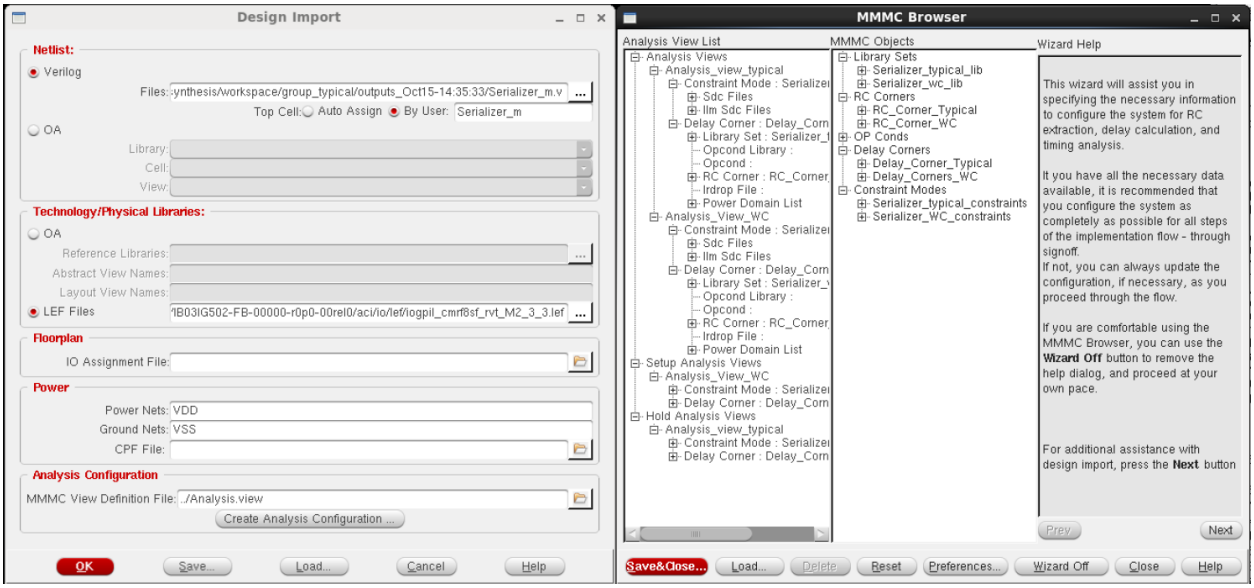

Figure 4-1. Encounter import design.

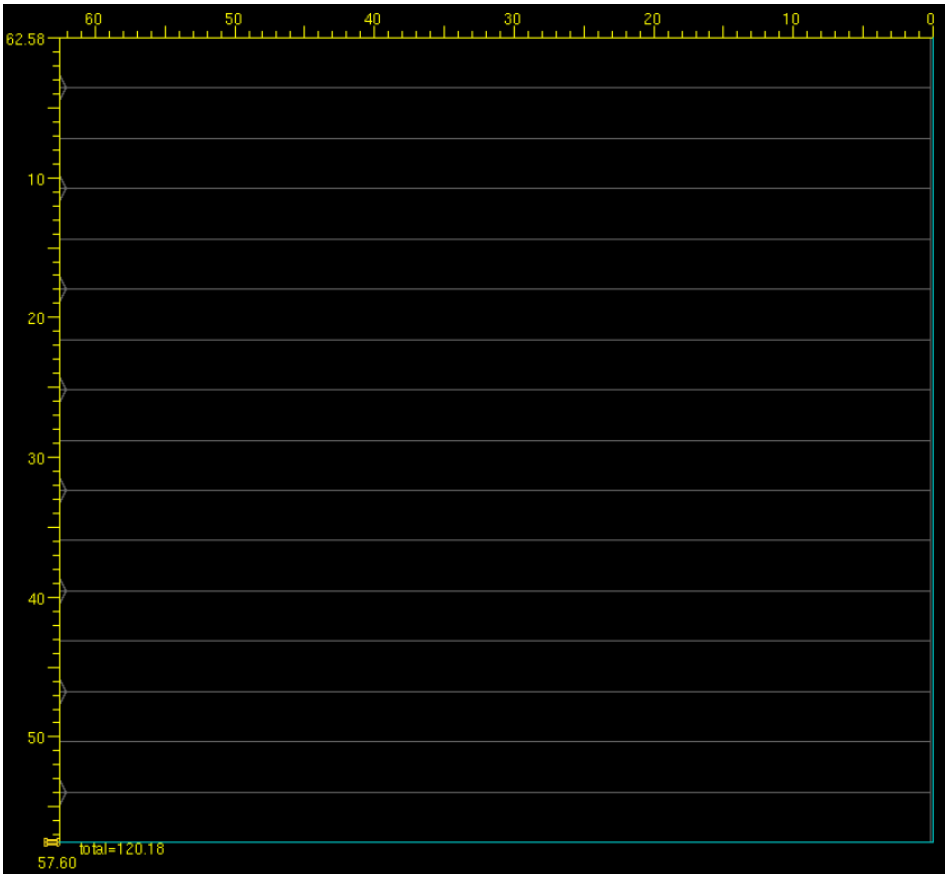

Figure 4-2. Serializer die initial area.

4.2 Floorplan definition

This stage of the physical synthesis is intended to specify the size of the core, considering adequate dimensions for the design and the space for the power rings; the core should be smaller than the die, but greater than the area needed for the standard cells.

The core perimeter is set to be 3.6 µm away to the I/O boundary on every side. Due to the specifications of the floorplan, the tool recalculates a proper size of the die and creates the core, resulting in a die of 6518.8 μm^2 and a core of 5407 μm^2. The floorplan also configures the orientation that the cells will take when placed in the layout and create rows of 3.6 µm each.

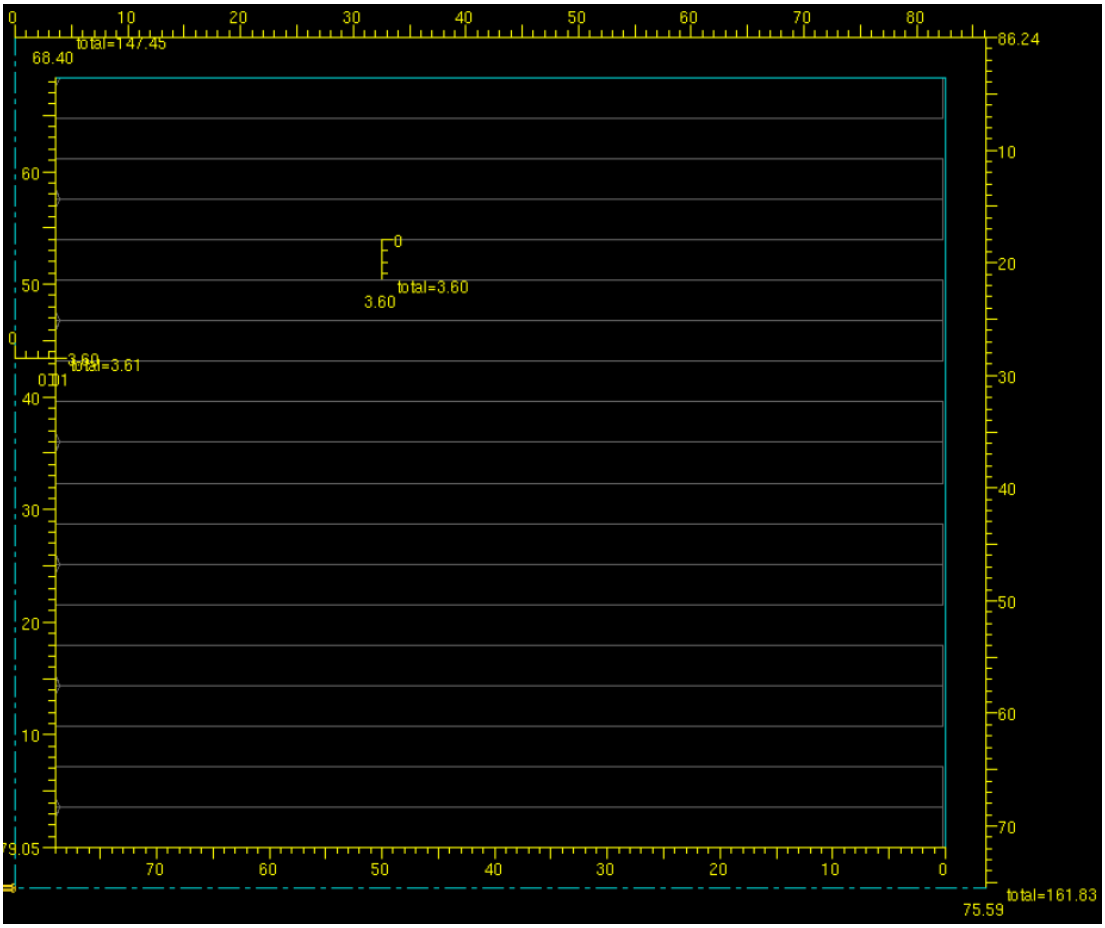

Figure 4-3. Serializer floorplan area.

4.3 Power Grid

The power grid is a power delivery technique used to ensure the correct operation of the chip, as every transistor is wired to voltage, ground or both. In addition, the power grid propagates the power sources through all the chip, avoiding large routings and voltage drops.

4.3.1 Power Rings

Rings for VDD and VSS are placed with two different metals around the core boundaries, granting a uniform power distribution all along the system. Encounter allows the designer to select the geometry and materials of the rings before being placed in the physical design. For this design, the horizontal lanes are placed with “Metal 1” and the vertical lanes with “Metal 2”, and every lane with a width of 1 µm and a space between lanes of 0.5 µm. In Figure 4-4, the external ring drives VSS and the internal ring drives VDD.

