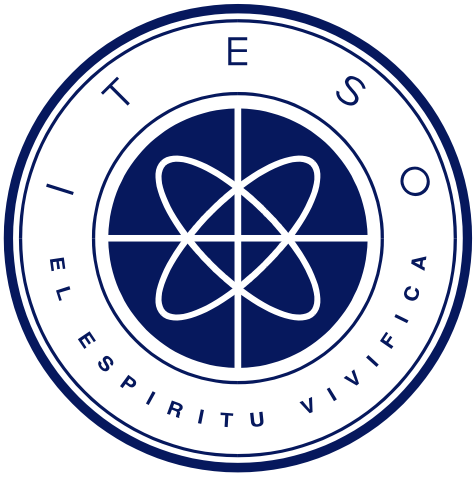
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Especialidad en Diseño de Sistemas en Chip

Reconocimiento de Validez Oficial de Estudios de nivel superior según Acuerdo Secretarial 15018,

publicado en el *Diario Oficial de la Federación* el 29 de noviembre de 1976

DEPARTAMENTO DE ELECTRÓNICA, SISTEMAS E INFORMÁTICA



Design and Integration of a Deserializer Module for a SerDes Mixed Signal System on Chip

Tesina para obtener el grado de:

Especialista en diseño de sistemas en chip

Presenta

José Manuel Centeno Quiñonez

Nombre de los directores de proyecto:

Dr. Víctor Avendaño Fernández, Mtro. Alexandro Girón Allende,

Mtro. Cuauhtémoc Rafael Aguilera Galicia

Guadalajara, Jalisco, Diciembre 2015

ACKNOWLEDGMENTS

The author would like to thank all the ITESO EDSEC teachers for their detailed lectures, expert consulting and approachable attitude. You proved to be extremely valuable for this endeavor. Special mention to Esteban and Efrain for going above and beyond your duties to help students achieve their dreams. You are a living testament of ITESO values.

My lasting gratitude to project advisors Alex, Victor, and Cuauhtemoc, without such generous and experienced mentors this project would have never made it this far.

The author would like to give his parents a big recognition. Thank you for teaching me that learning is a live long endeavor. Thank you for always being there for me, and for always having unlimited faith in me.

To my dearest classmate and wife Citlali. Your daily inspiration and love fills my life with joy. Thank you for sharing this journey with me, I can't wait to start the next great adventure by your side.

Finally, I want to thank PNPC-CONACYT for funding this research. CONACYT consistently empowers Mexican researchers to contribute to the scientific endeavor. I wish one day every institution in Mexico will motivate progress, across all disciplines and geographies, like CONACYT does.

ABSTRACT

This document presents a mixed signal System on Chip design developed at ITESO as a test vehicle for manufacturing at 180 nm node through ***MOSIS*** manufacturing services. The ITESO TV1 is a serializer and deserializer based on the PCIe protocol. The current report includes details on the design for the deserializer digital module for the SerDes. The testing done at RTL and gate level abstractions to the Deserializer are presented. The report finishes by discussing the VLSI technique used for the system integration of analog and digital modules in the SerDes. This research effort is a first attempt at ITESO to generate a mixed signal complex system for manufacturing.

TABLE OF CONTENTS

LIST OF FIGURES & LIST OF TABLES

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The ITESO Test Vehicle 1 (ITESO TV1) is a system defined and implemented to explore a modern design flow for mixed signal Integrated Circuits. The explored flow follows Very Large Scale Integration (VLSI) techniques. This effort will ultimately expand the institutional knowledge base and expertise in the university to design, implement and fabricate scalable Systems on Chip (SoC).

The main goal for this research is to reduce the amount of uncertainty towards the goal of designing and fabricating SoCs at a 180 nm process node with the current tools and capabilities available at the ITESO integrated circuit lab. The full ITESO TV1 brings together analog full custom cells, digital Register Transfer Level (RTL) hardware descriptions and test infrastructure into the system on chip for tape out.

Modern systems on chip are integrated by bringing together several subsystems also commonly called IP (Intellectual Property). Each subsystem or IP can be designed and implemented by applying one of the following layout techniques [1]:

- ASIC VLSI layout
- Datapath layout
- Analog layout
- Custom digital layout
- Cell layout.

At ITESO the datapath layout [2], and custom digital layout [3] have been explored in previous research efforts. This report shows the progress at exploring the ASIC VLSI flow for ITESO TV1. A serializer and Deserializer (SerDes) system, based on the PCIe protocol, is the overall system chosen to bring together modules designed with three of the previously stated techniques.

Specifically Chapter 1 and 2 present the design of a digital Deserializer module for the SerDes system. Chapter 3 and 4 will present great progress on how the Deserializer module can be integrated with other heterogeneous modules to accomplish the full system physical synthesis.

The digital subsystems of the SerDes design is developed using a hardware descriptions in Verilog. The Verilog description is synthesized using standard cells, and

it is automatically placed and routed by Electronic Design Automation (EDA) tools. The digital modules for the full system include:

- Serializer
- Deserializer
- Encoder
- Decoder
- Test infrastructure

The SerDes system also considers the integration of layouts for analog IPs. Modules planned to be integrated straight from an analog layout are the following modules:

- Analog Receiver Amplifier
- Custom Digital Transmitter Amplifier

In general, the SoC integration and physical synthesis are complex processes involving many IPs and levels of integration, which require verification of each synthesis step. The VLSI flow proposed explores some of the common techniques to verify the synthesis. Verification covers two objectives, first to ensure the functionality matches the original RTL description, and second to increase the probability of successful manufacturing of the design.

While the number of IPs and analog designs in the SerDes system is not impressive, the mix of ingredients makes this effort unique at the ITESO University. The automatable nature of the VLSI flow described on this report can set a new horizon for complexity and size in the systems designed at the ITESO integrated circuit lab. Mixed signal systems allow for the highest level of complexity in terms of system integration, which is the reason we have explored so many tools and need to define so many ingredients.

The final result of this effort is a SerDes system layout close to one that can be taped out for manufacture. For full SerDes manufacturing there are three ingredients missing at the time of creation of this document: Power grid design without crossing over analog cells, a pad ring for Inputs and Outputs (IOs), and a proper Layer Stream Definition (.map). These three ingredients would allow a system to be fabricated, even if PCIe performance goals are not met yet. After these ingredients are developed, A PLL would enable the circuit to run faster, and therefore be compliant with timing restrictions on the PCIe standard. This report and all the intellectual property developed is part of ITESO’s research effort in the field of System On Chip design.

CHAPTER 1: BACKGROUND OF A PCIE SERDES SYSTEM

A Serializer-Deserializer (SerDes) is a common Input/Output (IO) design used in digital communications. It consists of a pair of blocks that translate two different types of communication. A serial link of communication and a parallel bus of data. The SerDes is a common pair to connect an agent on a network to other blocks. [4]

A. PCIe Protocol For ITESO TV1

The PCIe protocol is a serial communication protocol popular in computer designs. The PCIe protocol is based on an 8b/10b encoder described by Widmer and Franaszek [5]. PCIe defines differential lanes for the physical layer. A PCIe interface can have several diffrential pairs, called lanes, to increase bandwidth.

The PCIe protocol has other sets of rules, strategies and protocols for various layers of communication such as software configurations and network topologies. All the protocol layers beyond the physical layer of the interface are irrelevant for the ITESO Test Vehicle 1 (ITESO TV1).

The 8b/10b encoding used in PCIe takes an 8 bit payload and converts it into 10 bit serial frames with a balanced set of zeros and ones, which makes it good for transmision lines. The encoding also guarantees a constant switch in the transmision lane. A Phase Lock Loop in the receiver can be used to keep a system clock in sync with the input stream of data. Using any technique to recover the data and the clocks from the differential lane in the PCIe is known as “clock-data recovery”.

As stated before, the encoding is an integral part of the serial transmision, as it eliminates the need to connect a clock between systems. Therefore, an encoder and decoder are included as part of the ITESO TV1 SerDes sytem.

B. ITESO TV1 SerDes Architecture

Figure 1 shows an ITESO TV1 SerDes architectural block diagram, which highlights the main functional blocks, a Serializer and a Deserializer. It also shows the inputs and outputs with a focus on the signals involved in the functionality of the SerDes. On the left the serial interfaces for transmisition and reception are presented. On the right analogus parallel interfaces are presented.

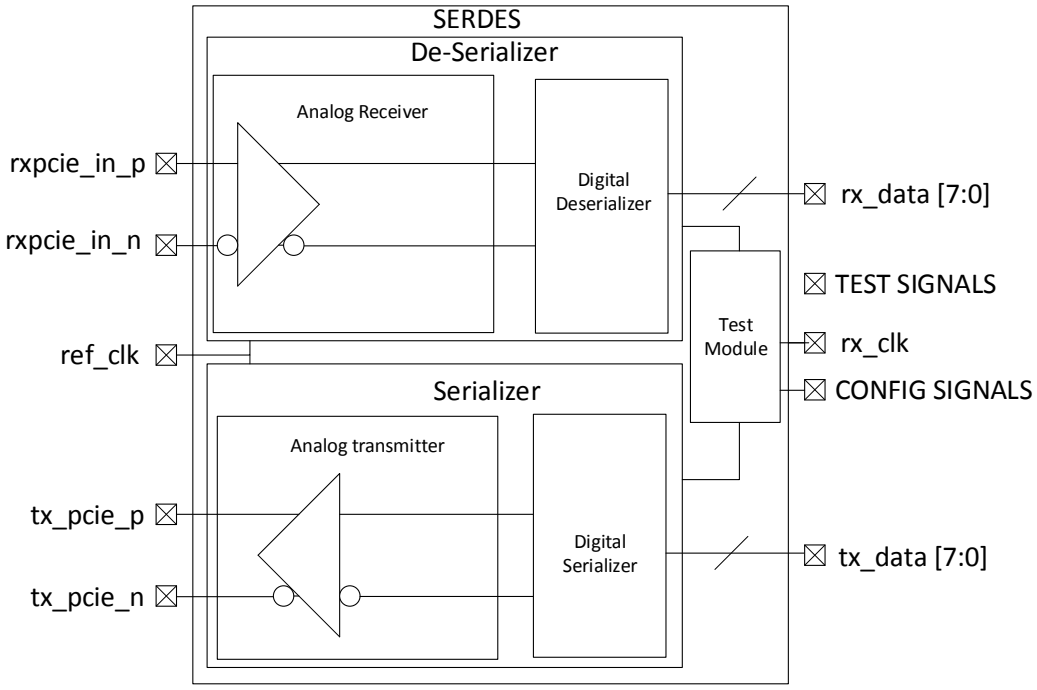


Figure 1. SerDes Architecture, I/Os and main modules.

The left inputs and outputs correspond to the high speed signals on the design. The rxpcie_in_p, rxpcie_in_n, tx_pcie_p and tx_pcie_p are differential pairs for serial communication based on the PCIe Gen1 protocol. The data rates for a serial PCIe protocol can be up to 1 Gbps. The reference clock needed by the design is 8 times the data rate expected on the serial pcie ports.

The analog receiver is in charge of doing signal conditioning. It transforms a differential signal with poor strenght and a lot of noise into a proper CMOS level single ended signal for the logic blocks downstream.

The analog transmitter takes a CMOS signal and converts it into a customizable digital signal with programmable amplitude, pre-emphasis and impedance. This allows the interface to be calibrated to match transmision lines with different characteristics.

The Test Module includes mainly digital components to enable use cases that allow a Built In Self Test. It also lets the designers interact with the different modules of the design in isolation.

The Serializer module transforms a parallel bus of data into a codified serial stream. It is designed by purely digital components at CMOS logic levels. It uses an open source encoder [6] as part of it's microarchitecture.

The Deserializer module converts a codified serial stream into a parallel bus of data. The input is packed every ten bits and decodified in parallel. The decoder used is an open source module [6]. Chapters 2 to 4 of this report present the development and integration of the Deserializer module in the ITESO TV1 SerDes system.

C. 8b/10b Encoding

The 8 bit to10 bit encoding transforms 8 bit payloads to 10 bit symbols with a balanced set of bits 0 and bits 1 in the long run. Each payload is coupled with one extra bit to allow for special characters used for control in a communication protocol.The encoded 10 bit symbol contains either three, five or seven bits 0. The number of bits 0 alternates between 7 and 3 to keep an average close to an even split of bits 0 and bits 1. To maintain the balance, some 8 bit payloads may be encoded by up to 4 diferent symbols.

Inside the 8b/10b encoder there are two different encoders working together. The 8 bit payload is divided into a subset of 5 bits and a subset of 3bits. Each subset is encoded using a different look up table, although both follow similar rules. Each encoding adds 1 bit to the input to create a 6 bit and 4 bit subset of the symbol.

The five least significant digits of the payload are mapped using a 5b/6b encoder, producing digits 8,4,3,2,1,0 in the encoded symbol.

The three most significant digits of the payload are mapped using a 3b/4b encoder, producing digits 9,7,6,5 in the encoded symbol.

Each payload value can be mapped to different symbols depending on what symbols have been used in previous cycles. The chosen output symbol depends on an internal bit registered in the encoder called "running disparity". The running disparity may alternate between -1 or +1 depending on the symbol used in the previous cycle. The disparity is a one bit register that helps the encoder keep the balance of total number of bits 0 and bits 1 used over time.

D. 5b/6b Encoding

Table 1 provides some examples of how 5 bits of a payload would be mapped into 6 bits of a symbol.

Name	payload [4:0]	payload [8]	symbol- [5:0]	symbol+ [5:0]
D.0	00000	0	111001	000110
D.1	00001	0	101110	010001
D.2	00010	0	101101	010010
D.3	00011	0	100011	100011
D.4	00100	0	101011	010100
D.6	00110	0	100110	100110
D.8	01000	0	011000	100111
D.10	01010	0	101010	101010
D.12	01100	0	101100	101100
D.14	01110	0	001110	001110
D.16	10000	0	110110	001001
D.31	11111	0	110101	001010

Table 1. Examples of 5b/6b encoding pairs [5]

E. 3b/4b Encoding

Table 2 provides some examples of how 3 bits of a payload would be mapped into 4 bits of a symbol.

Name	payload [5:7]	payload [8]	symbol - [9:6]	symbol + [9:6]
D.0	000	0	1101	0010
D.1	001	0	1001	1001
D.2	010	0	1010	1010
D.3	011	0	0011	1100
D.4	100	0	1011	0100
D.5	101	0	0101	0101
D.6	110	0	0110	0110
D.7	111	0	1110	0001

Table 2. 3b/4b encoding pairs [5]

F. 8b/10b Special Characters

Table 3 contains the full list of special characters used as commands on the original ***IBM*** 8b/10b encoder [5].

Name	**payload [4:0]**	**payload [7:5]**	**payload [8]**	**symbol - [5:0]**	**symbol - [9:6]**	**symbol + [5:0]**	**symbol + [9:6]**
K28.0	11100	000	1	111100	0010	000011	1101
K28.1	11100	001	1	111100	1001	000011	0110
K28.2	11100	010	1	111100	1010	000011	0101
K28.3	11100	011	1	111100	1100	000011	0011
K28.4	11100	100	1	111100	0100	000011	1011
K28.5	11100	101	1	111100	0101	000011	1010
K28.6	11100	110	1	111100	0110	000011	1001
K28.7	11100	111	1	111100	0001	000011	1110
K23.7	10111	111	1	010111	1000	000011	1110
K27.7	11011	111	1	011011	1000	000011	1110
K29.7	11101	111	1	011101	1000	000011	1110
K30.7	11110	111	1	011110	1000	000011	1110

Table 3. 8b/10b special characters [5]

G. Reference Clock

The Deserializer takes as an input a stream of data. Such stream is asynchronous to any other port or interface of the SerDes. There is no clock for a pure PCIe serial lane. The system has to find a way to keep the clock used to recover the data synchronous to the rxpcie_in interface described with the SerDes microarchitecture. The most common way to synchronize a system clock to an input is by using a module that works with a “Phase Locked Loop” (PLL). A PLL is a common module in communication blocks, but it is also one of the most complex analog blocks in a system on chip. For the ITESO TV1 a PLL was considered out of scope. Instead of a PLL, a reference clock should be provided to the system. The clock should be aligned with the input data. The clock is assumed to have a frequency 8 times higher than the data rate of the serial link.

With a reference clock 8 times faster than the serial data stream, a Clock and Data Recovery (CDR) technique from previous research efforts at ITESO [3] is used for the ITESO TV1. In essence the technique samples each data bit 8 times. The oversampling allows the system to decide if the input is a 0 or 1 based on how many times each level was sampled.

More details on the CDR used for the ITESO TV1 are provided in Chapter 2. From the system design perspective the CDR used requires a clock divider that produces 8 clocks with a frequency equal to one eight of the reference clock. All 8 clocks have the same frequency but different phase. The various phases will allow the oversampling of data to take place at 8 instants equally distributed over the period of one data bit.

H. ITESO TV1 SerDes Microarchitecture

Considering the unique aspects of the CDR a microarchitecture block diagram is presented in Figure 2 for the ITESO TV1 SerDes system.

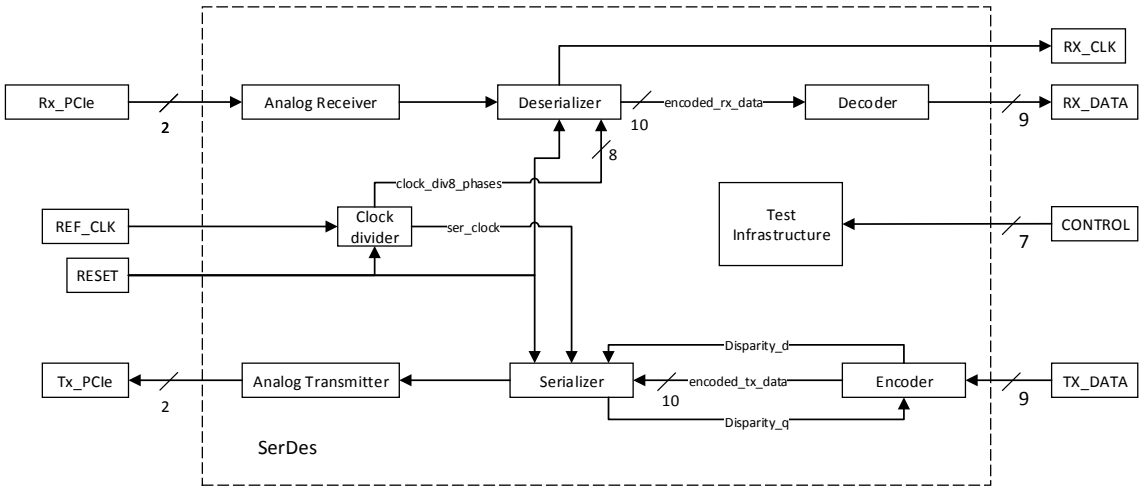


Figure 2. ITESO TV1 SerDes Microarchitecture

The microarchitecture presented in Figure 2 does not put emphasis on the testing infrastructure to allow a clear diagram of the main functional mode intended for the SerDes. In the following chapters the focus will be on the Deserializer and clock divider modules. The details on the other modules are out of the scope of this report, only the system integration is described in detail on Chapter 4.

CHAPTER 2: GATE LEVEL DESIGN OF DESERIALIZER MODULE

The Deserializer module is designed at a Register Transfer Level (RTL) of abstraction. This means the functionality is crafted by describing synchronous registers and the logic between them. At this level of abstraction main characteristics of the input to the Deserializer are the following:

- One bit.
- Asynchronous to the reference clock.
- Serial communication.
- 8b/10b encoded.
- Clock embedded in same pin.
- Up to 1 Gbps

The clock on the system has the following characteristics

- Frequency equal to 8 times the data rate of the data input.
- 50% duty cycle
- In phase with data input.
- Up to 8 GHz

On the other side, the outputs from the Deserializer main attributes are:

- 10 bits.
- Synchronous.
- Parallel data bus.
- 8b/10b encoded.
- A clock needs to be provided with the output data.
- Up to 125 MHz

A. Deserializer Microarchitecture

The Deserializer uses a clock data recovery technique based on 8 clocks at equidistant phases [3] [2]. Then a shift register is the core of the Deserializer functionality. Finally synchronization, control and output registers control the flow of data at the outputs of the Deserializer. Figure 3 shows a block diagram detailing the interactions of the various elements inside de Deserializer design.

