INSTITUTO TECNOLOGICO 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 de 29 de noviembre de 1976

DEPARTAMENTO DE ELECTRONICA, SISTEMAS E INFORMATICA

Serializer Design for a SerDes chip in 130nm CMOS Technology

Tesina para obtener el grado de:

Especialista en Diseño de Sistemas en Chip

Presenta:

Efraín Arrambide Barrón

Asesores:

Víctor Avendaño Fernández y Cuauhtémoc Aguilera

Guadalajara, Jalisco, Julio 2016

ACKNOWLEDGMENT

Thanks all my colleagues in the specialty program for all your support, time and help. To be more specific my friends R. Rivas, E. Conde, Cesar Limones and J. Nuñez.

A recognition for your dedication effort and advises to my mentors V. Avendaño. E. Juarez and C. Aguilera.

My gratitude to my mother for always support all my decisions and push me to acquire all my objectives and never give up.

My respects and gratitude to my brother for the economic and emotional support in each step of my life.

I want to thank the CONACYT for providing this opportunity to all the people who wants to keep studying and encourage the education and development programs.

ABSTRACT

This document presents the digital Serializer module description and development, which will be part of a SerDes system in 130nm CMOS technology for PCI Express protocol communication applications. The code of Serializer was written in Verilog description language, and the logical and physical synthesis were performed using the following tools, Cadence Encounter Digital Implementation System (EDI) and Encounter RTL compiler.

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1. Introduction

SerDes stands for Serializer/Deserializer. This is a high-speed communication circuit used to optimize and agile the communications between bus data. SerDes convert all the input parallel signals into one serial signal. This allows to reach higher speed and have advantages in performance and space implementation.

This document details the steps to develop a SerDes system on a chip. The SerDes will implement the PCIe generation 1 protocol communication at the speed of 1.25 Gbits/sec. The technology used is CMOS 130nm technology provide by IBM base libraries.

The SerDes system was divided in 5 main Digitals and Analogs blocks. The digital blocks are BIST, Serializer and Deserializer. The Analogs blocks are the Transmitter and Receiver. This document will explain in detail all about Serializer block.

Chapter 2 presents with detail each component of the SerDes system. This chapter provides information about the functionality, constraints, and interactions between each module.

Chapter 3 gives details about the Serializer architecture and RTL code description. This chapter explains the logical synthesis in RC compiler and describes the steps and configurations that were applied. Chapter 3 shows the results of timing, area, and power consumption testing.

Chapter 4 provides information of the test bench applied to the Serializer component. This chapter presents information of gate level and behavioral test simulations and timing analysis. Chapter 4 gives a description of the waveforms showing a proper respond to the stimulus applied and reports verifying the timing be right.

Finally, chapter 5 presents the physical synthesis in Cadence Encounter, this chapter describes the stages to perform all the final tests. This chapter describes the importation to Virtuoso and two important tests DRC and LVS and all the result and conclusion acquired.

Chapter 2 Background of a SerDes

2.1 SerDes description

SerDes (serializers/deserializer) are devices that can take several buses and convert these in a single ended signal bus. A differential signal that switches at a much higher frequency rate than the parallel buses. SerDes enables the movement of a large amount of data point to point while reducing the complexity, cost, power, and board space usage associated with having to implement wide parallel data buses. SerDes usage becomes especially beneficial as the frequency rate of parallel data buses moves beyond 500 MHz (1000 Mbps) [1].

Figure 1. Abstract representation of Serdes

At these higher-frequency rates, the problems associated with wide parallel buses are further aggravated. A faster switching parallel bus consumes more power and is much more difficult to route, given that timing tolerances are reduced [1].

2.2 SerDes protocol

PCI Express is a high performance, general purpose I/O interconnect defined for a wide variety of future computing and communication platforms. Key PCI attributes, such as its usage model, load and store architecture, and software interfaces, are maintained, whereas its parallel bus implementation is replaced by a highly scalable, fully serial interface. PCI Express takes advantage of recent advances in point to point interconnects, Switch-based technology, and packetized protocol to deliver new levels of performance and features. Power Management, Quality Of Service (QoS), Hot-Plug/HotSwap support, Data Integrity, and Error Handling are among some of the advanced features supported by PCI Express [2].

A Link represents a dual-simplex communications channel between two components. The fundamental PCI Express Link consists of two, low-voltage, differentially driven signal pairs: a Transmit and Receiver pair are shown in figure 2.

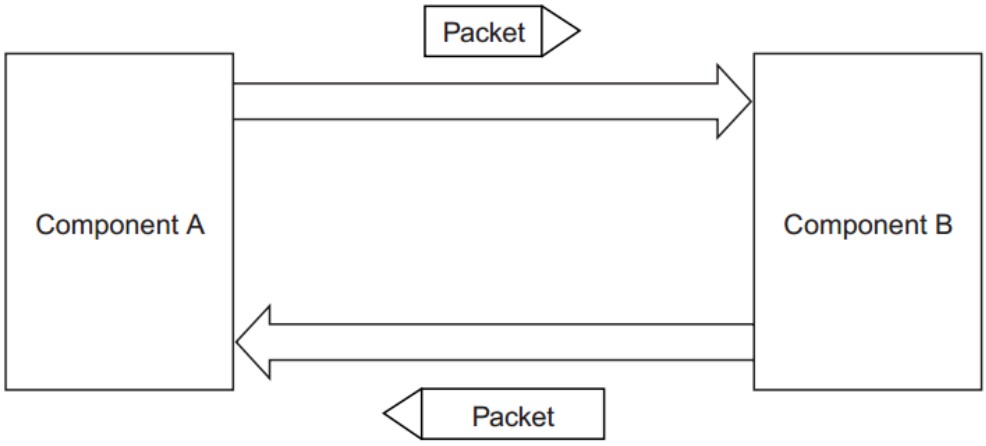


Figure 2. PCIe Link [2]

PCI Express uses packets to communicate information between components. Packets are formed in the Transaction and Data Link Layers to carry the information from the transmitting component to the receiving component. As the transmitted packets flow through the other layers, they are extended with additional information necessary to handle packets at those layers. At the receiving side the reverse process occurs and packets get transformed from their Physical Layer representation to the Data Link Layer representation and finally (for Transaction Layer Packets) to the form that can be processed by the Transaction Layer of the receiving device [2].

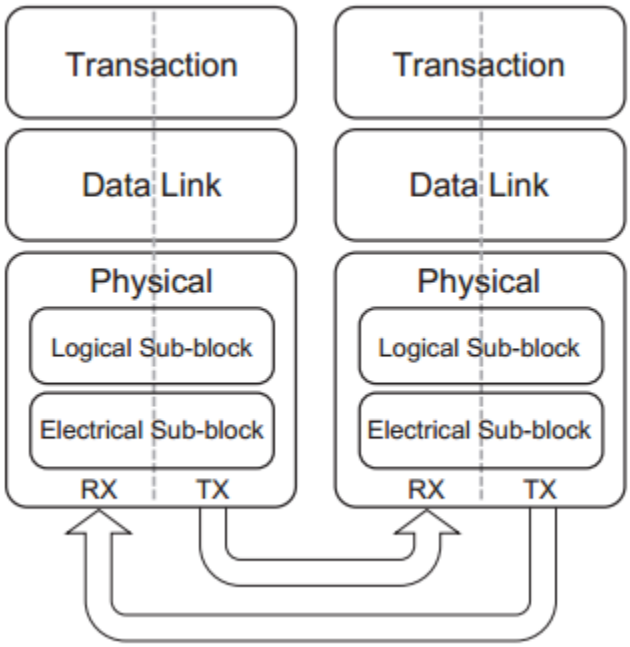


Figure 3. PCIe Transaction, Data Link and Physical layers [2]

The Transaction Layer's primary responsibility is the assembly and disassembly of Transaction Layer Packets (TLPs). TLPs are used to communicate transactions, such as read and write, as well as certain types of events. The Transaction Layer is also responsible for managing credit-based flow control for TLPs [2].

Every request packet requiring a response packet is implemented as a split transaction. Each packet has a unique identifier that enables response packets to be directed to the correct originator. The packet format supports different forms of addressing depending on the type of the transaction (Memory, I/O, Configuration, and Message). The Packets may also have attributes such as No Snoop and Relaxed Ordering [2].

The transaction Layer supports four address spaces: it includes the three PCI address spaces (memory, I/O, and configuration) and adds a Message Space. This specification uses Message Space to support all prior sideband signals, such as interrupts, power-management requests, and so on, as in-band Message transactions. You could think of PCI Express Message transactions as “virtual wires” since their effect is to eliminate the wide array of sideband signals currently used in a platform implementation [2].

The Data Link Layer serves as an intermediate stage between the Transaction Layer and the Physical Layer. The primary responsibilities of the Data Link Layer include Link management and data integrity, including error detection and error correction. The transmission side of the Data Link Layer accepts TLPs assembled by the Transaction Layer, calculates and applies a data protection code and TLP sequence number, and submits them to Physical Layer for transmission across the Link. The receiving Data Link Layer is responsible for checking the integrity of received TLPs and for submitting them to the Transaction Layer for further processing. On detection of TLP error(s), this Layer is responsible for requesting retransmission of TLPs until information is correctly received, or the Link is determined to have failed [2].

The Data Link Layer also generates and consumes packets that are used for Link management functions. To differentiate these packets from those used by the Transaction Layer (TLP), the term Data Link Layer Packet (DLLP) will be used when referring to packets that are generated and consumed at the Data Link Layer [2].

The Physical Layer includes all circuitry for interface operation, including driver and input buffers, parallel-to-serial and serial-to-parallel conversion, PLL(s), and impedance matching circuitry. It includes also logical functions related to interface initialization and maintenance. The Physical Layer exchanges information with the Data Link Layer in an implementation-specific format. This Layer is responsible for converting information received from the Data Link Layer into an appropriate serialized format and transmitting it across the PCI Express Link at a frequency and width compatible with the device connected to the other side of the Link. The PCI Express architecture has “hooks” to support future performance enhancements via speed upgrades and advanced encoding techniques. The future speeds, encoding techniques or media may only impact the Physical Layer definition [2].

Four transaction type by the transactions layer:

1. Memory Read or memory write. Use to transfer data from of to a memory mapped location.
2. I/O read or write. Use to transfer data from or to an I/O location.
3. Configuration read or write. Use to discover devices capabilities, program features, and check status in the 4KB PCI Express configuration space.
4. Messages. Handled like posted writes. Used for event signaling and general purpose messaging.

2.3 SerDes 8b/10b encryption

The first two generations of PCIe use 8b/10b encoding. Each Lane implements an 8b/10b Encoder that translates the 8-bits characters into 10-bits Symbols. This coding scheme was patented by IBM in 1984 and is widely used in many serial transports today, such as Gigabit Ethernet and Fiber Channel [3].

Encoding accomplishes several desirable goals for serial transmission. Three of the most important are listed here:

Embedding a Clock into the Data. Encoding ensures that the data stream has enough edges in it to recover a clock at the Receiver, with the result that a distributed clock is not needed. This avoids some limitations of a parallel bus design such as flight time and clock skew. It also eliminates the need to distribute a high/frequency clock that would cause other problems like increased electro-magnetic interference and difficult routing [3].

Maintaining DC Balance. PCIe uses an AC-couple link, placing a capacitor serially in the path to isolate the DC part of the signal from the other end of the Link. This allows the Transmitter and Receiver to use different common-mode voltages and makes the electrical design easier for cases where the path between them is long enough that they are less likely to have exactly the same reference voltages. That DC value, or common-mode voltage, can change during runtime because the line charges up when the signal is driven [3].

The 8b/10b encoder tracks the disparity of the last Symbol that was sent. The disparity, or inequality, simply indicates whether the previous Symbol had more ones that zeros (called positive disparity), more zeros than ones (negative disparity), or balance of ones and zeros (neutral disparity). If the previous Symbol had negative disparity, for example, the next one should balance that by using more ones [3].

Enhancing Error Detection. The encoding scheme also facilitates the detections of transmission errors. For a 10-bit value, 1024 codes are possible, but the character to be encoded only has 265 unique codes. To maintain the DC balance the design uses two codes for each character, and chooses which one base on the disparity of the las Symbol that was sent, so 512 codes would be needed. However, many of the neutral disparity encodings have the same values, so not all the 512 are used. As a result, more than half the possible encodings are not used and will be considered illegal if seen at a Receiver. If a transmission error does change the bit pattern of a Symbol, there is a good chance the result would be one of these illegal patterns that can be recognized right away [3].

The major disadvantage of 8b/10b encoding is the overhead it requires. The actual transmission performance is degraded by 20% from the Receiver's point of view because 10 bits are sent for each byte, but only 8 useful bits are recovered at the receiver. This is a non-trivial price to pay but is still considered acceptable to gain the advantages mentioned [3].

The figure 4 displays a new outgoing Symbol is created a base on three things: the incoming character, the D/K# indicator for that character, and the current run disparity. A new current run disparity value is computed based on the outgoing Symbol and is fed back for use in encoding the next character. After the encoding, the resulting Symbol is fed to a serializer that clocks out the individual bits [3].

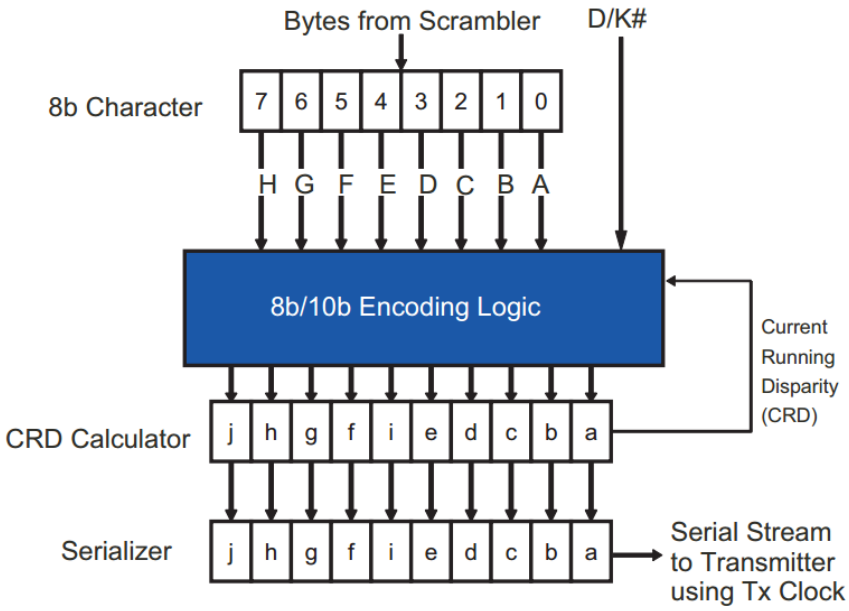


Figure 4. 8b/10b encoding scheme [3]

The 8b/10b encoding scheme used by PCI Express provides Special Symbols that are distinct from the Data Symbols used to represent Characters. These Special Symbols are used for various Link Management mechanisms described later in this chapter. Special Symbols are also used to frame DLLPs and TLPs, using distinct Special Symbols to allow these two types of Packets to be quickly 5 and easily distinguished [2].

Encoding	Symbol	Name	Description
K28.5	COM	Comma	Used for Lane and Link initialization and management
K27.7	STP	Start TLP	Marks the start of a Transaction Layer Packet
K28.2	SDP	Start DLLP	Marks the start of a Data Link Layer Packet
K29.7	END	End	Marks the end of a Transaction Layer Packet or a Data Link Layer Packet
K30.7	EDB	EnD Bad	Marks the end of a nullified TLP
K23.7	PAD	Pad	Used in Framing and Link Width and Lane ordering negotiations
K28.0	SKP	Skip	Used for compensating for different bit rates for two communicating Ports
K28.1	FTS	Fast Training Sequence	Used within an ordered set to exit from L0s to L0
K28.3	IDL	Idle	Symbol used in the Electrical Idle ordered set

Figure 5. Special Symbol used by PCIe protocol [2]

2.4 SerDes block diagram

Besides taking the 10 bit parallel data from the PCIe transmission, the Serializer can also generate its own parallel test data from the internal BIST (built-in self-test) pattern generator. Depending on the design, a variety of BIST patterns can be generated to not only facilitate the debugging and testing of the SerDes design but also to enhance the overall system level diagnostic capability.

