2016-08

Design, Implementation and Verification of a Deserializer Module for a SerDes Mixed Signal System on Chip in 130 nm CMOS Technology

Rivas-Villegas, Rogelio

Instituto Tecnológico y de Estudios Superiores de Occidente

Especialidad en Diseño de Sistemas en Chip

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

Departamento de Electrónica Sistemas e Informática

Design, Implementation and Verification of a Deserializer Module for a SerDes Mixed Signal System on Chip in 130 nm CMOS Technology

Tesina para obtener el grado de:

Especialista en diseño de sistemas en chip

Presenta

Rogelio Rivas Villegas

Nombre de los asesores del proyecto:

Dr. Víctor Avendaño Fernández, Mtro. Cuauhtémoc Rafael Aguilera Galicia

Guadalajara, Jalisco, Agosto 2016

ACKNOWLEDGEMENTS

The author would like to thank to his classmates César, Ernesto, Efraín and Joel in the specialty program. Your support and help were critical to the development of this project.

I want to thank PNPC-CONACYT for funding this research. I would also like to acknowledge V. Avendaño, C. Aguilera, E. Juárez, E. Martínez and A. Girón.

My sincere gratitude to Cata, Gera, Kenneth and Adrián. With their patience and motivation, they keep on telling me time and time again “don’t ever give up”.

Last but not the least, I would like to thank my parents for supporting me throughout this specialty, just like they have been performing in every single project in my life.

ABSTRACT

This document presents the design, verification and physical implementation process of a digital receiver of a mixed signal System on Chip called SerDes. This circuit consists in a communication system based on the PCI Express standard. The ITESO TV2 project walks through the full logic and physical design of an integrated circuit, starting from the specs definitions to the generation of the output files that are delivered to the MOSIS manufacturing team. A detailed description of the digital deserializer and decoding microarchitecture is presented, followed by the physical layout equivalent circuit implementation.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS 3
ABSTRACT 5
TABLE OF CONTENTS 7
TABLE OF FIGURES 11
CHAPTER 1. Background of a PCIExpress SerDes System 14
1.1 Introduction 14
1.2 SerDes VT1 previous work 16
CHAPTER 2. Design and Implementation of the SerDes Microarchitecture. 18
2.1 Digital Receiver Microarchitecture 18
2.1.1 Digital Rx 18
2.1.2 Clock divider 20
2.1.3 Input Buffer 20
2.1.4 Deserializer 21
2.1.5 Clock Divider Recovery (CDR) 22
2.1.6 Pipelined 8B10B decoder 23
2.2 RTL fixes and enhancements 25
2.2.1 Edge detector and best sampling enhanced technique 26
2.2.2 Number of samples per bit reduction 27
2.2.3 Pipelined 8B10B decoder 30
2.2.4 One bit adder microarchitecture changes 33
2.3 Logic Synthesis 34
2.4 Gate level simulation 35
CHAPTER 3. Physical Synthesis 39
3.1 Basic scripts and layout 39
3.2 Timing analysis 42
3.2.1 Negative slack fixes 42
3.3 PADS inclusion analysis 44
3.4 Geometry, DRC, VLS and equivalence verification 48
3.5 Import layout to Virtuoso 53
3.5.1 GRLOGIC issues 54
3.5.2 N-well connectivity errors 56
3.5.3 Latch up violations 58

3.5.4 Miscellaneous antenna violations 60
3.5.5 LVS analysis 62
Conclusions 64
References 66
Appendix 68
A. Verilog source files 68
a. digtalRx.v 68
b. clock_divider.v 68
c. CDR.v 69
d. deserializer.v 72
e. inputBuffer.v 74
f. decodePipe.v 75
g. adderA.v 87
h. adderB.v 87
i. deserializerTB.sv 88
B. Logic synthesis files 91
a. Netlist 91
b. Netlist with pads 107
c. Logic synthesis script 124
d. Constraint file 130
C. Physical Synthesis 132
a. Automation script 132
b. Analysis View 137
LIST OF FILES 139