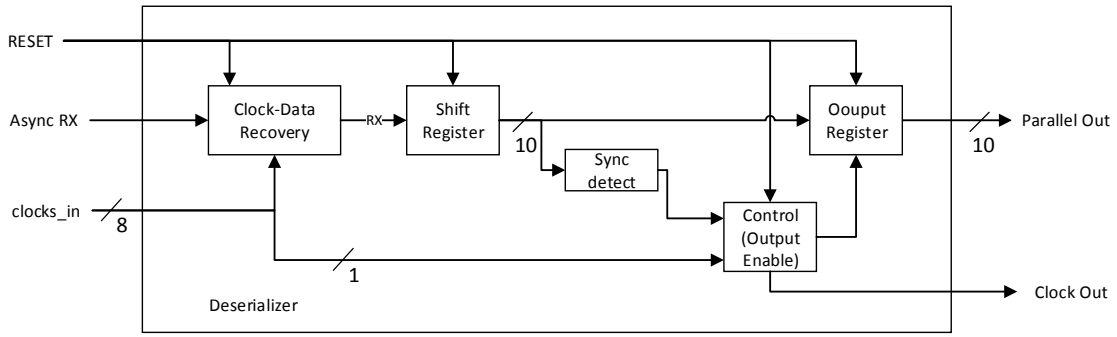


Figure 3. Deserializer block diagram

An array of flip flops clocked at different phases is used to effectively oversample the input at 8 different points in time. With the eight samples of the input data, a decision can be made to choose whether the bit being read is a 0 or a 1. The synchronous input is a 0 or 1 based on what the majority of the sampling flip flops capture.

Once data is synchronized by oversampling, it is passed to a shift register to group the data. In the shift register the information is transformed from a serial communication to a parallel bus.

One of the special characters from the 8b/10b encoding is used for synchronization. The comma character [5] provides the means to identify the start and end of a 10 bit frame at the serial link. Therefore the Deserializer needs to be aware and be capable of synchronizing the internal counters to the comma sequence detection.

The comma synchronization and the clock are combined to control when an output is ready. A 10 cycle counter is used to create a reference clock for the output. The clock output can be used as a reference to sample the parallel output bus in a synchronous fashion. The 10 cycle counter can be restarted by the synchronizing detection.

Finally the outputs are registered before going to the next stage. This helps to avoid large clouds of logic by isolating the internal logic in the serializer from the logic downstream. Avoiding large clouds of logic make the Deserializer IP easier to reuse in applications where the timing is restrictive.

B. Clock Infrastructure

The Deserializer microarchitecture presented in Figure 3 requires a synthesizable clock divider. The purpose for the clock divider is to generate 8 clocks at the same frequency, but different equidistant phases. Each clock should be one eight of a full

period apart from each other. The clocks are all generated based on the system reference clock. This phase difference between each of the eight clocks is one period of the reference clock, which implies the clocks used in the Deserializer will be one eight of the reference clock.

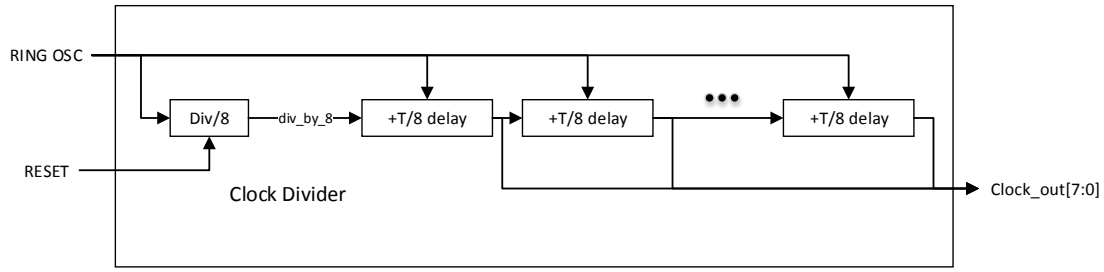


Figure 4. Clock divider with delays for outputs at equidistant phases

Figure 4 shows the functional block diagram of how to generate the 8 clocks required by the Deserializer. This module is implemented by combining a clock divider with 8 registers. Each register corresponds to a T/8 delay on Figure 4.

The clock divider needs to be very simple, since it will be clocked by a very fast clock in the system. The proposed microarchitecture for the clock divider uses only a single inverter and a group of 4 registers connected in series. Figure 5 shows a structural representation of the clock divider topology used in the ITESO TV1.

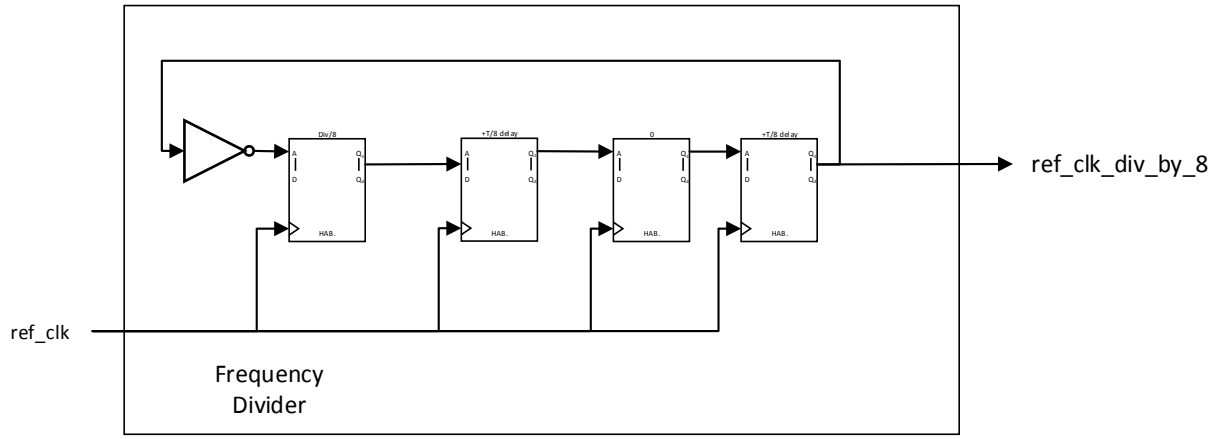


Figure 5. Clock divider topology for ITESO TV1.

It is worth mentioning that the whole clock divider and delays will add some level of complexity to the clocking constraints. This topology requires constraints that can define derived clocks and the phase shift relationship between them. The final constraints used for SerDes synthesis in this chapter are presented in appendix G.

C. Logic Synthesis

The first step in VLSI physical design is the logic synthesis of the validated RTL. Chapter 3 presents the results of the behavioral simulation of the RTL design, by presenting waveforms to verify the functionality of the design.

The standard cell library in the ***MOSIS cmrf7sf*** digital design kit was used for the gate level synthesis. Timing constraints were developed to guide the logic implementation. The timing constraints model the clock relationships in the clock divider module and set the requirements for inputs and outputs of the digital modules. A full list of timing constraints presented in appendix G. Close inspection of the timing constraints reveals a much slower reference clock than the one needed to meet proper PCIe protocol timing. A reference clock of 8GHz would be required to support up to 1 Gbps data rates. Instead a 250 MHz reference clock is used. From the initial gate level timing reports it is clear that oversampling the data input 8 times makes timing very difficult in the 180 nm technology. A relaxed set of constraints is used for physical synthesis exploration, timing analysis and optimization is out of the scope for this report. Figure 6 shows the ingredients involved in the logic synthesis.

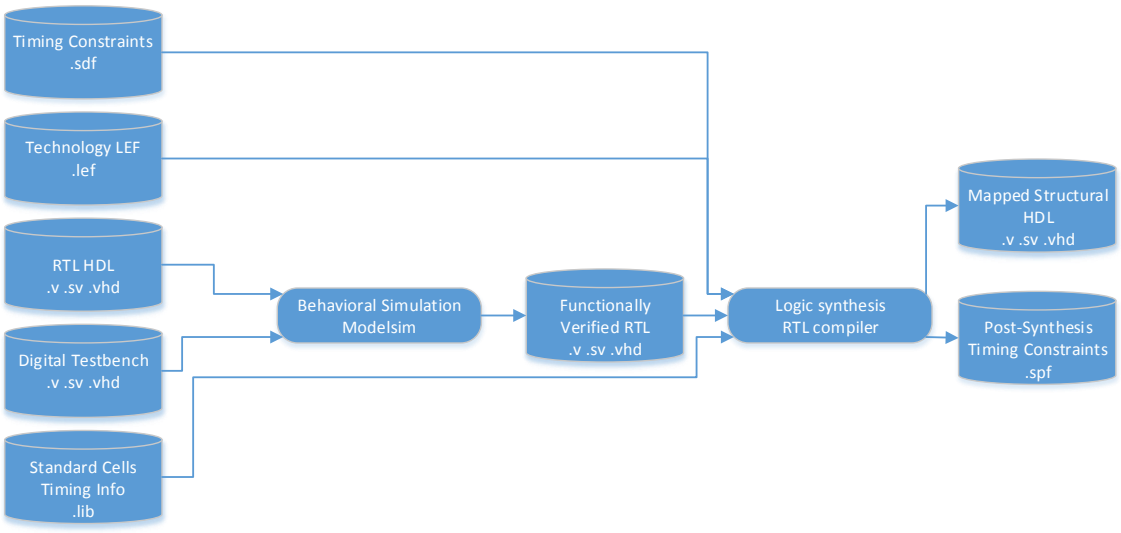


Figure 6. Logic synthesis of SerDes design

The ***RTL Compiler*** is controlled by TCL commands. TCL is common scripting language used to interact with Electronic Design Automation tools such as ***RTL Compiler*** or ***Encounter Digital Implementation System*** (EDI). Each command performs a step of digital synthesis to elaborate using ingredients from Figure 6 to

generate the desired netlists, verification scripts and unwrapped timing constraints. The full set of commands to perform logic synthesis with timing constraints is included in appendix H. Once the TCL script is developed launching ***RTL Compiler*** is simple as shown in Figure 7.

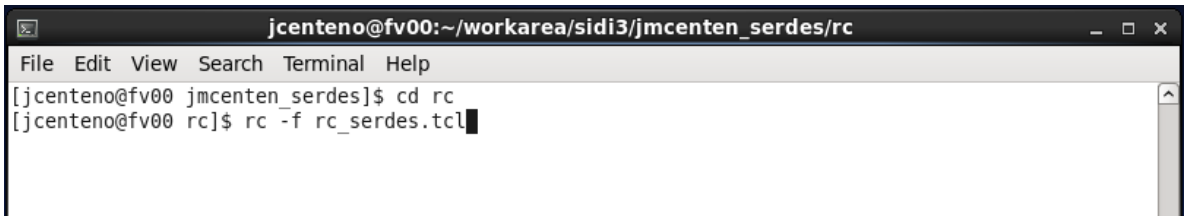


Figure 7. RTL compiler launched from design root.

At the end, in the RC prompt, the command *"gui_show"* displays a schematic representation of the structural hardware description created by ***RTL Compiler***. The schematic is shown in Figure 8.

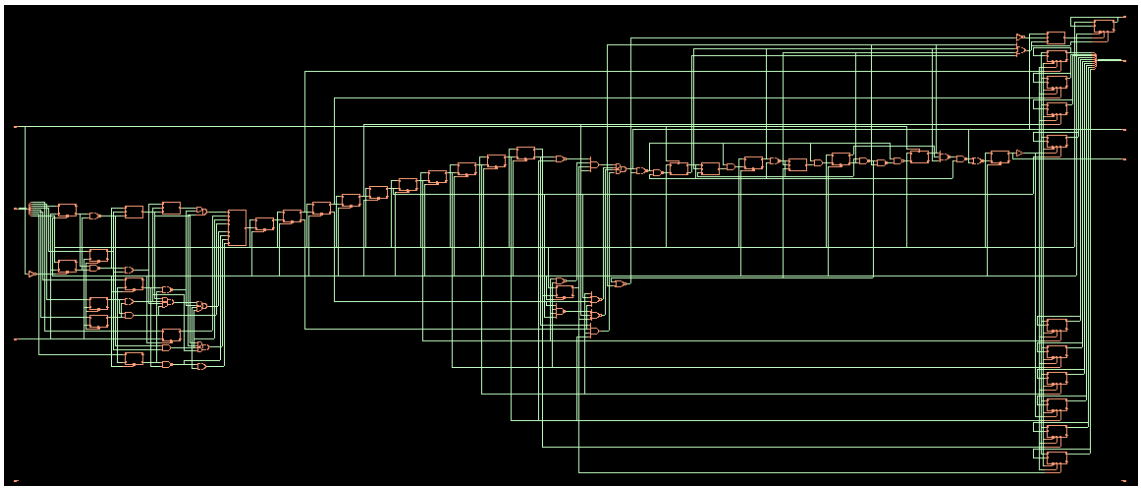


Figure 8. Deserializer module. Schematic view of gate level synthesis

There are two ports in the gate level implementation of the Deserializer which have no logic associated to them. They are disparity_d and disparity_q. Based on the RTL description this signals should be the inputs and outputs to a D type flip flop. A zoom into the unconnected ports is presented in Figure 9.

Figure 9. Deserializer disparity ports unconnected in the schematic view

Tracing through the design, we observe how the SerDes system ignores the error flags provided by the Decode module. The disparity_q output from the Decode module is only relevant for the error flags; it is not relevant for the recovered payload bits. Hence,

the flip flop is optimized out because the disparity_q does not affect any of the relevant decode module outputs connected in the SerDes system. This bit will only become relevant if the Deserializer module implements a new feature to handle error flags in the Decode module. At this stage of the SerDes design *disparity_d* and *disparity_q* stay untested in the system simulations and disconnected in the gate level netlist.

The gate level synthesis and constraints were implemented in the context of the full SerDes system, not just the Deserializer module. The full SerDes schematic view is presented in Figure 10.

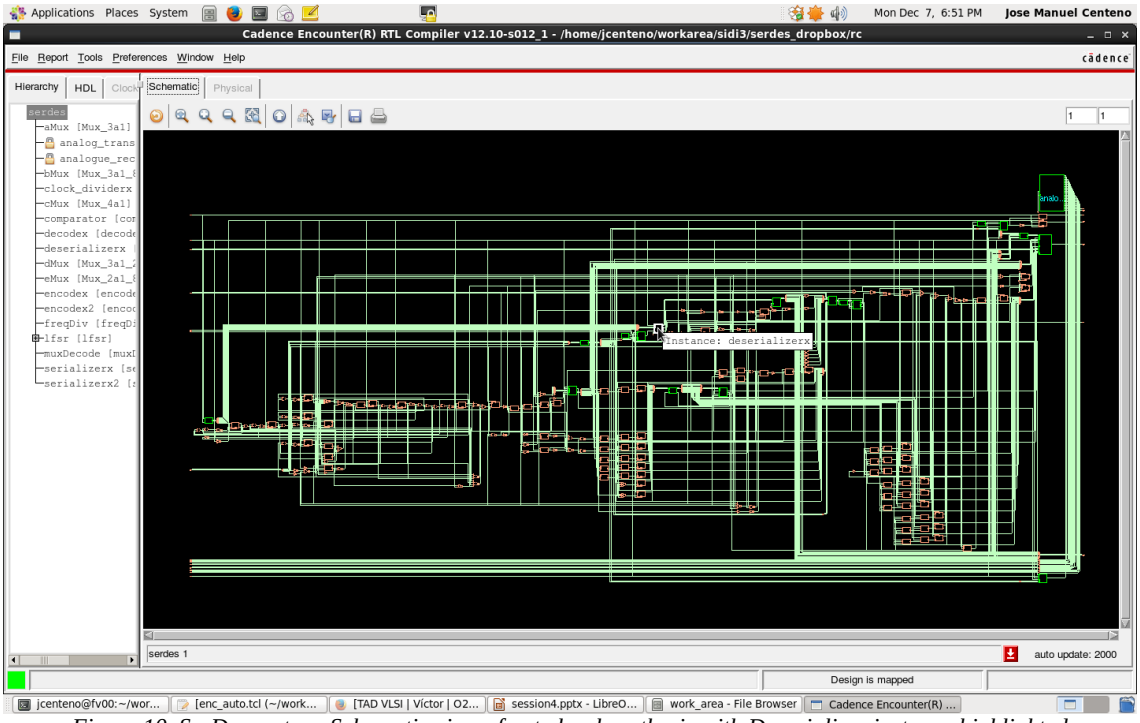


Figure 10. SerDes system. Schematic view of gate level synthesis with Deserializer instance highlighted

It is important to highlight that Figure 10 shows a schematic which includes black boxes for analog cells. These black boxes represent cells that will be developed using analog or custom digital techniques. Black boxes are not included in the HDL list for ***RTL Compiler***, and are implemented as their own instance with no optimization allowed inside of them. For digital simulations all black boxes are modeled. For analog receiver and transmitters in the SerDes design a simple wire and mux describe the functionality from a pure digital stand point. The black box models used for simulations are included in appendices I and J.

CHAPTER 3: TEST AND VERIFICATION OF DESERIALIZER MODULE

The RTL developed for the Deserializer was validated both in isolation, and as part of the full SerDes system. Figures 11 to 18 show waveforms that demonstrate the correct behavior of the design.

A. Clock divider

Figure 11 shows the inputs and outputs to the clock divider, were we can look at the equidistant phases for all output clocks, running at the same frequency.

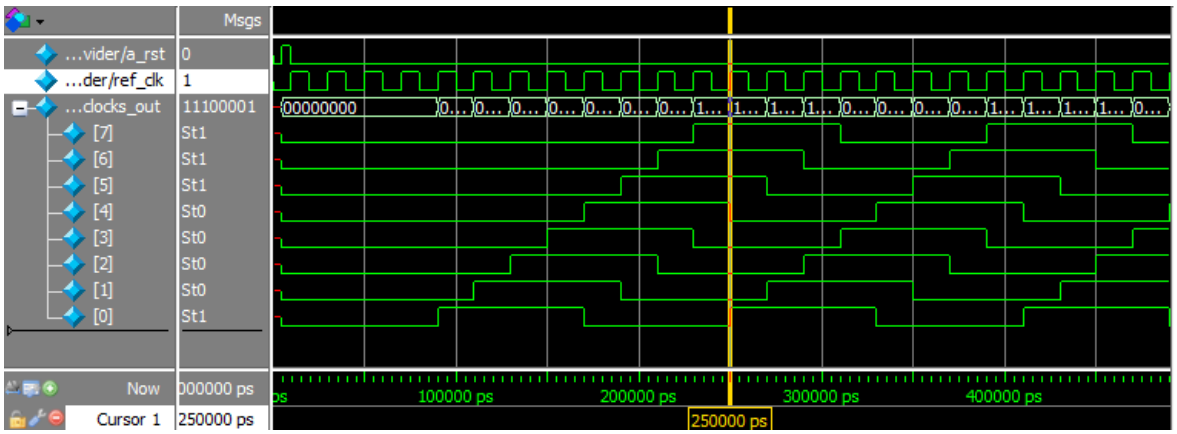


Figure 11. RTL simulation of Clock divider inputs and outputs

For the RTL simulation in Figure 11 the exact frequencies are not relevant. On the gate level simulation shown in Figure 22 a reference clock at 250 MHz is used. Therefore clocks_out[7:0] have a frequency of 31.25 MHz (32 ns period) with a 45° separation between each element (4ns).

The reference clock is divided by eight and delayed 8 times to produce the desired outputs. The clock division is obtained by connecting several flip-flops type D in series, with an inverter feedback connection between the first and last flip-flop. The delay is implemented by using a series of flip flops type D, using the reference clock to sample the D input.

B. Deserializer

The Deserializer is the largest part of the design. It implements several features as presented in Chapter 2. Figure 12 shows a waveform with the inputs and outputs to the Deserializer.