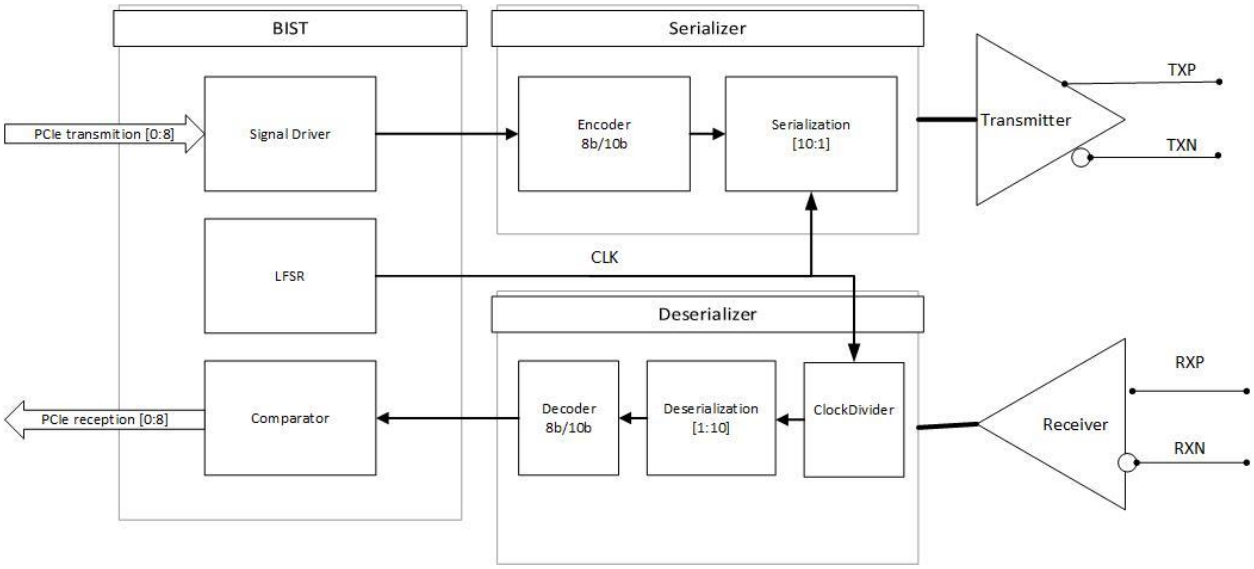


Figure 6. PCIe SerDes block diagram [4]

The Digital Serializer converts the 10 bit parallel data into a serial data stream. The conversion is done with the clocks generated from the transmit clock generator.

The Digital Deserializer sampling a signal coming from the Analog Receiver, this is synchronized with the clock thanks to ClockDivider module after this stage, the signal is stored in the registers till 10 cycles of the main clock, to then be send to the decoder and finally the Decoder send the data information for the PCIe protocol traffic.

The Analog Transmitter includes a driver strength selection and an equalization circuit to implement pre-emphasis and de-emphasis to account for channel loss.

The Analog Receiver aims to compensate for the attenuation in amplitude experienced by the transmission of the serial data through the communication channel. It also has a bias circuit to achieve compensation of voltage and temperature.

The BIST (Buil-In Self-Test) approach is to eliminate or reduce the need for an external tester by integrating active test infrastructure onto the chip.

2.5 SerDes specification

The SerDes specifications and implementations tools are defined as the following items:

- ARM Manufacturing technology for CMOS 130nm
- DIP40 packaging provides by MOSIS.
- Communication protocol PCIe 1.0 (8b/10b decoding and frequency up to 1.25 GHz)
- Operation frequency 123 Mhz (SerDes development)
- Area Restrictions 1.5mm x1.5mm provided by MOSIS
- VDD voltage 1.2v
- Cadence RC compiler (Logical Synthesis)
- Cadence Encounter(Physical Synthesis)
- Cadence Virtuoso(Schematic and Layout analysis and simulation)
- Model Sim-Altera 10.4d (RTL simulation and compilation tool)

Chapter 3. Serializer Architecture

3.1 Serializer top diagram architecture

The digital Serializer module architecture consists of 3 modules: the Encoder, the Serializer, and the top Serial Transmitter block.

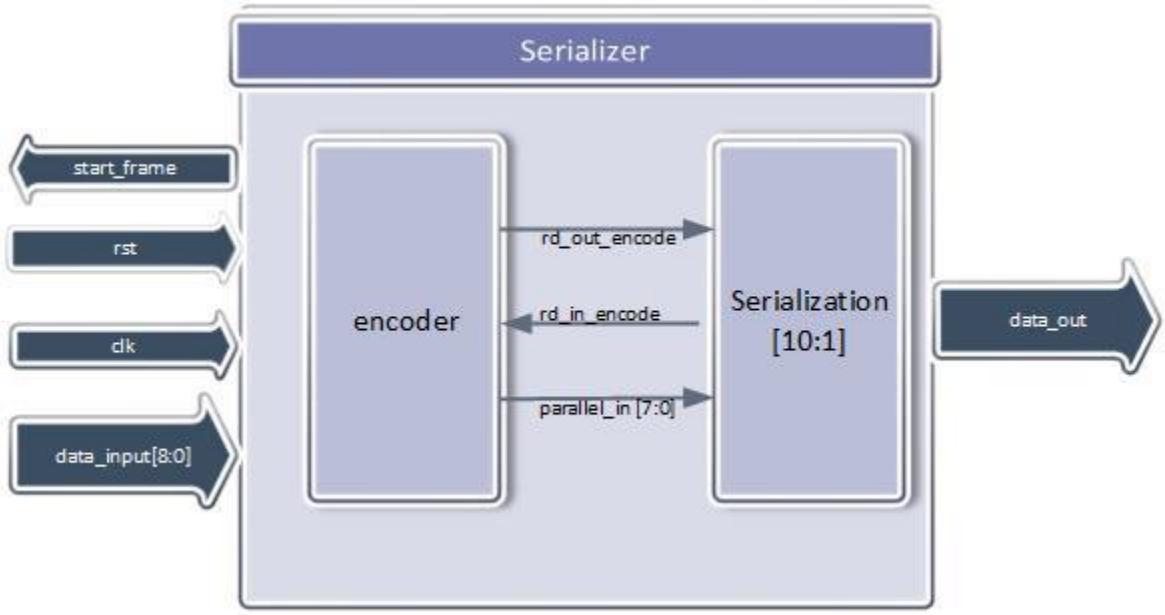


Figure 7. Serializer diagram

3.2 Serializer

The Serializer module transforms the data information received in parallel to a serial data stream. The incoming data is processed by the Encoder which generates a 10 bits output. When the encoding process is completed, the module sends the serial data at the rise clock edge. The input K bit takes action for the control symbols, for instance the coma character [5].

3.3 Encoder

The Encoder is a combinational module used to generate an encoded 10-bit output. The Encoder generates the same bit amount of 1's and 0's in order to get a balanced serial data stream. The combinations are limited and therefore it cannot have always the same quantity of 1s and 0s. The Encoder overcomes this restriction by implementing a Run Disparity bit which helps to control the number of 1's or 0's. If the Run Disparity is -1, it means that there are 6 zeros and 4 ones; when

the Run Disparity is +1, it means that there are 6 ones and 4 zeros; finally, when the Run Disparity is 0, it means that we have a balanced stream 5 zeros and 5 ones.

Furthermore, this helps the analog transmitter to balance the power consumption, and avoid the desynchronization: it is not allowed to transmit more than 3 ones or 3 zeros consecutively [5].

3.4 Serialization

The Serialization is a sequential module responsible for storing 10 bits. It has three input and two output ports: data_input, clock, reset, data_out and start_frame. The reset is an active high pin, and it clears all the information stored in the registers. The clock signal helps to synchronize the transmission to the analog side.

Figure 8 display the pin description of Serializer module.

Name	**I/O Direction**	**Description**
start_frame	output	Active high signal to synchronization with the PCIe protocol, when this in enable the Serializer is ready to receive the data from the pin data_input.
Rst	input	Active high asynchronous reset, this pin is common for all the components of the chip.
Clk	input	Transmission clock coming from the clock generator module 125 Mhz.
data_input [8:0]	Input	Data stream coming from the PCIe protocol waiting to be encoded.
data_out	output	Data bit transmitted to the analog transmitter.
Internal Signals	**I/O Direction**	**Description**
rd_out_encode	Encoder to Serialization	Run disparity out a bit, it saves during the stream transmitting to the next encoding stream.
rd_in_encode	Serialization to Encoder	Run disparity in a bit, it is returned by serialization module, the main function of it creates a balance stream data.
parallel_in[7:0]	Encoder to Serialization	This is encoded data ready to be transmitted by Serialization model.

Figure 8. Pin description

3.5 Encoder Architecture

Figure 9 defines the Encoder architecture. This block is divided into two main sub-blocks the 5B/6B and 3B/4B Encoder. The first one has as an input the less significant bits of the parallel data. Those bits are converted into six bits balanced according to the disparity. The decoder bits are ordered and labeled in the following order: "abcdei".

The 3B/4B Encoder has as an input the three most significant bits of the parallel data. Those bits are converted into four bits and balanced according to the disparity. The decoder bits are ordered and labeled in the following order: "fghj"

Finally, the Encoder merges all the encoded bits to create a stream in the following order: abcdeifgj". The K bit is used to generate control symbols that will be explained in more detail in the next paragraph.

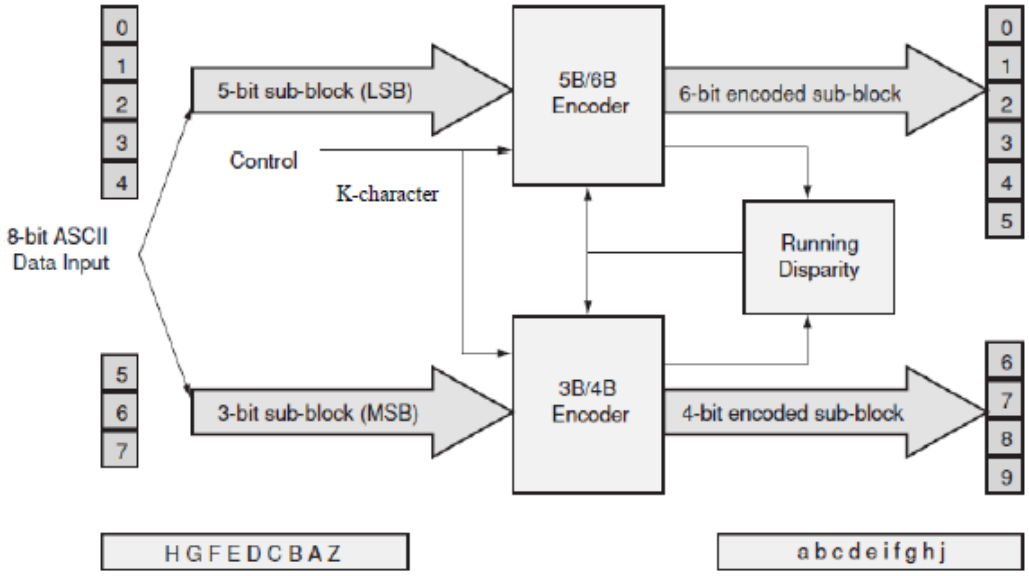


Figure 9. Encoder Architecture block diagram [5]

3.5.2 Encoding tables

The input data is divided and encoded according to the tables shown in figure 3. The table 5B/6B is showing how the output stream bits will be translated if the Run Disparity is positive or negative. The table 3B/4B explains how the output stream bits will be translated in the case that the Run Disparity is positive or negative. Both tables ensure the uniqueness of the special bit sequence depending on the disparity bit and the K-bit [6].

According to the Figure 10. The rules for Running Disparity (RD) are: if the previous RD is a -1 and the disparity of stream bits is 0, this means that the RD bit does not change. If the disparity of stream bits is +2 and the previous RD is -1, the RD became in +1 value. If the previous RD is a +1 and the disparity of stream bits is +2 that means that the RD bit does not change, but if the disparity of stream bits is positive, the RD became in -1 value.

Rules for running disparity

Previous RD	Disparity of code word	Disparity chosen	Next RD
−1	0	0	−1
−1	±2	+2	+1
+1	0	0	+1
+1	±2	−2	−1

5b/6b code

Input		RD = −1	RD = +1
	EDCBA	abcdei	
D.00	00000	100111	011000
D.01	00001	011101	100010
D.02	00010	101101	010010
D.03	00011	110001	
D.04	00100	110101	001010
D.05	00101	101001	
D.06	00110	011001	
D.07	00111	111000	000111
D.08	01000	111001	000110
D.09	01001	100101	
D.10	01010	010101	
D.11	01011	110100	
D.12	01100	001101	
D.13	01101	101100	
D.14	01110	011100	
D.15	01111	010111	101000

Input		RD = −1	RD = +1
	EDCBA	abcdei	
D.16	10000	011011	100100
D.17	10001	100011	
D.18	10010	010011	
D.19	10011	110010	
D.20	10100	001011	
D.21	10101	101010	
D.22	10110	011010	
D.23 †	10111	111010	000101
D.24	11000	110011	001100
D.25	11001	100110	
D.26	11010	010110	
D.27 †	11011	110110	001001
D.28	11100	001110	
D.29 †	11101	101110	010001
D.30 †	11110	011110	100001
D.31	11111	101011	010100
K.28	11100	001111	110000

3b/4b code

Input		RD = −1	RD = +1
	HGF	fghj	
D.x.0	000	1011	0100
D.x.1	001	1001	
D.x.2	010	0101	
D.x.3	011	1100	0011
D.x.4	100	1101	0010
D.x.5	101	1010	
D.x.6	110	0110	
D.x.P7 †	111	1110	0001
D.x.A7 †	111	0111	1000

Input		RD = −1	RD = +1
	HGF	fghj	
K.x.0	000	1011	0100
K.x.1 ‡	001	0110	1001
K.x.2 ‡	010	1010	0101
K.x.3 ‡	011	1100	0011
K.x.4	100	1101	0010
K.x.5 ‡	101	0101	1010
K.x.6 ‡	110	1001	0110
K.x.7 †	111	0111	1000

Figure 10. Tables for 5b/6b and 3b/4b encoder [4]

3.5.1 Control symbols

The control symbols within 8b/10b are 10b symbols that are valid sequences of bits (no more than six 1s or 0s) but do not have a corresponding 8b data byte. They are used for low-level control functions. For instance, in Fiber Channel, K28.5 is used at the beginning of four-byte sequences (called "Ordered Sets") that perform functions such as Loop Arbitration, Fill Words, Link Resets, etc. [3].

The following 12 control symbols are allowed to be sent:

Control symbols

Input				RD = −1	RD = +1
	DEC	HEX	HGF EDCBA	abcdei fghj	abcdei fghj
K.28.0	28	1C	000 11100	001111 0100	110000 1011
K.28.1 †	60	3C	001 11100	001111 1001	110000 0110
K.28.2	92	5C	010 11100	001111 0101	110000 1010
K.28.3	124	7C	011 11100	001111 0011	110000 1100
K.28.4	156	9C	100 11100	001111 0010	110000 1101
K.28.5 †	188	BC	101 11100	001111 1010	110000 0101
K.28.6	220	DC	110 11100	001111 0110	110000 1001
K.28.7 ‡	252	FC	111 11100	001111 1000	110000 0111
K.23.7	247	F7	111 10111	111010 1000	000101 0111
K.27.7	251	FB	111 11011	110110 1000	001001 0111
K.29.7	253	FD	111 11101	101110 1000	010001 0111
K.30.7	254	FE	111 11110	011110 1000	100001 0111

Figure 11. Control symbols table for K character [4]

Chapter 4. Serializer Logical synthesis

4.1 Logical Synthesis of Serializer

Table 12 displays the Chip_Serializer top module, this module implement the PADS which are necessary for getting a better timing accuracy during the physical synthesis.

The integration of the SerDes considered that the Serializer module has 10 pins with the external modules, 9 pins will be connected to the protocol PCIe parallel data and one pin will be connected to the analog transmitter. To have an accuracy timing analysis is necessary implement these pads for the logical and physical synthesis.