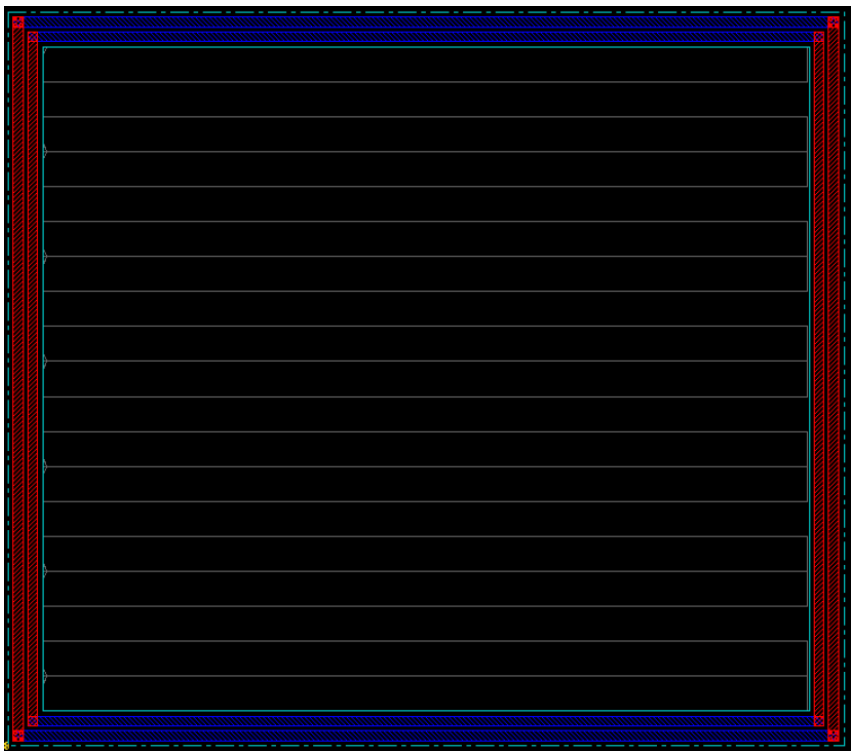

Figure 4-4. Serializer power rings.

4.3.2 Horizontal Power Stripes

To complete a grid, the power distribution must have horizontal and vertical stripes all along the core to effectively achieve any path of the device. The horizontal stripes are considered special routing and can also be automated. The serializer design routes a stripe per row, due to the orientation of the cells, specified at the floorplan, only one stripe for VSS is placed for every two stripes for VDD, as observed in Figure 4-5.

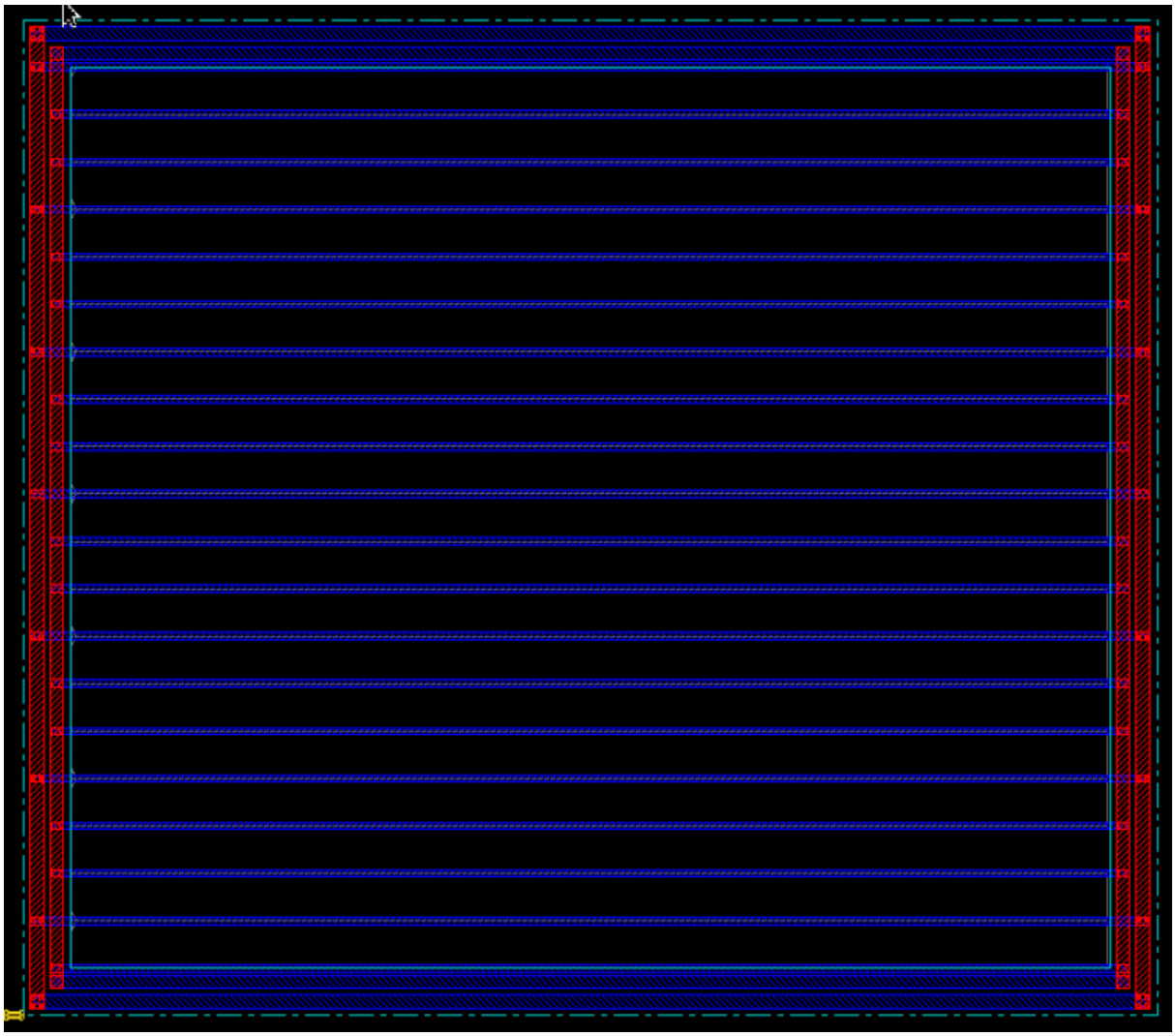

Figure 4-5. Serializer horizontal stripes.

4.3.3 Vertical Power Stripes

To finish with the power grid, vertical stripes are placed in the core, every stripe that is places specifically in the configuration, inserts a VDD and a VSS lane. In some occasions, this step needs to be repeated after routing, because these metal layers can interfere with the design rules. So, if the final design violates geometry or design rules, the vertical stripes might be placed accordingly to comply with the specifications. Considering the size of the device, four stripes are placed from the beginning of the physical synthesis (Figure 4-6).

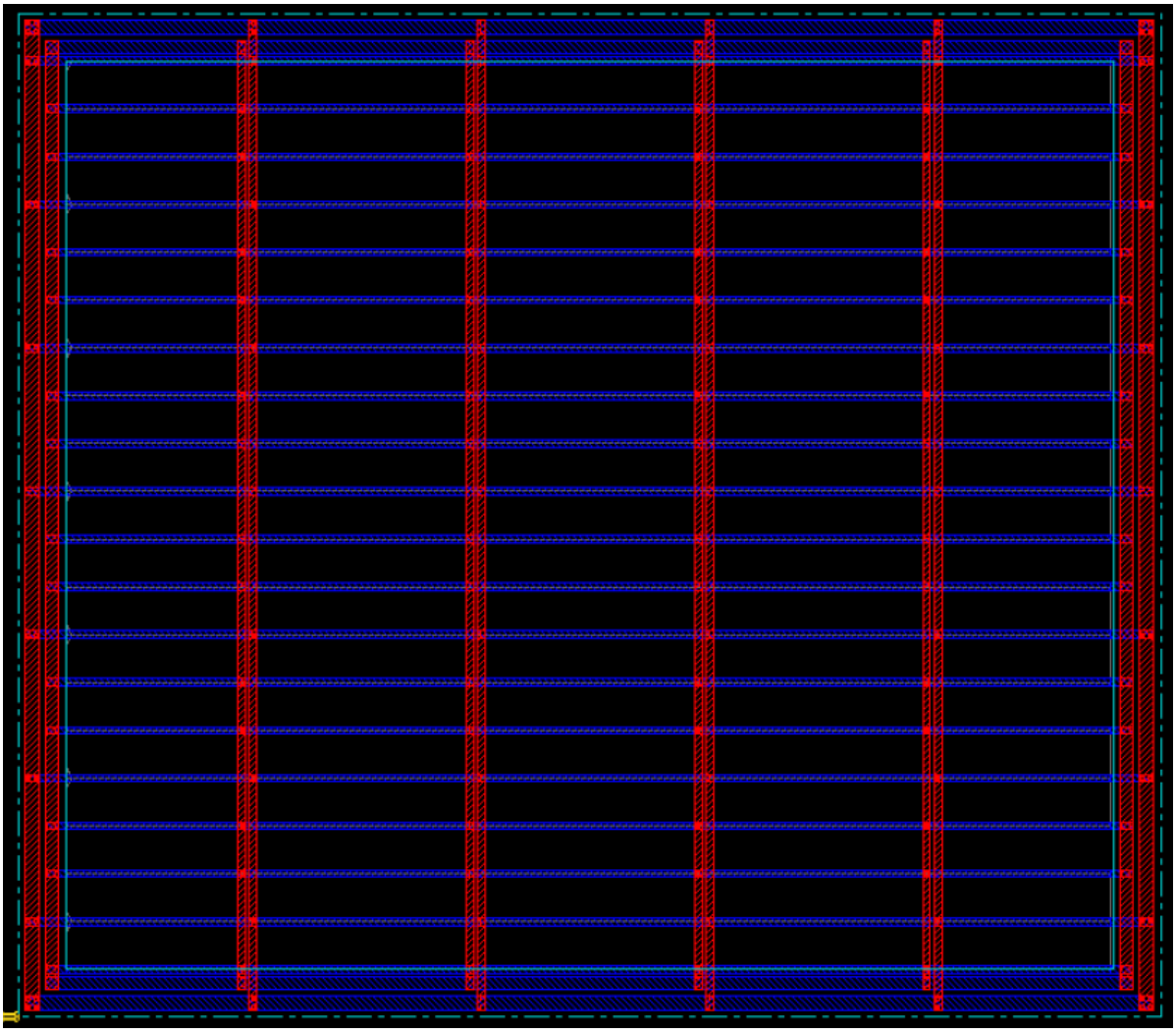

Figure 4-6. Serializer power grid.