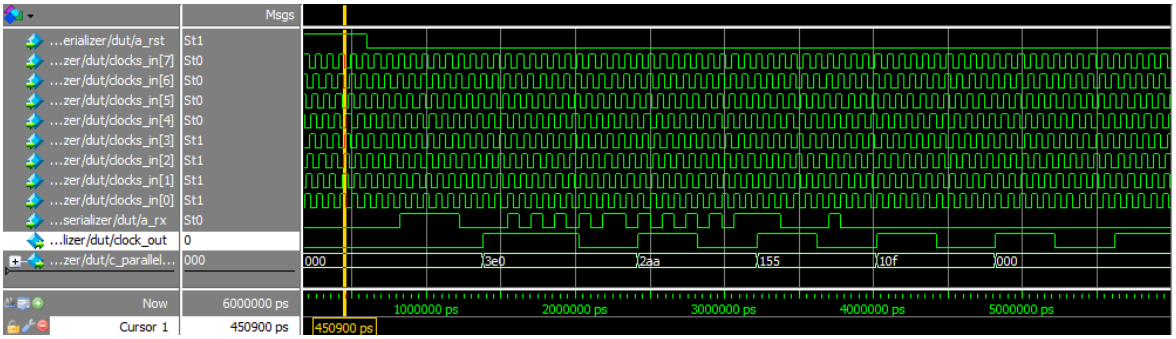


Figure 12. RTL simulation of Deserializer inputs and outputs

The clock inputs in the Deserializer sample the asynchronous serial input called “a_rx”. The “a” prefix in “a_rx” signal name indicates the asynchronous nature of the signal with respect to the clock inputs. The output of the Deserializer “c_parallel_out” is a parallel bus, which corresponds to the last 10 values recovered from “a_rx”. The 10 bits are decoded outside of the Deserializer module, inside the Decode module as presented on the SerDes microarchitecture in Figure 2.

To help understand the first stages in the data path, Figure 13 shows the upsampling and “sampled bit count” which decides if the input should be interpreted as a 0 or 1:

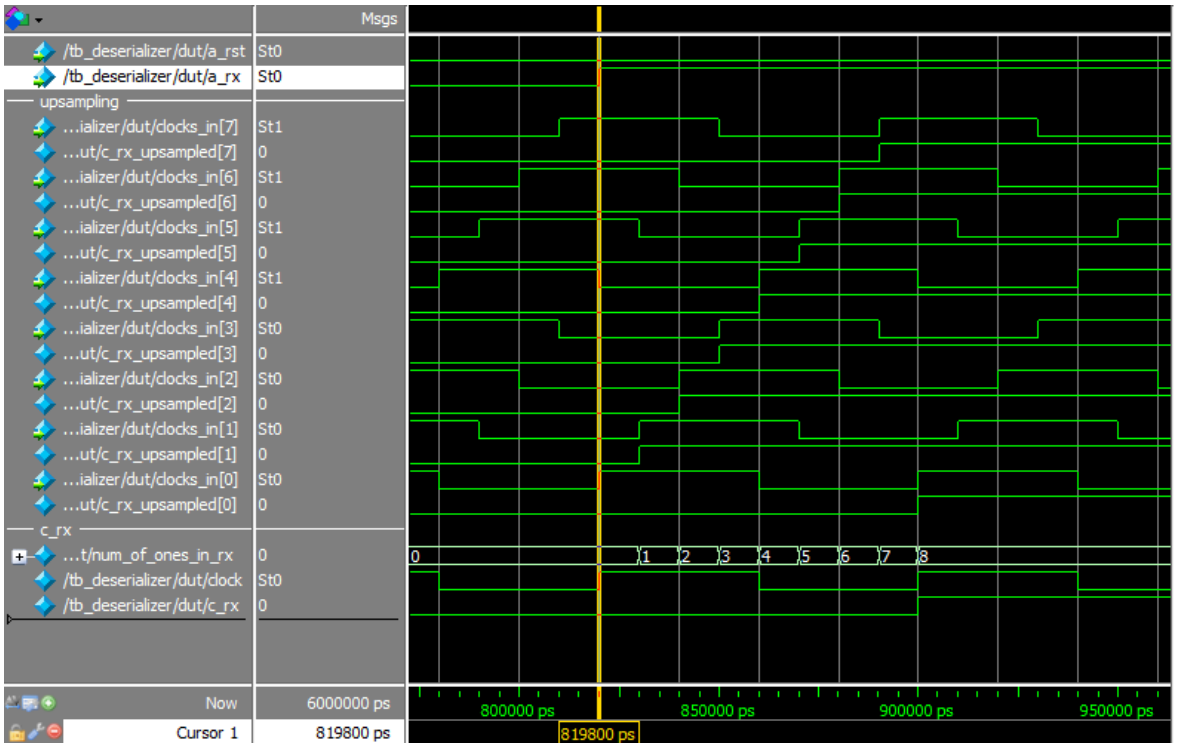


Figure 13. RTL simulation. Signals involved in up-sampling data recovery

In this example we see how the bit count increases as more rising edges occur in the input clocks, sampling the asynchronous input to a known clock domain.

Figure 14 presents a waveform of the detection of the special symbol for K28.7, also known as the “comma special character”. Figure 14 shows the characteristic stream of five bits 1 in a_rx corresponding o the comma character, which is used to synchronize the input stream. The encoded “0001111100” stream in “a_rx” is an unambiguous series. No other combination of encoded characters would produce this sequence.

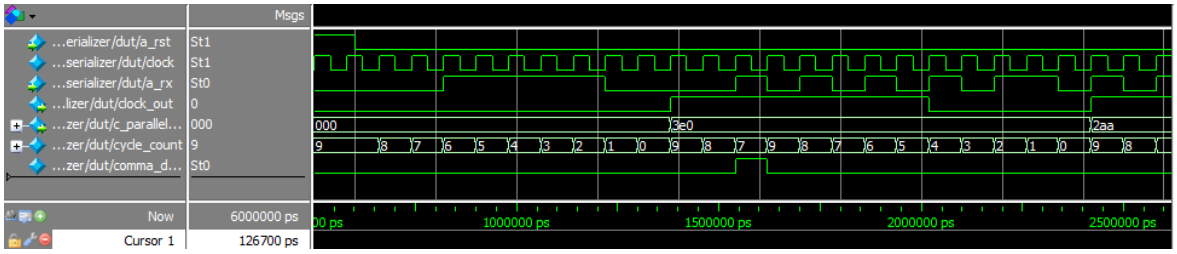


Figure 14. RTL simulation. 8b/10b special character “comma” detection

The key moment happens when the signal comma_detected asserts at count 7. In this cycle, the comma is identified. The control logic interprets the comma_detected assertion and restarts a 10 cycle counter to resynchronize the Deserializer internal count with the a_rx stream. The next clock_out edge will happen10 cycles after the comma character was detected. From that moment on, the parallel output produces

sets of 10 bits correctly aligned, based on the 8b/10b interpretation of the comma special character.

C. SerDes integration

A serial loopback was used as a first approach to system validation.

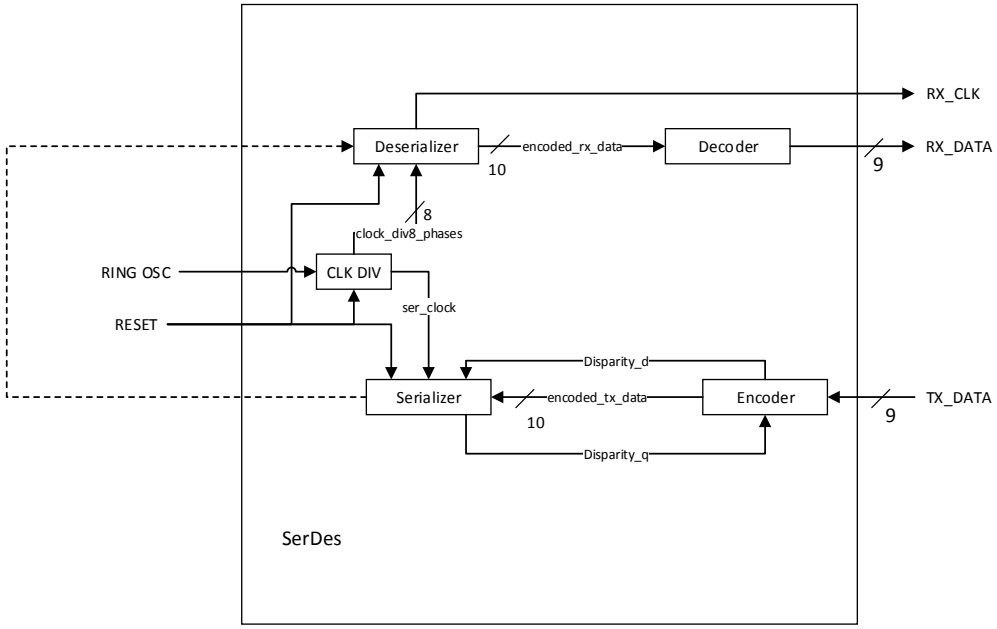


Figure 15. SerDes connected in loop-back mode for system validation

Figure 16 shows the sequence of bytes transmitted using the serializer and recovered using the Deserializer. Both elements include the 8b/10b encoding and decoding blocks, which lets the stream be a DC balanced signal.

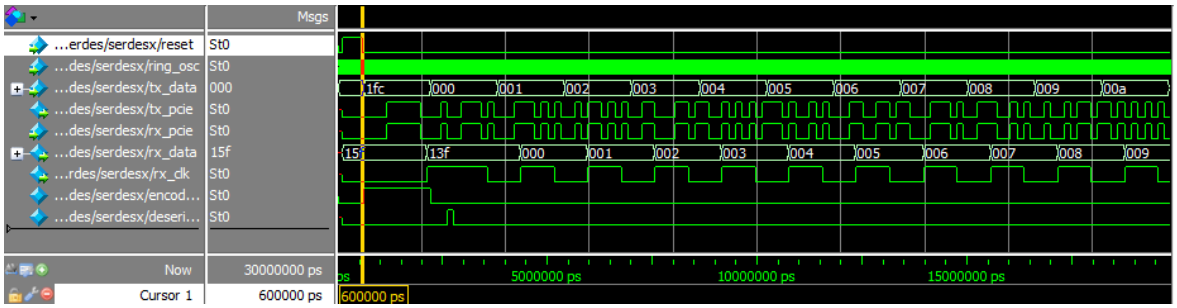


Figure 16. RTL simulation. SerDes loop-back comma detection and data recovery

We can see the Deserializer recovering the data with a latency of about one output clock cycle. The latency is the result of sum of latencies in the serializer and the Deserializer. Note how the first character sent by the Serializer, 0x1FC, is a comma special character. This character is not recovered, but is used to synchronize all the output bytes that follow.

Figure 17 shows the details on the comma transmission and detection:

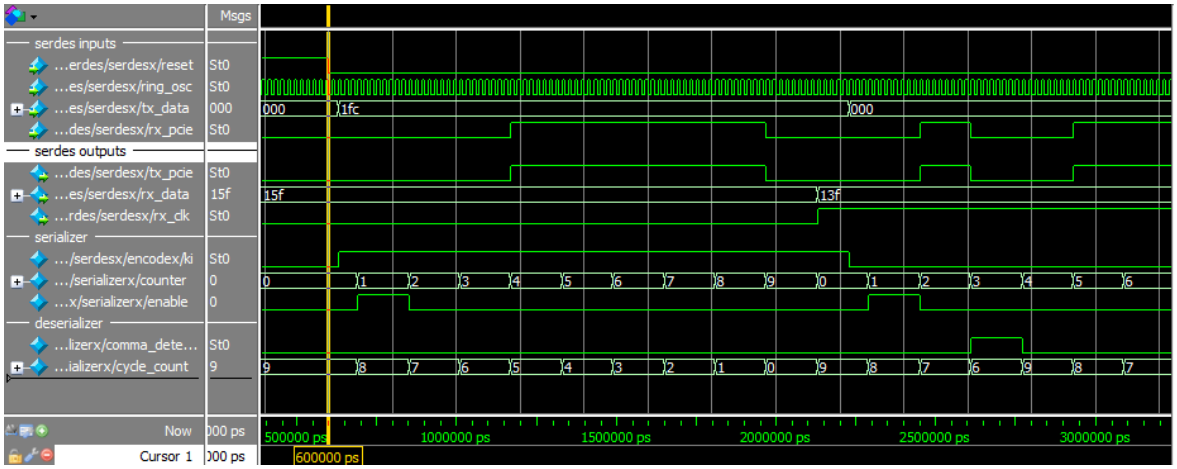


Figure 17. RTL simulation. SerDes loop-back detailed comma detection

The serializer encodes the characters in a special way. Note how the ninth bit in the input to the SerDes is a bit 1. This signal indicates to the encoder the special character table should be used for this symbol. The stream is very straightforward, basically a series of zeros, followed by five ones and finishing with zeros.

Finally we show the transmission of input signals that exercise more input and output bits in Figure 18.

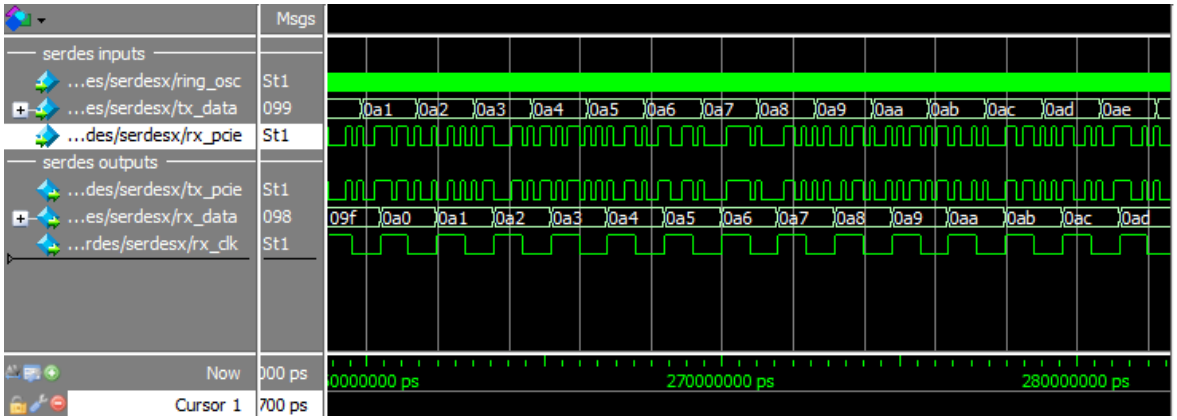


Figure 18. RTL simulation. SerDes loop-back data recovery

Note all characters are regular sequences, as the ninth bit input to the serializer is a zero. Again, the Deserializer continues to be in sync, and recovers the transmission after a few reference clock cycles.

D. Logic Equivalence Check for SerDes

A common modern technique used to verify synthesis is the logic equivalence check. This technique can be applied effortlessly by combining the RC tool used for gate level netlist and the ***Encounter Conformal Logic Equivalence Checker*** (LEC). ***RTL***

Compiler creates a .do file that can be consumed by the LEC. The LEC is able to compare various stages of the synthesis.

Figure 19 shows the results for a comparison between the RTL and the intermediate mapped, but not optimized netlist.

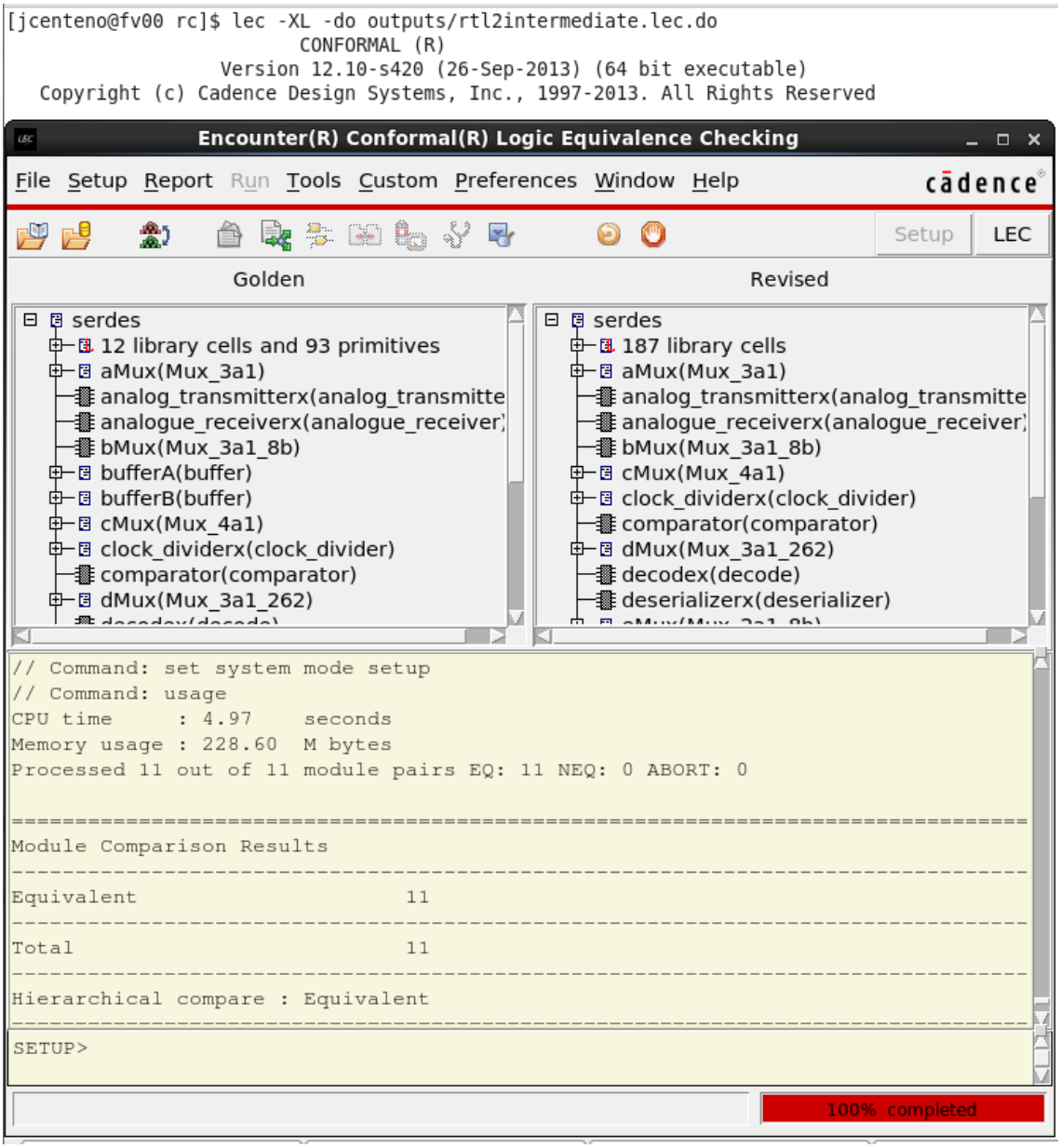


Figure 19. RTL to intermediate netlist logic equivalence check.