TABLE OF FIGURES

Figure 1 High level abstraction of the SerDes [3] 15
Figure 2 Parallel In Serial Out 15
Figure 3 Serial In Parallel Out 16
Figure 4 RTL diagram of the digital receiver....19
Figure 5 RTL diagram of the clock divider module 20
Figure 6 RTL diagram of the input buffer used for load balancing. 21
Figure 7 Deserializer block diagram 22
Figure 8 Clock and data recovery module block representation 23
Figure 9 8B10B decoder with pipelined architecture 24
Figure 10 Best sample is aligned to the center of the bit 26
Figure 11Best sample index update 27
Figure 12 A negative slack was obtained using the fast clock in the sampling section. 27
Figure 13 Positive slack after the sampling phases reduction. 30
Figure 14 Critical path showing a negative slack timing violation in the decoding stage. 31
Figure 15 Timing slack after pipelining techniques. 33
Figure 16 Daisy-chain adder 33
Figure 17 One-bit adder proposed 34
Figure 18 Digital receiver logic synthesis diagram. 35
Figure 19 Verilog libraries added for the standard cells and IO pads. 35
Figure 20 Test values injected into the DUT 36
Figure 21 Digital receiver top module simulation. 36
Figure 22 Phases reduction waveforms. 36
Figure 23 Best sampling technique after oversampling the incoming serial stream. 37
Figure 24 Pipelined output from the 8B10B decoder. 37
Figure 25 Floorplan dimensions 40
Figure 26 Physical synthesis of the digital receiver 41
Figure 27 Clock tree physical synthesis 42
Figure 28 Original tool generated netlist....43
Figure 29 Modified netlist with bigger cells 44
Figure 30 Floorplan dimensions included some pads to emulate the final implementation of the chip. 45
Figure 31 Physical synthesis of the proposed model with IO pads. 46
Figure 32 Clock tree synthesis. 47
Figure 33 Clock tree synthesis for the input reference clock and the resultant slow clock, ...48
Figure 34 Geometry verification results 49
Figure 35 Connectivity verification results 50
Figure 36 DRC verification results....50
Figure 37 Geometry verification 51
Figure 38 DRC verification 52
Figure 39 Connectivity verification....52
Figure 40 Virtuoso final layout 53
Figure 41 GRLOGIC error signature 54
Figure 42 GRLOGIC suround shape 55
Figure 43 Error signature of the n-well violations....56

Figure 44 Layout n-well connectivity issue 57
Figure 45 Layout manual n-well fix.....57
Figure 46 Latchup violation signature.....58
Figure 47 Latchup error layout 59
Figure 48 Fixed layout connecting the RX to M1.....60
Figure 49 Diode inserted as possible fix for the antenna violation 61
Figure 50 Final DRC report.....61
Figure 51 LVS translation issues.....62

CHAPTER 1. Background of a PCIExpress SerDes System

Novel electronic systems need to meet new requirements due to the rapid progress of semiconductor technologies, so the specifications of the devices face new challenges to ensure correct performance in various areas of the electronics industry. Communication systems have been impacted by this rapid growth, so they have implemented various techniques to meet these needs in data transmission. The SerDes is part of these efforts, so this chapter provides a general description of this system.

1.1 Introduction

One of the devices used in high-speed communications is the SerDes (serializer/deserializer) [1]. The SerDes device allows the transmission of parallel data with certain length of bits in a bus with fewer bits often single bit (serializer), sending through a channel to a receiver that performs the inverse operation converting the incoming stream from serial to parallel format (deserializer). This means that there are savings in system complexity, costs reduction, area and power consumption.

The main blocks of the SerDes consist in a PISO (Parallel Input Serial Output) module followed by a SIPO (Serial Input Parallel Output) module, however there are additional functional blocks that allow the execution of the tasks of this device.

As we have a system input parallel data bus, so that data is loaded into the system in parallel bit registers in the same clock edge.