4.4 Placement of Standard Cells

The standard cells represent all the logic behind the design and Encounter tries to make an effort to place them in such a way that the routing consists in short wires and then, rearranges the IO pins at the boundary of the die. This step is demonstrated in Figure 4-7.

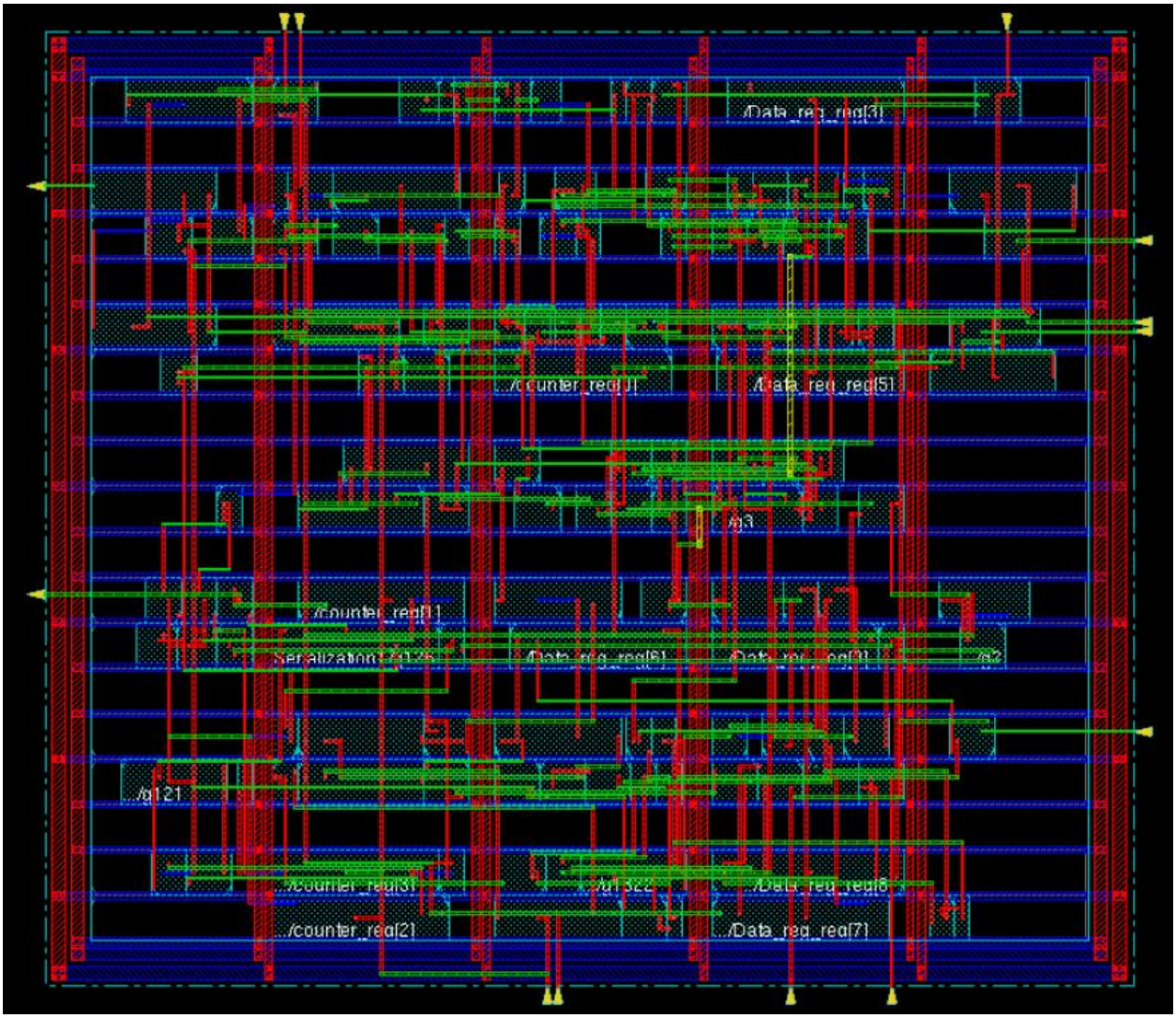

Figure 4-7. Serializer standard cells placement.

4.5 Clock Tree Synthesis

As the serializer implements a sequential logic, it depends on a common clock signal for every register and latch. As a result, the routing of the clock is a priority and its distribution must be done before any other interconnections, as the signal integrity of the entire device has dependency on this single signal. The tool provides the feature for synthetizing a clock tree by placing the buffers, inverters, and delays.

The serializer module physical synthesis places the input clock pin at the bottom of the die, so the tool starts the tree from this point and makes the required branches in the path (Figure 4-8).

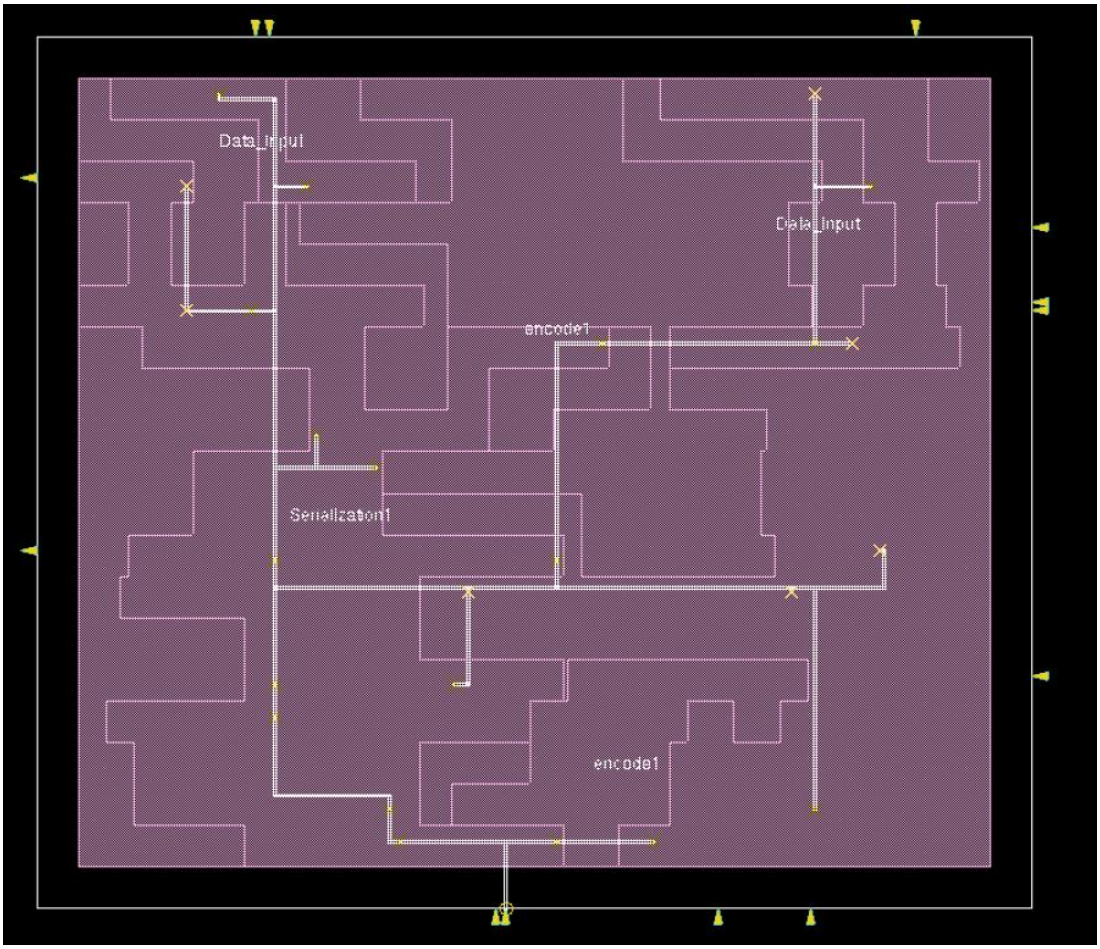

Figure 4-8. Serializer clock tree synthesis.

4.6 Design Routing

Design routing is an automatic process that the Encounter performs and places the corresponding metal layers and vias to connect the terminals of the standard cells based on the previously imported gate level netlist, when routing long wires, the integrity of the signal and the timing can be affected, so it is recommended to adjust the parameters of the tool to make an optimization while routing.