Figure 20 shows a complimentary logic equivalence check. The Intermediate to final comparison shows the optimized netlist is logically equivalent. Both netlist, intermediate and final, are gate level netlists based on logic elements mapped to the fabrication technology.

```
[jcenteno@fv00 rc]$ lec -XL -do outputs/intermediate2final.lec.do
                        CONFORMAL (R)
                Version 12.10-s420 (26-Sep-2013) (64 bit executable)
   Copyright (c) Cadence Design Systems, Inc., 1997-2013. All Rights Reserved
```

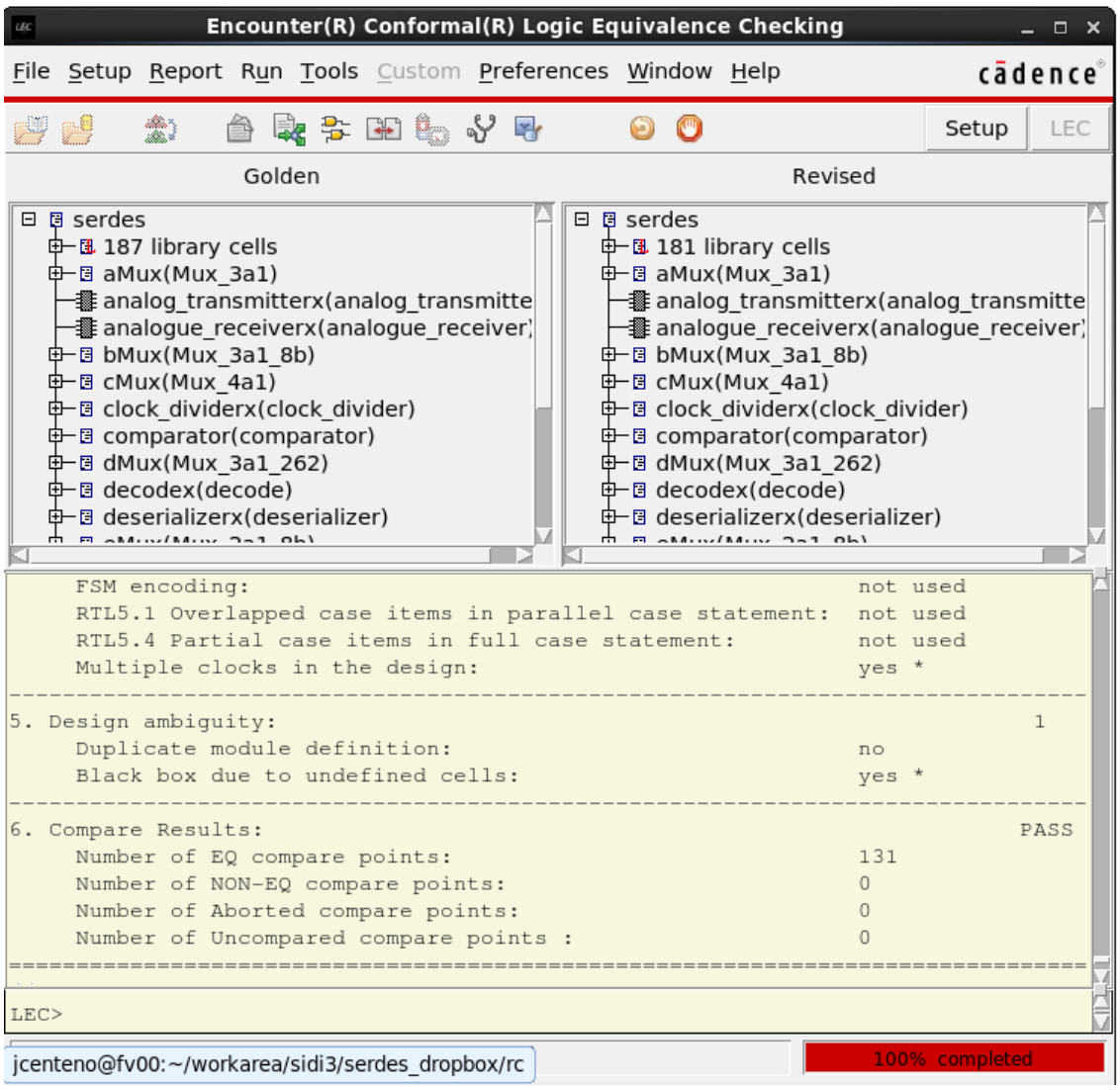


Figure 20. Intermediate netlist to final gate level netlist logic equivalence check.

Figure 21 shows the results of comparing the RTL directly to the final gate level netlist. As expected from the previous two checks, the logic equivalence check between RTL and final gate level netlist yields an “equivalent” verdict.

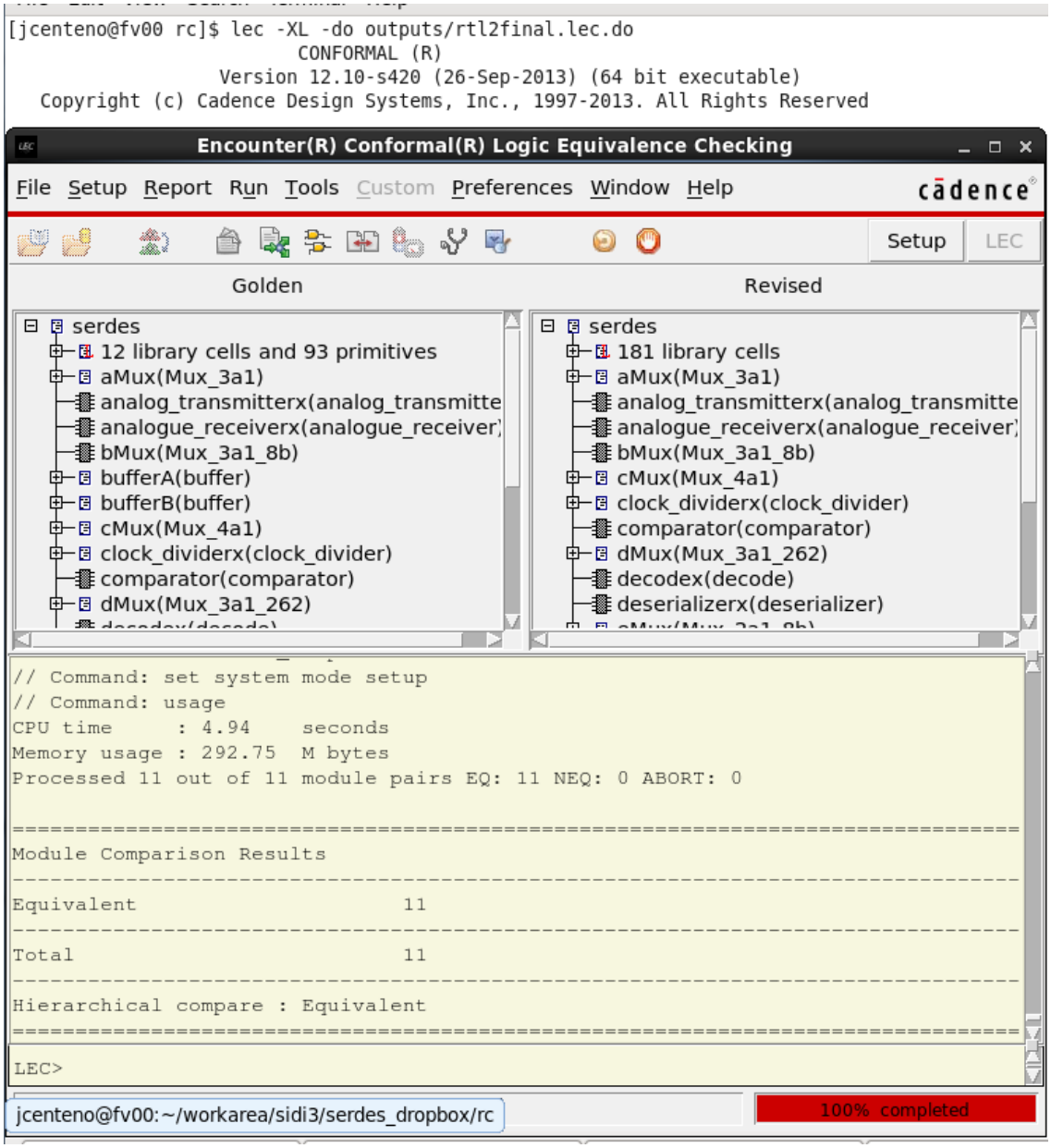


Figure 21 RTL to final gate level netlist logic equivalence check.

E. Gate Level Simulation

After the SerDes RTL was synthesized using RC Compiler, the produced netlist was simulated. The simulation reuses the same testbenches developed during the RTL design phase. To model the behavior of the logic elements in the netlist, a verilog description from the ***IBM*** design Kit was used. The ***IBM*** digital kit includes timing aware models for all gates potentially used in the mapped netlist. Figures 22 to 27 present the results from using the timing aware models of the ***IBM*** digital design kit.

Figure 22 shows the waveform of the clock divider module. We can observe the 8 clocks produced at equidistant phases with one reference clock period of difference.

The outputs are driven directly by flip-flops, which is why we don't observe any transition glitches on any of the clock_out bits.

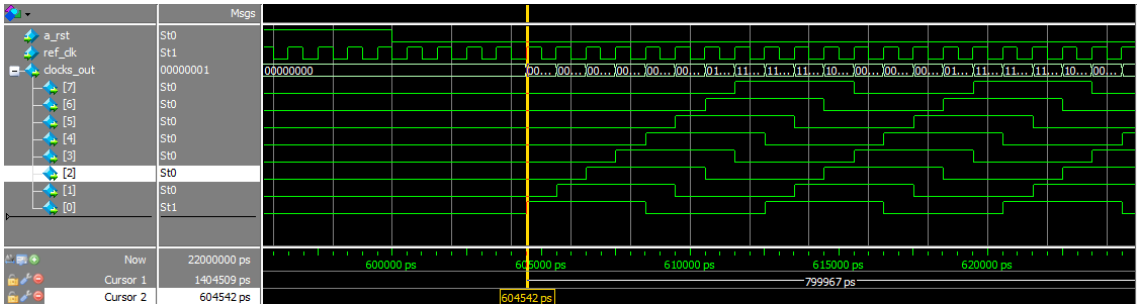


Figure 22. Gate level simulation of Clock divider inputs and outputs

Figure 23 shows a waveform of the Deserializer module inputs and outputs. We can observe clean outputs and the non-periodic clock_out generated as reference for the parallel bus. The reference clock named "clock_out" in figure 23 runs at one eightieth of the frequency of the ref_clk. For the initial gate level synthesis and simulations a reference clock of 1 GHz was used. Therefore "clock_out" runs at 12.5 MHz.

In Deserializer module the outputs are also driven directly by flip-flops which mask any transition values due to uneven logic delays in the internal blocks. The flip flops produce a time delay in Figures 22 and 23 of 10 ps between "ref_clk" and "clocks_out" which not observed on the RTL simulations. 10 ns is negligible compared to the 1000 ps period of the reference clock.

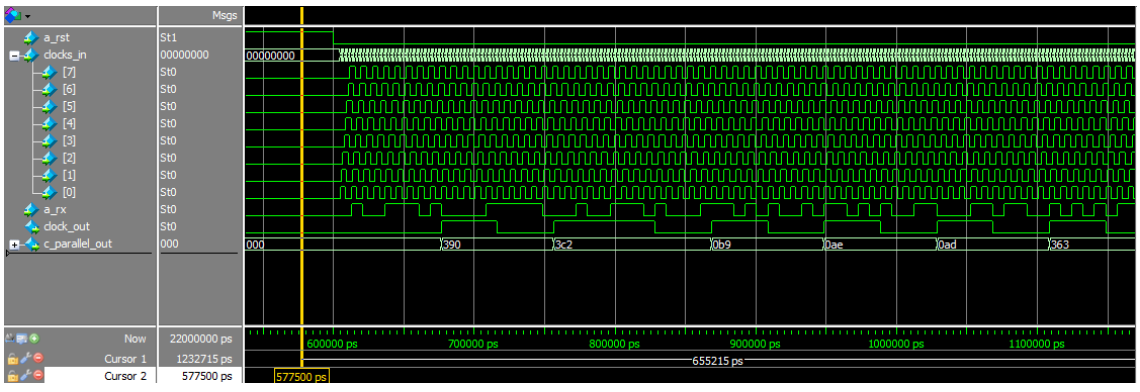


Figure 23. Gate level simulation of Deserializer inputs and outputs

Figure 24 shows a zoom into the comma detection signals. We observe how the cycle_count restarts to 9 when the comma symbol is detected. This synchronizes the frame packaging to match the PCIe standard in this regard. One difference between the gate level simulation and the RTL behavioral simulation can be observed on the comma_detected signal. Close to the 740 ns mark, a small unintended transition can be observed. Fortunately this transition does not cause any setup or hold issue, as we

can see the counter does not restart on that small glitch. In fact, setup and hold are not validated at any flip flop during any of the gate level testbenches exercised. The Verilog flip flop models provided by the ***IBM*** digital design kit include assertions for timing violations. This is an interesting feature that increases the level of confidence we have on the timing constraints provided during the logic synthesis.

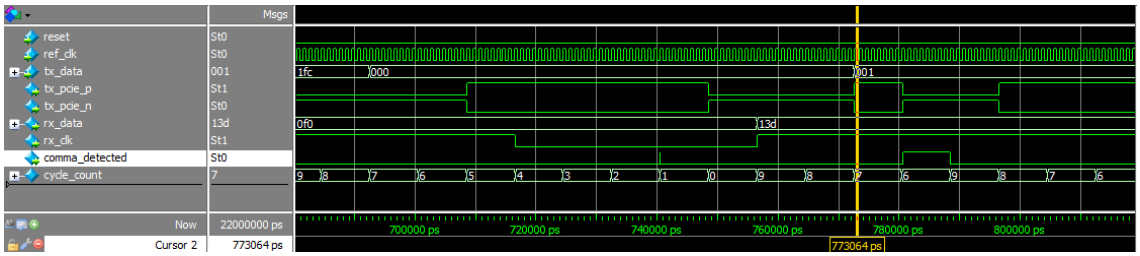


Figure 24. Gate level simulation. 8b/10b special character "comma" detection

Considering the design is not only made of registers exclusively, a purely combinatorial circuit portion of the design is highlighted in Figure 25. The open source decoder does not contain any registers, and we can observe multiple transitioning values. The transition portion is relatively small, but it is presented as sanity check to verify we are really simulating a gate level design with logic delays.

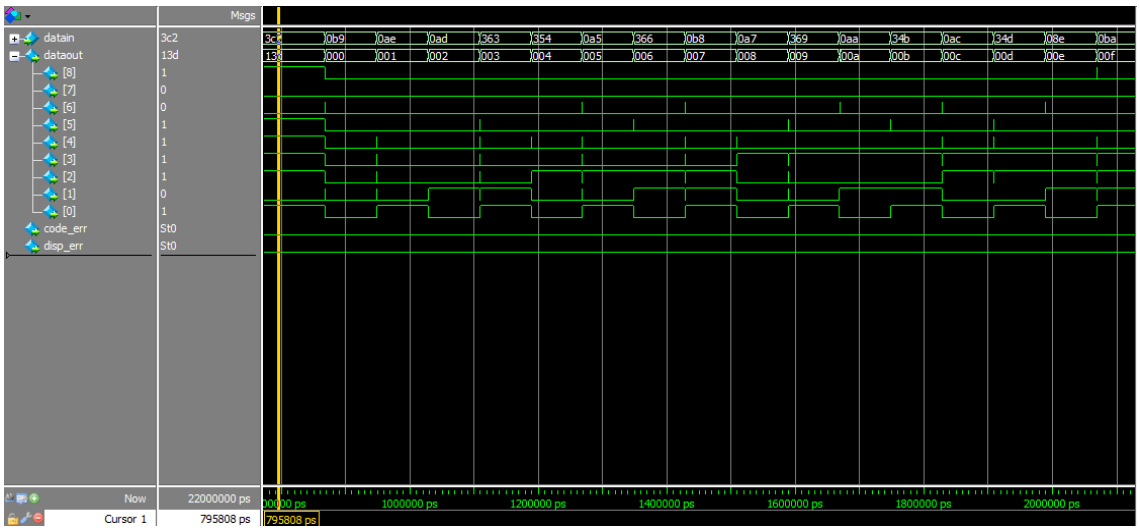


Figure 25. Gate level simulation. Open source decoder.

Figure 26 presents the system level testbench waveforms used to validate the SerDes design in a loop-back configuration. As expected the Deserializer detects the comma character in rx_pcie_in data and starts decoding the frames at the right time. We can observe in rx_data the same data sent a few cycles earlier on tx_data, which is how this test is supposed to work.

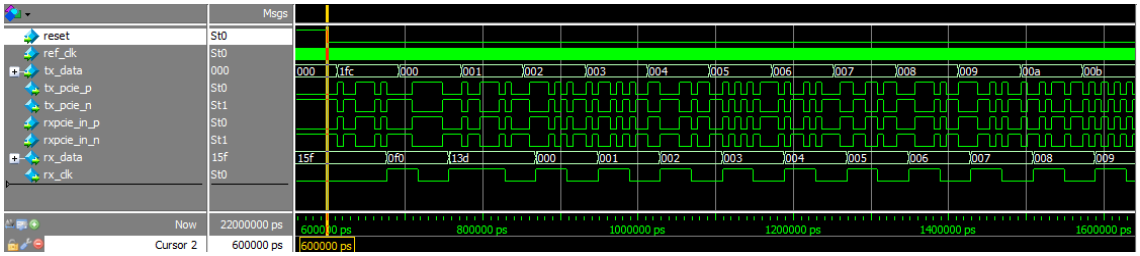


Figure 26. Gate level simulation. SerDes loop-back data recovery after comma detection

Figure 28 shows the same loopback testbench results as Figure 27 centered at a more advanced point in time where higher bits of tx_data are exercised. The delay between rx_data and tx_data stays the same since both the serializer and Deserializer use a common system clock produced by the clock divider circuit.

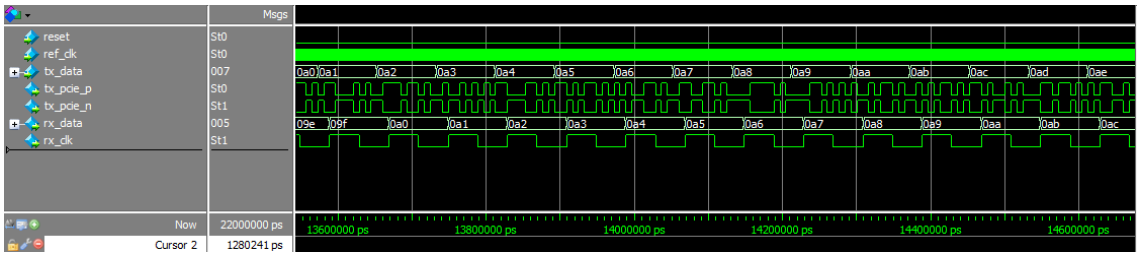


Figure 27. Gate level simulation. SerDes loop-back data recovery

CHAPTER 4: PHYSICAL DESIGN OF SERDES

The VLSI design technique followed by the ITESO TV1 has a flow of tools that enable the physical synthesis and integration of the full SerDes system [7]. These tools require a set of ingredients to aid design, perform validation, verification or some level of synthesis. The tools let us go from RTL and subsystem analog layouts to a data base for fabrication of the integrated circuit. Before zooming into different tools and flows, a high level summary of the full VLSI synthesis flow is presented in Figure 28. The main goal is to identify all the ingredients and tools involved, before analyzing in detail which parts of the flow have been explored for the ITESO TV1.

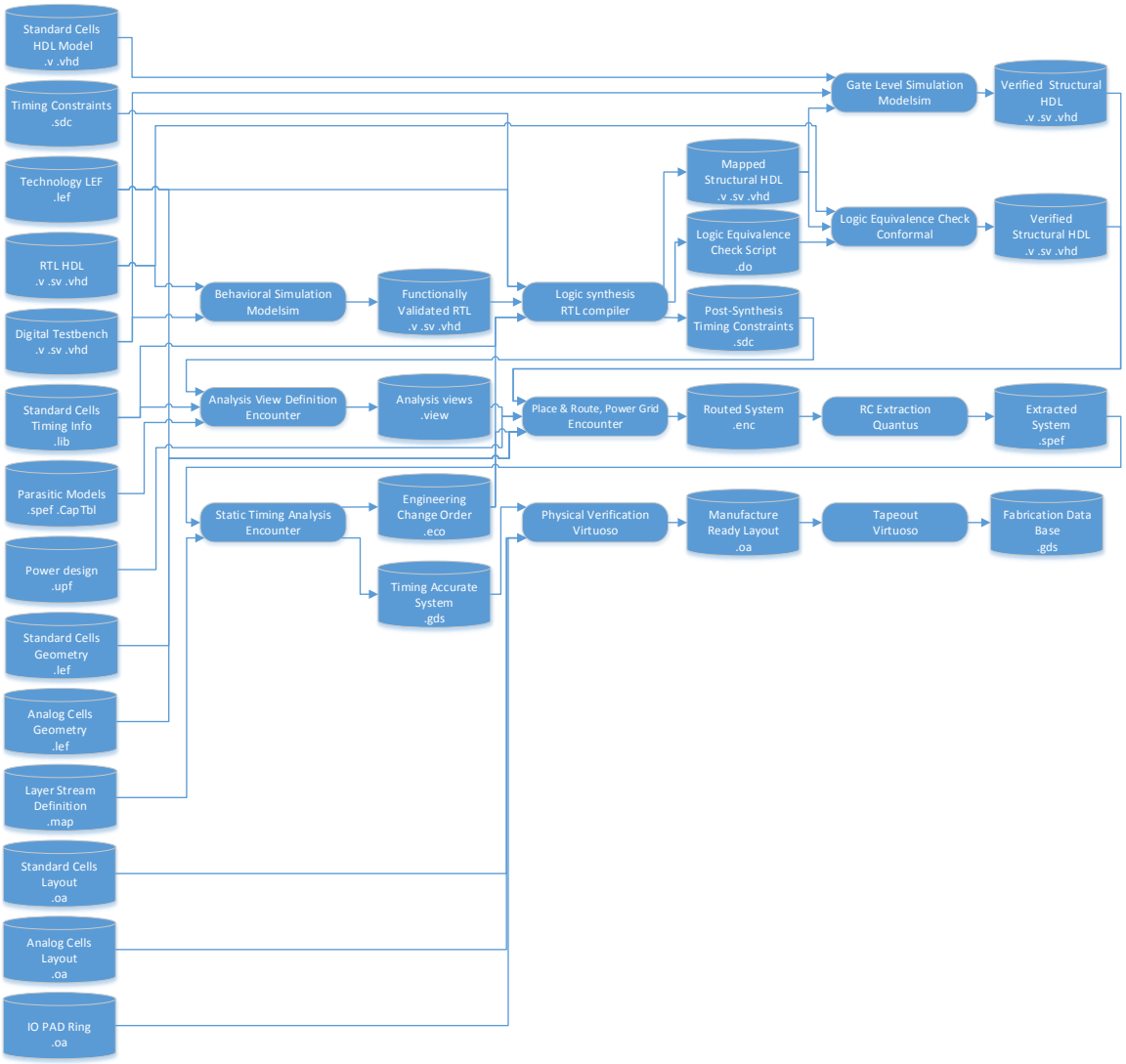


Figure 28. Full VLSI flow for mixed signal integrated circuit physical synthesis

As some ingredients are not available until the last period of the project development, mock ingredients have been used to test the different tools involved in the SerDes physical implementation. The use of mock ingredients allowed the development team to identify risks and mitigate them before some of the pre-required tools were enabled, and before all the actual SerDes ingredients were developed. Not all tools have been evaluated, verification and timing closure related tools are out of scope for this project. The focus for this research it to explore how RTL source code can be transform into a GDS database ready for fabrication.