To have a symmetrical chip, it is necessary to calculate the number of pads if these are not enough, it is required to use dummy pads, this module is using three dummy pads.

```
module Serializer(VDD, VSS, DVDD, DVSS, clk, rst, data_in, data_out, frame, dummy_in1, dummy_in2,
dummy_in3);
input      VDD;
input      VSS;
input      DVDD;
input      DVSS;
input      clk;
input                  rst;
input [8:0] data_in;
input      dummy_in1;
input      dummy_in2;
input                  dummy_in3;
output data_out;
output frame;

wire rd_out_encode, rd_in_encode;
wire [9:0] dataout;
wire [8:0] data_in_w;
wire clk_w, rst_w, dout_w;
wire start_frame;

// Addding power pads
PDVDD pad_DVDD(.DVDD (DVDD));
PDVSS pad_DVSS(.DVSS (DVSS));
PVDD  pad_VDD(.VDD (VDD));
PVSS  pad_VSS(.VSS (VSS));

// Adding input pads
PIC pad_clk(.P(clk), .IE(VDD), .Y(clk_w));
PIC pad_rst(.P(rst), .IE(VDD), .Y(rst_w));
PIC pad_data_in0(.P(data_in[0]), .IE(VDD), .Y(data_in_w[0]));
PIC pad_data_in1(.P(data_in[1]), .IE(VDD), .Y(data_in_w[1]));
PIC pad_data_in2(.P(data_in[2]), .IE(VDD), .Y(data_in_w[2]));
PIC pad_data_in3(.P(data_in[3]), .IE(VDD), .Y(data_in_w[3]));
PIC pad_data_in4(.P(data_in[4]), .IE(VDD), .Y(data_in_w[4]));
PIC pad_data_in5(.P(data_in[5]), .IE(VDD), .Y(data_in_w[5]));
PIC pad_data_in6(.P(data_in[6]), .IE(VDD), .Y(data_in_w[6]));
PIC pad_data_in7(.P(data_in[7]), .IE(VDD), .Y(data_in_w[7]));
PIC pad_data_in8(.P(data_in[8]), .IE(VDD), .Y(data_in_w[8]));
```

```
// Adding output pads
POC4C pad_dout  (.A(dout_w), .P(data_out));
POC4C pad_frame (.A(start_frame), .P(frame));

//Adding dummy pads
PIC pad_dummy1(.P (dummy_in1), .IE (VSS), .Y (VSS));
PIC pad_dummy2(.P (dummy_in2), .IE (VSS), .Y (VSS));
PIC pad_dummy3(.P (dummy_in3), .IE (VSS), .Y (VSS));

encode encode1(.datain(data_in_w),
          .dispin(rd_in_encode),
          .dataout(dataout),
          .dispout(rd_out_encode));

Serialization Serialization(.par_in(dataout),
                .clk(clk_w),
                .rst(rst_w),
                .rd_in(rd_out_encode),
                .rd_out(rd_in_encode),
                .ser_out(dout_w),
                .start_frame(start_frame));
Endmodule
```

Figure 12. Pads and instances of Serializer

4.2 Clock definition

There are several constraints, one of the most important is the clock speed. It is limited by the standards cells speed. The fastest speed that the standards cells can handle is 1Ghz. The receptor defines this constraint, this is using the fastest clock of 1Ghz and it is divide 4 times. The receptor is implementing a module called clock_divider, this module split the clock signal of 1Gz into 4 signals at the speed of 125Mhz. The motivation to use this 125Mhz is because the Deserializer cannot handle the speed of 250Mhz, to solve this problem. Our SerDes is sampling 4 bit of the Stream. Finally, this was define:

```
define_clock -name 125MHz_CLK -period 8000 [get_ports clk]
```

4.3 Logical synthesis timing

Setup and hold times: Setup time is the amount of time that data on the parallel data bus must be stable before it can be clocked into the parallel register. Hold time is the amount of time that the data must remain valid after being clocked into the parallel register. Set/hold time violations are a common cause of implementation issues for SerDes.

Rise and fall times: Commonly referred to as the signal edge rate. Rise and fall times are commonly specified for the serial I/O as a measure of serial switching performance. A rule of thumb is that the rise or fall time should be no greater than 25 to 30 percent of the unit interval for a given data rate of operation.

Figure 13 displays the worst slack in the rc.log. The critical path reported positive slack of 191 ps.

The Serializer's critical path is passing through the start_frame pin and the counter register.

```
Cost Group 'C2O' target slack:   191 ps
Target path end-point (Port: Chip_Serial_Transmisor/frame)

Warning : Possible timing problems have been detected in this design. [TIM-11]
      : The design is 'Chip_Serial_Transmisor'.
       Pin                  Type        Fanout Load Arrival
                                         (fF)   (ps)
-------------------------------------------------------------------------
(clock 125MHz_CLK)     <<<  launch                          0 R
                       latency
Serializador1
  cb_seqi
    counter_reg[1]/clk
    counter_reg[1]/q   (u)  unmapped_d_flop       5 31.0
  cb_seqi/g58_in_1
  cb_oseqi/cb_seqi_g58_in_1
    g167/in_1
    g167/z             (u)  unmapped_or2          2 12.4
    g169/in_0
    g169/z                  unmapped_nor2         1 34.8
  cb_oseqi/start_frame
Serializador1/start_frame
pad_frame/A
pad_frame/P             (P)  POC4C                1 40.0
frame                  <<<  out port
(ODelay)                    ext delay
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 125MHz_CLK)          capture                    8000 R
                       latency
                       uncertainty
-------------------------------------------------------------------------
```

Figure 13. Logical Synthesis timing analysis

Figure 14 shows the block diagram of the Serial Transmitter top module, the green rectangles are the Encoder and the Serializer, the other rectangles are pads to make the connection with the external signals of the chip die.

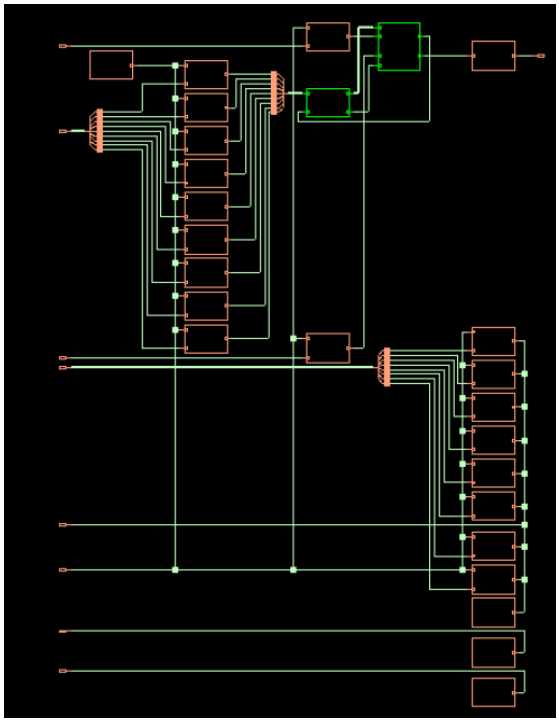


Figure 14. Block diagram of Serializer module

Figure 15 presents a block diagram of the Encoder, here there are highlighted gates that being part of a path with the worst slack.

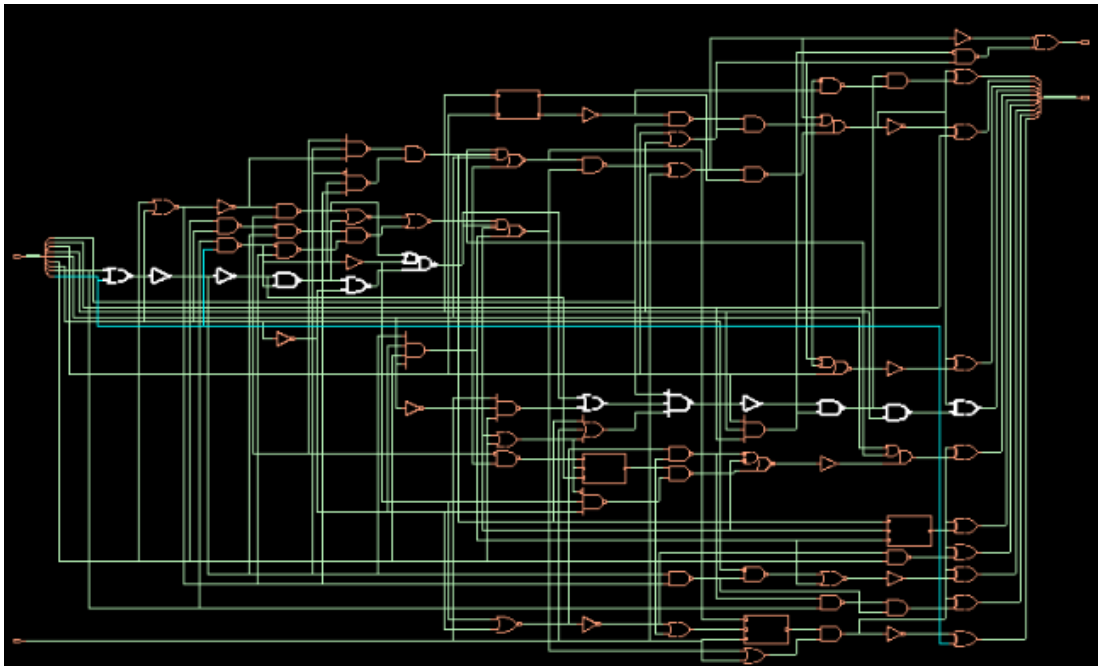


Figure 15. Block diagram of Encoder

Figure 16 shows the logical synthesis for the Serializer module, and the highlight gates are a worst slack path.

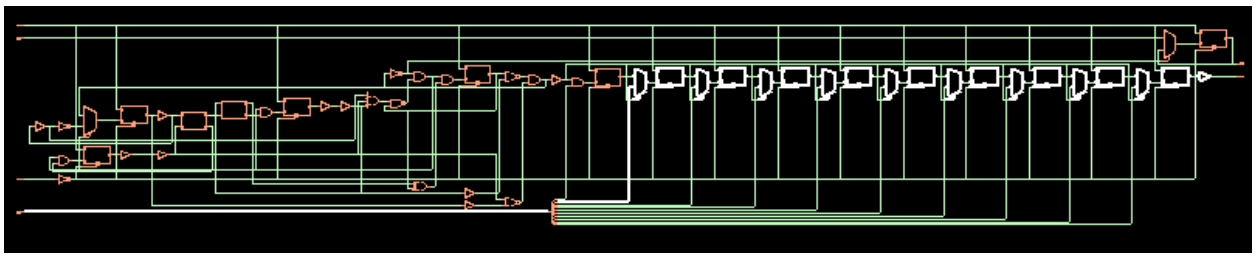
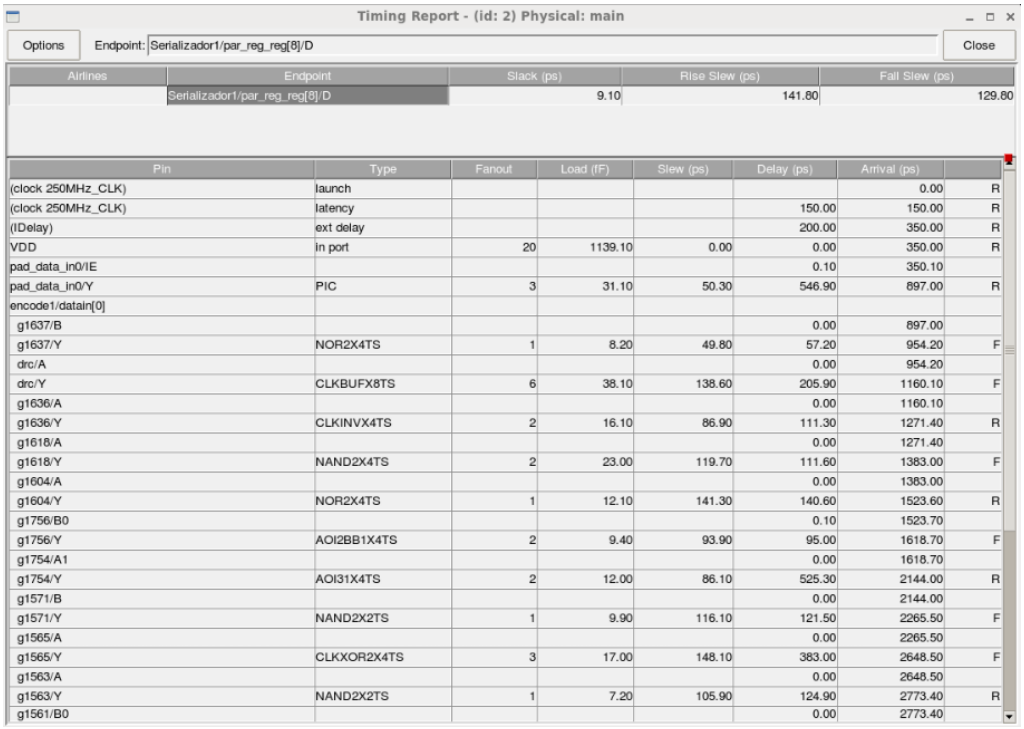


Figure 16. Block diagram of Encoder

According to the data presented in Figure 17, the Serializer module can work at the speed of 250 MHz, because it shows positive slack of 94ps, but for constrains in the Deserializer (receiver). The Serializer module will work at the speed of 125Mhz determinates by the clock divider module inside of Deserializer module.

The report indicates that the Encoder module produces more than 80% of the total slack. This slack can be improved if registers are allocated between the gates with the worst delay arrival (this can be implemented in the next revision of the project). The timing report log can be found in appendix F, and in Figure 17.



Timing Report - (id: 2) Physical: main

Options | Endpoint: Serializador1/par_reg_reg[8]/D | Close

Airlines	Endpoint	Slack (ps)	Rise Slew (ps)	Fall Slew (ps)
	Serializador1/par_reg_reg[8]/D	9.10	141.80	129.80

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)	
(clock 250MHz_CLK)	launch					0.00	R
(clock 250MHz_CLK)	latency				150.00	150.00	R
(IDelay)	ext delay				200.00	350.00	R
VDD	in port	20	1139.10	0.00	0.00	350.00	R
pad_data_in0/IE					0.10	350.10	
pad_data_in0/Y	PIC	3	31.10	50.30	546.90	897.00	R
encode1/datain[0]							
g1637/B					0.00	897.00	
g1637/Y	NOR2X4TS	1	8.20	49.80	57.20	954.20	F
drc/A					0.00	954.20	
drc/Y	CLKBUFX8TS	6	38.10	138.60	205.90	1160.10	F
g1636/A					0.00	1160.10	
g1636/Y	CLKINVX4TS	2	16.10	86.90	111.30	1271.40	R
g1618/A					0.00	1271.40	
g1618/Y	NAND2X4TS	2	23.00	119.70	111.60	1383.00	F
g1604/A					0.00	1383.00	
g1604/Y	NOR2X4TS	1	12.10	141.30	140.60	1523.60	R
g1756/B0					0.10	1523.70	
g1756/Y	AOI2BB1X4TS	2	9.40	93.90	95.00	1618.70	F
g1754/A1					0.00	1618.70	
g1754/Y	AOI31X4TS	2	12.00	86.10	525.30	2144.00	R
g1571/B					0.00	2144.00	
g1571/Y	NAND2X2TS	1	9.90	116.10	121.50	2265.50	F
g1565/A					0.00	2265.50	
g1565/Y	CLKXOR2X4TS	3	17.00	148.10	383.00	2648.50	F
g1563/A					0.00	2648.50	
g1563/Y	NAND2X2TS	1	7.20	105.90	124.90	2773.40	R
g1561/B0					0.00	2773.40	

Figure 17. RC timing report

According to the area report showing below the Serial_Transmitter has 177 cells, the Serializer is provide 69 cells and the Encoder 84.

Instance	Cells	Cell Area	Net Area	Total Area
Chip_Serial_Transmisor	177	434829	2229	437059
Serializador1	69	1191	783	1974
encode1	84	894	953	1847

4.4 Serializer final version

Table 18 displays the final version of the Serializer module, it includes the instance of the Encoder and the Serialization.

The Serialization module has modifications, it is using just one always block, it makes a small change in the start_frame pin before this pin use to be a register, now it is just a wire assigned to a condition.

In the previous version of the Serialization, the start_frame signal was removed, but for the purpose of integration and synchronization, this pin is required.

The target for this module is operating at 1.25 GHz, to make this module operate at that speed is necessary to implement customer cell instead use the standards cell.

The SerDes operation for this project is 125Mhz speed, which means that Serializer modules are in the range of the required operation.

```
module Serializer(clk, rst, data_in,
data_out, frame);
input          clk;
input          rst;
input [8:0]   data_in;
output       data_out;
output        frame;

wire rd_out_encode, rd_in_encode;
wire [9:0] dataout;
encode encode1(
          .datain(data_in),
         .dispin(rd_in_encode),
         .dataout(dataout),
         .dispout(rd_out_encode));

Serialization Serialization1(
                  .par_in(dataout),
                  .clk(clk),
                  .rst(rst),
              .rd_in(rd_out_encode),
                  .rd_out(rd_in_encode),
                  .ser_out(data_out),
                  .start_frame(frame));
Endmodule
```

```
module Serialization(
input [9:0]  par_in,
input        clk,
input        rst,
input        rd_in,
output reg rd_out,
output       ser_out,
output       start_frame
);

reg [9:0] par_reg;
reg [3:0] counter;

always @(posedge clk, posedge rst) begin
          if (rst) begin
                    counter <= 4'h0;
                    par_reg <= 10'h0;
                    rd_out <=  1'h0;
          end
          else if (counter == 4'h0) begin
                    counter <= counter + 1'h1;
                    rd_out <= rd_in;
                    par_reg <= par_in;
          end
          else if (counter < 4'h9) begin
                    counter <= counter + 1'h1;
                    rd_out <= rd_out;
                    par_reg <= {1'h0,
par_reg[9:1]};
          end
          else begin
                    counter <= 4'h0;
                    rd_out <= rd_out;
                    par_reg <= {1'h0,
par_reg[9:1]};
          end
end

assign ser_out = par_reg[0];
assign start_frame = (counter == 9) ? 1'h1:
1'h0;

endmodule
```

Figure 18. Serialization (right) and top file (left) RTL code.