The routing for the serializer cannot be easily identified in the design, but it is possible to perform several automatic verification processes. In order to verify that the routing was done correctly, the connectivity and antenna verifications are called and no error is reported (Figure 4-9). The connectivity verification checks that all the proper wires are placed correctly in the design, while the antenna verification ensures whether any antenna rule is violated. The design must have specific geometry of connections to prevent the deterioration of the transistor gates with the antenna effects.

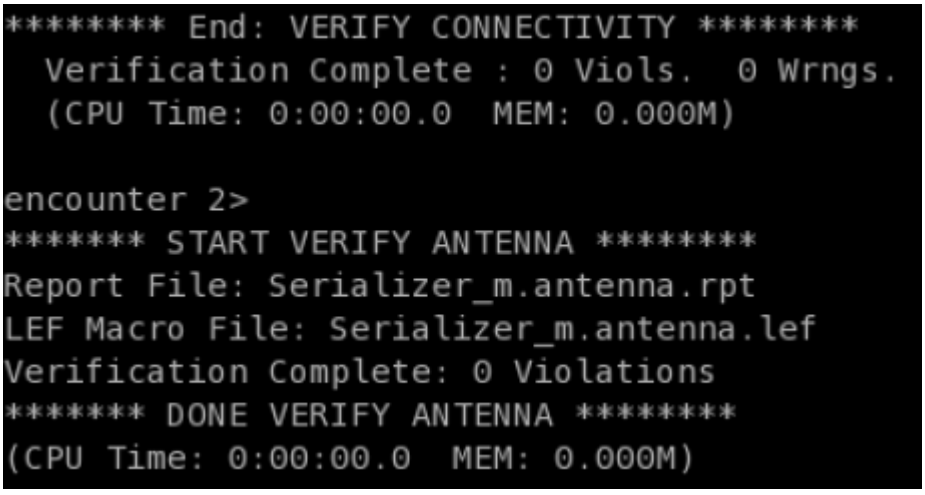

Figure 4-9. Serializer connectivity and antenna verifications

The layout of any system on chip with CMOS technology needs to be continuously tested by performing the different verifications such as: geometry, drc, connectivity, and antenna. All

specifications need to be achieved successfully before manufacturing as they grant that the design is less susceptible to errors during build-up and the yield of the physical device is as expected.

4.7 Timing Optimization

For every stage of a digital design, the timing analysis is essential as it considers all the environment and specifications of the logic gates to determine the time it takes for every signal to reach the endpoints passing through every possible path, in such a way that there are some specific techniques to enhance the data propagation and reach the best possible performance. In the physical synthesis stage, the timing can be compromised by the routing and placement of the cells, this is why a timing optimization during the physical synthesis consists in an iterative process of place and route and thus, selecting as final design the one with the most positive worst slack.

The procedure of the previous physical synthesis is merged with the timing analysis and optimization in the tcl script of Appendix J Physical synthesis encounter scriptThis script is run from the encounter terminal after loading the design and the proper libraries for the physical synthesis. After executing the script, the display shows the final design including floorplan, power grid, placement, clock tree synthesis, and routing.

The timing analysis results at different stages of the place and route with optimization are shown in Table 4-1. The tool is set to employ the worst case library for setup analysis and the typical library for hold mode.

Table 4-1. Place and Route timing optimization

	Before CTS		After CTS		After DRV fixes		After Setup Fixes		After Hold Fixes		After Route	
Mode	*Setup*	*Hold*	*Setup*	*Hold*	*Setup*	*Hold*	*Setup*	*Hold*	*Setup*	*Hold*	*Setup*	*Hold*
Worst Slack	-2.951 ns	0.014 ns	-2.926 ns	-0.064 ns	-3.499 ns	0.024 ns	-0.748 ns	0.028 ns	-0.748 ns	0.028 ns	-0.777 ns	0.019 ns
Total Negative Slack	-22.833 ns	0 ns	-22.574 ns	-0.533 ns	-28.306 ns	0	-5.401 ns	0	-5.401 ns	0	-5.664 ns	0
Violating Paths	11/51	0/51	11/51	10/51	11/51	0/51	11/51	0/51	11/51	0/51	11/51	0/51

The result of the physical synthesis (Figure 4-10) includes eleven out of 51 setup violation paths, so the device is susceptible to errors while working at high temperatures and a low voltage, even when the serializer can be manufactured and works properly in typical or ideal conditions.

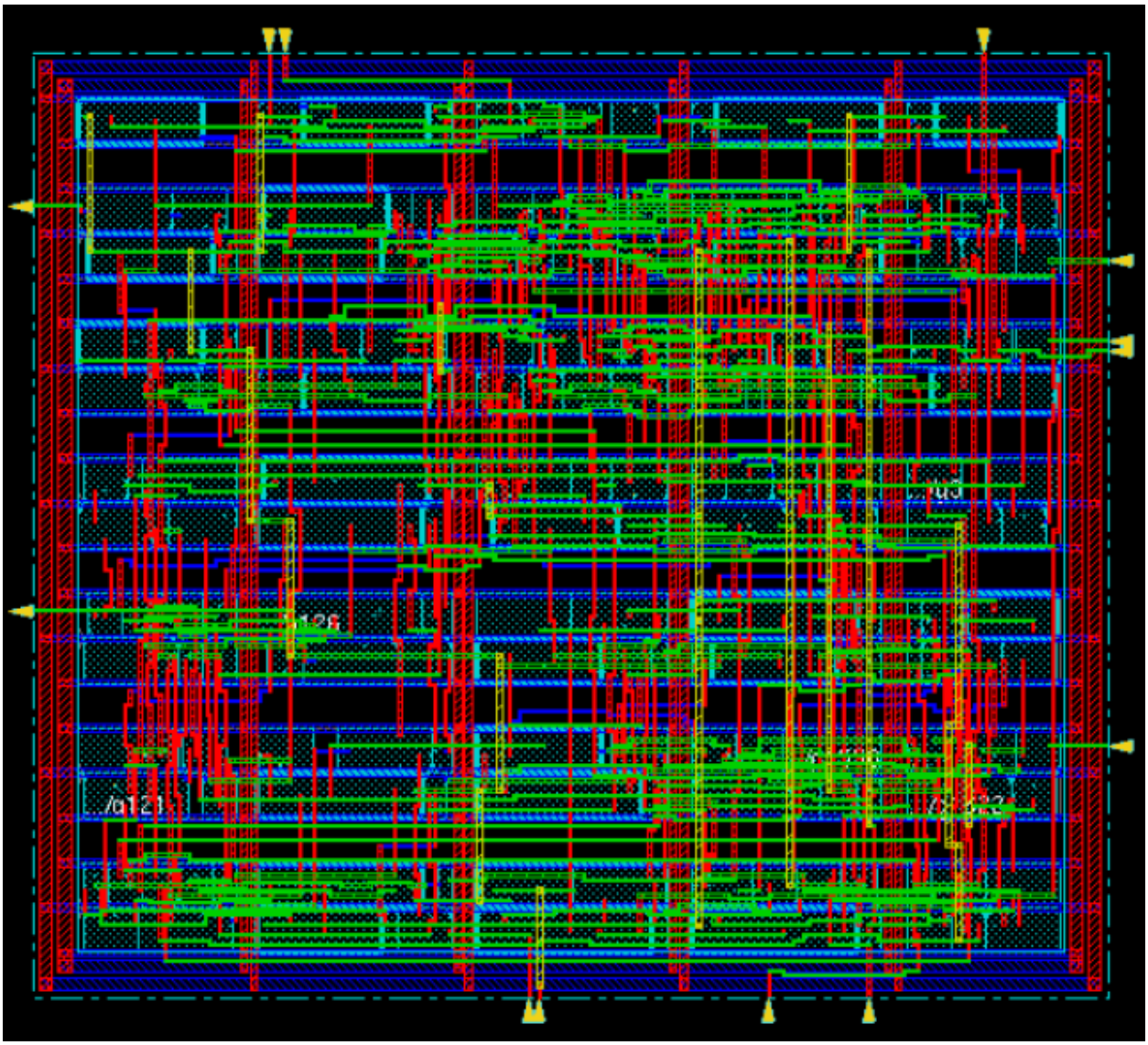

Figure 4-10. Serializer physical synthesis layout with timing optimization.

At this point, the serializer meets all requirements for manufacturing except timing, which is met partially, meaning that at the worst operation conditions, its performance would be affected.

4.8 Filler cells

By observing the cells on the physical design in a floorplan view (Figure 4-11), several spare areas inside the core can be noted. These gaps must be filled with unutilized cells in order to prevent manufacturing errors, extend N-well and P-well regions, and decouple capacitance, creating an electrostatic discharge (ESD).

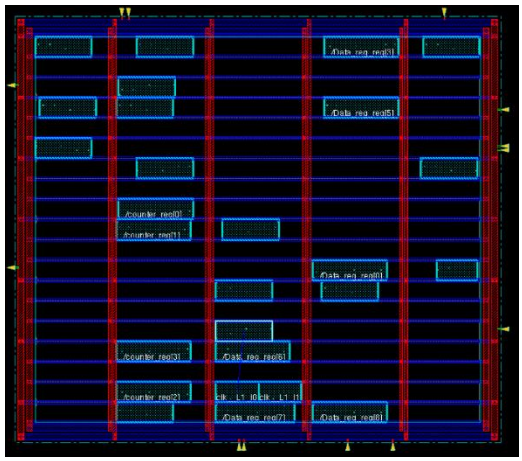

Figure 4-11. Serializer floorplan view.

The filler tool from the encounter selects from the available filler cells in the lef library and places them where they fit and fills the remaining space in the cell rows of the core, obtaining the layout in Figure 4-12 after 134 new instances are placed in the core.