With the logic synthesis completed as described in Chapter 2, the next ingredient required for physical synthesis is to develop analysis views [7] that will let the place and route tools know what the design goals are.

A. Analysis Views

The analysis views developed for the SerDes consider a worst case and best case corners. In Encounter Digital Implementation System (EDI) analysis views can be declared through a TCL file that can be easily loaded. It will specify the corners where our performance will be evaluated. It consumes timing data from .lib files for the standard cells, mock parasitic model for our routing technology and the timing constraints generated by ***RTL Compiler***. Figure 29 shows the diagram for analysis view using mock parasitic models.

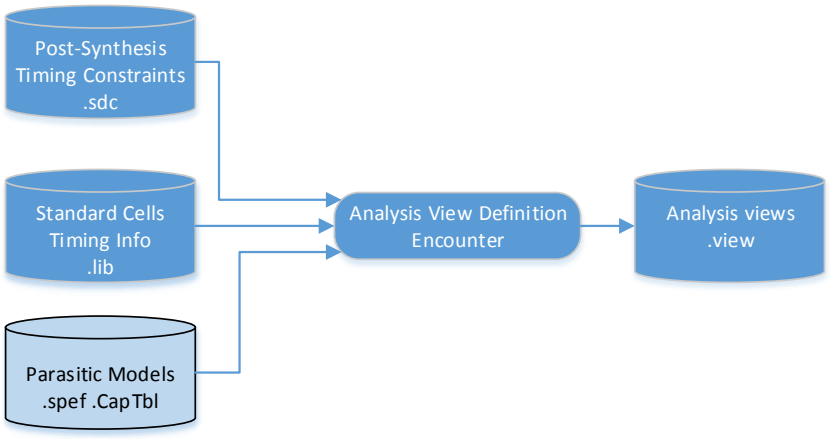


Figure 29. Analysis view using mock parasitic models

The analysis view definition used for the SerDes system is included in appendix K. All .lib files are part of ***IBM*** digital design kit.

B. Power grid, Place and Route

Figure 30 shows the required ingredients and tools used for place, power grid definition and routing [7].

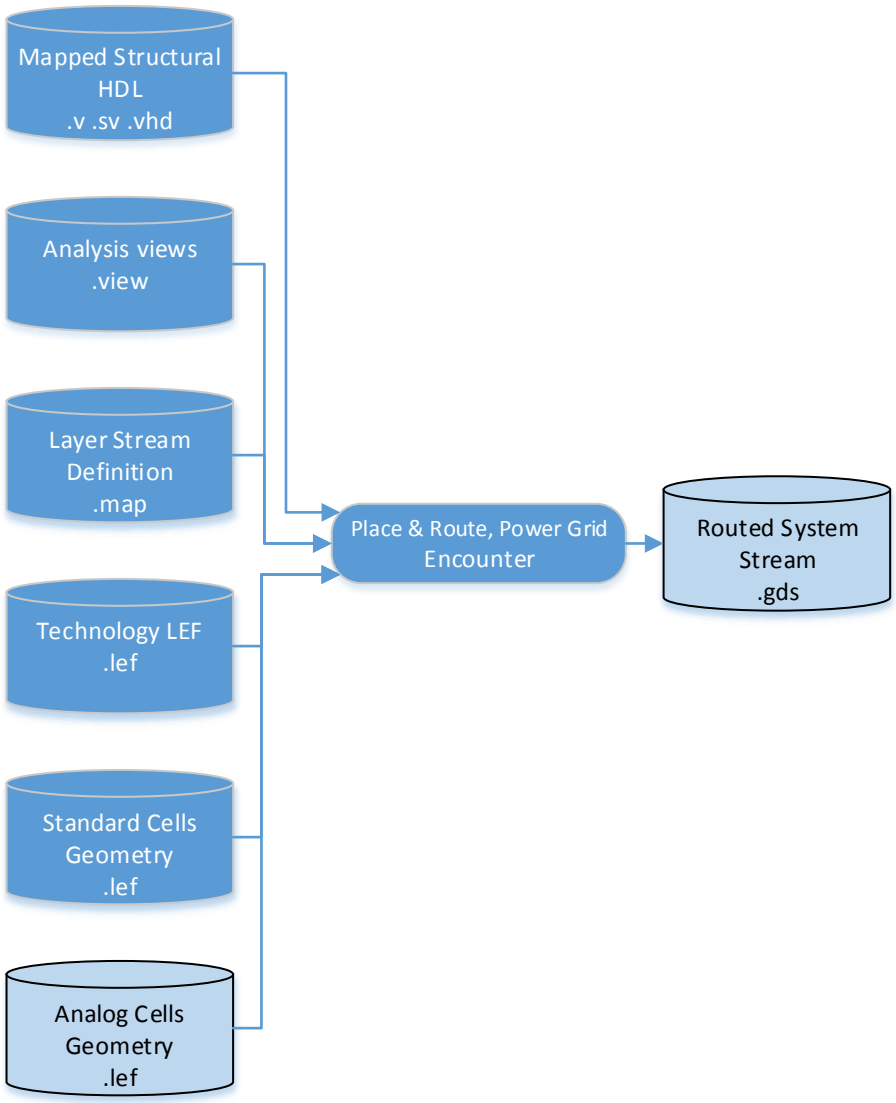


Figure 30. Ingredients to explore EDI Place & Route

The analog cells shown in Figure 30 are simple mock LEF files that describe the geometry of analog modules with the correct pin names based on analog schematic designs. The mock LEF used in EDI is included in appendix L.

The EDI tool requires some non-trivial manipulation to perform power, place and route. Figures 31 to 40 include screenshots of how the mock routed system was generated to explore the GDS2 generation tools.

EDI loads the analysis views through a “globals” TCL file that also defines RTL paths and other miscellaneous file locations. The global specification used for the SerDes in the ITESO Integrated circuit lab is included in appendix M. These settings can be loaded to EDI by using the “Import Design” GUI shown in Figure 31.

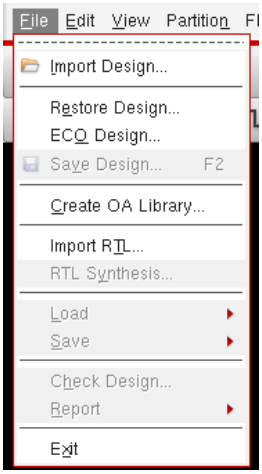


Figure 31. EDI. File menu to access import design utility

After launching the Import Design window, clicking the Load button opens a browser to let the designer search for the TCL file with the global specification.

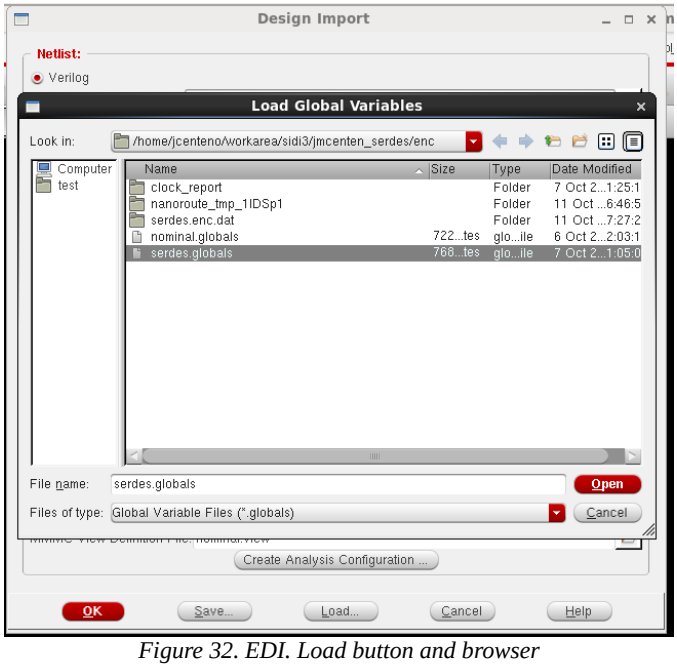


Figure 32. EDI. Load button and browser

After EDI knows where the ingredients come from by loading the globals, a floorplan is defined to set the size of the chip. For this mock flow exploration a floorplan that could fit the serializer and Deserializer gates is enough. Figure 33 shows the configurations used to explore the floorplan in EDI:

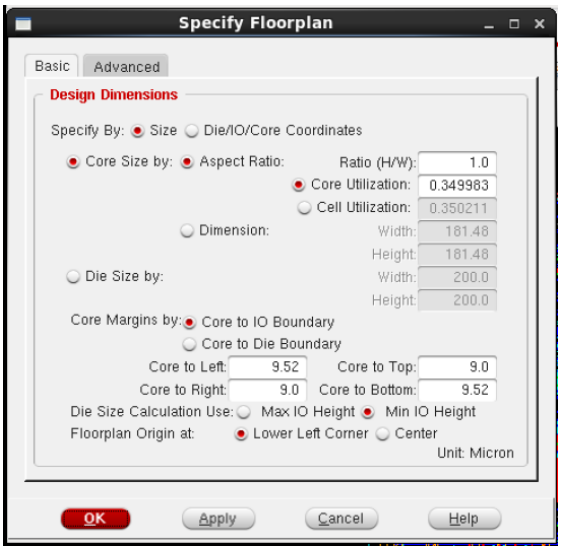


Figure 33. EDI. Floorplan specification

Once the floorplan is specified, the power ring with VDD and GND connections is defined using the EDI wizards.

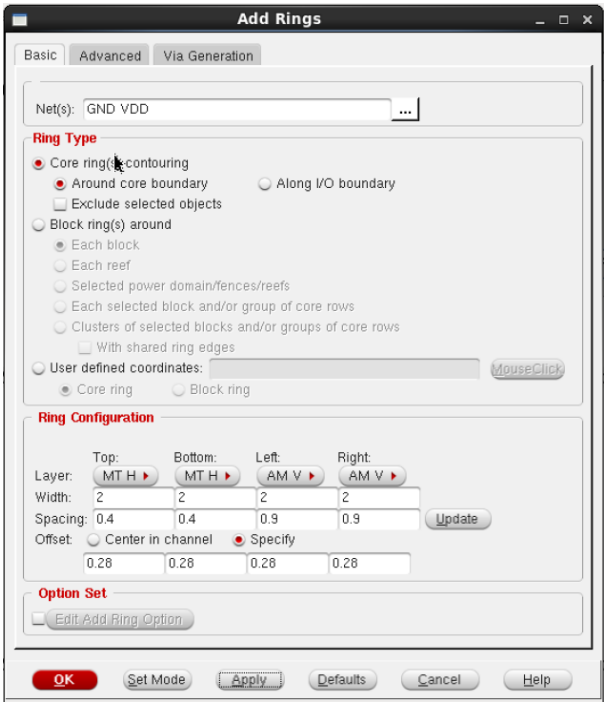


Figure 34. EDI. Power rings specification

With rings available, stripes are created to let the various gates to connect to power with small ground return paths. As a first approach, 10 stripes are created with default configuration and the Basic tab options [7]. From the advanced tab all defaults were used except for “Omit stripes inside block rings” as shown in Figure 35.

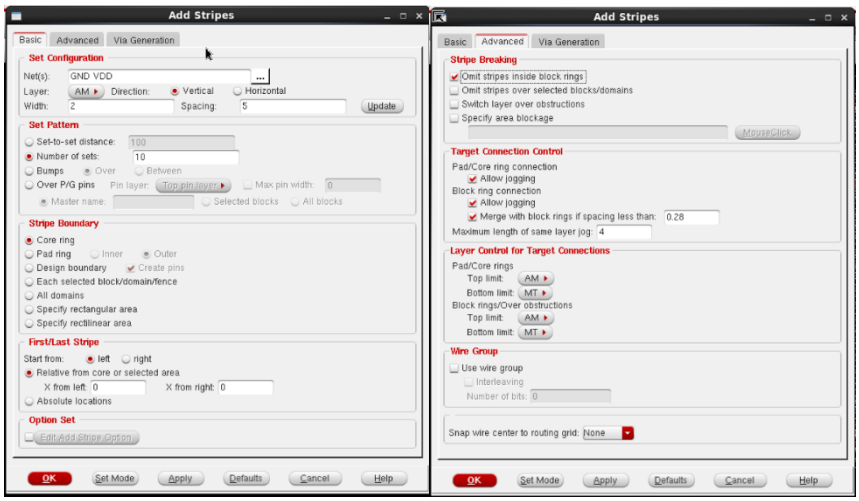


Figure 35. EDI. Power stripes specification

The SerDes digital IPs are synchronous designs, which require a clock. This clock needs some additional information to be created and placed. The standard cell placement does not implement the clock tree on its own. Buffers need to be selected as approved elements to create the clock trees. Figure 36 shows a GUI from EDI used to create the clock specification file which sets the approved blocks to be used during clock tree synthesis.

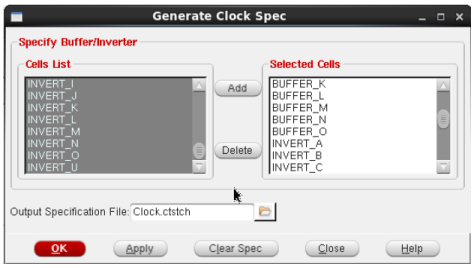


Figure 36. EDI. Clock spec generation

The clock specification created on Figure 37 needs manual changes to include fan-out specifications and route types. The modified Clock.ctstch file is included in appendix N.

Once the power, clock trees, and floorplan are ready we can proceed to use EDI for placing the standard cells that are specified by the mapped structural gate level netlist. A change in the mode setup is required to avoid scan connection reordering.

Figure 37. EDI. Placement setup and launch

Inside EDI, ***Nanoroute*** is the underlying tool used for routing the placed cells and creates the connections between them. Figure 38 shows the setup used for routing exploration.

Figure 38. EDI. Nanoroute setup

Once the design is routed the layout created can be explored in the EDI GUI.

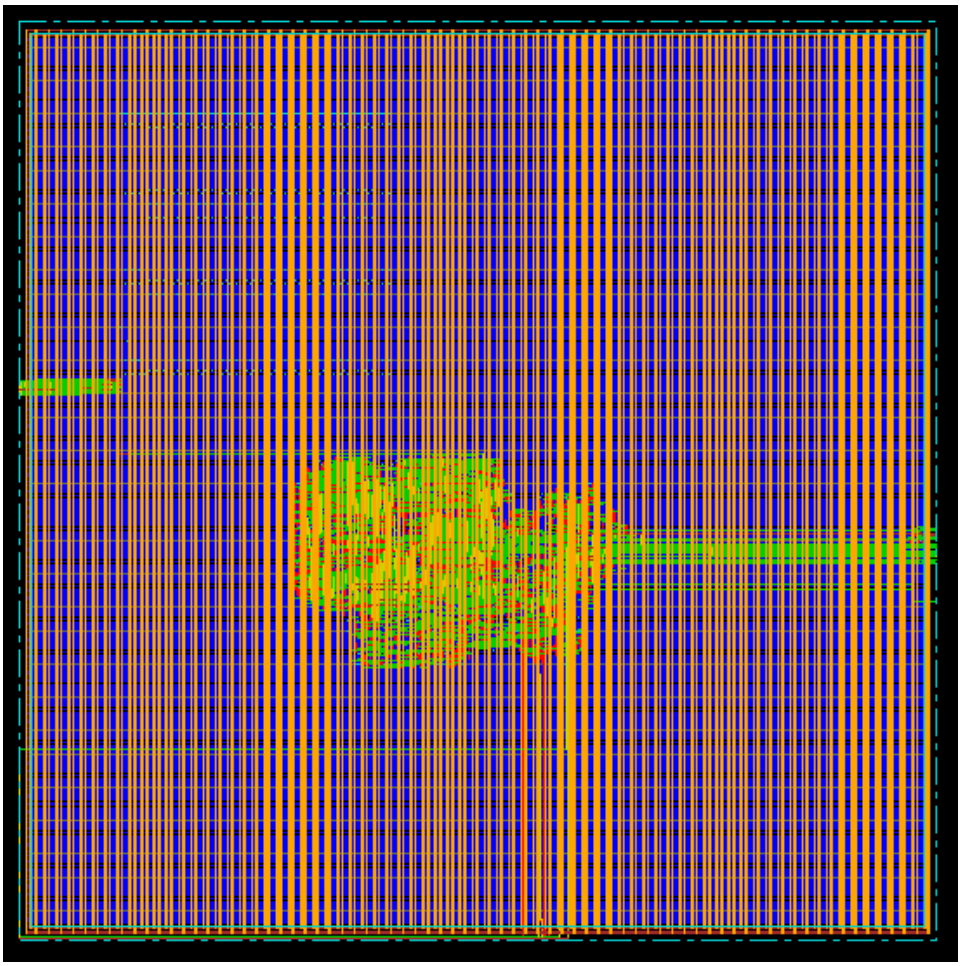


Figure 39. EDI Layout View. Full SerDes system.

While the several windows of the EDI GUI make the physical synthesis approachable and easier to understand. Reproducing the steps can be time consuming and error prone. Once again a TCL script was developed to rerun synthesis up to the layout in Figure 39. The TCL script used to reproduce the results up to this point is available in appendix O.

The routed design is exported to a stream GDS that can be imported to ***Virtuoso. Virtuoso*** substitutes the LEF based layout views in Figure 40 for full layer layouts [7].

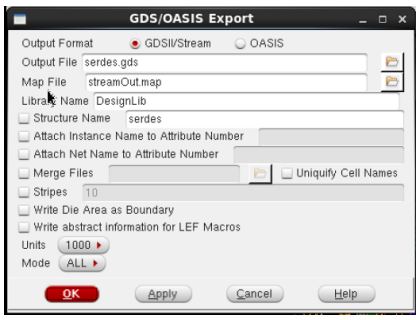


Figure 40. EDI. GDS export using default .map for EDI

The StreamOut.map was not found in the IBM digital design kit. The .map file translates between the EDI way of describing the fabrication layers and a ***Virtuoso*** ready GDS stream. For exploration purposes a mock map file was used were all the unknown layers are mapped to the drawing type of the corresponding layer. This .map would most certainly cause Design Rules Violations but it allows us to explore the EDI export and ***Virtuoso*** import flows. Stand Alone Layout View

C. Self-contained Layout View

Virtuoso is used to bring together the .oa layout views of the various ingredients placed and routed in EDI. Figure 41 shows the import flow using ***Virtuoso***.