Chapter 5. Serializer Verification

5.1 Verification of Encoder-Decoder

8b10b Encoder-Decoder simulation.

To perform a verification, the code from Hollis Chuck was ported to ModelSim and simulated. In the simulation, all the possible combinations of the 8 bits data to feed the Encoder were injected. RTL code was taken from [7]. Once the output of the Encoder was verified, it was connected to the input of the decoder, creating a loopback test bench.

Figure 19 is showing the waveforms of the simulation between the Encoder and decoder, all the signals with DUTE correspond to the Encoder and the DUTD signals correspond to the decoder. In the first transaction, the Encoder received in datain pins “00000000” and this is translated by the Encoder into “001011101” and send in the dataout pins to the decoder. Ones received this information is decoded into “00000000” and send into the dataout pins [7].

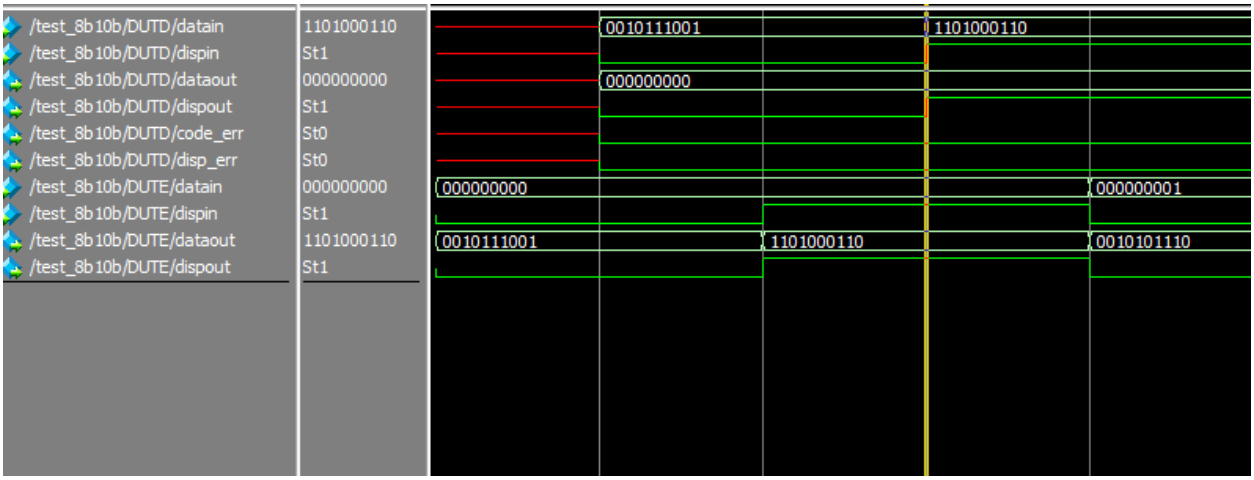


Figure 19. Encoder 8b/10b simulations result [7]

5.2 Serializer Verification

The test bench created for the Serializer verification can be found in the appendix E. A checker is used in the test bench to verify the data information by comparing the input stream with the result output, this comparison will be perform every 10 cycles, generating the pass/fail information in a log file.

Figure 20 displays the waveforms produced by the Serializer module. This module is pure behavioral description without logical and physical synthesis. This module is described on the right side of Figure 18.

The first signal at the top of Figure 20 is the par_in (parallel data input), this signal will change when start frame is enable. The clk signal change periodically in an amount of time specifies by the test bench. The reset is a control signal to clear all the data register information, this is used at the start to initialize all the values of the Serializer. The run_disparity_in is tied to 0 all the time, same case apply for run_disparity_out. The ser_out pin is the data output pin, it is connected to the analog part of the SerDes, this signal can be compared with the input signal, it is “1001110100” in this case it is matching with the par_in (parallel data input).

The par_reg signal is the same data than the input parallel data, it will be stored in a register and this is shifting to the par_out pin. The counter controls the number of cycles to trigger the Ack pin and the output of the ser_out pin. The Ack signal is used to sync the capture of the data information into the par_reg.

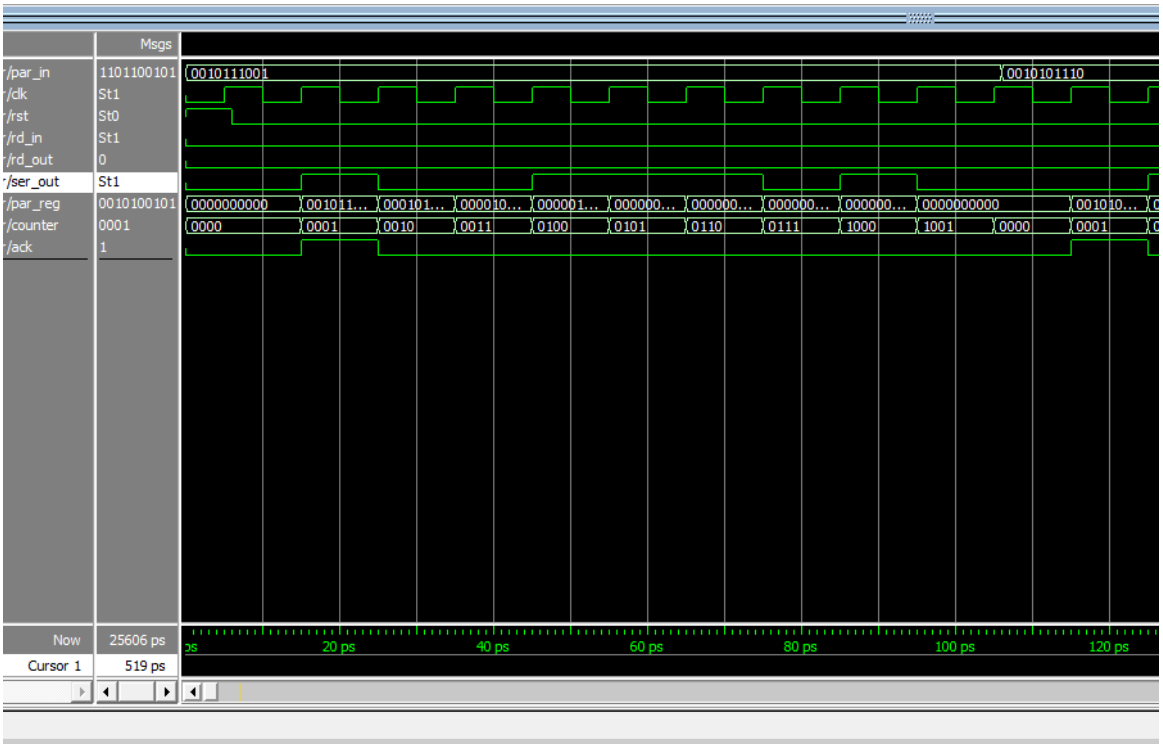


Figure 20. Serializer module RTL simulations waveforms

5.3 Gate Level Simulation

Figure 21 displays the waveforms and the files that were used in the gate level simulation of the Chip_Serial_Transmisor_m.v module. This code is in the appendix B and the TB_serializador.sv module is in appendix E.

The files ibm13rflpvt.v and the ibm13rflpvt_neg.v contain all the information of the standard cells and the iogpil_cmrf8sf_rvt.v contain all the information of the pads. These three Verilog files are imported from the ARM130nm libraries.

The test bench makes the simulation with a speed of 250 Mhz with a period of 4ns which is faster than the 125Mhz clock that was selected for the logical synthesis. The verification at 250 Mhz prove that the module is able to work properly at that speed.

According to the tables 5B\6B and 3B\4B from the Figure 1, the Serializer output corresponds with the values of the tables.

In the waveforms, there is a file handler signal that corresponds to the file.txt. This text file is the output log generated by the test bench. Figure 22 displays all the information about this file.

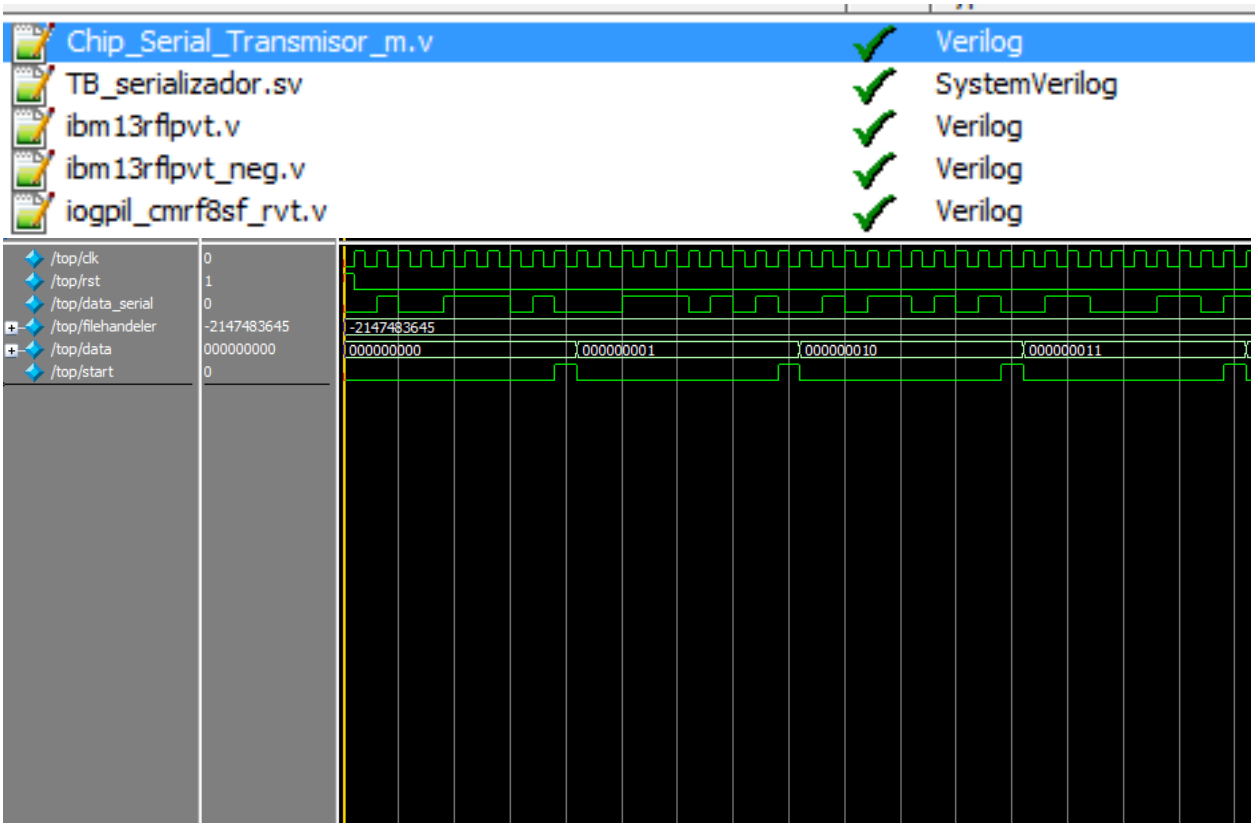
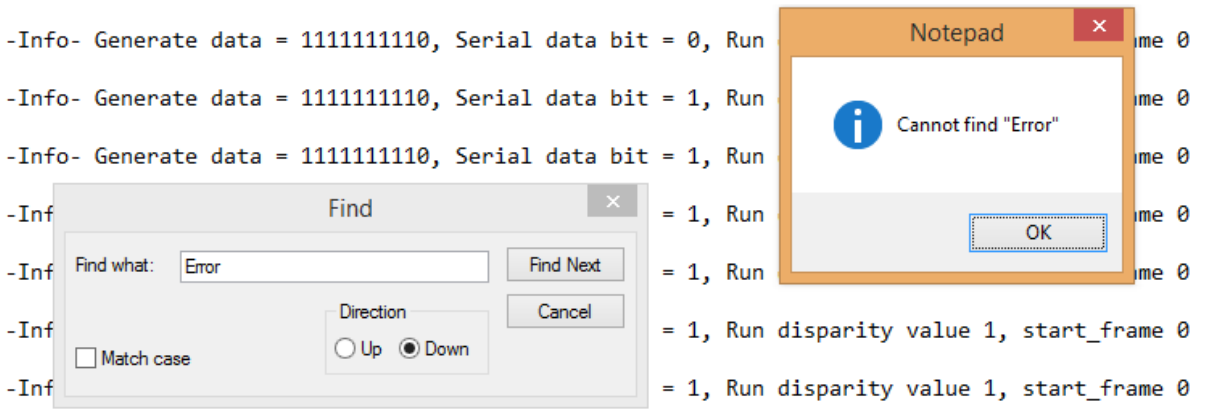


Figure 21. Waveforms running at the speed of 250 Mhz post logic synthesis.




```
-Info- Generate data = 1111111110, Serial data bit = 0, Run                    me 0
-Info- Generate data = 1111111110, Serial data bit = 1, Run                    me 0
-Info- Generate data = 1111111110, Serial data bit = 1, Run                    me 0
-Inf                                                    = 1, Run               me 0
-Inf                                                    = 1, Run               me 0
-Inf                                                    = 1, Run disparity value 1, start_frame 0
-Inf                                                    = 1, Run disparity value 1, start_frame 0
-Info- Generate data = 1111111110, Serial data bit = 1, Run disparity value 1, start_frame 0
-Info- Generate data = 1111111110, Serial data bit = 1, Run disparity value 1, start_frame 1
-Info- Generate data = 1111111110, Serial data bit = 1, Run disparity value 1, start_frame 0
-Info- Match Generate data = 1022,  Expected data = 1022, Serial data bit = 1, Run disparity 1
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 1
-Info- Generate data = 1111111111, Serial data bit = 1, Run disparity value 0, start_frame 0
-Info- Match Generate data = 1023,  Expected data = 1023, Serial data bit = 1, Run disparity 0
```

Figure 22. Test-bench log

Chapter 6. Physical Synthesis

6.1 Physical synthesis description.

To perform the physical synthesis, it is necessary to follow the steps showing the figure 22. The constraints and netlist files are generated in the logical synthesis, these must be imported from RC to Encounter.

Another important step is to create the Power Grid, the definition of the width and spacing of the vertical and horizontal stripes. The placement is one elementary step, here the imported design will be set in the floorplan inside of the power grid.

A couple of buffers are selected to generate the clock synthesis. The nano-routing must be launched to create the connection between different standards cells. The last step is the verification of the module where several tests are required to be run like DRC, connectivity, geometry and timing. The main recommendation for the physical synthesis is to create a script to make the automation of all the steps described above. [2].

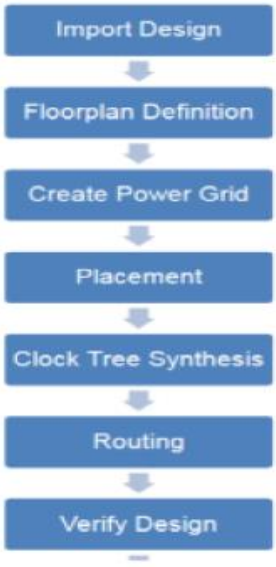


Figure 23. Physical synthesis stages [2].

Figure 23 display the power grid, it was set a 4 width for the ring with space of 5, it is important to pay attention because for the floorplan create with the pads need to fit with this design. The figure shows the modules without the optimization of nano-routing that helps to make improvements in the timing and creating buffers in the nets.