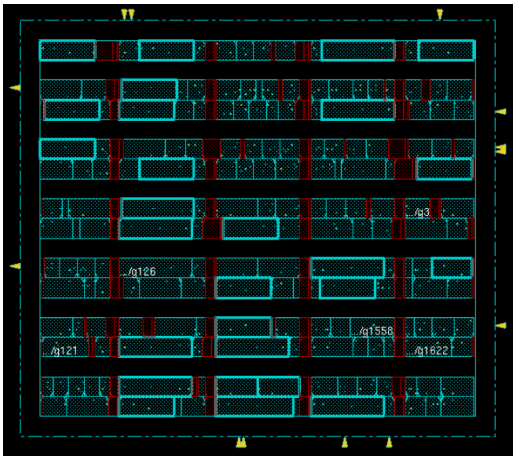

Figure 4-12. Serializer standard cells and filler cells.

4.9 Place and route compilation

At this point, the digital serializer module meets the functional and electrical requirements and does not generate violations at any stage of the design. By adding pads and exporting it into a gds file, the design can be manufactured through a 130 nm manufacturing process.

As mentioned before, the complete integrated circuit is a mixed signal device that consists in the joinning of 5 different modules and it must be integrated in just one die.

Since it has been verified that the digital serializer module can go through the complete design flow without facing any error, the rtl is provided to the one in charge of the integration, expecting him to go from the logic synthesis through the insertion of the pads to achieve the objective of the project.

Chapter 5. SerDes Integration

According to Figure 5-1, the complete system is composed by an analog and a digital module for the serializer deserializer and a digital BIST. The five modules must be routed and comply with the design rules, as well as work correctly while interacting with the rest of the modules with a positive slack.

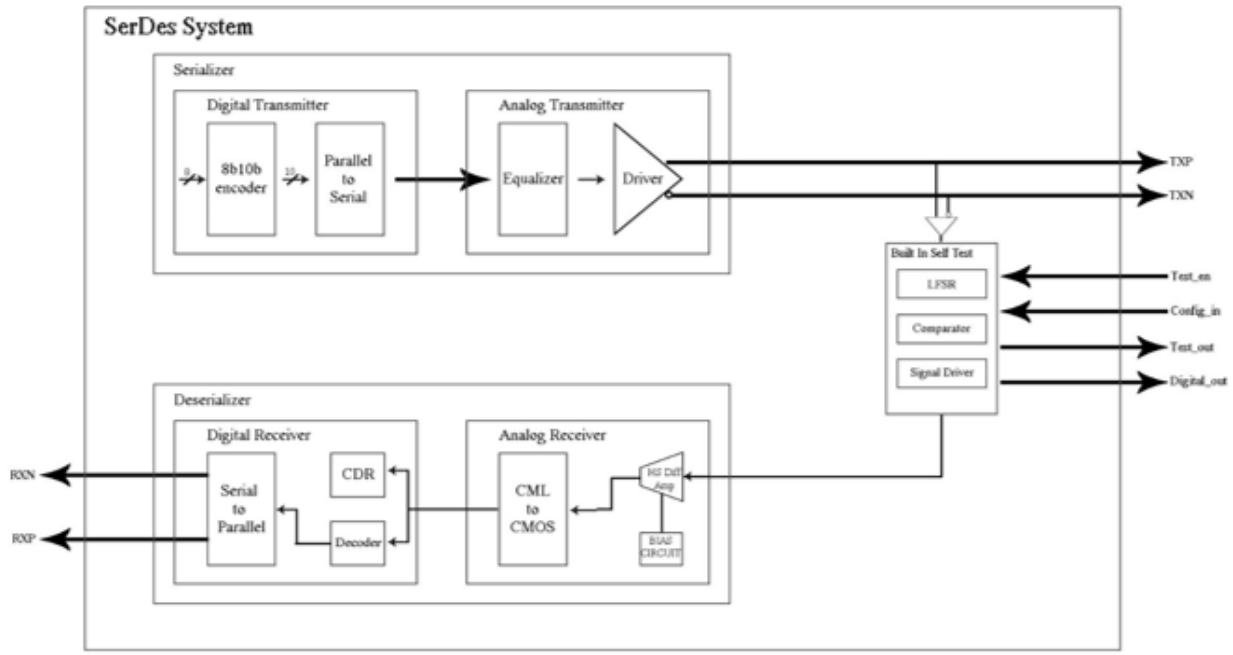

Figure 5-1. SerDes top level architecture.

The integration is done at the rtl level, where the analog modules are represented as black boxes only with inputs and outputs, in such a way that each analog layout module is linked to a lef file generated by the designers with the same 130 nm process implemented by the digital modules.

A lef file is a text description of the placement of metals and vias in a layout, as a result of the work from the analog designers after reaching a clean and working design, with successful DRC and LVS verifications.

The analog designs, from the point of view of the integration, are considered primitive cells and must have a pin for every input and output located at a chipedge layer that represents the boundary of the entire block. The routing to be performed by Encounter should take the pins as the communication interfaces between the analog modules and the rest of the SerDes system.

For the functional device, the top module expresses instances of every analog and digital module in verilog and represents the correct wiring, while the chip top module makes an instance of the functional top module and wire each pin to a specific pad.

Every pad is a verilog description of the interface in charge of the interaction between the outside of the chip and the physical pin implementation of the top module for the serializer-deserializer architecture.

For the further steps, the chip top module is the representation of the full design and contains the information of all the submodules, so the process of integration is the same that has been followed for each digital module, starting with the logic synthesis, going through a static time analysis and exporting a gds file from Encounter to Virtuoso. Encounter should be able to route both analog modules equal to the standard cells, the only difference remains in the size of the cells, so the power grid should be done having in mind that some specific area must be preserved for them.

Despite having every individual effort working correctly, the integration has not been achieved on time for manufacturing. The design flow is followed and reached without errors until the physical synthesis stage, where the layout generates violations that could not been solved in time. The system verifications return two antenna, four DRC and five connectivity violations, as a consequence of a wrong export configurations of the analog modules as Encounter tried to route inside the boundaries of the analog modules and it only should place vias at their pins.

A manual correction in Virtuoso was considered as a possible solution, but with a lack of schematic, the routing became risky and mistakes can be committed with ease because of the amount of components within the design. Therefore, the analog modules should be modified, as well as their lef settings in order to export them as required.

The final design and the closest approach to the correct integration of the SerDes system is in Figure 5-2, the power grid has been modified, as well as the die and core sizes, leaving some specific blank spaces for the analog modules and placing fillers.

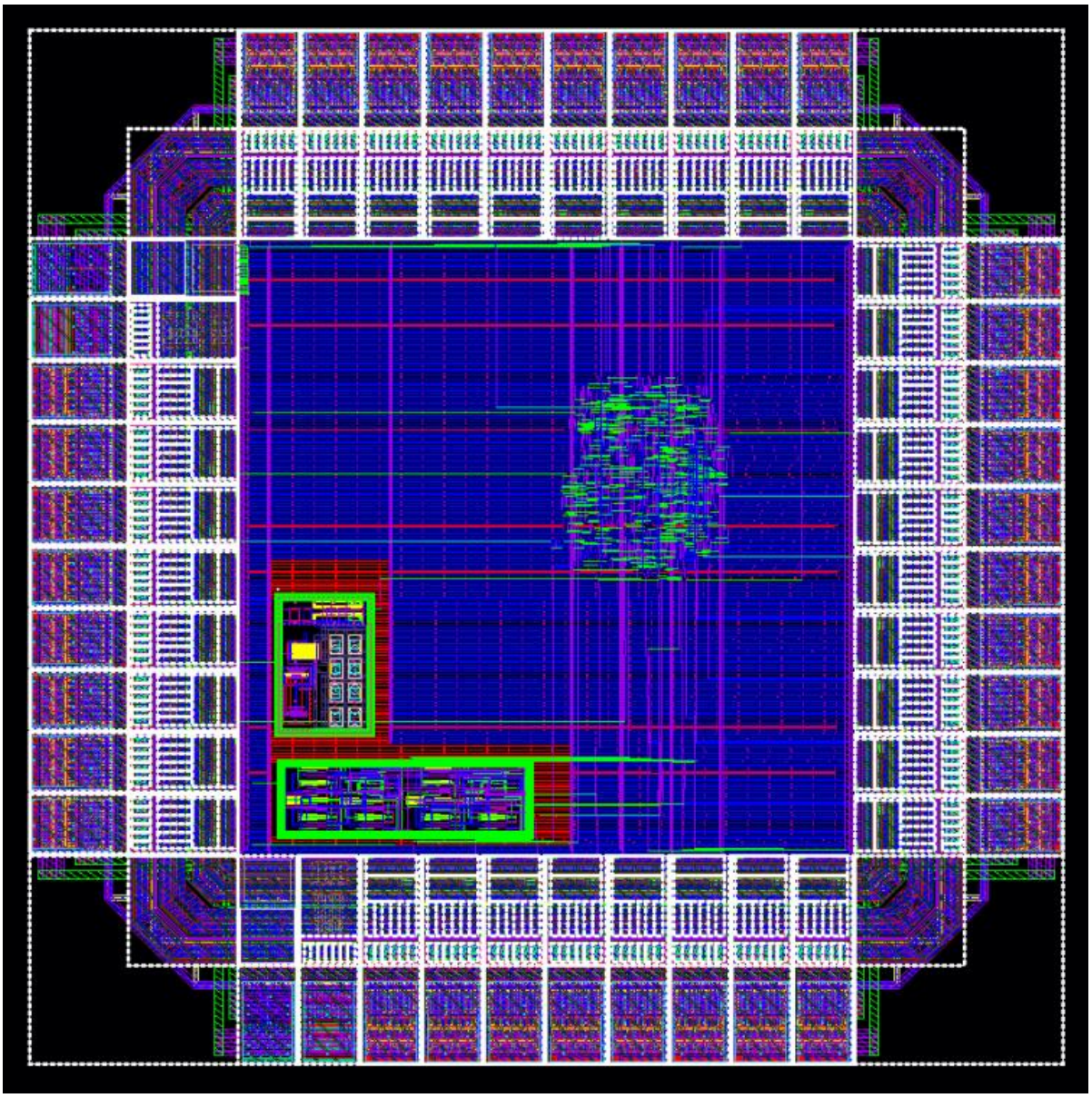

Figure 5-2. Virtuoso final schematic with violations

Conclusion

The SerDes system was defined from the beginning as the effort of five SOC specialty students, splitting the main objective of manufacturing a complete system into individual modules, followed by an integration led by student Miguel Hoil and support from the rest of the designers. All the individual modules specifications were achieved successfully, meeting design rules and timing, in such a way that the simulations turned to be as expected when all the spare blocks were involved. Still, the communication between the members of this project was deficient and the time for integration was too short, leading to the cancelation of the chip's manufacturing process.