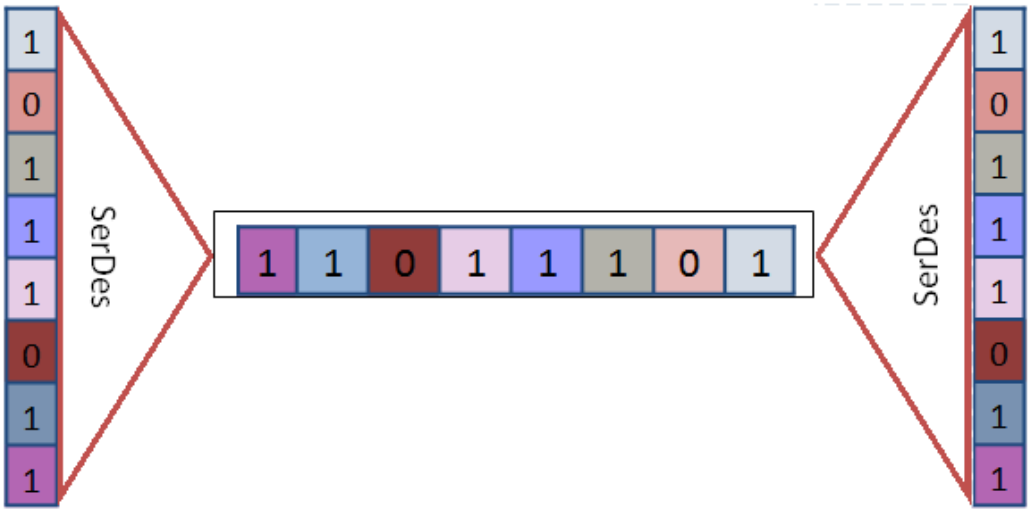

Figure 1 High level abstraction of the SerDes [3]

The parallel data needs to be encoded according to different standards. This helps to prepare the data for an adequate serial transmission and also improves the quality of the signal for its reception in the deserializer block. The designer can choose which encoding technique is required depending on the characteristics of the transmission.

After the data encoding we proceed to the serialization of the encoded bit word. This data stream is transmitted through the channel to the receiver using a differential output buffer.

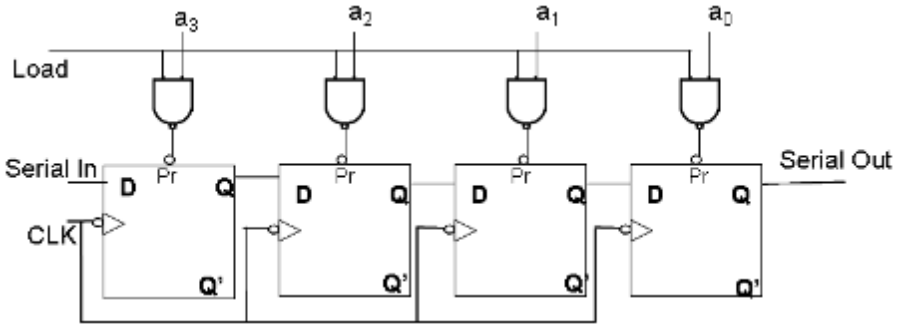

Figure 2 Parallel In Serial Out

Serial data received by deserializer are processed once a differential input buffer converts them into digital inputs. A clock and data recovery (CDR) block is required to extract the data and clock from the incoming data steam.. This stage has a strong dependence on the coding stage has been carried out in the transmitting device, since the retrieval of information requires a minimum transition density bit for the process to avoid certain errors in the reception.

After the data is received, the next step is the serial-to-parallel conversion. This is accomplished by shift registers that store the incoming serial data stream as a parallel bit vector structure to be read as a code word of a certain number of bits (Fig. 3).