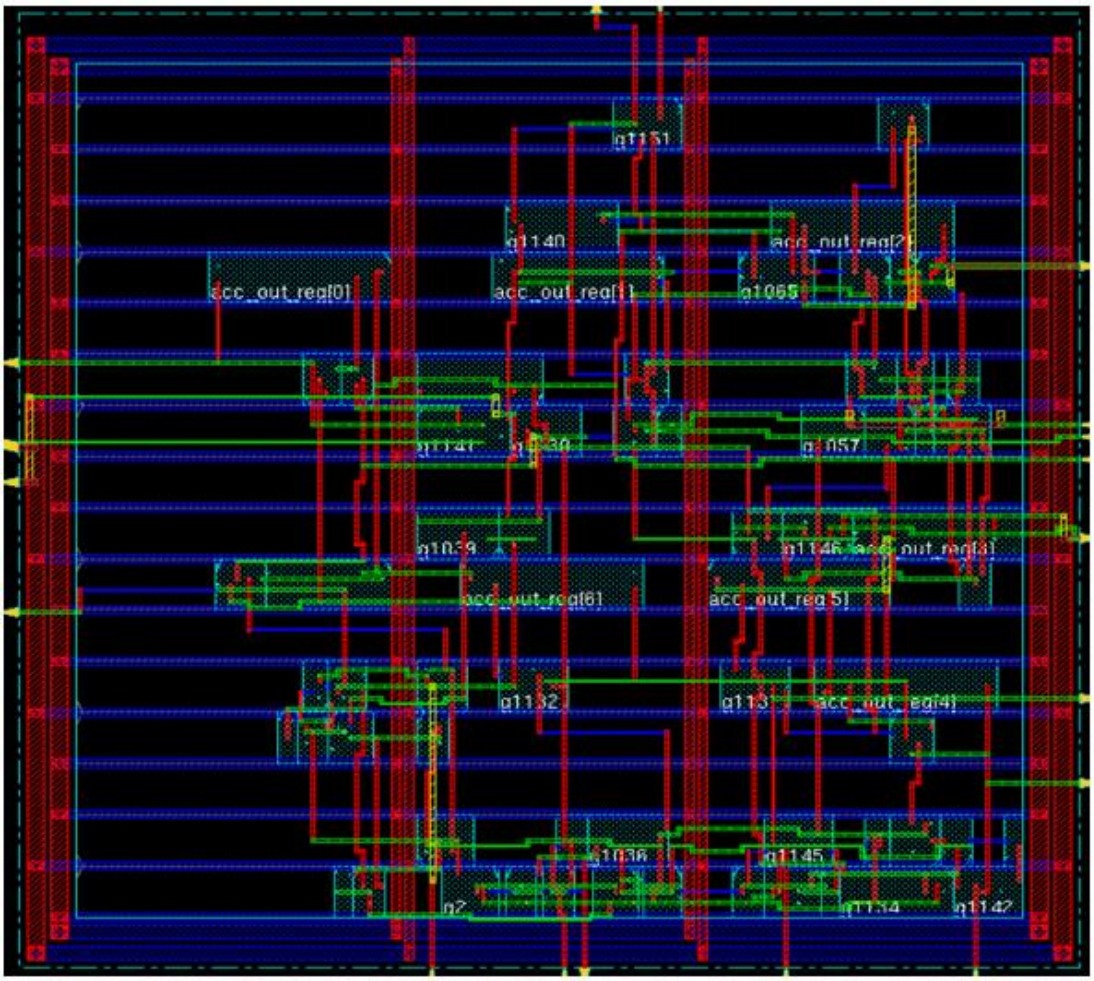


Figure 24. Floorplan with horizontal and vertical nets and ring

6.2 Results physical synthesis of Serial Transmitter

After the nano-routing is applied, it is necessary to verify the DRC, geometry, connectivity and timing test to verify that the module is working properly.

Figure 25 displays the DRC verification, this is checking any violation about connections and dimension between layers, at the begin of this project, Serializer hit many issues in the DRC verification because the pads and the power ring were not configure correctly.

```
*** Starting Verify DRC (MEM: 920.9) ***

 VERIFY DRC ..... Starting Verification
 VERIFY DRC ..... Initializing
 VERIFY DRC ..... Deleting Existing Violations
 VERIFY DRC ..... Creating Sub-Areas
 VERIFY DRC ..... Using new threading
 VERIFY DRC ..... Sub-Area : 1 of 4
 VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.
 VERIFY DRC ..... Sub-Area : 2 of 4
 VERIFY DRC ..... Sub-Area : 2 complete 0 Viols.
 VERIFY DRC ..... Sub-Area : 3 of 4
 VERIFY DRC ..... Sub-Area : 3 complete 0 Viols.
 VERIFY DRC ..... Sub-Area : 4 of 4
 VERIFY DRC ..... Sub-Area : 4 complete 0 Viols.

 Verification Complete : 0 Viols.
```

Figure 25. Log result of physical synthesis post-optimization DRD verification

Figure 26 represents the result of the geometry, this test verify the correct dimensions of the cells. This test also check the wiring, overlapping and short between nets, pads, cells and buffers.

Figure 27 display the result of connectivity test, this verification check the net connections if there is not short circuit and missing net connections.

Figure 28 show the result of timing test. This is one of the most important test. This will give you an approach if your design will work with the frequency tentative to use in the design. If this test fail there are a few paths to improve the timing. The first is with script optimization and others increasing the nano-routiong level and last one is creating another architecture for your design.

```
 *** Starting Verify Geometry (MEM: 837.4) ***

  VERIFY GEOMETRY ..... Starting Verification
  VERIFY GEOMETRY ..... Initializing
  VERIFY GEOMETRY ..... Deleting Existing Violatio
  VERIFY GEOMETRY ..... Creating Sub-Areas
                  ..... bin size: 2560
VG: elapsed time: 0.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
```

Figure 26. Log result of physical synthesis post-optimization Geometry verification

```
******** Start: VERIFY CONNECTIVITY ********
Start Time: Sun Jun 19 15:38:29 2016

Design Name: Chip_Serial_Transmisor
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (1159.0000, 1159.0000)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Sun Jun 19 15:38:29 2016
Time Elapsed: 0:00:00.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
```

Figure 27. Log result of physical synthesis post-optimization connectivity verification

```
------------------------------------------------------------
     optDesign Final Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  0.155  |  0.236  |   N/A   |  0.155  |
|           TNS (ns):|  0.000  |  0.000  |   N/A   |  0.000  |
|    Violating Paths:|    0    |    0    |   N/A   |    0    |
|          All Paths:|   32    |   15    |   N/A   |   28    |
+--------------------+---------+---------+---------+---------+

+--------------------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  0.486  |  1.116  |   N/A   |  0.486  |
|           TNS (ns):|  0.000  |  0.000  |   N/A   |  0.000  |
|    Violating Paths:|    0    |    0    |   N/A   |    0    |
|          All Paths:|   32    |   15    |   N/A   |   28    |
+--------------------+---------+---------+---------+---------+
```

Figure 28. Timing summary log result of physical synthesis post-optimization

Figure 28 displays the chip design, the most of the placement was in the below-left corner, this happened because the inputs and outputs of the circuit are located in that area, the location of the pads were thinking to have the best performance in timing.

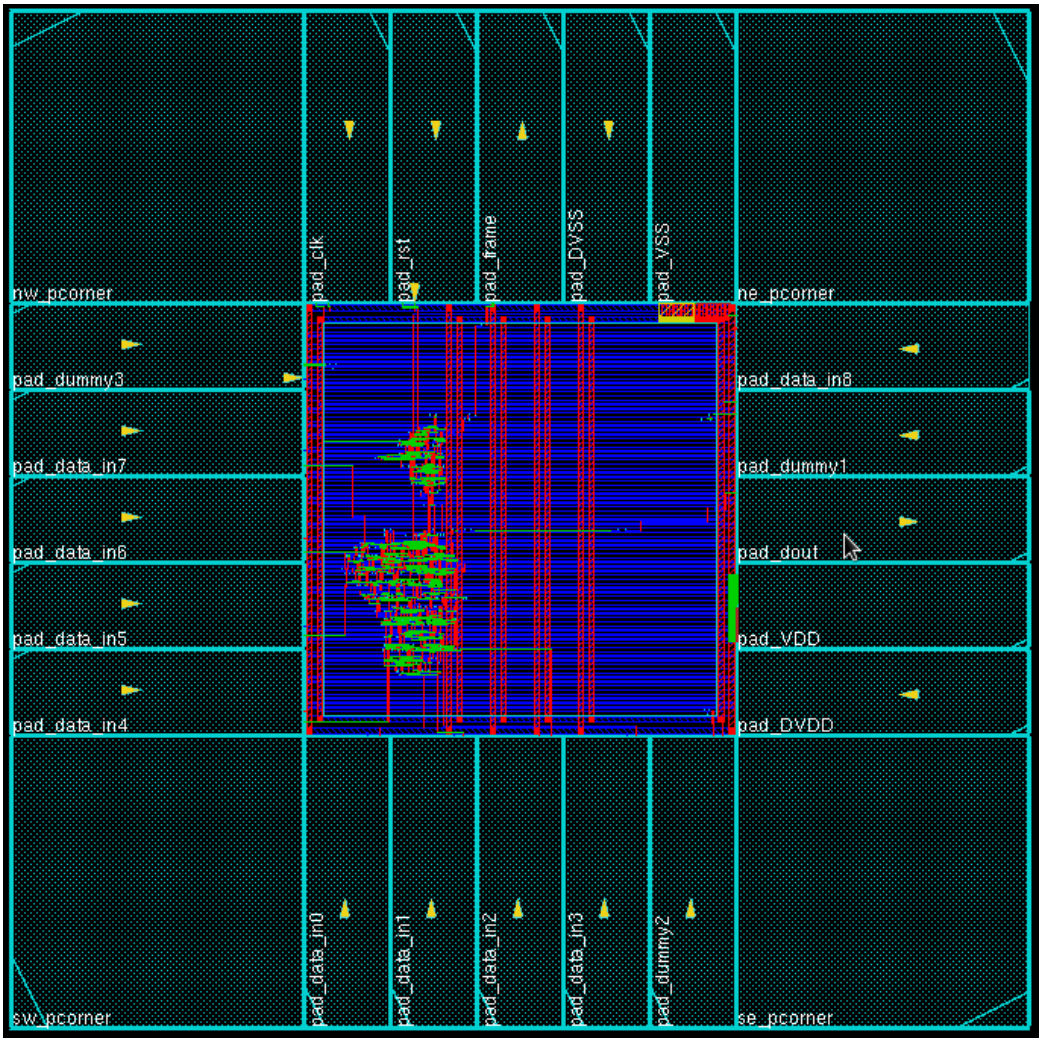


Figure 29. Final result chip

6.3 Serializer Virtuoso

After the completion of all the verification test in Encounter. A gds file importation should be done to Virtuoso.

The libraries .map are provided to create a gds file and make the importation to Virtuoso. The directory where these libraries are located is the follow: opt/libs/AMD_PDK/libraries.map

The Virtuoso tool is used to create schematics and layout design. The objective of work in Virtuoso is to perform DRC and LVS verification.

Figure 29 display the Serializer full chip imported to virtuoso.

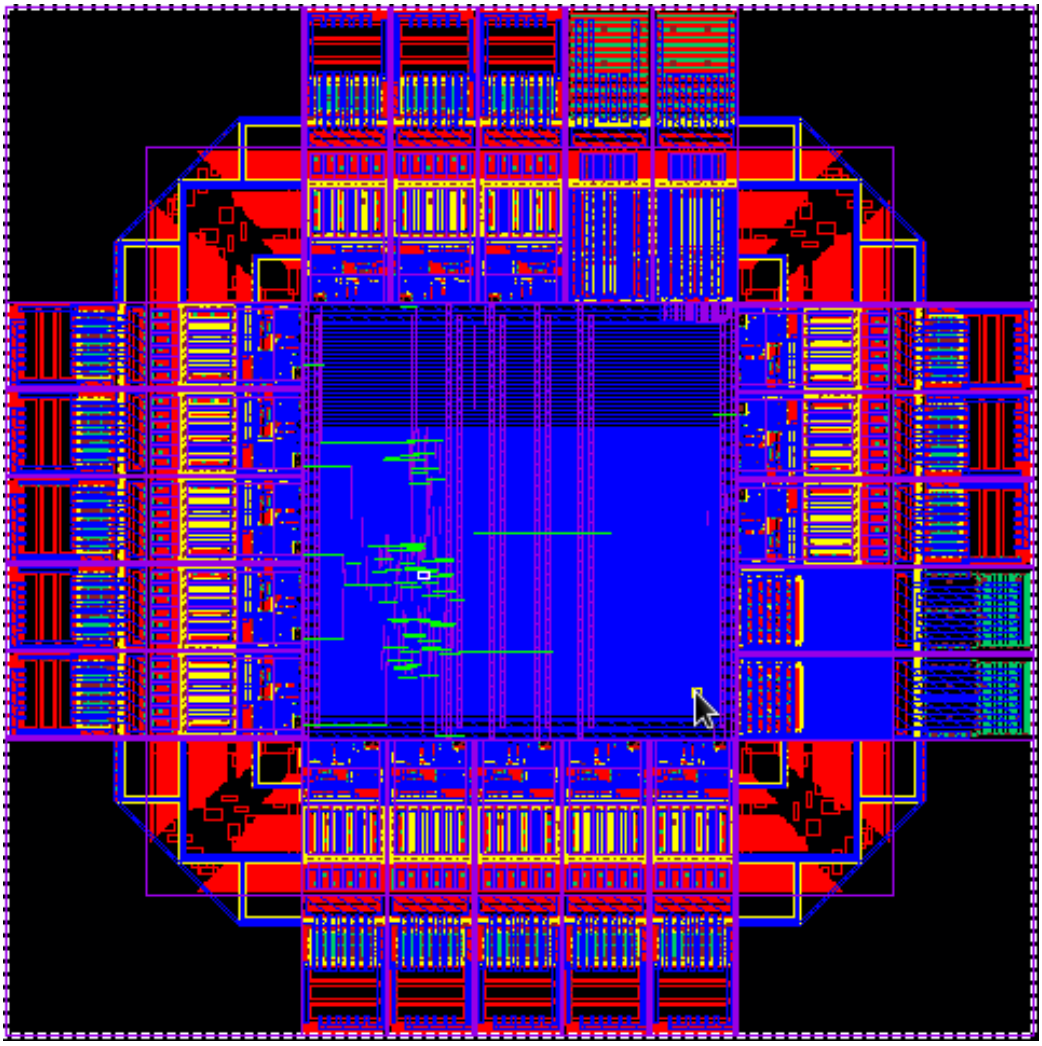


Figure 30. Chip imported to virtuso

The DRC verification consists in dimensional, antenna, and latch up rules.

For the DRC test, environment variables were selected. Figure 30 displays all the switches used for the configurations of DRC test, if the user doesn't use this configuration, the DRC testing will hit a lot of problems, to solve this a configuration was provided by MOSIS and some mentoring people:

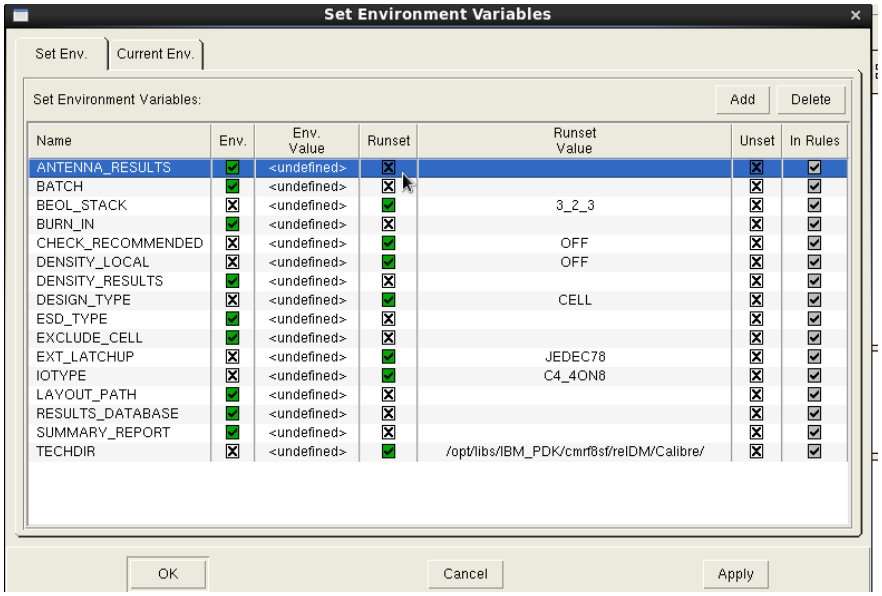


Figure 31. DRC switches.

After the corrected switched where selected, DRC testing was launching and it showed these problems:

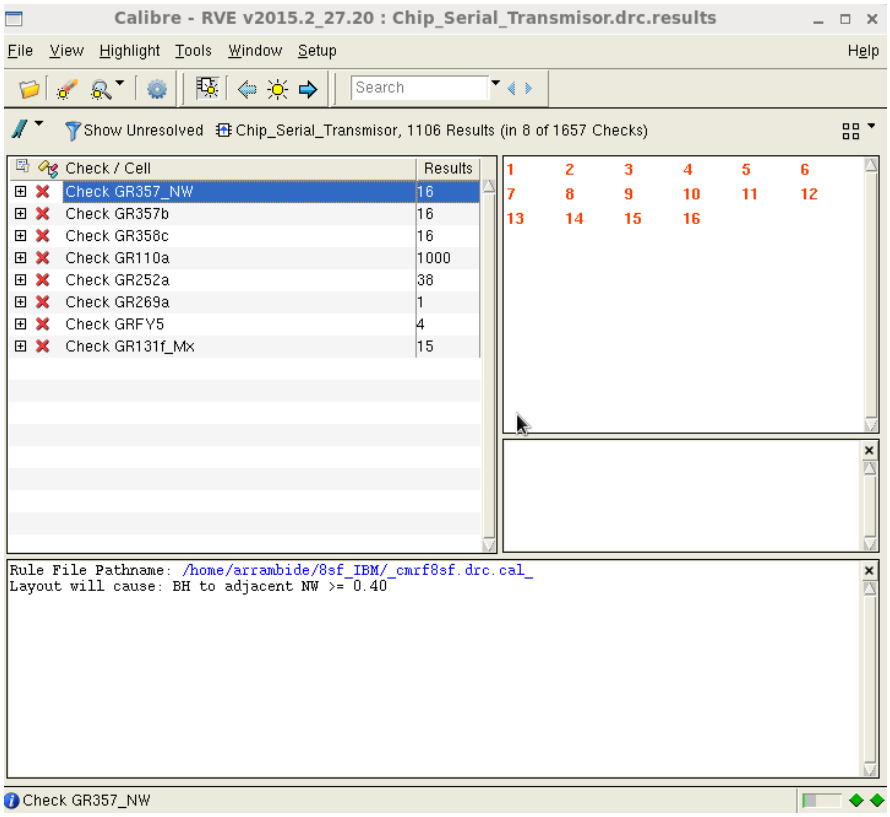


Figure 32. DRC violation rules.