There is still work to do involving the integration of mixed signal devices, the information in this and in the parallel thesis should be considered by further generations as the specialty program is established to teach many of this processes in parallel to the development of the project and it usually stops the development of the project as it is common to reach a point where the following tasks correspond to what have not been learnt in the classroom, leaving short time for the next steps.

There were some changes that have been done to the provided rtl of the serializer module, first of all the code was simplified, expecting little changes in the synthesis but not in behavior. A control unit was added, mainly implemented with counters and some XOR based comparator instances. Simulations demonstrated how the disparity was not stored, impacting the real purpose of the encoder, this issue was solved by the addition of a register, while the control unit performs the synchronization of the system.

On the other hand, a testbench for every hierarchy of the serializer was created or enhanced from previous design of Efrain Arrambide, making easier the debug and reducing the chance of errors in the response of the hardware.

The 255 possible data frames were tested with 0 and 1 disparity having the complete serializer module and less testing was performed while working together with the remaining digital modules, confirming that the module works correctly.

The integration process was only a 3-day job, where most of the difficulties appeared, as there was no reference to follow and few time to perform it. Even though, an irregular power grid was created with a script, leaving blank spaces for the analog modules. The placement was done, only having problems during routing as the way the analog modules were exported from Virtuoso was probably incomplete, this is definitely a task in which further work must be done, not only for this project.

In conclusion, the work done during the 2017 specialization partially satisfies the objectives of the program itself, as the yield of the device improved and reached a further step, standing close to the main objective of manufacturing. This effort implies the execution of the complete system on chip design flow and strengthened the individual (specialty students) either digital or analog design skills, which matches the intention of the postgraduate degree.

This project can be retaken further as the proposals of enhancement reside in migrating the same design into a more recent technology and comparing the yield, as well as adding dft (design for testability) in order to validate the manufacturing process.

Appendix A Encoder rtl.

```
/*****************************************************************
* Name:
* Encode.v
* Description: This modules executes de codification from 8 bits to 10b
depending in the last Run Disparity
* Fully combinational
* Inputs:
* data_in: 8 bit word input to transmit and 1 bit for control signals
* disp_in: 0 means last disparity was negative
* Outputs
* dataout: 10 bit output word
* disp_out: disparity of the actual word (10 bit)
* Version:
*     2.0
* Author:
* Chuck Benz, Hollis, NH   Copyright (c)2002
* Permission is granted for any reuse of this information and description as
* long as this copyright notice is preserved.  Modifications may be made as
* long as this notice is preserved.
* Review by:
*     Christian Aparicio Zuleta
* Date:
*     24/04/2017
*****************************************************************/
module encode (datain, dispin, dataout, dispout);
  input [8:0]   datain ;
  input     dispin ;  // 0 = neg disp; 1 = pos disp
  output [9:0]    dataout ;
  output    dispout ;

  wire ai = datain[0] ;
  wire bi = datain[1] ;
  wire ci = datain[2] ;
  wire di = datain[3] ;
  wire ei = datain[4] ;
  wire fi = datain[5] ;
  wire gi = datain[6] ;
  wire hi = datain[7] ;
  wire ki = datain[8] ;

  wire aeqb = (ai & bi) | (!ai & !bi) ;
  wire ceqd = (ci & di) | (!ci & !di) ;
  wire l22 = (ai & bi & !ci & !di) |(ci & di & !ai & !bi)| ( !aeqb & !ceqd) ;
  wire l40 = ai & bi & ci & di ;
  wire l04 = !ai & !bi & !ci & !di ;
  wire l13 = ( !aeqb & !ci & !di) | ( !ceqd & !ai & !bi) ;
  wire l31 = ( !aeqb & ci & di) | ( !ceqd & ai & bi) ;


  // The 5B/6B encoding
  wire ao = ai ;
```

```
  wire bo = (bi & !l40) | l04 ;
  wire co = l04 | ci | (ei & di & !ci & !bi & !ai) ;
  wire d0 = di & ! (ai & bi & ci) ;
  wire eo = (ei | l13) & ! (ei & di & !ci & !bi & !ai) ;
  wire io = (l22 & !ei) |(ei & !di & !ci & !(ai&bi)) |  // D16, D17, D18
            (ei & l40) |(ki & ei & di & ci & !bi & !ai) | // K.28
            (ei & !di & ci & !bi & !ai) ;

  // pds16 indicates cases where d-1 is assumed + to get our encoded value
  wire pd1s6 = (ei & di & !ci & !bi & !ai) | (!ei & !l22 & !l31) ;
  // nds16 indicates cases where d-1 is assumed - to get our encoded value
  wire nd1s6 = ki | (ei & !l22 & !l13) | (!ei & !di & ci & bi & ai) ;
  // ndos6 is pds16 cases where d-1 is + yields - disp out - all of them
  wire ndos6 = pd1s6 ;
  // pdos6 is nds16 cases where d-1 is - yields + disp out - all but one
  wire pdos6 = ki | (ei & !l22 & !l13) ;


  // some Dx.7 and all Kx.7 cases result in run length of 5 case unless
  // an alternate coding is used (referred to as Dx.A7, normal is Dx.P7)
  // specifically, D11, D13, D14, D17, D18, D19.
  wire alt7 = fi&gi&hi&(ki|(dispin ? (!ei & di & l31) : (ei & !di & l13))) ;

  wire fo = fi & ! alt7 ;
  wire go = gi | (!fi & !gi & !hi) ;
  wire ho = hi ;
  wire jo = (!hi & (gi ^ fi)) | alt7 ;
  // nd1s4 is cases where d-1 is assumed - to get our encoded value
  wire nd1s4 = fi & gi ;
  // pd1s4 is cases where d-1 is assumed + to get our encoded value
  wire pd1s4 = (!fi & !gi) | (ki & ((fi & !gi) | (!fi & gi))) ;
  // ndos4 is pd1s4 cases where d-1 is + yields - disp out - just some
  wire ndos4 = (!fi & !gi) ;
  // pdos4 is nd1s4 cases where d-1 is - yields + disp out
  wire pdos4 = fi & gi & hi ;
  wire compls6 = (pd1s6 & !dispin) | (nd1s6 & dispin) ;
  wire disp6 = dispin ^ (ndos6 | pdos6) ;
  wire compls4 = (pd1s4 & !disp6) | (nd1s4 & disp6) ;
  assign dispout = disp6 ^ (ndos4 | pdos4) ;
  assign dataout = {(jo ^ compls4), (ho ^ compls4),
                (go ^ compls4), (fo ^ compls4),
                (io ^ compls6), (eo ^ compls6),
                (d0 ^ compls6), (co ^ compls6),
                (bo ^ compls6), (ao ^ compls6)} ;

endmodule
```

Appendix B Encoder testbench.

```
/************************************************************/
* Name:
*     validate_8b10b.v
* Description:
* This module stresses the inputs of the encoder module in order to validate
it
* Version:
*     1.0
* Author:
*     Christian Aparicio Zuleta
* Date:
*     21/05/2017
************************************************************/
`timescale 1ns / 1ns
module validate_8b10b ;
//Input Ports
reg [8:0] testin;
reg dispin;

//Output
wire [9:0] testout;
wire dispout;

//Internal signals
reg error, errorp;
reg [8:0] control [0:11];
reg [9:0] result [0:11];
reg [9:0] goldendatcontroldisp0 [0:11];
reg [9:0] goldendatcontroldisp1 [0:11];
reg [3:0] i;
integer filehandeler;
encode DUT (testin, dispin, testout, dispout) ;
/*********************************INIT**********************/
initial begin
      $readmemh ("8b10b_control.mem", control) ;
      $readmemb ("8b10b_control_disp0.mem", goldendatcontroldisp0);
      $readmemb ("8b10b_control_disp1.mem", goldendatcontroldisp1);
      filehandeler = $fopen("encoder.log","w");
      error = 1'b0;
      i = 5'b0;
      testin = control[0];
      errorp = 1'b0;
end
/**********Negative Disparrity Control Data*************/
initial begin
      #5;
      for (i = 0 ; i <= 11 ; i = i + 1) begin
            dispin = 0 ;
            testin = control[i];
            #1;
               if (testout != goldendatcontroldisp0[i]) begin
```

```
                  $fdisplay (filehandeler, "Error during verification of
transaction %d expecting %b and received %b", i, goldendatcontroldisp0[i],
testout);
                  error = 1'b1;
                end
                else begin
                   if (error == 0 && i == 11)$fdisplay (filehandeler, "Control
data encode with negative disparity has been succesfully achieved");
                   end
              end

/******************Positive Disparrity Control Data******************/
      #5;
      i = 5'b0;
      testin = control[0];
      #1;
      for (i = 0 ; i <= 11 ; i = i + 1) begin
             dispin = 1 ;
             testin = control[i];
             #1;
                if (testout != goldendatcontroldisp1[i]) begin
                  $fdisplay (filehandeler, "Error during verification of
transaction %d expecting %b and received %b", i, goldendatcontroldisp1[i],
testout);
                  errorp = 1'b1;
                end
                else begin
                   if (errorp == 0 && i == 11)$fdisplay (filehandeler,
"Control data encode with positive disparity has been succesfully achieved");
                end
              end
              #1;
              $fclose(filehandeler);
              $stop;
end
endmodule
```