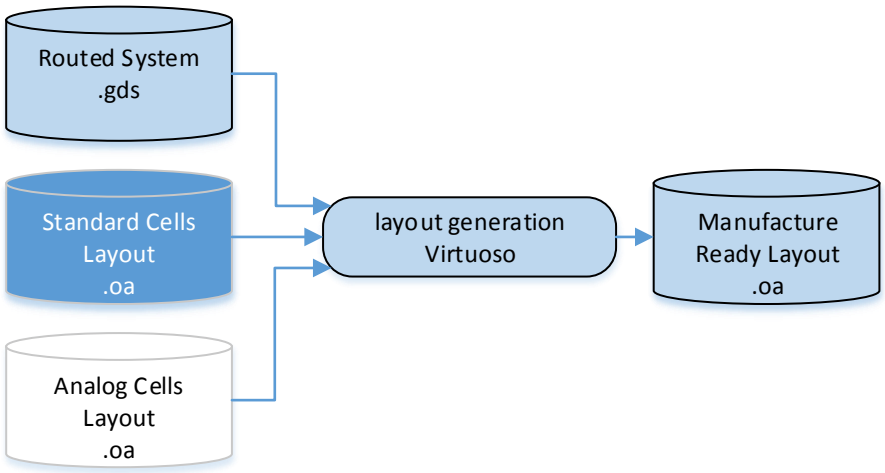


Figure 41. Virtuoso used only to create a self-contained layout

The standard cell layouts from the IBM design kit were used to create the manufacture ready layout, but there was no mock layout for analog cells at the moment of system exploration. This means the layout has an empty area that will be replaced by actual layouts, when the analog design teams release a bundle of a layout views with their corresponding LEF views.

Note how virtuoso is only used to create a manufacture ready layout. The team chose to skip physical validations since the layout generated by encounter did not have a proper layer map file anyway.

Figure 42 shows screenshots of how to import the LEF based layout stream from EDI into ***Virtuoso***, where virtuoso will use the layout views instead of the LEF views.

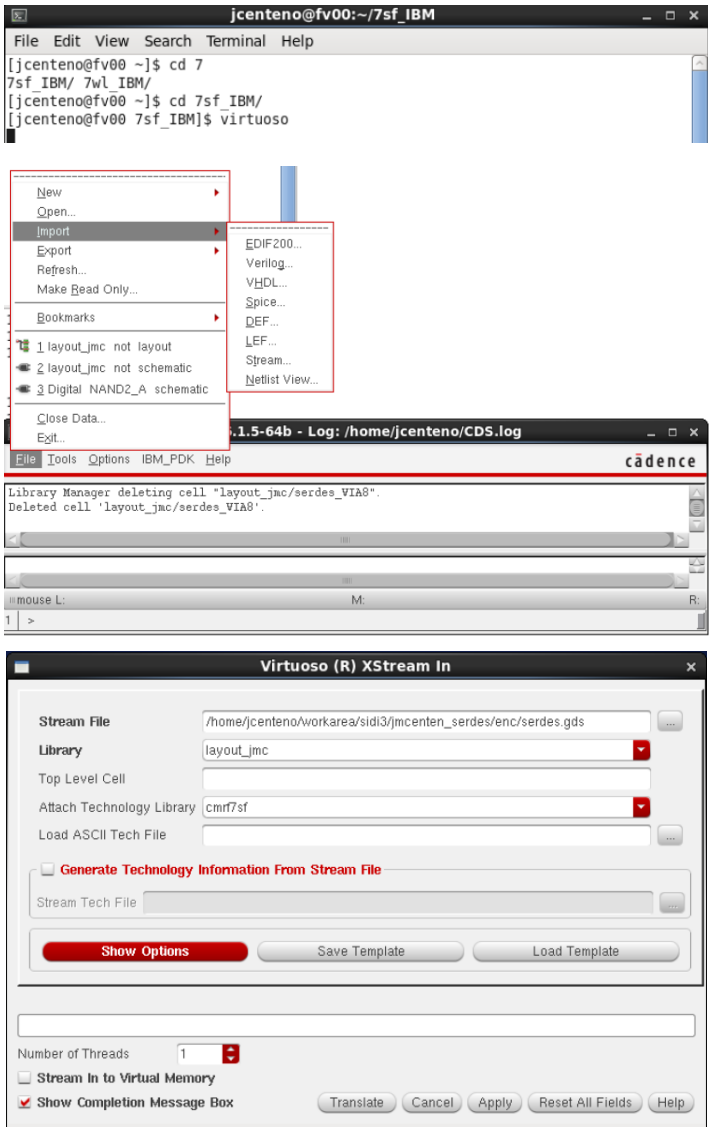


Figure 42. Virtuoso. Importing a stream from EDI to generate a layout view

Figure 43 shows the layout view created by ***Virtuoso***. The white squares are placeholders for the actual layout of the analog modules, analog transmitter and analog receiver.

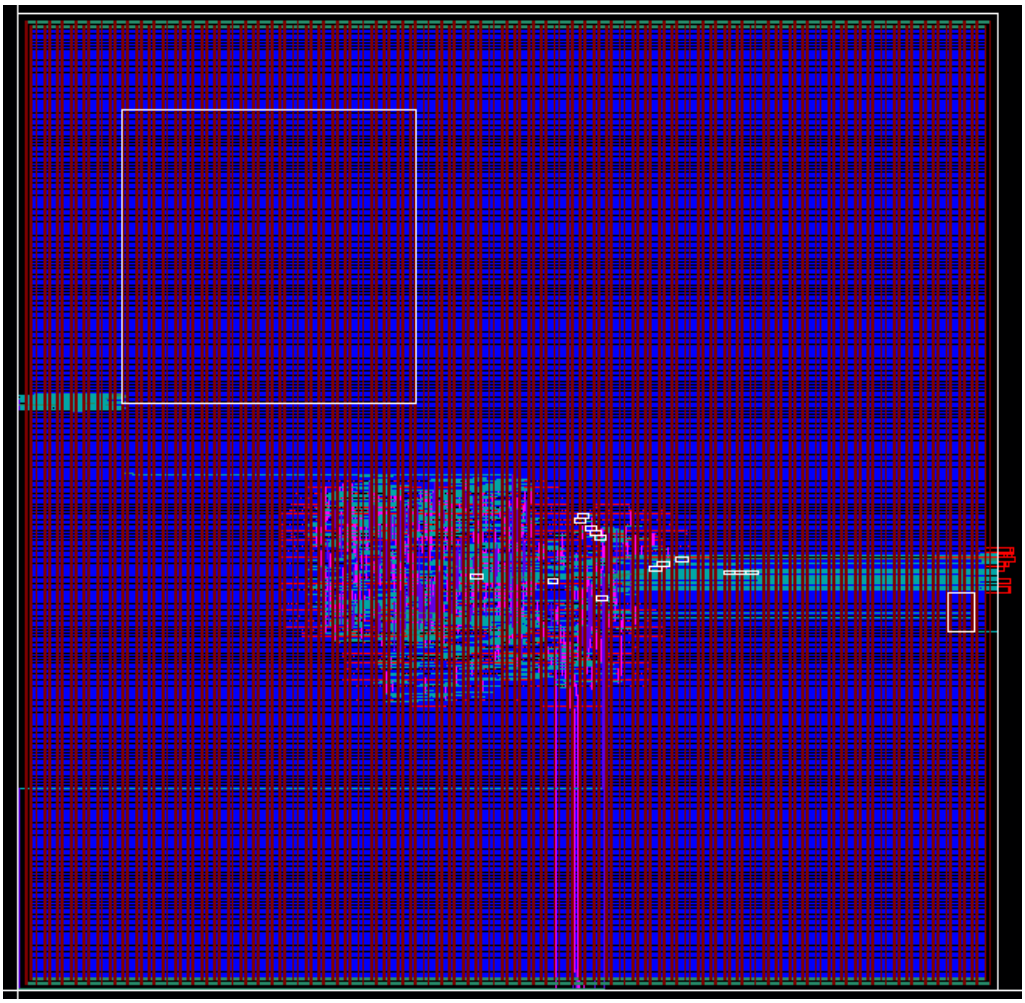


Figure 43. Virtuoso Layout View. SerDes System Core.

The core layout in Figure 43 has some visible problems. A proper layout should not have power grid routes on top of the analog cells. This problem can be addressed by using some advanced features during the EDI place and route. It is also important to remark that the layer map used for this implementation was just a mock ingredient. There may be some layers were text, labels or other shapes not meant for fabrication appear in the drawing purpose of ***Virtuoso*** layers. This problem can be solved by procuring a correct layer map from ***MOSIS***. Finally, the ***IBM*** design kit provides pads for each type of Inputs and Outputs in our design. We need to create a PAD frame, also known as IO PAD ring. The IO PAD frame layout would enclose the core layout in Figure 43. Finally, before attempting system wide Design Rule Checking we need to insert fillers in the empty areas of the SerDes core layout.

D. VLSI Physical Design Results

To summarize the progress in the VLSI exploration the diagram in Figure 28 is presented again in in Figure 40. The diagram in Figure 40 includes highlights on

unexplored tools and undeveloped ingredients. The diagram shows more than half of the ingredients needed for physical synthesis were completely developed. All the tools involved in RTL to GDS synthesis were explored. Half of them have been explored with final ingredients; some others would require to being re-run when final ingredients become available. The tools for physical verification, timing closure and manufacturability testing were not explored as they are out of the scope of this research effort.

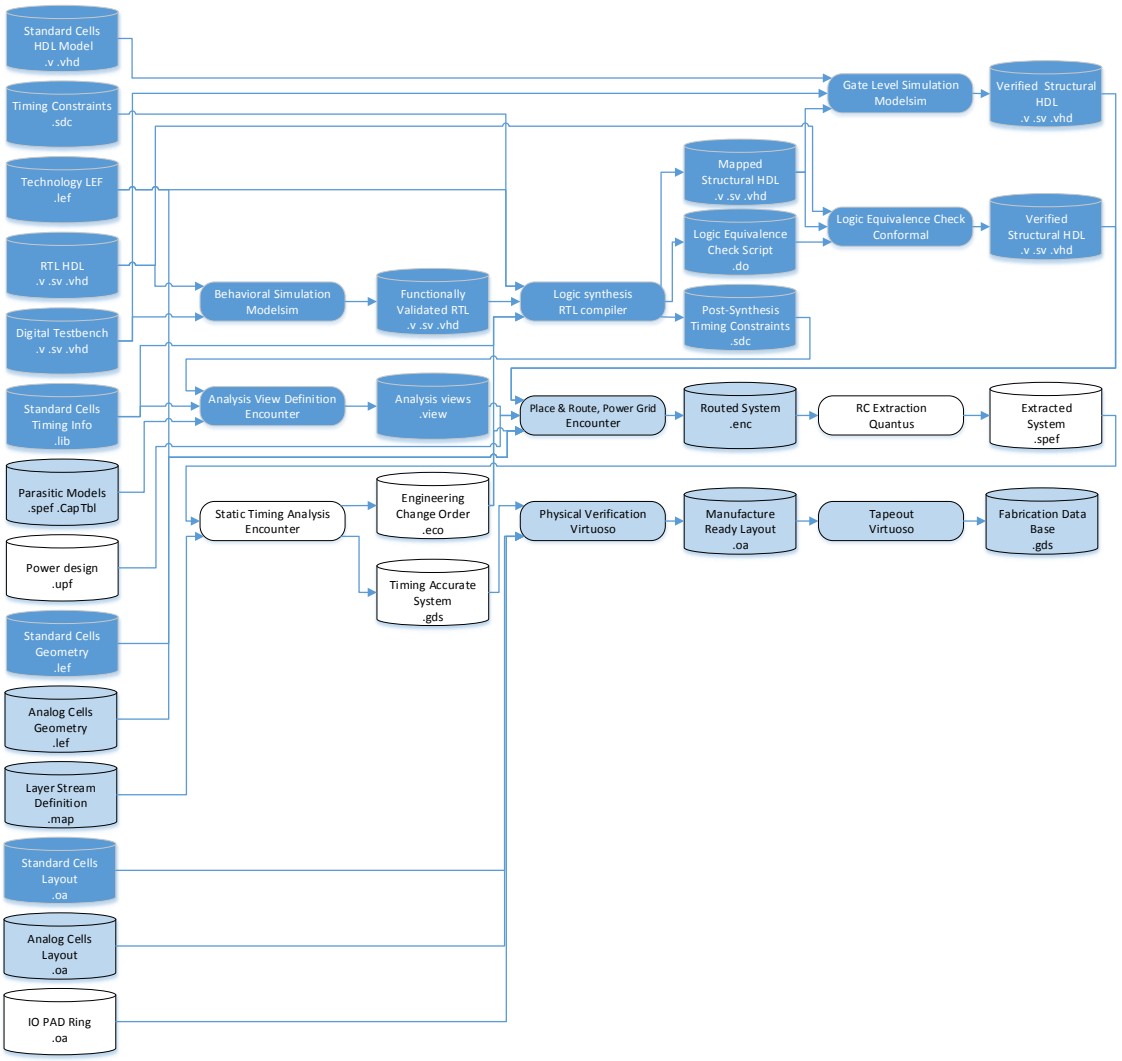


Figure 44. ITESO TV1 VLSI exploration summary. White = unexplored. Clear blue = Use of mock ingredients

CONCLUSIONS

The ITESO Test Vehicle 1 (ITESO TV1) is a SerDes system defined and implemented to explore a modern design flow for mixed signal integrated circuits. The Very Large Scale Integration (VLSI) flow presented includes several necessary tools to design and implement a system on chip. This research narrowed the gap required to successfully fabricate mixed signal SoCs at a 180 nm process node with the current tools and capabilities available at the ITESO integrated circuit lab.

This document presented how the Deserializer digital module was developed and tested for the SerDes System. The deserializer was implemented using standard cells and VLSI flows. The SoC presented includes the Deserializer module, full custom analog cells, other digital Register Transfer Level (RTL) hardware descriptions and test infrastructure for tape out. A first approach to physical synthesis produced a layout view of a full SerDes system which can be transformed into a standalone manufacturing database (GDS).

The ITESO TV1 SerDes system is the first design in the ITESO university to attempt automated integration of modules designed using a wide variety of techniques. For full SerDes manufacturing there are three ingredients missing at the time of creation of this document: power grid design without crossing over analog cells, an I/O pad ring, and a proper layer stream definition (.map). These three ingredients would allow a system to be fabricated, even if PCIe performance goals are not met yet.

The next logical step for SerDes integration is to use real analog layouts during the place and route flow. This requires the generation of a corresponding LEF view for each analog cell in to the design.

Timing constraints were relaxed from original PCIe protocol requirements because the design of a PLL module was not in the scope of this project. When a PLL is integrated into the SerDes, timing closure may require some redesign in the Deserializer module to meet the PCIe aggressive specifications for a 180 nm process node.

Beyond the SerDes system design, ITESO can use this IP as a basis to design and explore more novel or interesting architectures in SoCs. The VLSI technique used for integration of the SerDes has a very big potential for scalability. It can meet the needs for much higher complex designs.

LIST OF REFERENCES

[1] D. Clein, "What is Full Custom Layout Design," UBM Electronics, 08 06 2001. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1277368. [Accessed 20 08 2015].

[2] S. Arámbula, *Diseño de un Sistema Serializador / Des-Serializador de Datos (SERDES) en tecnología CMOS de 0.5 micras,* Guadalajara: ITESO, 2008.

[3] M. Lomeli, *Diseño y mejora de un sistema serdes,* Guadalajara: ITESO, 2008.

[4] A. Athavale and C. Christensen, High-Speed Serial I/OMade Simple, San Jose, California: Xilinx, 2015.

[5] P. A. Franaszek and A. X. Widmer, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code," *IBM Journal of Research and Development,* 1983.

[6] C. Benz, "Chuck Benz ASIC and FPGA Design," 16 06 2015. [Online]. Available: http://asics.chuckbenz.com/.

[7] E. Castorena, *Physical Design,* Tlaquepaque, JAL: ITESO, 2015.

[8] P. E. A. N. R. S. Randall L.Geiger, VLSI Design Techniques for Analog and Digital Circuits, McGrawHill, 1990.

APPENDICES

Appendix A. Clock Divider RTL

```
// Clock Divider
// This block takes a clock reference and divides it by 8.
// It will generate 8 clocks at equidistant phases.

module clock_divider(a_rst, ref_clk, clocks_out);
    input a_rst;
    input ref_clk;
    output reg [7:0] clocks_out;

    reg [3:0] div_by_8_q;
    wire ref_clk_div_by_8;

    // Use a chain of 4 flip flops to divide the clock by 8.
    // The flip flop chain with a not feedback follows this
    // sequence:
    always @(posedge ref_clk, posedge a_rst) begin
        if (a_rst)
            div_by_8_q <= 4'b0;
        else
            div_by_8_q <= { div_by_8_q[2:0], ~div_by_8_q[3]};
    end
    assign ref_clk_div_by_8 = div_by_8_q[3];

    // Delay the divided clock 7 times to generate the equidistant phases
    // Use the divided clock as the first phase
    always @(posedge ref_clk, posedge a_rst) begin
        if (a_rst)
            clocks_out <= 8'd0;
        else
            clocks_out <= {clocks_out[6:0], ref_clk_div_by_8};
    end
endmodule
```

Appendix B. Clock Divider Testbench

```
`timescale 1ns / 100ps

module tb_clock_divider();

    reg a_rst;
    reg ref_clk;
    wire [7:0] clocks_out;

    // reset signal generation
    initial
    begin
        a_rst    <= 1'b0;
        #5 a_rst <= 1'b1;
        #5 a_rst <= 1'b0;
    end

    // clock signal generation (period = 20)
    initial
    begin
        ref_clk <= 1'b0;
        forever #10 ref_clk <= ~ref_clk;
    end

    clock_divider clock_divider_dut(a_rst, ref_clk, clocks_out);
endmodule
```

Appendix C. Deserializer RTL

```
// Deserializer
// This block samples an asynchronous signal. and transforms it into
// a source synchronous parallel bus.
// It uses up-sampling data recovery and a shift register to transform the
// serial input stream into a parallel bus.
//
// This block does not decode or encode any of the inputs.

module deserializer(a_rst, clocks_in, a_rx, c_parallel_out, clock_out, disparity_d,
disparity_q, c_data_valid, comma_detected);
    input a_rst;
    input [7:0] clocks_in;
    input a_rx;
    input disparity_d ;
    output reg [9:0] c_parallel_out;
    output reg clock_out;
    output reg  disparity_q;
    output reg c_data_valid;
    output comma_detected;

    reg [7:0] c_rx_upsampled;
    reg [3:0] num_of_ones_in_rx;
    reg c_rx;
    wire clock;
    reg [9:0] shift_reg;
    reg [3:0] cycle_count;
    integer i;

    // Use the phase 0 clock as the system clock
    assign clock = clocks_in[0];


    // Use a flip flop to remember the running disparity in the decoder
        always @(posedge clock, posedge a_rst) begin
        if (a_rst)
            disparity_q <= 1'b0;
        else
            disparity_q <= disparity_d;
    end

    // CDR
    // Up-sample de serial input with clocks_in at different
    // phases.
    genvar index;
    generate
        for (index=0; index < 8; index=index+1) begin : gen_upsample
            always @(posedge clocks_in[index], posedge a_rst) begin
                if (a_rst)
                     c_rx_upsampled[index] <= 1'b0;
                else
                     c_rx_upsampled[index] <= a_rx;

            end
        end
    endgenerate

    // Count the number of "1"s in the upsamples array
    always @(c_rx_upsampled) begin
        num_of_ones_in_rx = 8'h00;
        for(i=0; i < 8; i=i+1) begin
            num_of_ones_in_rx = num_of_ones_in_rx + c_rx_upsampled[i];
        end
    end

    // Consider a_rx a 1, if a_rx stayed asserted on
    // most of the sampling phases.
    always @(posedge clock, posedge a_rst)
    begin
        if (a_rst)
            c_rx <= 1'b0;
        else
```

```
            c_rx <= (num_of_ones_in_rx > 4'd4)? 1'b1 : 1'b0;
    end

    // sipo 10-bit buffer
    // 10 bit shift register
    always@(posedge clock, posedge a_rst) begin
        if (a_rst)
            shift_reg <= 10'h000;
        else
            shift_reg <= {c_rx, shift_reg[9:1]};
    end

    // Look for comma symbol
    assign comma_detected = (shift_reg == 10'b0001111100) || (shift_reg ==
10'b1110000011);

    // Use a 10 cycle counter to generate a data_valid signal.
    // Reset the counter if a special sync character, such as
    // a comma is identified
    always @(posedge clock, posedge a_rst) begin
        if (a_rst) begin
            cycle_count <= 4'd9;
            c_data_valid <= 1'b0;
            clock_out <= 1'b0;
        end
        else begin
            // 10 cycle counter
            if(comma_detected || (cycle_count == 4'd0))
                // Restart
                cycle_count <= 4'd9;
            else
                //Count down
                cycle_count <= cycle_count - 1;

            // Data is valid when the count down expires.
            if(comma_detected)
                c_data_valid <= 1'b0;
            else if (cycle_count == 4'd1)
                c_data_valid <= 1'b1;
            else
                c_data_valid <= 1'b0;

            // Assert clock_out one cycle after data_valid
            // De-assert clock_out in cycle 5
            if(cycle_count == 4'd5)
                clock_out <= 1'b0;
            else if (cycle_count == 4'd0 &&(!comma_detected))
                clock_out <= 1'b1;
        end
    end

    // reg outputs
    always @(posedge clock, posedge a_rst) begin
        if(a_rst)
            c_parallel_out <= 10'd0;
        else
            if (c_data_valid)
                c_parallel_out <= shift_reg;
    end

endmodule
```