The GR110a was fixing using the GRLOGIC, DRC was complying because the standard cells don't have a connection to the substrate. A wrap of GRLOGIC was created around the chip to tell DRC test these zone of layout should be traded as a standard cell and avoid this violation rule [8].

The GR357_NW is regarding NW connexion it was a violation the distance so we procedure to join all the spaces between the NW material [8].

The GR37b and GR358c were solved when the GR357_NW was clean. This mean that these rules were correlated with the previously errors [8].

The GR252 this complained about a distance NW to NW space, this recommend to be bigger than 0.92 to solve this error, the NW was joined with the other NW [8].

After putting the GRLOGIC wrapper some error rules were detected, LUP10B this is a latch-up rule. To solve this violation, a 130nm technology manual was consulted to understand what the problem was. This explanation suggests that the NWell contact needs to have a minimum of area related to the BP contact area, this is to avoid short circuits between the substratum and ground [8].

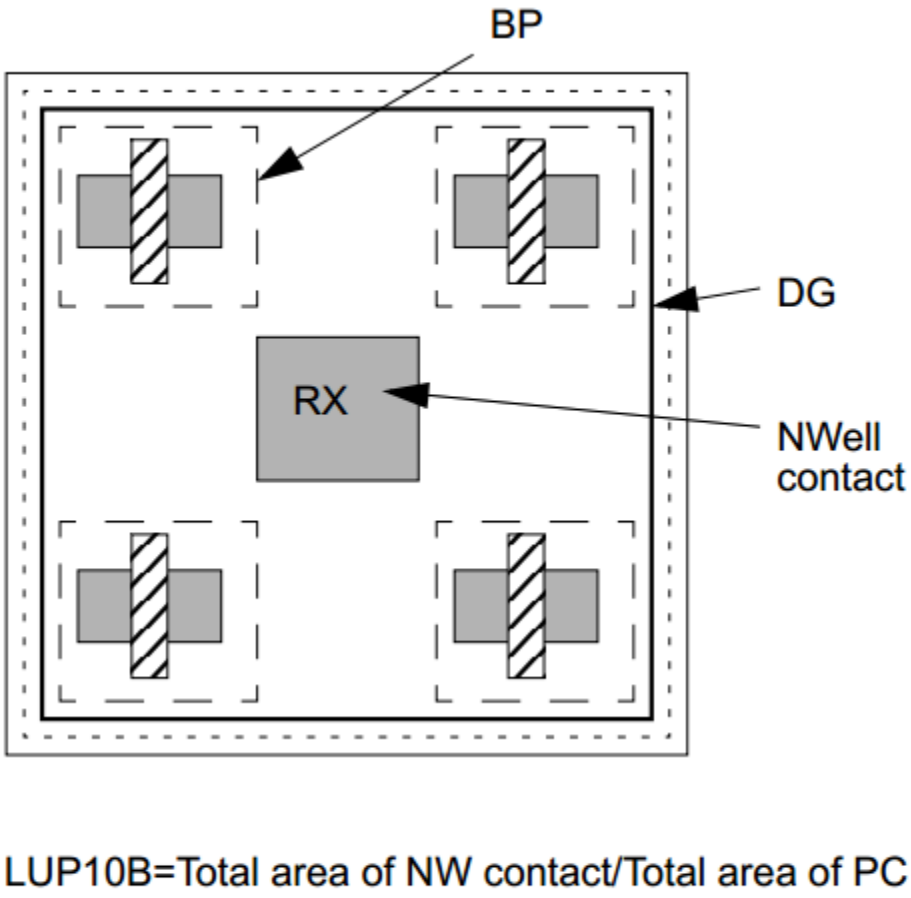


Figure 33. Diagram of LATCHUP rule layout [8]

After the fixing of all the problems with the DRC test. Finally, figure 33 displays the result of clean DRC test.

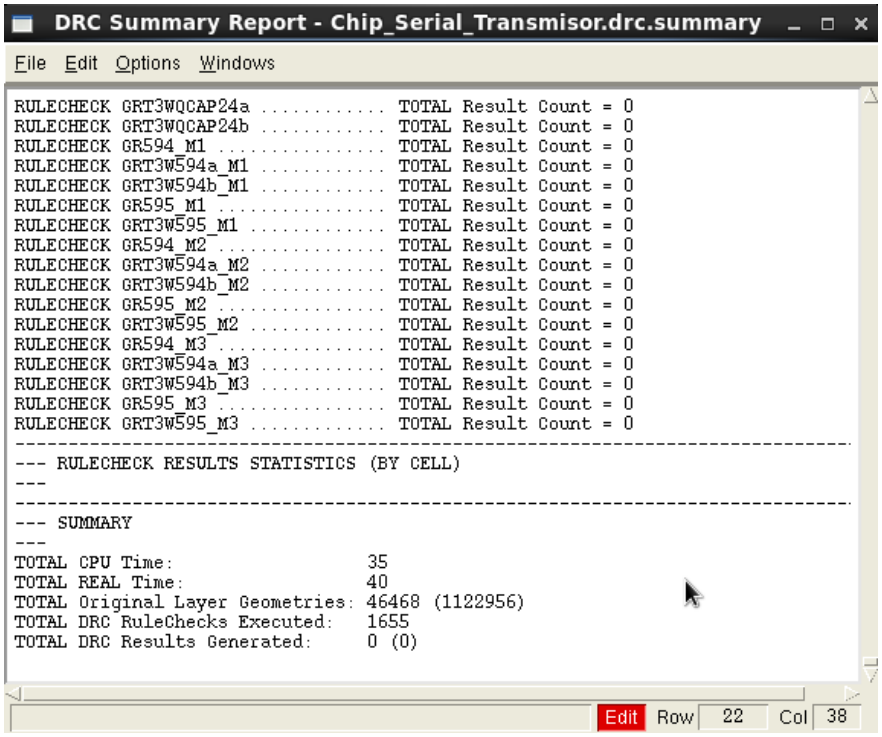


Figure 34. DRC result log.

Conclusion

This project give an explanation with detail of all the steps to develop and implement a Serializer, which will be part of a SerDes system using the technology of 130nm. This is an inheritance project started by a previous generation of the specialty program in System On a Chip at ITESO University. This have helped to create interaction and receive feedback from the pass generation, allowing create new proposals to enhancement the design and increase the performance.

Once the design was evaluated some enhancement were detect, these were evaluated and verify, creating a Serializer module with a clean time, meaning that Serializer lack of setup and hold timing violation. It passed pre-post logic synthesis (timing, area, gate level simulation, behavioral pure simulation) and pre-post physical synthesis (timing, DRC, connectivity, geometry) validation test. The design is ready to run at the speed of 125Mhz and the Serialization block is compatible with test module block and analog transmitter block.

The main problems faced in the implementation of the Serializer module were in the timing area, new architectures were presented to solve the problem of timing, but none of them work, the problem is that the standards cells cannot support the speed above 1Ghz. After feedback from the Receiver or DeSerializer module the speed was fix to 125Mhz even if the Serializer can work at 250Mhz. Another facing problem was the DRC in virtuoso, this verification take a considerable time because some of the violated rules were difficult to understand and find a solution for it.

The Serializer module had two main modifications, the first one is the modification of one register and it was changed to an assignment to a condition. The second one was moving all the RTL code to one always block and the run disparity is capture at the same time that the parallel data. As part of the verification a new testbench was proposed creating a log file helping the debugging process.

References

[1] Atul Patel. What is a SerDes? [Onlie] Available: http://www.planetanalog.com/document.asp?doc_id=528099

[2] Joe Winkles 2006. PCI Express Basic [Online] Available: www.mindshare.com

[3] Mike Jackson, Ravi Budruk 2006 . PCI Express Technology [Online] Available: www.mindshare.com

[4] José Manuel Centeno Quiñonez, Citlali Gonzalez Morales, Cuauhtémoc Aguilera, Victor Avendaño, and Alexandro Giron, “ITESO TV1. RTL Design for the PCIe deserializer module”, Tesina ESC-ITESO, Guadalajara Jal. Diciembre 2015.

[5] Patel, A. (2010, September 16). The basics of SerDes (serializers/deserializers) for interfacing. [Online] Available: http://www.planetanalog.com/document.asp?doc_id=528099/

[6] Al X. Widmer, Peter A. Franaszek (1983). "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code". IBM Journal of Research and Development 27 (5): 440–451.

[7] Chuck Benz, “Chuck Benz ASIC and FPGA Design”, [Online] Available: http://asics.chuckbenz.com/

[8] CMOS8RF Desing Manual, 1st ed, IBM Corporation, 2010.

Appendix

A. Constraint file to RC

```
set_time_unit -picoseconds
set_load_unit -femtofarads

# Clock definition
define_clock -name 125MHz_CLK -period 8000 [get_ports clk]

# slew rate definitions (min rise, min fall, max rise, max fall).
# The values coming from IBM typical specification.
set_attribute slew { 28 28 28 28 } 125MHz_CLK

# network clock latency
set_attribute clock_network_late_latency 100 125MHz_CLK
set_attribute clock_network_early_latency 90 125MHz_CLK
# source clock latency
set_attribute clock_source_late_latency 50 125MHz_CLK
set_attribute clock_source_early_latency 40 125MHz_CLK

# clock skew
set_attribute clock_setup_uncertainty {17 10} 125MHz_CLK
set_attribute clock_hold_uncertainty {13 5} 125MHz_CLK

set_attribute hdl_error_on_blackbox true

# Input delay definition: This is the delay coming from outside the design
# for this design it's defined at 10% the period of the clock.
external_delay -clock [find / -clock 125MHz_CLK] -input 200 -name IDelay [find /des* -
port ports_in/*]

# Output delay definition: This is the delay going outside the design
# for this design it's defined at 10% the period of the clock.
external_delay -clock [find / -clock 125MHz_CLK] -output 200 -name ODelay [find /des*
-port ports_out/*]

# Driving cell definition
set_attribute external_driver [find [find / -libcell BUFX3TS] -libpin Z]
/designs/Chip_Serial_Transmisor/ports_in/*


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

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

# Considering a pad output buffer POC2A load
set_attribute external_pin_cap 40 /designs/Chip_Serial_Transmisor/ports_out/*

# Setting maximum value of fanout
set_attribute max_fanout 15 /designs/*

set_attribute lp_power_unit {uW}
set_attribute lp_pso_aware_estimation true
set_attribute leakage_power_effort high
set_attribute max_leakage_power 100 "/designs/$DESIGN"
set_attribute lp_power_optimization_weight 0.5 "/designs/$DESIGN"
set_attribute max_dynamic_power 100 "/designs/$DESIGN"
```

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

#set_attribute lp_power_unit {uW}
#set_attribute lp_pso_aware_estimation true
#set_attribute max_leakage_power 100 "/designs/$DESIGN"
#set_attribute lp_power_optimization_weight 0.5 "/designs/$DESIGN"
#"


#Specifies the maximum leakage-power constraint of the design. The constraint enables
RTL Compilerto perform timing
#and leakage power optimization simultaneously during mapping
# Specifies the maximum dynamic-power constraint of the design. The dynamic power is
the sum of the internal power dissipated in #all instances and the switching power
dissipated in all nets.
```

B. Netlist file imported to EDI

a. Serialization.v

```
module Serialization(par_in, clk, rst, rd_in, rd_out, ser_out,
    start_frame);
  input [9:0] par_in;
  input clk, rst, rd_in;
  output rd_out, ser_out, start_frame;
  wire [9:0] par_in;
  wire clk, rst, rd_in;
  wire rd_out, ser_out, start_frame;
  wire [9:0] par_reg;
  wire [3:0] counter;
  wire UNCONNECTED, UNCONNECTED0, UNCONNECTED1, UNCONNECTED2,
      UNCONNECTED3, UNCONNECTED4, UNCONNECTED5, UNCONNECTED6;
  wire UNCONNECTED7, UNCONNECTED8, UNCONNECTED9, UNCONNECTED10, n_0,
      n_1, n_2, n_3;
  wire n_4, n_5, n_6, n_7, n_8, n_12, n_13, n_14;
  wire n_15, n_16, n_17, n_18, n_19, n_20, n_21, n_22;
  wire n_28, n_29, n_30, n_32, n_33, n_34, n_35, n_36;
  wire n_37, n_38, n_39, n_40, n_41, n_42, n_57, n_59;
  wire n_70, n_71;
  CLKBUFX8TS g273(.A (n_42), .Y (ser_out));
  MX2X1TS g274(.S0 (n_30), .B (par_in[0]), .A (par_reg[1]), .Y (n_41));
  MX2X1TS g276(.S0 (n_30), .B (par_in[1]), .A (par_reg[2]), .Y (n_40));
  MX2X1TS g278(.S0 (n_30), .B (par_in[2]), .A (par_reg[3]), .Y (n_39));
  MX2X1TS g280(.S0 (n_30), .B (par_in[3]), .A (par_reg[4]), .Y (n_38));
  MX2X1TS g282(.S0 (n_30), .B (par_in[4]), .A (par_reg[5]), .Y (n_37));
  MX2X1TS g284(.S0 (n_30), .B (par_in[5]), .A (par_reg[6]), .Y (n_36));
  MX2X1TS g286(.S0 (n_30), .B (par_in[6]), .A (par_reg[7]), .Y (n_35));
  MX2X1TS g288(.S0 (n_30), .B (par_in[7]), .A (par_reg[8]), .Y (n_34));
  MX2X1TS g290(.S0 (n_30), .B (par_in[8]), .A (par_reg[9]), .Y (n_33));
  AND2X1TS g292(.A (n_30), .B (par_in[9]), .Y (n_32));
  AND3X6TS g293(.A (n_29), .B (counter[3]), .C (counter[0]), .Y
     (start_frame));
  AND2X8TS g294(.A (n_29), .B (n_28), .Y (n_30));
  NOR2X6TS g295(.A (counter[2]), .B (counter[1]), .Y (n_29));
  AND2X2TS g296(.A (n_57), .B (n_1), .Y (n_28));
  INVX4TS g298(.A (counter[0]), .Y (n_1));
  INVX2TS g299(.A (rst), .Y (n_2));
  BUFX4TS drc_bufs302(.A (n_30), .Y (n_59));
  CLKAND2X2TS g230(.A (n_17), .B (n_14), .Y (n_19));
  DFFRHQX2TS \counter_reg[0] (.RN (n_2), .CK (clk), .D (n_18), .Q
     (n_22));
  MX2X1TS g232(.S0 (n_14), .B (n_1), .A (n_59), .Y (n_18));
  CLKXOR2X2TS g233(.A (n_12), .B (counter[3]), .Y (n_17));
  CLKAND2X2TS g236(.A (n_14), .B (n_8), .Y (n_16));
  AND2X1TS g237(.A (n_14), .B (n_13), .Y (n_15));
  NOR2X6TS g238(.A (n_59), .B (n_70), .Y (n_14));
  CMPR22X2TS g239(.A (n_3), .B (n_7), .S (n_13), .CO (n_12));
  DFFRHQX8TS rd_out_reg(.RN (n_2), .CK (clk), .D (n_0), .Q (rd_out));
  CLKMX2X2TS g242(.S0 (n_59), .B (rd_in), .A (rd_out), .Y (n_0));
  CLKXOR2X2TS g244(.A (n_22), .B (counter[1]), .Y (n_8));
  NOR2BX4TS g245(.AN (counter[1]), .B (n_1), .Y (n_7));
  NAND2BX2TS g246(.AN (n_57), .B (n_4), .Y (n_6));
  NAND2BX2TS g247(.AN (counter[1]), .B (n_1), .Y (n_5));
  CLKBUFX2TS g251(.A (n_3), .Y (n_4));
  BUFX4TS g252(.A (n_20), .Y (n_3));
  BUFX3TS drc_bufs256(.A (n_20), .Y (counter[2]));
  CLKBUFX6TS drc_bufs259(.A (n_21), .Y (counter[1]));
  BUFX3TS drc_bufs262(.A (n_22), .Y (counter[0]));
```