Appendix C Serialization module rtl.

```
/////////////////////////////////////////////////////
* Name:
*     Serialization.v
* Description:
* This module takes a parallel input and delivers the same data in serial at
the frequency given by itÂ´s clock
* Parameters:
* DATA_LENGTH: size of the word that is going to be serialized
* Inputs:
*     Data_in
*     clk: clock signal
* reset: reset signal (Active Low)
*     rd_in: run disparity received from encoder
* Outputs:
*     rd_out: bypasses rd_in when count = 1
*     serial_out: This port will give the pulses of the information to
transmit
*     start_frame: Gives a pulse 1 clock cycle before the data begins sending
when the
* VersiÃ³n:
*     2.0
* Author:
*     Efrain Arrambide
* Review by:
*     Christian Aparicio Zuleta
* Date:
*     24/04/2017
/////////////////////////////////////////////////////////////*/
module Serialization
#(
      parameter DATA_LENGTH,
      parameter COUNT_LENGTH = CeilLog2(DATA_LENGTH)
)
(
input [DATA_LENGTH-1:0]  Data_in,
input clk, reset, rd_in,
output rd_out, serial_out, start_frame, nine
);

reg [DATA_LENGTH-1:0] par_reg;
reg [COUNT_LENGTH-1:0] counter;
reg disparity;
wire ten, eight;
always @(posedge clk or negedge reset) begin
      if          (~reset) {counter, par_reg} <=
{{COUNT_LENGTH{1'b0}},{DATA_LENGTH{1'b0}}};
      else begin
            if    (!ten)  {counter, par_reg} <= {counter +
1'b1,1'b0,par_reg[DATA_LENGTH-1:1]};
            else  {counter, par_reg} <= {{COUNT_LENGTH-
1'b1{1'b0}},1'b1,Data_in};
```

```
            if (eight) disparity <= rd_in;
            else         disparity <= 0;
      end
end

//Combinational logic
Control_serial #(COUNT_LENGTH) Control_PISO(counter, eight, nine, ten);
assign rd_out = disparity;
assign serial_out = par_reg[0];
assign start_frame = (ten)? 1'b1: 1'b0;

/*************************************************************************/
 /*Log Function*/
     function integer CeilLog2;
       input integer data;
       integer i,result;
       begin
          for(i=0; 2**i < data; i=i+1)
             result = i + 1;
          CeilLog2 = result;
       end
    endfunction
/*************************************************************************/
endmodule
```

Appendix D Control unit rtl.

```
/////////////////////////////////////////////////////////////
* Name:
*     Control_serial.v
* Description:
* This module delivers enable signals, based in comparators
* Parameters:
* COUNT_LENGTH: size of the counter
* Inputs:
*     Counter
* Outputs:
* eight, nine, ten
* VersiÃ³n:
*     1.0
* Author:
*     Christian Aparicio Zuleta
* Date:
*     24/04/2017
/////////////////////////////////////////////////////////////*/
module Control_serial
#(
      parameter COUNT_LENGTH
)
(
      input [COUNT_LENGTH-1:0] counter,
      output eight, nine, ten
);

Comparator #(COUNT_LENGTH) Comp_Counter_eight(4'd8,counter,eight);
Comparator #(COUNT_LENGTH) Comp_Counter_nine(4'd9,counter,nine);
Comparator #(COUNT_LENGTH) Comp_Counter_ten(4'd10,counter,ten);

endmodule
```

Appendix E Comparator rtl.

```
/*************************************************************
* Name:
*     Comparator.v
* Description:
*     This module returns 1 when the 2 input n bits variable is wired.
*     Combinational implementation
* Inputs:
*     Data A
*     Data B
* Outputs
*     Comp_out
* VersiÃ³n:
*     1.0
* Author:
*     Christian Aparicio Zuleta
* Fecha:
*     09/09/2017
*************************************************************/
module Comparator
#(
      parameter WORD_LENGTH = 8
)
(
      // Input ports
      input [WORD_LENGTH-1 : 0] Data_A,
      input [WORD_LENGTH-1 : 0] Data_B,

      // Output Ports
      output Comp_out
);
//Internal wires
wire [WORD_LENGTH-1 : 0] XOR_out;
wire OR_out;
//Combinational logic
assign XOR_out = Data_A ^ Data_B;
assign OR_out = | XOR_out;
assign Comp_out =  !OR_out;
endmodule
```

Appendix F Comparator testbench.

```
module Comparator_TB;
parameter WORD_LENGTH = 8;
//Input ports
reg [WORD_LENGTH-1 : 0] regData_A;
reg [WORD_LENGTH-1 : 0] regData_B;

//Output Ports
wire Comp_out;

Comparator
#(
      // Parameter Declarations
      .WORD_LENGTH(WORD_LENGTH)
)
DUV
(
        //Input Ports
        .Data_A(regData_A),
        .Data_B(regData_B),
        .Comp_out(Comp_out)
);

/*****************************************/
initial begin
    regData_A = 0;
    regData_B = 0;
    forever #2 begin
        regData_A = regData_A+1;
        #1
        regData_B = regData_B+1;
    end
end
/********************* Kill sim **********/
initial begin
      #3600
      $stop;
end
/******** Save trn info ****/
initial begin
      $recordfile("/home/caparicio/Documents/SERDES/Serializer_Digital/simulat
ion_trn/comparatorwaveform.trn");
      $recordvars();
end
endmodule
```

Appendix G Top module testbench.

```
/************************************************************
* Name:
*     TB.sv
* Description:
* This module stresses the inputs of the serializer module in order to
validate it
* Version:
*     2.0
* Author:
*     Efrain Arrambide
* Review by:
*     Christian Aparicio Zuleta
* Date:
*     24/04/2017
************************************************************/
module TB();
reg clk, rst, set_rd,compare;
reg [8:0] data;
reg [9:0] compare_data;
reg [3:0] counter;
logic [9:0] golden_data;
logic [9:0] error_count, match, iterations;
logic rd_out_encode;
wire rd_out, frame, data_serial;
wire [9:0] dataout;
integer filehandeler;
/*****************************************************/
//Clock Generator
always begin
      #4 clk = !clk;
end
/*****************************************************/
//DUT
Serializer #(10) DUT (clk, rst, data, data_serial, frame);
/*****************************************************/
//Data Generation (1 to 1024)
initial begin
      START;
    while (data < 255) begin
              set_rd = rd_out_encode;
              @(posedge clk) begin #1
                 if (counter == 9 ) begin
                  golden_data = TB.DUT.encode1.dataout;
                  rd_out_encode = TB.DUT.encode1.dispout;
                 end
                     compare_data[counter - 1'b1] = data_serial;
              end
            @(negedge clk) begin #1
                  if (counter == 10) begin
                    if (compare == 1 ) begin
                      iterations  = iterations + 1;
                          if (compare_data === golden_data) begin
```

```
                                $fdisplay(filehandeler, "\n -Info- Match
Generate data = %d,  Expected data encoded= %h, Run disparity
%d",data,compare_data,rd_out_encode);
                                match = match + 1;
                            end else begin
                                    $fdisplay(filehandeler, "\n -Error-
Mismatch- Valor del dato generado = %h, valor esperado = %h",compare_data,
golden_data);
                                    error_count = error_count + 1;
                            end
                        end
                      #1 data = data + 1;
                  end
            end
    end
    $fdisplay(filehandeler, "\n\n------Simulation Report---------\nNumber of
iterations: %d\nNumber of errors: %d\nNumber of matching data: %d",
iterations, error_count, match);
    #2
    $fclose(filehandeler);
    $stop;
end

/*****************************************************/
always @(negedge frame) begin
  counter = 4'b1;
  compare = (compare_data != 3'h0)? 1'b1: 1'b0;
end
/*****************************************************/
always @(posedge clk) begin
      counter = counter + 1'b1;
end
/*****************************************************/
task START;
      begin
        filehandeler =
$fopen("/home/caparicio/Documents/SERDES/Serializer_Digital/Logs/transactions
.log","w");
        {clk, rst, data, match, iterations, error_count, set_rd, compare_data,
compare, counter} = {{57{1'b0}}, 1'b1};
         @(posedge clk) #5 rst = 1;
      end
endtask

/******** Save trn info ****/
initial begin
      $recordfile("/home/caparicio/Documents/SERDES/Serializer_Digital/simulat
ion_trn/serializerwaveform.trn");
      $recordvars();
end
endmodule
```