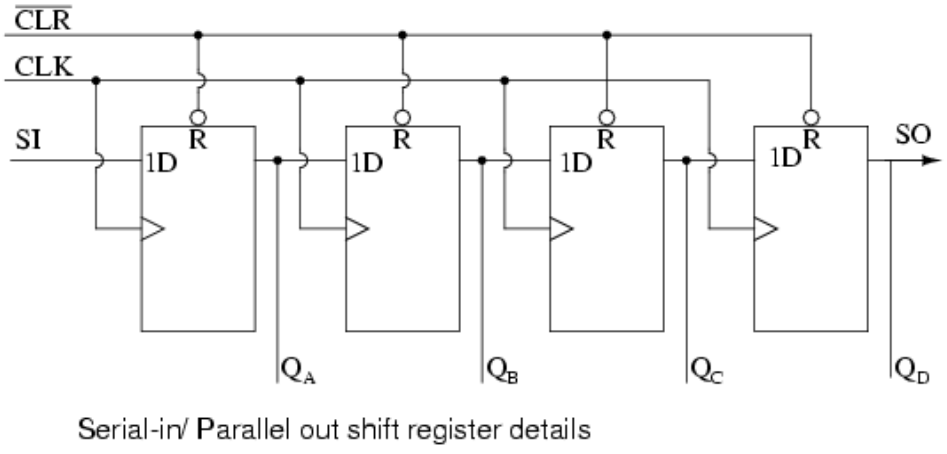

Figure 3 Serial In Parallel Out

The bits obtained from the shift registers are sent to the output buffer, typically accompanied by a synchronization signal that indicates when the data in the port are valid for processing in later stages.

1.2 SerDes VT1 previous work

For the development of this project we chose the PCI Express Gen 1 technology as described in the introductory section. This communication standard is aligned to the goals of the specialty, since the physical layer of this data transmission protocol can be implemented as a SoC. It is intended to be a protocol for high-speed communications [4]. Furthermore, it implies the implementation of a mixed signal system in which some analog and digital components are interconnected to make possible the signal integrity and data consistency in the transmission process.

Based on the previous statements we defined the following specifications for this project:

- ARM standard cells for 130 nm technology
- Supply voltage of 1.2 V
- Operating frequency up to 1.25 GHz
- 8B10B decoding scheme
- The verification rules will be the ones contained in the Cadence software.
- DIP 40 package restrictions provided by MOSIS.
- Area restrictions are 1.5x1.5mm provided by MOSIS.
- The project should be ready by July 30, 2016.
- Mentor Graphics tools for layout verification.
- Intel tools for RTL design

CHAPTER 2. Design and Implementation of the SerDes Microarchitecture.

Based on the specifications set by the system clients, a microarchitecture was designed and described in Verilog to obtain a synthesizable model of the digital receiver. A hierarchy of modules will be explained in the following chapter, describing the design criteria, input/output ports and a block diagram representation of the components that perform the data recovery, serial to parallel conversion and decoding processes.

2.1 Digital Receiver Microarchitecture

2.1.1 Digital Rx

The top module of the digital section is named digital receiver (Rx). It includes the deserializer, decoder and clock divider blocks and all the modules used to interconnect each unit. It receives the serial data from the analog receiver, recovers the signal clock, samples the data to convert the serial stream into a parallel data and performs the required decoding technique to obtain the transmitted data.The RTL code of this module was used as input to generate the Logic and Physical Synthesis. This module interacts with the analog receiver and with the user.

The hierarchy of this digital receiver is conformed by the following functional units:

- Clock divider: Receives and manages the reference clock to generate a four times slower clock frequency for each of the modules.
- Input buffer: Interface between the analog and digital parts of the receiver. It permits a correct load balancing for the analog stage.
- Deserializer: Samples the input data and converts the serial input stream into a parallel encoded data.
- CDR: The clock data recovery module receives the serial stream and determines the best sampling timing, recovering the embedded clock signal as well.
- Decoder: It is an 8B10B pipelined decoder that converts the 10 bit parallel encoded frame into an 8 bit paralled data.