Appendix D. Deserializer Testbench

```
// Deserializer
// This block samples an asynchronus signal. and transforms it into
// a source synchronous parallel bus.
// It uses up-sampling data recovery and a shift register to transform the
// serial input stream into a parallel bus.
//
// This block does not decode or encode any of the inputs.

`timescale 1ns / 100ps
module tb_deserializer();
    wire       clk;
    reg        a_rst;
    reg        a_rx;

    reg  [7:0] clocks_in;
    wire [9:0] c_parallel_out;
    wire clock_out;
    integer i,j;

    // clock signal generation (period = 20)
    initial
    begin
        clocks_in = 8'h0f;
        #50;
        forever begin
            for (i=0; i < 4; i=i+1) begin
                #10 clocks_in[i] = ~clocks_in[i];
                clocks_in[4+i] = ~clocks_in[4+i];
            end
        end
    end

    assign clk = clocks_in[0];


    // serial input generation
    initial
    begin
        // reset
        a_rst    = 1'b0;
        a_rx = 1'b0;
        #100  a_rst = 1'b1;
        #500 a_rst = 1'b0;

        @(posedge clk);
        // send comma
        serial_mblb(10'h0f8);

        // send test patterns
        serial_mblb(10'h2aa);
        serial_mblb(10'h155);
        serial_mblb(10'h10f);
        serial_mblb(10'h000);
    end


    task serial_mblb(input[9:0] data);
    begin
        for (j=9; j >= 0; j=j-1) begin
            a_rx = data[j];
            @(posedge clk);
        end
    end

    endtask

    deserializer dut(a_rst, clocks_in, a_rx, c_parallel_out, clock_out);
endmodule
```

Appendix E. SerDes RTL with test infrastructure included

```
module serdes(
    input reset, //reset
    input ref_clk, //main system clock
    input rxpcie_in_p, //RXA In positive
    input rxpcie_in_n, //RXA In negative
    output rx_clk, //RXD clock
    output [8:0] rx_data, //RXD Out
    input[8:0]tx_data, //TXD parallel in
    output tx_pcie_p, //TXA Out Positive
    output tx_pcie_n, //TXA Out Negative
    input[1:0] analog_serializer_premphasis_p, //TXA Configuration Pins
    input[1:0] analog_serializer_premphasis_n, //TXA Configuration Pins
    input[3:0] analog_serializer_impedance_p, //TXA Configuration Pins
    input[3:0] analog_serializer_impedance_n, //TXA Configuration Pins
    input[1:0] analog_serializer_amplitude_p, //TXA Configuration Pins
    input[1:0] analog_serializer_amplitude_n, //TXA Configuration Pins
    input [3:0] bistSel,//bist register selector
    input testIn, //Bist test signal in
    input [3:0]mode,//Bist Configuration pins
    output testOut_P, //Bist test signal out
    output testOut_N,
    output bist //Bist success/fail output pin
);

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

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

//Register Declarations
    reg [8:0]cnt;//LFSR Seed Load Signal
    reg fFlag; //Mode 13 First Valid Data Received for Bist Syncronization
    reg [9:0]seed;
    reg [9:0]compVal;
    reg load;//activate lfsr
    reg [8:0]lfsr_out; //LFSR data out
    reg LFSRxenable;//Lfsr flip flop enable
    reg ldflag; //LFSR Load flag Signal
```

```
    reg compflag; //BIST comparator activation if mode 11
    reg compflagS; //BIST comparator activation if mode 13
    reg[8:0]rx_IN; //Digital Receiver output to connect to Mux B
    reg commaFlag;//First comma detected Flag (Mode 13)
    reg validFlag;//First Valid data decoded after the first commadetected Flag (Mode
13)
    reg FF_testOut_P; //Register Containing the Positive signal of the Testout
    reg FF_testOut_N; //Register Containing the Negative signal of the Testout
//Register Declarations

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

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

    always@(*) begin

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

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

        //Mode 13 - Send data to the Digital Transmitter until valid data is decoded on
the Digital Receiver
        if (mode==4'd13) begin
            if(validFlag) begin
                rx_IN=rx_data;
            end else begin
                rx_IN=0;
            end
        end else begin
            rx_IN=rx_data;
        end

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

```
            end else begin
                LFSRxenable=0;
            end
        end else begin
            LFSRxenable=1;
        end
    end
//Combinational Logic

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

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

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

    //LFSR output assign - Make sure the first output of the LFSR is a valid comma for
the Digital Receiver
    always @(posedge ser_clock or posedge reset) begin
        if (reset) begin
            cnt <= 0;
            load <= 1;
            lfsr_out<=9'd0;
            compVal<=10'b1111111111;
            fFlag<=0;
        end else begin
            if(ldflag && cnt==4'b0000) begin
                load <= 1;
                cnt<=cnt+1'b1;
            end    else if(!ldflag) begin
                cnt <=9'd0;
                load <= 0;
            end else begin
                load <= 0;
                cnt<=cnt+1'b1;
            end
            if (mode==4'd11||mode==4'd10) begin
                if (load && LFSRenable) begin
                    lfsr_out<=9'b111111100;
                end else if(!load && LFSRenable) begin
                    lfsr_out<=state_out[8:0];
                end
            end
            if (mode==4'd13) begin
                if (load && LFSRenable2) begin
                    lfsr_out<=9'b111111100;
                end else if(!load && LFSRenable2) begin
```

```
                    lfsr_out<={1'b0,state_out[7:0]};
                    if(fFlag)begin
                        if(compVal==10'b1111111111) begin
                            compVal<=encoded_tx_data2;
                        end
                    end else begin
                        if(LFSRenable)begin
                            fFlag<=1;
                        end
                    end
                end
            end    else begin
                compVal<=10'b1111111111;
                fFlag<=0;
            end
        if(cnt==8'd159)begin
            cnt<=9'd0;
        end
        end
    end
//Sequential Logic

//Module Instantiation
    // clocking section
    clock_divider clock_dividerx(
        .a_rst(reset),
        .ref_clk(ref_clk),
        .clocks_out(clock_div8_phases));
    // Analog Receiver
    analogue_receiver analogue_receiverx(
        .rxpcie_in_p(rxpcie_in_p),
        .rxpcie_in_n(rxpcie_in_n),
        .rxpcie_out(rx_pcie));
    //Digital Receiver - Deserializer
    deserializer deserializerx(
        .a_rst(reset),
        .clocks_in(clock_div8_phases),
        .a_rx(muxA),//rx_pcie
        .c_parallel_out(encoded_rx_data),
        .clock_out(rx_clk),
        .disparity_d(rx_disparityd),
        .disparity_q(rx_disparityq),
        .c_data_valid(c_data_valid),
        .comma_detected(comma_detected));
    //Digital Receiver - Decoder (8b/10b)
    decode decodex(
        .datain(encoded_rx_data),
        .dispin(rx_disparityq),
        .dataout(rx_data),
        .dispout(rx_disparityd),
        .code_err(),
        .disp_err());
    //Digital Transmitter - Encoder (8b/10b)
    encode encodex(
        .datain(muxB),
        .dispin(tx_disparityq),
        .dataout(encoded_tx_data),
        .dispout(tx_disparityd)
    );
    //Digital Transmitter - Serializer
    serializer serializerx(
        .par_in(encoded_tx_data),
        .clk(ser_clock),
        .rst(reset),
        .disparity_d(tx_disparityd),
        .ser_out(tx_pcie),
        .disparity_q(tx_disparityq),
        .tx_frame_started(LFSRenable)
    );
    //Analog Transmitter
    analog_transmitter analog_transmitterx(
        .Data(muxC), //tx_pcie
        .Random(1'b0),
        .Data_Selector(1'b0),
```

```
    .Eq_A_P(analog_serializer_premphasis_p[0]),
    .Eq_A_N(analog_serializer_premphasis_n[0]),
    .Eq_B_P(analog_serializer_premphasis_p[1]),
    .Eq_B_N(analog_serializer_premphasis_n[1]),
    .Imp_A_P(analog_serializer_impedance_p[0]),
    .Imp_A_N(analog_serializer_impedance_n[0]),
    .Imp_B_P(analog_serializer_impedance_p[1]),
    .Imp_B_N(analog_serializer_impedance_n[1]),
    .Imp_C_P(analog_serializer_impedance_p[2]),
    .Imp_C_N(analog_serializer_impedance_n[2]),
    .Imp_D_P(analog_serializer_impedance_p[3]),
    .Imp_D_N(analog_serializer_impedance_n[3]),
    .Amp_A_P(analog_serializer_amplitude_p[0]),
    .Amp_A_N(analog_serializer_amplitude_n[0]),
    .Amp_B_P(analog_serializer_amplitude_p[1]),
    .Amp_B_N(analog_serializer_amplitude_n[1]),
    .Direct_Out_P(),
    .Direct_Out_N(),
    .Trans_Out_N(tx_pcie_n),
    .Trans_Out_P(tx_pcie_p));
//Test Module Instances
    //Mode Decoder
    muxDecode muxDecode(
        .mode(mode),
        .sel(sel)
    );
    //MUX A
    Mux_3a1 aMux(
        .A(tx_pcie),
        .B(testIn),
        .C(soutB),
        .Sel(sel[9:8]),
        .Q(muxA)
    );
    //MUX B
    Mux_3a1_8b bMux(
        .A(tx_data),
        .B(rx_IN),
        .C(lfsr_out),
        .Sel(sel[7:6]),
        .Q(muxB)
    );
    //MUX C
    Mux_4a1 cMux(
        .A(tx_pcie),
        .B(testIn),
        .C(soutB),
        .D(q),
        .Sel(sel[5:4]),
        .Q(muxC)
    );
    //MUX D
    Mux_3a1 dMux(
        .A(tx_pcie),
        .B(rxAnDiv),
        .C(tx_pcie2),
        .Sel(sel[3:2]),
        .Q(muxD)
    );
    //MUX E
    Mux_2a1_8b eMux(
        .A({8'b00000000,testIn}),
        .B(rx_data),
        .Sel(sel[1:0]),
        .Q(muxE)
    );
    //BUFFER A
    buffer bufferA(
        .sin(rx_pcie),
        .sout(soutA)
    );
    //Frequency Divider
    freqDiv freqDiv(
        .rxAn(soutA),
```

```
            .rst(reset),
            .rxAnDiv(rxAnDiv)
        );
        //BUFFER B
        buffer bufferB(
            .sin(soutA),
            .sout(soutB)
        );
        //Linear Feedback Shift Register
        lfsr lfsr(
            .q(q),
            .clk(ser_clock),
            .rst(reset),
            .seed(seed),
            .load(load),
            .state_out(state_out),
            .enable(LFSRxenable)
        );
        //LFSR - Parallel out Encoder (8b/10b)
        encode encodex2(
            .datain(lfsr_out),
            .dispin(tx_disparityq2),
            .dataout(encoded_tx_data2),
            .dispout(tx_disparityd2)
        );
        //LFSR - Encoded data Serializer
        serializer serializerx2(
            .par_in(encoded_tx_data2),
            .clk(ser_clock),
            .rst(reset),
            .disparity_d(tx_disparityd2),
            .ser_out(tx_pcie2),
            .disparity_q(tx_disparityq2),
            .tx_frame_started(LFSRenable2)
        );
        //Comparator
        comparator comparator(
            .cIn(muxE),
            .clk(ser_clock),
            .rst(reset),
            .lfsr_in(state_out[9:0]),
            .compflag(compflag),
            .compflagS(compflagS),
            .LFSRenable(LFSRenable),
            .bist(bist),
            .bistSel(bistSel),
            .ser_in(tx_pcie2),
            .compVal(compVal)
        );
    //Test Module Instances
//Module Instantiation
endmodule
```

Appendix F. SerDes Testbench

```
`timescale 1ns/1ps

module tb_serdes();
    reg ref_clk;
    reg reset;
    reg[8:0] tx_data;
    wire rx_clk;
    wire [8:0] rx_data;
    wire rxpcie_in_p;
    wire rxpcie_in_n;
    wire tx_pcie_n;
    wire tx_pcie_p;
    integer i;

    // ref_clk signal generation (period = 1000ps )
    initial
    begin
        ref_clk <= 1'b0;
        forever #0.5 ref_clk <= ~ref_clk;
    end

    assign tx_clk = serdesx.serializerx.clk;

    // input generation
    initial
    begin
        // reset
        reset    = 1'b0;
        tx_data = 9'd0;
        #100  reset = 1'b1;
        #500 reset = 1'b0;

        @(posedge tx_clk);
        // tx_data input generation
            parallel(9'b111111100);
            for (i = 0; i < 256; i = i + 1)
                parallel(i[8:0]);

    end

    // Loop-Back
    assign rxpcie_in_p = tx_pcie_p;
    assign rxpcie_in_n = tx_pcie_n;

    task parallel(input[8:0] data);
    begin
      @(posedge tx_clk);
           tx_data = data;
       repeat (9)
           @(posedge tx_clk);
    end
    endtask

    //DUT
    serdes serdesx(
        .reset(reset),
        .ref_clk(ref_clk),
        .rxpcie_in_p(rxpcie_in_p),
        .rxpcie_in_n(rxpcie_in_n),
        .rx_clk(rx_clk),
        .rx_data(rx_data),
        .tx_data(tx_data),
        .tx_pcie_p(tx_pcie_p),
        .tx_pcie_n(tx_pcie_n),
        .analog_serializer_premphasis_p('b0),
        .analog_serializer_premphasis_n('b0),
        .analog_serializer_impedance_p('b0),
        .analog_serializer_impedance_n('b0),
        .analog_serializer_amplitude_p('b0),
        .analog_serializer_amplitude_n('b0),
        .bistSel('b0),
```

```
        .testIn('b0),
        .mode('b0),
        .testOut_P(),
        .testOut_N(),
        .bist());
endmodule
```

Appendix G. Timing constraints

```
dc::set_time_unit -picoseconds
dc::set_load_unit -femtofarads

current_design serdes

set_clock_gating_check -setup 0.0

set_wire_load_mode "enclosed"

# Using a slow clock while a PLL is designed
set ref_clk_period 1000

# PCIe gen1 speed is 2 Gbps. Overclocked by 8, requires a reference
# clock of 16 GHz -> 62.5 ps period. Using 1000 for now
create_clock -name ref_clk -period $ref_clk_period [get_ports ref_clk]

create_generated_clock  \
    -name clocks_out_reg_0 \
    -source clock_dividerx/ref_clk  \
    -divide_by 8 [get_pins clock_dividerx/clocks_out_reg[0]/q]

create_generated_clock  \
    -name clocks_out_reg_1 \
    -source clock_dividerx/clocks_out_reg[0]/q  \
    -divide_by 1 [get_pins clock_dividerx/clocks_out_reg[1]/q] \
    -edges {1 2 3} \
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock  \
    -name clocks_out_reg_2 \
    -source clock_dividerx/clocks_out_reg[0]/q  \
    -divide_by 1 [get_pins clock_dividerx/clocks_out_reg[1]/q] \
    -edges {1 2 3} \
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock  \
    -name clocks_out_reg_3 \
    -source clock_dividerx/clocks_out_reg[0]/q  \
    -divide_by 1 [get_pins clock_dividerx/clocks_out_reg[1]/q] \
    -edges {1 2 3} \
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock  \
    -name clocks_out_reg_4 \
    -source clock_dividerx/clocks_out_reg[0]/q  \
    -divide_by 1 [get_pins clock_dividerx/clocks_out_reg[1]/q] \
    -edges {1 2 3} \
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock  \
    -name clocks_out_reg_5 \
    -source clock_dividerx/clocks_out_reg[0]/q  \
    -divide_by 1 [get_pins clock_dividerx/clocks_out_reg[1]/q] \
    -edges {1 2 3} \
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock  \
    -name clocks_out_reg_6 \
    -source clock_dividerx/clocks_out_reg[0]/q  \
    -divide_by 1 [get_pins clock_dividerx/clocks_out_reg[1]/q] \
    -edges {1 2 3} \
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock  \
    -name clocks_out_reg_7 \
    -source clock_dividerx/clocks_out_reg[0]/q  \
    -divide_by 1 [get_pins clock_dividerx/clocks_out_reg[1]/q] \
    -edges {1 2 3} \
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock  \
```

```
    -name clocks_out_reg_8 \
    -source clock_dividerx/clocks_out_reg[0]/q  \
    -divide_by 1 [get_pins clock_dividerx/clocks_out_reg[1]/q] \
    -edges {1 2 3} \
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]


set_attribute slew { 5 5 6 6 } [get_clocks ref_clk]

clock_uncertainty -setup 3.0 [get_clocks ref_clk]

clock_uncertainty -hold 1.0 [get_clocks ref_clk]

set_clock_latency -source 10 [get_clocks ref_clk]


set_input_delay -name IDelay -clock clocks_out_reg_0 100 [all_inputs]

set_output_delay -name ODelay -clock clocks_out_reg_0 100 [all_outputs]

set_driving_cell -lib_cell BUFFER_O -pin "Z" -input_transition_rise 100 -
input_transition_fall 100 [all_inputs]

set_max_transition 200 [current_design]

set_max_fanout 10 [current_design]

set_load 40 [all_outputs]
```

Appendix H. *RTL compiler*

```
#### Template Script for RTL->Gate-Level Flow (generated from RC RC14.23 - v14.20-
s027_1)


################################################################################
## Preset global variables and attributes
################################################################################

set DESIGN serdes
set SYN_EFF medium
set MAP_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH outputs
set _REPORTS_PATH reports
set _LOG_PATH logs/tcl_logs_${DATE}
##set ET_WORKDIR <ET work directory>
set_attribute lib_search_path {.
/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_20111130
/std_cell/v.20111130/synopsys/nom} /
set_attribute hdl_search_path {. ../rtl} /

##set_attribute wireload_mode <value> /
set_attribute information_level 9 /
set_attribute auto_ungroup none /

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


#set_attribute library <libname>
set_attr library {
    PnomV180T025_STD_CELL_7RF.lib
}
## PLE
set_attribute lef_library
/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_20111130
/std_cell/v.20111130/lef/cmos7rf_6AM_tech.lef
## Provide either cap_table_file or the qrc_tech_file
##set_attribute cap_table_file <file> /
#set_attribute qrc_tech_file <file> /
##generates <signal>_reg[<bit_width>] format
#set_attribute hdl_array_naming_style %s\[%d\] /
######

#####################################################################
## Load Design
#####################################################################
#read_hdl <hdl file name(s)>
read_hdl -v2001 {
    buffer.v
    clock_divider.v
    comparator.v
    decode.v
    deserializer.v
    encode.v
    flipflop.v
    freqDiv.v
    lfsr.v
    Mux_2a1_8b.v
    Mux_3a1_8b.v
    Mux_3a1.v
    Mux_4a1.v
    muxDecode.v
    mux.v
    serdes.v
    serializer.v
}

set_attribute hdl_track_filename_row_col true
```

```
elaborate $DESIGN
puts "Runtime & Memory after 'read_hdl'"
timestat Elaboration

check_design -unresolved

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

read_sdc ../constraints/serdes.sdc

puts "The number of exceptions is [llength [find /designs/$DESIGN -exception *]]"


#set_attribute force_wireload <wireload name> "/designs/$DESIGN"

if {![file exists ${_LOG_PATH}]} {
  file mkdir ${_LOG_PATH}
  puts "Creating directory ${_LOG_PATH}"
}

if {![file exists ${_OUTPUTS_PATH}]} {
  file mkdir ${_OUTPUTS_PATH}
  puts "Creating directory ${_OUTPUTS_PATH}"
}

if {![file exists ${_REPORTS_PATH}]} {
  file mkdir ${_REPORTS_PATH}
  puts "Creating directory ${_REPORTS_PATH}"
}
report timing -lint


#############################################################################################
## Define cost groups (clock-clock, clock-output, input-clock, input-output)
#############################################################################################

## Uncomment to remove already existing costgroups before creating new ones.
## rm [find /designs/* -cost_group *]

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
}


#### To turn off sequential merging on the design
#### uncomment & use the following attributes.
##set_attribute optimize_merge_flops false /
##set_attribute optimize_merge_latches false /
#### For a particular instance use attribute 'optimize_merge_seqs' to turn off
sequential merging.