```
    OAI2BB1X4TS g2(.A0N (n_5), .A1N (counter[3]), .B0 (n_6), .Y (n_70));
    CLKBUFX3TS g342(.A (n_71), .Y (n_20));
    DFFRX4TS \counter_reg[3] (.RN (n_2), .CK (clk), .D (n_19), .Q
       (counter[3]), .QN (n_57));
    DFFRX2TS \par_reg_reg[0] (.RN (n_2), .CK (clk), .D (n_41), .Q (n_42),
       .QN (UNCONNECTED));
    DFFRX2TS \counter_reg[2] (.RN (n_2), .CK (clk), .D (n_15), .Q (n_71),
       .QN (UNCONNECTED0));
    DFFRX2TS \counter_reg[1] (.RN (n_2), .CK (clk), .D (n_16), .Q (n_21),
       .QN (UNCONNECTED1));
    DFFRX2TS \par_reg_reg[1] (.RN (n_2), .CK (clk), .D (n_40), .Q
       (par_reg[1]), .QN (UNCONNECTED2));
    DFFRX2TS \par_reg_reg[2] (.RN (n_2), .CK (clk), .D (n_39), .Q
       (par_reg[2]), .QN (UNCONNECTED3));
    DFFRX2TS \par_reg_reg[3] (.RN (n_2), .CK (clk), .D (n_38), .Q
       (par_reg[3]), .QN (UNCONNECTED4));
    DFFRX2TS \par_reg_reg[4] (.RN (n_2), .CK (clk), .D (n_37), .Q
       (par_reg[4]), .QN (UNCONNECTED5));
    DFFRX2TS \par_reg_reg[5] (.RN (n_2), .CK (clk), .D (n_36), .Q
       (par_reg[5]), .QN (UNCONNECTED6));
    DFFRX2TS \par_reg_reg[6] (.RN (n_2), .CK (clk), .D (n_35), .Q
       (par_reg[6]), .QN (UNCONNECTED7));
    DFFRX2TS \par_reg_reg[7] (.RN (n_2), .CK (clk), .D (n_34), .Q
       (par_reg[7]), .QN (UNCONNECTED8));
    DFFRX2TS \par_reg_reg[8] (.RN (n_2), .CK (clk), .D (n_33), .Q
       (par_reg[8]), .QN (UNCONNECTED9));
    DFFRX2TS \par_reg_reg[9] (.RN (n_2), .CK (clk), .D (n_32), .Q
       (par_reg[9]), .QN (UNCONNECTED10));
  endmodule
```

b. Encode.v

```
module encode(datain, dispin, dataout, dispout);
  input [8:0] datain;
  input dispin;
  output [9:0] dataout;
  output dispout;
  wire [8:0] datain;
  wire dispin;
  wire [9:0] dataout;
  wire dispout;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
  wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
  wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
  wire n_32, n_33, n_34, n_35, n_36, n_37, n_38, n_39;
  wire n_40, n_41, n_42, n_43, n_45, n_46, n_47, n_48;
  wire n_49, n_50, n_51, n_52, n_54, n_55, n_56, n_58;
  wire n_59, n_63, n_66, n_67, n_68, n_69, n_70, n_71;
  wire n_72, n_73, n_74, n_75, n_76, n_77, n_78, n_79;
  wire n_81, n_82, n_97, n_98, n_99, n_100;
  CLKXOR2X2TS g1572(.A (n_82), .B (n_76), .Y (dataout[9]));
  CLKXOR2X2TS g1573(.A (n_82), .B (n_77), .Y (dataout[6]));
  CLKXOR2X2TS g1574(.A (n_81), .B (datain[7]), .Y (dataout[8]));
  CLKXOR2X2TS g1575(.A (n_82), .B (n_2), .Y (dataout[7]));
  CLKXOR2X2TS g1576(.A (n_75), .B (n_63), .Y (dataout[5]));
  CLKXOR2X2TS g1577(.A (n_75), .B (n_99), .Y (dataout[4]));
  CLKXOR2X2TS g1578(.A (n_75), .B (n_31), .Y (dataout[3]));
  CLKXOR2X2TS g1579(.A (n_75), .B (n_46), .Y (dataout[2]));
  CLKXOR2X2TS g1580(.A (n_75), .B (n_54), .Y (dataout[1]));
  CLKXOR2X2TS g1581(.A (n_74), .B (datain[0]), .Y (dataout[0]));
  INVX2TS g1582(.A (n_82), .Y (n_81));
```

```
AND2X4TS g1583(.A (n_79), .B (n_78), .Y (n_82));
CLKXOR2X2TS g1584(.A (n_73), .B (n_3), .Y (dispout));
NAND2BX2TS g1585(.AN (n_1), .B (n_28), .Y (n_79));
NAND2BX2TS g1586(.AN (n_4), .B (n_73), .Y (n_78));
NAND2X2TS g1587(.A (n_72), .B (datain[5]), .Y (n_77));
CLKAND2X2TS g1588(.A (n_72), .B (n_43), .Y (n_76));
CLKINVX2TS g1589(.A (n_75), .Y (n_74));
CLKAND2X8TS g1590(.A (n_71), .B (n_69), .Y (n_75));
CLKXOR2X4TS g1591(.A (dispin), .B (n_68), .Y (n_73));
OR2X2TS g1592(.A (n_70), .B (n_19), .Y (n_72));
NAND2X2TS g1593(.A (dispin), .B (n_67), .Y (n_71));
NOR2BX2TS g1594(.AN (n_66), .B (datain[8]), .Y (n_70));
NAND2BX2TS g1595(.AN (dispin), .B (n_97), .Y (n_69));
NOR2BX4TS g1596(.AN (n_98), .B (n_97), .Y (n_68));
NAND2X2TS g1597(.A (n_98), .B (n_33), .Y (n_67));
CLKMX2X2TS g1598(.S0 (dispin), .B (n_55), .A (n_56), .Y (n_66));
OA21X2TS g1601(.A0 (n_48), .A1 (datain[4]), .B0 (n_59), .Y (n_63));
NOR2X2TS g1605(.A (n_47), .B (n_100), .Y (n_59));
CLKAND2X2TS g1606(.A (n_50), .B (n_11), .Y (n_58));
OR2X2TS g1608(.A (n_49), .B (n_16), .Y (n_56));
NAND3BX2TS g1609(.AN (n_50), .B (n_11), .C (datain[3]), .Y (n_55));
CLKAND2X2TS g1610(.A (n_45), .B (n_27), .Y (n_54));
AOI21X2TS g1612(.A0 (datain[8]), .A1 (n_5), .B0 (n_15), .Y (n_52));
NOR2X2TS g1613(.A (n_41), .B (datain[6]), .Y (n_51));
OA21X2TS g1614(.A0 (n_25), .A1 (n_17), .B0 (n_39), .Y (n_50));
CLKAND2X3TS g1615(.A (n_40), .B (n_37), .Y (n_49));
CLKAND2X3TS g1616(.A (n_38), .B (n_0), .Y (n_48));
CLKAND2X2TS g1617(.A (n_35), .B (datain[4]), .Y (n_47));
AND3X2TS g1618(.A (n_34), .B (n_27), .C (n_10), .Y (n_46));
NAND2BX2TS g1619(.AN (n_35), .B (datain[1]), .Y (n_45));
NAND2BX2TS g1621(.AN (datain[7]), .B (n_5), .Y (n_43));
NOR2BX2TS g1622(.AN (n_19), .B (n_15), .Y (n_42));
CLKAND2X2TS g1623(.A (n_15), .B (n_9), .Y (n_41));
NAND2X2TS g1624(.A (n_24), .B (n_13), .Y (n_40));
NAND2X2TS g1625(.A (n_24), .B (datain[2]), .Y (n_39));
OR2X2TS g1626(.A (n_25), .B (n_23), .Y (n_38));
NAND2BX2TS g1627(.AN (n_25), .B (n_7), .Y (n_37));
AOI2BB1X2TS g1628(.A0N (n_12), .A1N (n_17), .B0 (n_26), .Y (n_36));
CLKAND2X2TS g1629(.A (n_21), .B (datain[3]), .Y (n_35));
OR2X2TS g1630(.A (n_22), .B (datain[3]), .Y (n_33));
NAND3BX2TS g1631(.AN (n_16), .B (n_17), .C (n_10), .Y (n_32));
NAND2X2TS g1632(.A (n_22), .B (datain[3]), .Y (n_31));
OAI2BB1X2TS g1633(.A0N (datain[8]), .A1N (n_18), .B0 (n_16), .Y
   (n_30));
NAND2X4TS g1634(.A (n_20), .B (n_7), .Y (n_34));
ADDHX4TS g1635(.A (datain[5]), .B (datain[6]), .S (n_29), .CO (n_28));
NAND2X2TS g1636(.A (n_7), .B (n_13), .Y (n_27));
CLKAND2X2TS g1637(.A (n_18), .B (n_6), .Y (n_26));
OR2X2TS g1638(.A (n_18), .B (n_13), .Y (n_25));
INVX1TS g1639(.A (n_24), .Y (n_23));
CLKAND2X4TS g1640(.A (n_14), .B (n_17), .Y (n_24));
INVX2TS g1641(.A (n_21), .Y (n_22));
NOR2X4TS g1642(.A (n_17), .B (n_10), .Y (n_21));
AND3X4TS g1643(.A (n_10), .B (datain[3]), .C (datain[4]), .Y (n_20));
NAND3X4TS g1644(.A (datain[5]), .B (datain[6]), .C (datain[7]), .Y
   (n_19));
AND2X2TS g1645(.A (datain[3]), .B (datain[2]), .Y (n_18));
NAND2X6TS g1646(.A (datain[0]), .B (datain[1]), .Y (n_17));
OR2X4TS g1647(.A (n_11), .B (datain[3]), .Y (n_16));
NOR2X6TS g1648(.A (datain[6]), .B (datain[5]), .Y (n_15));
BUFX4TS g1652(.A (n_6), .Y (n_7));
OR2X2TS g1654(.A (datain[1]), .B (datain[0]), .Y (n_14));
CLKINVX2TS g1655(.A (n_13), .Y (n_12));
AND2X4TS g1656(.A (n_8), .B (n_10), .Y (n_13));
INVX3TS g1657(.A (datain[4]), .Y (n_11));
CLKINVX6TS g1658(.A (datain[2]), .Y (n_10));
```

```
    INVX1TS g1659(.A (datain[7]), .Y (n_9));
    CLKINVX2TS g1660(.A (datain[3]), .Y (n_8));
    INVX2TS drc_bufs(.A (n_14), .Y (n_6));
    CLKBUFX2TS drc_bufs1700(.A (n_52), .Y (n_4));
    CLKBUFX2TS drc_bufs1702(.A (n_42), .Y (n_3));
    CLKBUFX2TS drc_bufs1706(.A (n_51), .Y (n_2));
    CLKBUFX2TS drc_bufs1715(.A (n_73), .Y (n_1));
    CLKBUFX2TS drc_bufs1716(.A (n_36), .Y (n_0));
    BUFX3TS drc1734(.A (n_29), .Y (n_5));
    OAI2BB1X4TS g2(.A0N (n_58), .A1N (n_48), .B0 (n_34), .Y (n_97));
    AOI31X4TS g1759(.A0 (n_49), .A1 (n_48), .A2 (datain[4]), .B0
        (datain[8]), .Y (n_98));
    OAI2BB1X2TS g1760(.A0N (n_49), .A1N (n_11), .B0 (n_34), .Y (n_99));
    OAI2BB1X2TS g1761(.A0N (n_30), .A1N (n_7), .B0 (n_32), .Y (n_100));
  endmodule
```

c. Chip_Serializer.v

```
  module Chip_Serializer(VDD, VSS, DVDD, DVSS, clk, rst, data_in,
     data_out, frame, dummy_in1, dummy_in2, dummy_in3);
    input VDD, VSS, DVDD, DVSS, clk, rst, dummy_in1, dummy_in2, dummy_in3;
    input [8:0] data_in;
    output data_out, frame;
    wire VDD, VSS, DVDD, DVSS, clk, rst, dummy_in1, dummy_in2, dummy_in3;
    wire [8:0] data_in;
    wire data_out, frame;
    wire [9:0] dataout;
    wire [8:0] data_in_w;
    wire clk_w, dout_w, n_10, n_11, n_12, rd_in_encode, rd_out_encode,
        rst_w;
    wire start_frame;
    Serializador Serializador1(.par_in (dataout), .clk (clk_w), .rst
        (rst_w), .rd_in (rd_out_encode), .rd_out (rd_in_encode),
        .ser_out (dout_w), .start_frame (start_frame));
    encode encode1(.datain (data_in_w), .dispin (rd_in_encode), .dataout
        (dataout), .dispout (rd_out_encode));
    CLKBUFX2TS cdn_loop_breaker(.A (VSS), .Y (n_12));
    CLKBUFX2TS cdn_loop_breaker1(.A (VSS), .Y (n_11));
    CLKBUFX2TS cdn_loop_breaker2(.A (VSS), .Y (n_10));
    PDVDD pad_DVDD(.DVDD (DVDD));
    PDVSS pad_DVSS(.DVSS (DVSS));
    PVDD pad_VDD(.VDD (VDD));
    PVSS pad_VSS(.VSS (VSS));
    PCORNER se_pcorner();
    PCORNER sw_pcorner();
    PCORNER ne_pcorner();
    PCORNER nw_pcorner();
    PIC pad_clk(.P (clk), .IE (VDD), .Y (clk_w));
    PIC pad_data_in0(.P (data_in[0]), .IE (VDD), .Y (data_in_w[0]));
    PIC pad_data_in1(.P (data_in[1]), .IE (VDD), .Y (data_in_w[1]));
    PIC pad_data_in2(.P (data_in[2]), .IE (VDD), .Y (data_in_w[2]));
    PIC pad_data_in3(.P (data_in[3]), .IE (VDD), .Y (data_in_w[3]));
    PIC pad_data_in4(.P (data_in[4]), .IE (VDD), .Y (data_in_w[4]));
    PIC pad_data_in5(.P (data_in[5]), .IE (VDD), .Y (data_in_w[5]));
    PIC pad_data_in6(.P (data_in[6]), .IE (VDD), .Y (data_in_w[6]));
    PIC pad_data_in7(.P (data_in[7]), .IE (VDD), .Y (data_in_w[7]));
    PIC pad_data_in8(.P (data_in[8]), .IE (VDD), .Y (data_in_w[8]));
    POC4C pad_dout(.A (dout_w), .P (data_out));
    PIC pad_dummy1(.P (dummy_in1), .IE (n_12), .Y (VSS));
    PIC pad_dummy2(.P (dummy_in2), .IE (n_11), .Y (VSS));
    PIC pad_dummy3(.P (dummy_in3), .IE (n_10), .Y (VSS));
    POC4C pad_frame(.A (start_frame), .P (frame));
    PIC pad_rst(.P (rst), .IE (VDD), .Y (rst_w));
endmodule
```

C. IOC file to assign the position of the pads for EDI

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

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

#Pad: ne_corner NE

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

   ( left
    ( locals ring_number = 1)
( inst name= "pad_data_in4" )
( inst name= "pad_data_in5" )
( inst name= "pad_data_in6" )
( inst name= "pad_data_in7" )
( inst name= "pad_dummy3" )
 )

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

 ( bottom
    ( locals ring_number = 1    )

( inst name= "pad_data_in0" )
( inst name= "pad_data_in1" )
( inst name= "pad_data_in2" )
( inst name= "pad_data_in3" )
( inst name= "pad_dummy2" )
)


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

 ( right
    ( locals ring_number = 1    )
```

```
( inst name= "pad_DVDD" )
( inst name= "pad_VDD" )
( inst name= "pad_dout" )
( inst name= "pad_dummy1" )
( inst name= "pad_data_in8" place_status = placed)
)

( topright
      (locals ring_number = 1)
( inst name= "ne_pcorner")
   )
( top
     ( locals ring_number = 1    )
( inst name= "pad_clk" )
( inst name= "pad_rst" )
( inst name= "pad_frame" )
( inst name= "pad_DVSS" )
( inst name= "pad_VSS" )
)
)
```

D. RTL code of Encoder

```
// Chuck Benz, Hollis, NH   Copyright (c)2002
//
// The information and description contained herein is the
// property of Chuck Benz.

module encode (datain, dispin, dataout, dispout) ;
 input [8:0]   datain ;
 input       dispin ;  // 0 = neg disp; 1 = pos disp
 output [9:0]              dataout ;
 output     dispout ;


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

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

 // The 5B/6B encoding

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

 // pds16 indicates cases where d-1 is assumed + to get our encoded value
 wire pd1s6 = (ei & di & !ci & !bi & !ai) | (!ei & !l22 & !l31) ;
 // nds16 indicates cases where d-1 is assumed - to get our encoded value
 wire nd1s6 = ki | (ei & !l22 & !l13) | (!ei & !di & ci & bi & ai) ;

 // ndos6 is pds16 cases where d-1 is + yields - disp out - all of them
 wire ndos6 = pd1s6 ;
 // pdos6 is nds16 cases where d-1 is - yields + disp out - all but one
 wire pdos6 = ki | (ei & !l22 & !l13) ;


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