Appendix H Serializer_performance_synthesis.tcl

```
#### Template Script for RTL->Gate-Level Flow (generated from RC RC14.26 -
v14.20-s058_1)

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 Serializer
set SYN_EFF high
###set MAP_EFF medium
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}
# Variable to specify the technology .lib file name
set timing_library scx3_cmos8rf_lpvt_tt_1p2v_25c.lib
set my_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}
set_attribute lib_search_path {/opt/libs/ARM/IB03LB501-FB-00000-r0p0-
00rel0/aci/sc-x/synopsys} /
set_attribute script_search_path {..} /
set_attribute hdl_search_path
{/home/caparicio/Documents/SERDES/Serializer_Digital/rtl} /
set_attribute information_level 9 /

##############################################################
## Library setup
##############################################################

set_attribute library $timing_library
set_attribute lef_library $my_lef_library /
set_attribute auto_ungroup none /

##############################################################
## Load Design
##############################################################

read_hdl -v2001 { Comparator.v Control_serial.v Serialization.v encode.v
Register.v Serializer.v }
elaborate $DESIGN
puts "Runtime & Memory after 'read_hdl'"
timestat Elaboration
```

```
check_design -unresolved

#####################################################################
## Constraints Setup
#####################################################################

read_sdc
/home/caparicio/Documents/SERDES/Serializer_Digital/rc_synthesis/Serializer_o
pt.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

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
puts "Runtime & Memory after 'synthesize -to_generic'"
timestat GENERIC
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_generic.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

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 -verbose -no_exit -revised_design
${_OUTPUTS_PATH}/${DESIGN}_intermediate.v -logfile
${_LOG_PATH}/rtl2intermediate.lec.log >
${_OUTPUTS_PATH}/rtl2intermediate.lec.do

################################################################################
## Incremental Synthesis
################################################################################

## Uncomment to remove assigns & insert tiehilo cells during Incremental
synthesis
##set_attribute remove_assigns true /
##set_remove_assign_options -buffer_or_inverter <libcell> -design
<design|subdesign>
##set_attribute use_tiehilo_for_const <none|duplicate|unique> /
synthesize -to_mapped -eff $MAP_EFF -incr
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
}

report qor > $_REPORTS_PATH/${DESIGN}_qor.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_hdl  > ${_OUTPUTS_PATH}/${DESIGN}_m.v
write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_m.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
##Uncomment if the RTL is to be compared with the final netlist..
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
puts "============================"
puts "Synthesis Finished ........."
puts "============================"

file copy [get_attr stdout_log /] ${_LOG_PATH}/.

##quit
```

Appendix I Serializer_opt.sdc

```
# ITESO University
# Cuauhtemoc Aguilera
# User Constraint File

set_time_unit -picoseconds
set_load_unit -femtofarads

# Clock definition
define_clock -name 156MHz_CLK -period 6400 -rise 10 -fall 90 [clock_ports]

# slew rate definitions (min rise, min fall, max rise, max fall).
# The values coming from IBM typical specification.
set_attribute slew { 28 28 28 28 } 156MHz_CLK

# network clock latency
set_attribute clock_network_late_latency 90 156MHz_CLK
set_attribute clock_network_early_latency 70 156MHz_CLK
# source clock latency
set_attribute clock_source_late_latency 50 156MHz_CLK
set_attribute clock_source_early_latency 40 156MHz_CLK

# clock skew
set_attribute clock_setup_uncertainty {17 10} 156MHz_CLK
set_attribute clock_hold_uncertainty {14 8} 156MHz_CLK


# Input delay definition: This is the delay coming from outside the design
# for this design it's defined at 1% the period of the clock.
external_delay -clock [find / -clock 156MHz_CLK] -input 64 -name IDelay [find
/des* -port ports_in/*]

# Output delay definition: This is the delay going outside the design
# for this design it's defined at 1% the period of the clock.
external_delay -clock [find / -clock 156MHz_CLK] -output 64 -name ODelay
[find /des* -port ports_out/*]

# Driving cell definition
set_attribute external_driver [find [find / -libcell BUFX8TS] -libpin Y]
/designs/Serializer/ports_in/*


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

# The input capacitance for a NOR4X8 cell is 24.9fF  considering fanout of 5
and the wires caps:
set_attribute max_capacitance 130 /designs/Serializer

# Considering a pad output buffer POC2A load
set_attribute external_pin_cap 31 /designs/Serializer/ports_out/*
```

```
# Setting maximum value of fanout
set_attribute max_fanout 5 /designs/*

#set_attribute lp_power_unit {uW}
set_attribute lp_power_optimization_weight 0.2 [current_design]

## To enable the recommended leakage power optimization flow,  use the root
## attribute  leakage_power_effort  set to low, medium or high-
## with an optional specification of max_leakage_power attribute for a
specific power budget.
## Setting leakage_power_effort to 'none' will enable the backward compatible
mode.
#set_attribute leakage_power_effort high
set_attribute max_leakage_power 300 [current_design]
set_attribute max_dynamic_power 900 [current_design]
```

Appendix J Physical synthesis encounter script

```
#Serializer design for a SerDes system in CMOS130 nm
#Christian Aparicio
#November 2017
#
#Run in encounter terminal after loading the design
#Defining process mode
setDesignMode -process 130

#Define floorplan
getIoFlowFlag
setFPlanRowSpacingAndType 3.6 2
setIoFlowFlag 0
floorPlan -dieSizeByIoHeight max -site IBM13SITE -r 0.920421860019 0.699904
3.6 3.6 3.6 3.6

#Define global nets
clearGlobalNets
globalNetConnect VDD -type pgpin -pin VDD -inst * -module {} -verbose
globalNetConnect VSS -type pgpin -pin VSS -inst * -module {} -verbose
globalNetConnect VDD -type tiehi -pin VDD -inst * -module {} -verbose
globalNetConnect VSS -type tielo -pin VSS -inst * -module {} -verbose

#Create power rings
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 1 -spacing 0.5 -offset 0.2

#Create horizontal stripes
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 } -nets { VDD VSS } -allowLayerChange 1 -
blockPin useLef -targetViaLayerRange { M1 MA }

#Create vertical stripes
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 4 -skip_via_on_pin Standardcell -stacked_via_top_layer MA -
padcore_ring_top_layer_limit M3 -spacing .28 -xleft_offset 13 -xright_offset
13 -merge_stripes_value 0.2 -layer M2 -block_ring_bottom_layer_limit M1 -
width .56 -nets {VDD VSS} -stacked_via_bottom_layer M1
```

```
Puts "Power Grid Done"
Puts "Starting placement"
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
Puts "Placement Done"

#FE-CTS
setCTSMode -engine ck
createClockTreeSpec -bufferList {BUFX12TS BUFX16TS BUFX20TS BUFX2TS BUFX3TS
BUFX4TS BUFX6TS BUFX8TS CLKBUFX12TS CLKBUFX16TS CLKBUFX20TS CLKBUFX2TS
CLKBUFX3TS CLKBUFX4TS CLKBUFX6TS CLKBUFX8TS CLKINVX12TS CLKINVX16TS
CLKINVX1TS CLKINVX20TS CLKINVX2TS CLKINVX3TS CLKINVX4TS CLKINVX6TS CLKINVX8TS
DLY1X1TS DLY1X4TS DLY2X1TS DLY2X4TS} -file Clock.ctstch
clockDesign -specFile Clock.ctstch -outDir clock_report -fixedInstBeforeCTS
displayClockTree -skew -allLevel -clkRouteOnly

#CTS Optimization
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
clockDesign -specFile Clock.ctsch -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

Puts "Setting Optimizaiton Mode Options for DRV fixes"
setOptMode -fixFanoutLoad 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
```

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

Puts "Optimizing for Setup"
optDesign -postCTS

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

setOptMode -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

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

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

References

[1] "High-speed digital system design, a handbook of interconnect theory and design practices.[Book Review]", in IEEE Signal Processing Magazine, vol. 18, no. 2, pp. 58-58, March 2001.

[2] Song Yu-yun, Hu Qing-sheng and Hu Liming, "A 0.18μm pipelined 8B10B encoder for a high-speed SerDes", *2010 IEEE 12th International Conference on Communication Technology*, Nanjing, 2010, pp. 1039-1042.

[3] Bui Chinh Hien, Seok-Man Kim and Kyoungrok Cho, "Design of a wave-pipelined serializer-deserializer with an asynchronous protocol for high speed interfaces", *2012 4th Asia Symposium on Quality Electronic Design (ASQED)*, Penang, 2012, pp. 265-268.

[4] K. Iniewski and M. Syrzycki, "Low power 2.5 Gb/s serializer for SOC applications", *IEEE Computer Society Annual Symposium on VLSI*, 2004, pp. 211-212.

[5] W. Y. Tsai, C. T. Chiu, J. M. Wu, S. S. H. Hsu, Y. S. Hsu and Y. F. Tsao, "A novel low gate-count serializer topology with Multiplexer-Flip-Flops", *2012 IEEE International Symposium on Circuits and Systems*, Seoul, 2012, pp. 245-248.

[6] G. E. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.", in *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33-35, Sept. 2006.

[7] Intel Corporation, "Excerpts A Conversation with Gordon Moore: Moore's Law", United States. 2005.

[8] Flamm Kenneth "The end of Moore's law", *2017 University of Texas at Austin*, April 2017.

[9] D. R. Stauffer, Ed., *High speed serdes devices and applications*. New York: Springer, 2008.

[10] X. Bai, P. Patel, y X. Zhang, "A new statistical setup and hold time definition», en *IC Design & Technology (ICICDT)," 2012 IEEE International Conference on*, 2012, pp. 1–4.

[11] Y.-C. Chu, "Serial-GMII specification", *Revision*, vol. 1, p. 3, 2001.

[12] A. X. Widmer y P. A. Franaszek, "A DC-balanced, partitioned-block, 8B/10B transmission code", *IBM J. Res. Dev.*, vol. 27, n.º 5, pp. 440–451, 1983.

[13] S. Yu-yun, H. Qing-sheng, y H. Liming, "A 0.18 μm pipelined 8B10B encoder for a high-speed SerDes", en *Communication Technology (ICCT), 2010 12th IEEE International Conference on*, 2010, pp. 1039–1042.