#############################################################################################
#############
## Synthesizing to generic
#############################################################################################
#############


synthesize -to_generic -eff $SYN_EFF
```

```
write_hdl > ${DESIGN}_generic.v
puts "Runtime & Memory after 'synthesize -to_generic'"
timestat GENERIC
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_generic.rpt
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 -num_paths 10 -cost_group [list $cg] >
$_REPORTS_PATH/${DESIGN}_[basename $cg]_post_map.rpt
}
generate_reports -outdir $_REPORTS_PATH -tag map
summary_table -outdir $_REPORTS_PATH



##Intermediate netlist for LEC verification..
write_hdl -lec > ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v
write_do_lec -revised_design ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v -logfile
${_LOG_PATH}/rtl2intermediate.lec.log > ${_OUTPUTS_PATH}/rtl2intermediate.lec.do

## ungroup -threshold <value>

################################################################################################
###############
## 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
write_hdl > ${DESIGN}_syn.v

puts "Runtime & Memory after incremental synthesis"
timestat INCREMENTAL

foreach cg [find / -cost_group *] {
  report timing -num_paths 10 -cost_group [list $cg] >
$_REPORTS_PATH/${DESIGN}_[basename $cg]_post_incr.rpt
}


################################################
## Spatial mode optimization
################################################

## Uncomment to enable spatial mode optimization
##synthesize -to_mapped -spatial


################################################################################################
##############
## write Encounter file set (verilog, SDC, config, etc.)
################################################################################################
##############
```

```
##write_encounter design -basename <path & base filename> -lef <lef_file(s)>

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
report design_rules > $_REPORTS_PATH/${DESIGN}_drv.rpt
write_design -basename ${_OUTPUTS_PATH}/${DESIGN}_m
write_hdl  > ${_OUTPUTS_PATH}/${DESIGN}_m.v
write_script > ${_OUTPUTS_PATH}/${DESIGN}_m.script
write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_m.sdc


################################
### write_do_lecl
################################


write_do_lec -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 -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. Analog Receiver Blocks Behavioral Description for Black Box

```
// analogue_receiver
// This block is a behavioural model for the analogue front end of the
// Deserializer. The analog front end transforms a weak differential signal
// into a CMOS single ended signal.

module analogue_receiver(rxpcie_in_p, rxpcie_in_n, rxpcie_out);
    input rxpcie_in_p;
    input rxpcie_in_n;
    output rxpcie_out;

    assign rxpcie_out = rxpcie_in_p;
endmodule
```

Appendix J. Analog Transmitter Behavioral Description for Black Box

```
// analog_transmitter
// This block is a behavioural model for the
// analogue front end of the Serializer
// transforms a CMOS Single Ended
// signal into a differential signal with programmable amplitude, impedance
// and pre-emphasis.
//NOTE: Eq_A, Eq_b, Imp_A, Imp_B, Imp_C, Imp_D, Amp_A, Amp_B are not implemented
//      in this module since they don't have behavioural digital consequences.
//For convention signals starting with "a" are the LSB of their configuration
//signals
s
module analog_transmitter(
    Data,
    Random,
    Data_Selector,
    Eq_A_P,
    Eq_A_N,
    Eq_B_P,
    Eq_B_N,
    Imp_A_P,
    Imp_A_N,
    Imp_B_P,
    Imp_B_N,
    Imp_C_P,
    Imp_C_N,
    Imp_D_P,
    Imp_D_N,
    Amp_A_P,
    Amp_A_N,
    Amp_B_P,
    Amp_B_N,
    Direct_Out_P,
    Direct_Out_N,
    Trans_Out_N,
    Trans_Out_P
    );
    input Data;
    input Random;
    input Data_Selector;
    input Eq_A_P;
    input Eq_A_N;
    input Eq_B_P;
    input Eq_B_N;
    input Imp_A_P;
    input Imp_A_N;
    input Imp_B_P;
    input Imp_B_N;
    input Imp_C_P;
    input Imp_C_N;
    input Imp_D_P;
    input Imp_D_N;
    input Amp_A_P;
    input Amp_A_N;
    input Amp_B_P;
    input Amp_B_N;
    output Direct_Out_N;
    output Direct_Out_P;
    output Trans_Out_N;
    output Trans_Out_P;
    wire transmit_signal;

   //Implement MUX for choose Random when Data_Selector is 1, choose Data when
Data_Selector is 0.
    assign transmit_signal = Data_Selector ? Random:Data;

    assign Direct_Out_P = transmit_signal;
    assign Direct_Out_N = ~transmit_signal;
    assign Trans_Out_P = transmit_signal;
    assign Trans_Out_N = ~transmit_signal;

endmodule
```

Appendix K. Analysis View

```
# Version:1.0 MMMC View Definition File
# Do Not Remove Above Line
create_rc_corner \
    -name rcbc_tm40 \
    -T {-40} \
    -preRoute_res {1.0} \
    -preRoute_cap {1.0} \
    -preRoute_clkres {0.0} \
    -preRoute_clkcap {0.0} \
    -postRoute_res {1.0} \
    -postRoute_cap {1.0} \
    -postRoute_xcap {1.0} \
    -postRoute_clkres {0.0} \
    -postRoute_clkcap {0.0}
create_rc_corner \
    -name rcwc_t125 \
    -T {125} \
    -preRoute_res {1.0} \
    -preRoute_cap {1.0} \
    -preRoute_clkres {0.0} \
    -preRoute_clkcap {0.0} \
    -postRoute_res {1.0} \
    -postRoute_cap {1.0} \
    -postRoute_xcap {1.0} \
    -postRoute_clkres {0.0} \
    -postRoute_clkcap {0.0}
create_op_cond \
    -name op_cond_1v98_tm40 \
    -library_file
{/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_2011113
0/std_cell/v.20111130/synopsys/bc/PbcV198Tm40_STD_CELL_7RF.lib} \
    -P {180} \
    -V {1.98} \
    -T {-40}
create_op_cond \
    -name op_cond_1v62_t125 \
    -library_file
{/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_2011113
0/std_cell/v.20111130/synopsys/wc/PwcV162T125_STD_CELL_7RF.lib} \
    -P {180} \
    -V {1.62} \
    -T {125}
create_library_set \
    -name bc_1v98_tm40 \
    -timing
{/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_2011113
0/std_cell/v.20111130/synopsys/bc/PbcV198Tm40_STD_CELL_7RF.lib}
create_library_set \
    -name wc_1v62_t125 \
    -timing
{/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_2011113
0/std_cell/v.20111130/synopsys/wc/PwcV162T125_STD_CELL_7RF.lib}
create_constraint_mode \
    -name serdes_func \
    -sdc_files {../rc/outputs/serdes_m.sdc}
create_delay_corner \
    -name dc_rcbc_1v98_tm40 \
    -library_set {bc_1v98_tm40} \
    -opcond {op_cond_1v98_tm40} \
    -rc_corner {rcbc_tm40}
create_delay_corner \
    -name dc_wcbc_1v62_t125 \
    -library_set {wc_1v62_t125} \
    -opcond {op_cond_1v62_t125} \
    -rc_corner {rcwc_t125}
create_analysis_view \
    -name av_rcbc_1p98_tm40 \
    -constraint_mode {serdes_func} \
    -delay_corner {dc_rcbc_1v98_tm40}
create_analysis_view \
    -name av_rcwc_1p62_t125 \
```

```
    -constraint_mode {serdes_func} \
    -delay_corner {dc_wcbc_1v62_t125}
set_analysis_view \
    -setup {av_rcwc_1p62_t125} -hold {av_rcbc_1p98_tm40}
```

Appendix L. Mock Analog Cell LEF

```
MACRO analogue_receiver
    CLASS CORE ;
    FOREIGN analogue_receiver -0.28 -0.28 ;
    ORIGIN 0 0 ;
    SIZE 2.24 BY 3.36 ;
    SYMMETRY X Y ;
    SITE CORE ;
    PIN rxpcie_in_p
        DIRECTION INPUT ;
        ANTENNAMODEL OXIDE1 ;
        ANTENNAGATEAREA 0.16  LAYER M1  ;
        PORT
        LAYER M1 ;
        RECT  0 0 0.84 0.24 ;
        END
    END rxpcie_in_p
    PIN rxpcie_in_n
        DIRECTION INPUT ;
        ANTENNAMODEL OXIDE1 ;
        ANTENNAGATEAREA 0.16  LAYER M1  ;
        PORT
        LAYER M1 ;
        RECT  0 0.48 0.84 0.72 ;
        END
    END rxpcie_in_n
    PIN rxpcie_out
        DIRECTION OUTPUT ;
        ANTENNADIFFAREA 0.49  LAYER M1  ;
        PORT
        LAYER M1 ;
        RECT  1.68 0.24 2.52 0.48 ;
        END
    END rxpcie_out

    PIN GND!
        DIRECTION INOUT ;
        USE GROUND ;
        SHAPE ABUTMENT ;
        PORT
        LAYER M1 ;
        RECT  0 4.32 0.84 4.56 ;
        END
    END GND!
    PIN VDD!
        DIRECTION INOUT ;
        USE POWER ;
        SHAPE ABUTMENT ;
        PORT
        LAYER M1 ;
        RECT  0 5.04 0.84 5.28 ;
        END
    END VDD!
    OBS
        LAYER PC ;
        RECT  0.16 3.06 0.34 3.56 ;
        RECT  0.20 1.58 0.38 1.98 ;
        RECT  0.62 1.58 0.80 1.98 ;
        RECT  0.86 3.06 1.04 3.56 ;
    END

END analogue_receiver

MACRO analog_transmitter
    CLASS CORE ;
    FOREIGN analog_transmitter -0.28 -0.28 ;
    ORIGIN 0 0 ;
    SIZE 4.48 BY 6.72 ;
    SYMMETRY X Y ;
    SITE CORE ;
    PIN Data
        DIRECTION INPUT ;
```

```
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 0 0.84 0.24 ;
    END
END Data
PIN Random
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 0.48 0.84 0.72 ;
    END
END Random
PIN Data_Selector
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 0.96 0.84 1.2 ;
    END
END Data_Selector
PIN Eq_A
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 1.44 0.84 1.68 ;
    END
END Eq_A
PIN Eq_B
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 5.28 0.84 5.52 ;
    END
END Eq_B
PIN Imp_A
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 1.92 0.84 2.16 ;
    END
END Imp_A
PIN Imp_B
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 2.4 0.84 2.64 ;
    END
END Imp_B
PIN Imp_C
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 2.88 0.84 3.12 ;
    END
END Imp_C
PIN Imp_D
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
```

```
    PORT
    LAYER M1 ;
    RECT  0 3.36 0.84 3.6 ;
    END
END Imp_D
PIN Amp_A
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 3.84 0.84 4.08 ;
    END
END Amp_A
PIN Amp_B
    DIRECTION INPUT ;
    ANTENNAMODEL OXIDE1 ;
    ANTENNAGATEAREA 0.16  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  0 4.32 0.84 4.56 ;
    END
END Amp_B


PIN Direct_Out_P
    DIRECTION OUTPUT ;
    ANTENNADIFFAREA 0.49  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  1.68 0 2.52 0.24 ;
    END
END Direct_Out_P

PIN Direct_Out_N
    DIRECTION OUTPUT ;
    ANTENNADIFFAREA 0.49  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  1.68 0.48 2.52 0.72 ;
    END
END Direct_Out_N
PIN Trans_Out_P
    DIRECTION OUTPUT ;
    ANTENNADIFFAREA 0.49  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  1.68 0.96 2.52 1.2 ;
    END
END Trans_Out_P
PIN Trans_Out_N
    DIRECTION OUTPUT ;
    ANTENNADIFFAREA 0.49  LAYER M1  ;
    PORT
    LAYER M1 ;
    RECT  1.68 1.44 2.52 1.68 ;
    END
END Trans_Out_N



PIN GND!
    DIRECTION INOUT ;
    USE GROUND ;
    SHAPE ABUTMENT ;
    PORT
    LAYER M1 ;
    RECT  0 4.80 0.84 5.04 ;
    END
END GND!
PIN VDD!
    DIRECTION INOUT ;
    USE POWER ;
    SHAPE ABUTMENT ;
```

```
        PORT
        LAYER M1 ;
        RECT  0 5.04 0.84 5.28 ;
        END
    END VDD!
    OBS
        LAYER PC ;
        RECT  0.16 3.06 0.34 3.56 ;
        RECT  0.20 1.58 0.38 1.98 ;
        RECT  0.62 1.58 0.80 1.98 ;
        RECT  0.86 3.06 1.04 3.56 ;
    END
END analog_transmitter
```

Appendix M. Globals Definition

```
# Serdes Globals
set defHierChar {/}
set delaycal_input_transition_delay {0.1ps}
set fpIsMaxIoHeight 0
set init_gnd_net {GND}
set init_lef_file
{/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_2011113
0/std_cell/v.20111130/lef/cmos7rf_6AM_tech.lef
/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_20111130
/std_cell/v.20111130/lef/ibm_cmos7rf_sc_12Track.lef
../lef/ibm_cmos7rf_analog_cells_serdes.lef}
set init_mmmc_file {serdes_mmmc.tcl}
set init_oa_search_lib {}
set init_pwr_net {VDD}
set init_verilog {../rc/outputs/serdes_m.v}
set lsgOCPGainMult 1.000000
```

Appendix N. Clock.ctstch

```
################################################################
#  Generated by:      Cadence Encounter 13.17-s018_1
#  OS:                Linux x86_64(Host ID fv00)
#  Generated on:      Sun Nov 22 19:58:55 2015
#  Design:            serdes
#  Command:           createClockTreeSpec -bufferList {CLKI_I CLKI_K CLKI_M ...
################################################################
#
# Encounter(R) Clock Synthesis Technology File Format
#

#-- MacroModel --
#MacroModel pin <pin> <maxRiseDelay> <minRiseDelay> <maxFallDelay> <minFallDelay>
<inputCap>


#-- Special Route Type --
RouteTypeName specialRoute
TopPreferredLayer 4
BottomPreferredLayer 3
PreferredExtraSpace 1
End

#-- Regular Route Type --
#RouteTypeName regularRoute
#TopPreferredLayer 4
#BottomPreferredLayer 3
#PreferredExtraSpace 1
#End

#-- Clock Group --
#ClkGroup
#+ <clockName>


#-------------------------------------------------------------
# Clock Root   : ref_clk
# Clock Name   : ref_clk
# Clock Period : 0.2ns
# Clock Name   : ref_clk
# Clock Period : 0.2ns
#-------------------------------------------------------------
AutoCTSRootPin ref_clk
Period         0.2ns
SrcLatency     0.02ns # set_clock_latency -source
MaxDelay       0.01ns # sdc driven default
MinDelay       0ns # sdc driven default
MaxSkew        8ps # sdc driven default
SinkMaxTran    110ps # set_clock_transition
BufMaxTran     110ps # set_clock_transition
Buffer         CLKI_I CLKI_K CLKI_M CLKI_O CLKI_Q CLK_I CLK_K CLK_M CLK_O CLK_Q
MaxFanout      4
ForceMaxFanout Yes
NoGating       NO
DetailReport   YES
#SetDPinAsSync  NO
#SetIoPinAsSync NO
#SetASyncSRPinAsSync  NO
#SetTriStEnPinAsSync NO
#SetBBoxPinAsSync NO
RouteClkNet    YES
PostOpt        YES
OptAddBuffer   YES
#RouteType      specialRoute
#LeafRouteType  regularRoute
END
```

Appendix O. EDI RTL to layout automated physical synthesis script

```
set_global _enable_mmmc_by_default_flow      $CTE::mmmc_default
win
source serdes.globals
init_design

# Define a 180 nm process node.
setDesignMode -process 180

# Floorplan specification using 20 microns for IO pad ring.
set tiny_chip_size 1500.0
set io_pad_ring 20.0
#     option a) Specify using aspect ratio of 1 (the tiny chip is a squere)
#               Mosis recommends core utilization 0.75
#floorPlan -site CORE -r 1.0 0.5 \
#    $io_pad_ring $io_pad_ring $io_pad_ring $io_pad_ring
#     option b) Specify floorplan using die size of 1.5mm by 1.5 mm
floorPlan -site CORE -d $tiny_chip_size $tiny_chip_size \
    $io_pad_ring $io_pad_ring $io_pad_ring $io_pad_ring

# Define power nets:
globalNetConnect VDD! -type pgpin -pin VDD! -all
globalNetConnect GND! -type pgpin -pin GND! -all
globalNetConnect VDD! -type tiehi
globalNetConnect GND! -type tielo

addRing\
    -stacked_via_top_layer AM\
    -around core\
    -jog_distance 0.28\
    -threshold 0.28\
    -nets {GND! VDD!}\
    -stacked_via_bottom_layer M1\
    -layer {bottom MT top MT right AM left AM}\
    -width 2\
    -spacing 5\
    -offset 0.28

addStripe\
    -block_ring_top_layer_limit AM\
    -max_same_layer_jog_length 4\
    -padcore_ring_bottom_layer_limit MT\
    -set_to_set_distance 20\
    -stacked_via_top_layer AM\
    -padcore_ring_top_layer_limit AM\
    -spacing 5\
    -merge_stripes_value 0.28\
    -layer AM\
    -block_ring_bottom_layer_limit MT\
    -width 2\
    -nets {GND! VDD!}\
    -stacked_via_bottom_layer M1\
    -break_stripes_at_block_rings 1
sroute\
    -connect { padRing corePin floatingStripe }\
    -layerChangeRange { M1 AM }\
    -blockPinTarget { nearestTarget }\
    -stripeSCpinTarget { blockring padring ring stripe ringpin blockpin }\
    -checkAlignedSecondaryPin 1\
    -allowJogging 1\
    -crossoverViaBottomLayer M1\
    -allowLayerChange 1\
    -targetViaTopLayer AM\
    -crossoverViaTopLayer AM\
    -targetViaBottomLayer M1\
    -nets { GND! VDD! }
setPlaceMode -reset
setPlaceMode \
    -congEffort auto \
    -timingDriven 1 \
    -modulePlan 1 \
    -clkGateAware 1 \
```

```
    -powerDriven 0 \
    -ignoreScan 1 \
    -reorderScan 0 \
    -ignoreSpare 1 \
    -placeIOPins 1 \
    -moduleAwareSpare 0 \
    -checkPinLayerForAccess {  1 } \
    -preserveRouting 0 \
    -rmAffectedRouting 0 \
    -checkRoute 0 \
    -swapEEQ 0
setPlaceMode -fp false
placeDesign -prePlaceOpt

# clock tree
clockDesign -specFile Clock.ctstch \
    -outDir clock_report \
    -fixedInstBeforeCTS
displayClockTree \
    -skew \
    -allLevel \
    -clkRouteOnly

#nanoroute
setNanoRouteMode -quiet -timingEngine {}
setNanoRouteMode -quiet -routeWithTimingDriven 1
setNanoRouteMode -quiet -routeWithSiDriven 1
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven true
setNanoRouteMode -quiet -routeWithSiDriven true
routeDesign -globalDetail
```