```
                                (dispin ? (!ei & di & l31) : (ei & !di & l13))) ;

  wire fo = fi & ! alt7 ;
  wire go = gi | (!fi & !gi & !hi) ;
  wire ho = hi ;
  wire jo = (!hi & (gi ^ fi)) | alt7 ;

  // nd1s4 is cases where d-1 is assumed - to get our encoded value
  wire nd1s4 = fi & gi ;
  // pd1s4 is cases where d-1 is assumed + to get our encoded value
  wire pd1s4 = (!fi & !gi) | (ki & ((fi & !gi) | (!fi & gi))) ;

  // ndos4 is pd1s4 cases where d-1 is + yields - disp out - just some
  wire ndos4 = (!fi & !gi) ;
  // pdos4 is nd1s4 cases where d-1 is - yields + disp out
  wire pdos4 = fi & gi & hi ;

  // only legal K codes are K28.0->.7, K23/27/29/30.7
  //          K28.0->7 is ei=di=ci=1,bi=ai=0
  //          K23 is 10111
  //          K27 is 11011
  //          K29 is 11101
  //          K30 is 11110 - so K23/27/29/30 are ei & l31
  wire illegalk = ki &
                          (ai | bi | !ci | !di | !ei) & // not K28.0->7
                          (!fi | !gi | !hi | !ei | !l31) ; // not K23/27/29/30.7

  // now determine whether to do the complementing
  // complement if prev disp is - and pd1s6 is set, or + and nd1s6 is set
  wire compls6 = (pd1s6 & !dispin) | (nd1s6 & dispin) ;

  // disparity out of 5b6b is disp in with pdso6 and ndso6
  // pds16 indicates cases where d-1 is assumed + to get our encoded value
  // ndos6 is cases where d-1 is + yields - disp out
  // nds16 indicates cases where d-1 is assumed - to get our encoded value
  // pdos6 is cases where d-1 is - yields + disp out
  // disp toggles in all ndis16 cases, and all but that 1 nds16 case

  wire disp6 = dispin ^ (ndos6 | pdos6) ;

  wire compls4 = (pd1s4 & !disp6) | (nd1s4 & disp6) ;
  assign dispout = disp6 ^ (ndos4 | pdos4) ;

  assign dataout = {(jo ^ compls4), (ho ^ compls4),
                            (go ^ compls4), (fo ^ compls4),
                            (io ^ compls6), (eo ^ compls6),
                            (d0 ^ compls6), (co ^ compls6),
                            (bo ^ compls6), (ao ^ compls6)} ;

endmodule
```

E. Testbench for Serial Transmitter

```
module top();
reg      clk;
reg      rst;
integer filehandeler;
reg [9:0]data;
reg rd_out;
reg set_rd;
reg start_pkt;
reg [9:0] compare_data;
reg [3:0] counter;
wire start_frame;
wire      data_serial;

initial begin
   clk = 0;
   data = 0;
   set_rd = 0;
   rd_out  = 0;
   filehandeler = $fopen("file.txt","w");
        compare_data = 0;
        counter = 0;
end
always begin
        #5 clk = !clk;
end


initial begin
        START;
   for (bit [10:0]gen_data = 0; gen_data<1024; gen_data = gen_data+1) begin
      data = gen_data;
                set_rd = !rd_out;
                counter = 0;
                repeat (10) begin
                        @(posedge clk);
                        #1
                        compare_data[counter] = data_serial;
                        $fdisplay(filehandeler, "\n -Info- Generate data = %b, Serial data bit = %d, Run
disparity value %d, start_frame %d",data,data_serial, rd_out, start_frame);
                        if (counter == 9) begin
                                if (compare_data == data) begin
                                        $fdisplay(filehandeler, "\n -Info- Match Generate data =
%d,  Expected data = %d, Serial data bit = %d, Run disparity %d",data,compare_data,data_serial,rd_out);
                                end
                                else begin
                                        $fdisplay(filehandeler, "\n -Error- No Match- Valor del
dato generado = %d, valor de serial bit = %d",data, compare_data);
                                end
                        end
                        counter = counter + 1;
                end
   end
   $fclose(filehandeler);
   $finish ;
end

Serializador Serializador1(.par_in(data), .clk(clk), .rst(rst), .rd_in(set_rd), .rd_out(rd_out),
.ser_out(data_serial), .start_frame(start_frame));


task START;
        begin
        rst = 1;
```

```
        @(posedge clk);
       #1;
        rst = 0;
        end
endtask
endmodule
```

F. Script for EDI implementation

```
## script Full EDI Flow ##

## import design ##

set_global _enable_mmmc_by_default_flow      $CTE::mmmc_default
suppressMessage ENCEXT-2799
win
set ::TimeLib::tsgMarkCellLatchConstructFlag 1
set conf_qxconf_file NULL
set conf_qxlib_file NULL
set defHierChar /
set distributed_client_message_echo 1
set gpsPrivate::dpgNewAddBufsDBUpdate 1
set gpsPrivate::lsgEnableNewDbApiInRestruct 1
set init_gnd_net {VSS DVSS}
set init_io_file ../serial_arm.ioc
set init_lef_file {/opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-
x/lef/ibm13_8lm_2thick_3rf_tech.lef /opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-
x/lef/ibm13rflpvt_macros.lef /opt/libs/ARM/IB03IG502-FB-00000-r0p0-
00rel0/aci/io/lef/iogpil_cmrf8sf_rvt_M2_3_3.lef}
set init_mmmc_file ../Typ_WC.view
set init_pwr_net {VDD DVDD}
set init_verilog ../Chip_Serial_Transmisor_m.v
set lsgOCPGainMult 1.000000
set pegDefaultResScaleFactor 1.000000
set pegDetailResScaleFactor 1.000000
set timing_library_float_precision_tol 0.000010
set timing_library_load_pin_cap_indices {}
set tso_post_client_restore_command {update_timing ; write_eco_opt_db ;}
init_design

# Defining process mode
setDesignMode -process 130

## Floor plan definition

getIoFlowFlag
setFPlanRowSpacingAndType 3.6 2
setIoFlowFlag 0
floorPlan -site IBM13SITE -s 331.4 331.4 16.8 16.8 16.8 16.8
uiSetTool select
getIoFlowFlag

## Defining Power Global Nets

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

## Adding power Ring

addRing -skip_via_on_wire_shape Noshape -skip_via_on_pin Standardcell -stacked_via_top_layer MA -
type core_rings -jog_distance 0.2 -threshold 0.2 -nets {VDD VSS} -follow io -stacked_via_bottom_layer M1 -
layer {bottom M1 top M1 right M2 left M2} -width 4 -spacing 5 -offset 2

## Adding Horizontal lines

sroute -connect { blockPin padPin padRing corePin floatingStripe } -layerChangeRange { M1 MA } -
blockPinTarget { nearestTarget } -padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -
corePinTarget { firstAfterRowEnd } -floatingStripeTarget { blockring padring ring stripe ringpin blockpin
```

```
followpin } -allowJogging 1 -crossoverViaLayerRange { M1 MA } -allowLayerChange 1 -nets { VDD VSS } -
blockPin useLef -targetViaLayerRange { M1 MA }

## Adding strip lines


addStripe -skip_via_on_wire_shape Noshape -block_ring_top_layer_limit M3 -max_same_layer_jog_length
8 -padcore_ring_bottom_layer_limit M1 -number_of_sets 4 -skip_via_on_pin Standardcell -
stacked_via_top_layer MA -padcore_ring_top_layer_limit M3 -spacing 5 -xleft_offset 103.9 -xright_offset
103.9 -merge_stripes_value 0.2 -layer M2 -block_ring_bottom_layer_limit M1 -width 4 -nets {VDD VSS} -
stacked_via_bottom_layer M1


## Place Standard Cells

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

## Clock synthesis CTS

# Use the FE-CTS
setCTSMode -engine ck

# Create clock tree using the clock buffers list:
createClockTreeSpec -bufferList {CLKBUFX2TS CLKBUFX3TS CLKBUFX4TS CLKBUFX6TS CLKBUFX8TS
CLKBUFX12TS CLKBUFX16TS CLKBUFX20TS} -file ../Clock.ctstch

# Display Clock Tree
displayClockTree -skew -allLevel -preRoute

# Edit the .ctstch that was created to complete constraints...

## Route design with nano route

setNanoRouteMode -quiet -timingEngine {}
setNanoRouteMode -quiet -routeWithTimingDriven 1
setNanoRouteMode -quiet -routeWithSiDriven 1
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -routeTdrEffort 10
setNanoRouteMode -quiet -drouteStartIteration default
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven true
setNanoRouteMode -quiet -routeWithSiDriven true
routeDesign -globalDetail


####### Run optimization script based on class script ###########

Puts "Timing the design before CTS"

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

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

Puts "Running CTS"
dbDeleteTrialRoute
```

```
clockDesign -specFile ../Clock.ctstch -outDir clock_report -fixedInstBeforeCTS
Puts "Finished running CTS"

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

Puts "Setting Optimizaiton Mode Options for DRV fixes"
setOptMode -fixFanoutLoad true
setOptMode -fixDRC true
setOptMode -addInstancePrefix postCTSdrv
setOptMode -setupTargetSlack 0.05


Puts "Optimizing for DRV"
optDesign -postCTS -drv

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

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

Puts "Optimizing for Setup"
optDesign -postCTS

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

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

Puts "Routing the Design"
setNanoRouteMode -quiet -timingEngine {}
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -routeTdrEffort 10
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven True
setNanoRouteMode -quiet -routeWithSiDriven True
routeDesign -globalDetail

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

## Verify connectivity,DRC & Geometry

## Geometry

setVerifyGeometryMode -area { 0 0 0 0 } -minWidth true -minSpacing true -minArea true -sameNet true -
short true -overlap true -offRGrid false -offMGrid true -mergedMGridCheck true -minHole true -
implantCheck true -minimumCut true -minStep true -viaEnclosure true -antenna false -insuffMetalOverlap
true -pinInBlkg false -diffCellViol true -sameCellViol false -padFillerCellsOverlap true -routingBlkgPinOverlap
true -routingCellBlkgOverlap true -regRoutingOnly false -stackedViasOnRegNet false -wireExt true -
useNonDefaultSpacing false -maxWidth true -maxNonPrefLength -1 -error 1000
verifyGeometry
setVerifyGeometryMode -area { 0 0 0 0 }

## DRC
```

```
setVerifyGeometryMode -area { 0 0 0 0 }
verify_drc -report AccumulatorBehavioral.drc.rpt -limit 1000

## Connectivity

verifyConnectivity -type all -error 1000 -warning 50

# report power

set_power_analysis_mode -method static -analysis_view Typ_Analysis_View -corner max -
create_binary_db true -write_static_currents true -honor_negative_energy true -ignore_control_signals
true
set_power_output_dir -reset
set_power_output_dir power_report
set_default_switching_activity -reset
set_default_switching_activity -input_activity 0.2 -period 4.0 -seq_activity .5 -clock_gates_output 0
read_activity_file -reset
set_power -reset
set_powerup_analysis -reset
set_dynamic_power_simulation -reset
report_power -rail_analysis_format VS -outfile power_report/serial_top.rpt


# save design

saveDesign serial_opt_script.enc
```

G. *Timing report log*

```
path  1:

     Pin             Type      Fanout                    Load        Slew        Delay Arrival
                               (fF)                      (ps)        (ps)        (ps)
-------------------------------------------------------------------------------
(clock 250MHz_CLK)      launch                                                  0 R
                       latency                          +150                   150 R
(IDelay)                                 ext delay                           +200          350 R
VDD                     in port          20 1139.1   0   +0                  350 R
pad_data_in0/IE                                                                +0          350
pad_data_in0/Y          (P)  PIC    3   31.1             50                    +547          897 R
encode1/datain[0]
 g1637/B                                                                       +0          897
 g1637/Y                NOR2X4TS         1   8.2   50                         +57          954 F
 drc/A                                                                         +0          954
 drc/Y                CLKBUFX8TS       4    30.7  120                         +194         1148 F
 g1601/AN                                                                      +0         1148
 g1601/Y              NAND3BX2TS    1    7.0              142                  +238         1386 F
 g1590/B                                                                       +0         1386
 g1590/Y              AND2X4TS       3   13.8     77     +229                  1616 F
 g1580/B                                                                       +0         1616
 g1580/Y              AND3X2TS       1   13.0    127    +296                  1912 F
 g1578/A                                                                       +0         1912
 g1578/Y              NOR2X4TS       2   11.4   137   +140                    2052 R
 g1571/A                                                                       +0         2052
 g1571/Y              NAND2X2TS      1    9.9   116   +128                    2181 F
 g1565/A                                                                       +0         2181
 g1565/Y              CLKXOR2X4TS    3   17.0   148   +383                    2564 F
 g1563/A                                                                       +0         2564
 g1563/Y              NAND2X2TS      1    7.2   106   +125                    2688 R
 g1561/B0                                                                      +0         2688
 g1561/Y              OA21X4TS       4   24.5   146   +259                    2948 R
 g1560/A                                                                       +0         2948
 g1560/Y              INVX2TS        1    7.6              66    +103         3051 F
 g1551/A                                                                       +0         3051
 g1551/Y              CLKXOR2X2TS    1    5.6   127   +317                    3368 F
encode1/dataout[8]
Serializador1/par_in[8]
 g240/B                                                +0                    3368
 g240/Y              MX2X1TS        1    5.1   130   +366                    3734 F
 par_reg_reg[8]/D     DFFRHQX1TS                       +0                    3734
 par_reg_reg[8]/CK    setup                      28   +305                  4039 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 250MHz_CLK)      capture                                              4000 R
                        latency                  +150                        4150 R
                        uncertainty               -17                        4133 R
-------------------------------------------------------------------------------
Cost Group   : 'I2C' (path_group 'I2C')
Timing slack :     94ps
Start-point  : VDD
End-point    : Serializador1/par_reg_reg[8]/D
============================================================
```

H. Analisis View

Version:1.0 MMMC View Definition File

Do Not Remove Above Line

create_rc_corner -name RC_Corner_Typ -T {25} -preRoute_res {1.0} -preRoute_cap {1.0} -preRoute_clkres {0.0} -preRoute_clkcap {0.0} -postRoute_res {1.0} -postRoute_cap {1.0} -postRoute_xcap {1.0} -postRoute_clkres {0.0} -postRoute_clkcap {0.0}

create_rc_corner -name RC_Corner_WC -T {125} -preRoute_res {1.0} -preRoute_cap {1.0} -preRoute_clkres {0.0} -preRoute_clkcap {0.0} -postRoute_res {1.0} -postRoute_cap {1.0} -postRoute_xcap {1.0} -postRoute_clkres {0.0} -postRoute_clkcap {0.0}

create_library_set -name Typ_LibSet -timing {/opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-x/synopsys/scx3_cmos8rf_lpvt_tt_1p2v_25c.lib /opt/libs/ARM/IB03IG502-FB-00000-r0p0-00rel0/aci/io/synopsys/iogpil_cmrf8sf_rvt_tt_1p2v_2p5v_25c.lib}

create_library_set -name WC_LibSet -timing {/opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-x/synopsys/scx3_cmos8rf_lpvt_ss_1p08v_125c.lib /opt/libs/ARM/IB03IG502-FB-00000-r0p0-00rel0/aci/io/synopsys/iogpil_cmrf8sf_rvt_ss_1p08v_2p3v_125c.lib}

create_constraint_mode -name General_Constraint_Mode -sdc_files {../Chip_serializer_m.sdc}

create_delay_corner -name Typ_DelayCorner -library_set {Typ_LibSet} -rc_corner {RC_Corner_Typ}

create_delay_corner -name WC_DelayCorner -library_set {WC_LibSet} -rc_corner {RC_Corner_WC}

create_analysis_view -name Typ_Analysis_View -constraint_mode {General_Constraint_Mode} -delay_corner {Typ_DelayCorner}

create_analysis_view -name WC_Analysis_View -constraint_mode {General_Constraint_Mode} -delay_corner {WC_DelayCorner}

set_analysis_view -setup {WC_Analysis_View} -hold {Typ_Analysis_View}

I. Lis of archives of Serializer module design in Virtuoso cadence

Name of archive	Library	Views
8B/10B encoder	scx3_cmos8rf_lpvt_tt iogpil_cmrf8sf_rvt_tt	Schematic, layout
Serilization	scx3_cmos8rf_lpvt_tt iogpil_cmrf8sf_rvt_tt	Schematic, layout
Serializer	scx3_cmos8rf_lpvt_tt iogpil_cmrf8sf_rvt_tt	Schematic, layout
Test_bench	scx3_cmos8rf_lpvt_tt iogpil_cmrf8sf_rvt_tt	N,A