The input and output ports of this module are the following:

Name	Direction	Width	Description
rst	input	1 bit	Asynchronous active low reset shared with the rest of the chip.
clk	input	1 bit	Reference clock running at 500 MHz.
a_rx	input	1 bit	Serial input received from the analog module. This signal contains the serialized data and embedded clock.
a_rx_n	input	1 bit	Complementary serialized input signal. This is required in order to meet the load balancing requirements for the analog receiver.
disparity_d	input	1 bit	Initial disparity. It is tied to '0' for this implementation.
data_out	output	8 bit	Final recovered parallel data.
data_valid_pipe	output	1 bit	Data valid signal for the output data.
dispout	output	1 bit	Resultant disparity obtained from the decoding process.
code_err	output	1 bit	Flag to indicate that an error occurred during the decoding process.
disp_err	output	1 bit	Flag to indicate that an error occurred during the disparity calculation process.
clock_4f	output	1 bit	Transmission clock obtained from the clock generator module.

A block diagram of this module was obtained as follows:

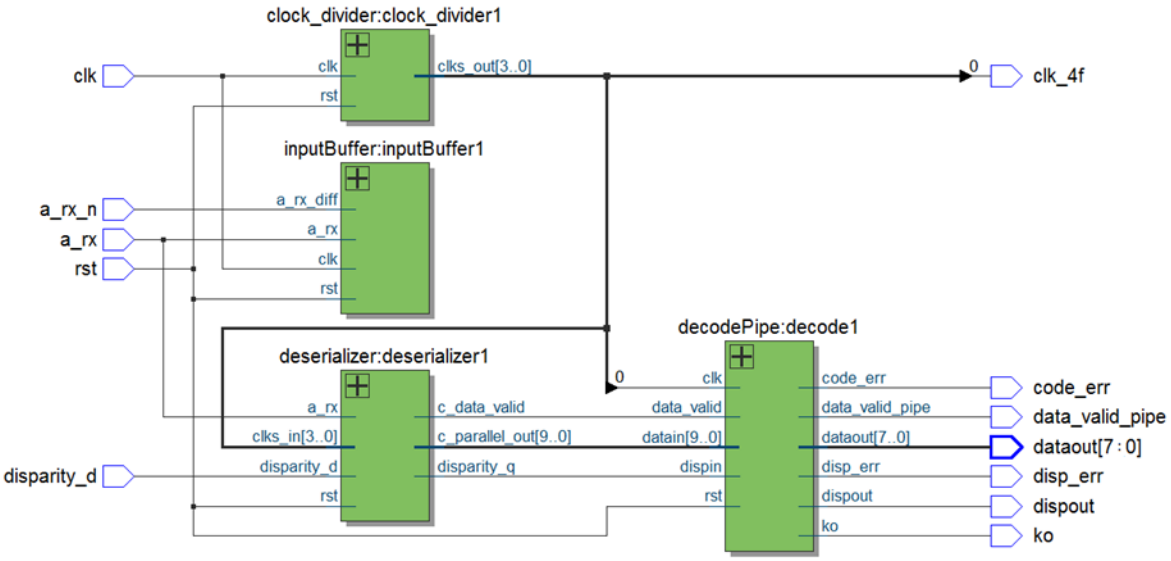

Figure 4 RTL diagram of the digital receiver

In the following sections the internal modules that perform the data recovering will be presented, describing each module as a stand-alone functional unit but also providing details of how do they fit into the whole system.

2.1.2 Clock divider

The clock divider is in charge of managing the different clock signals required for each clock domain of the system. It receives the reference clock from the external world and generates four different phases to oversample the input serial data. This module generates a system clock reducing the frequency of the input clock by a factor of four. The system clock is fed to the transmission and testing modules.

The main challenge for this module is the generation of the clock phases without compromising the timing requirements of the system. In this case we are using four clock phases in order to oversample the incoming bit based on the previous work of the SerDes [6]. This module is connected with the external world and feeds a processed clock signal into the system.

The input and output ports of this module are the following:

Name	Direction	Width	Description
rst	input	1 bit	Asynchronous active low reset shared with the rest of the chip.
clk	input	1 bit	Reference clock running at 500 MHz.
clocks_out	output	4 bit	Vector with four 125 MHz clock phases.

Block representation of this module is shown in Fig. 5:

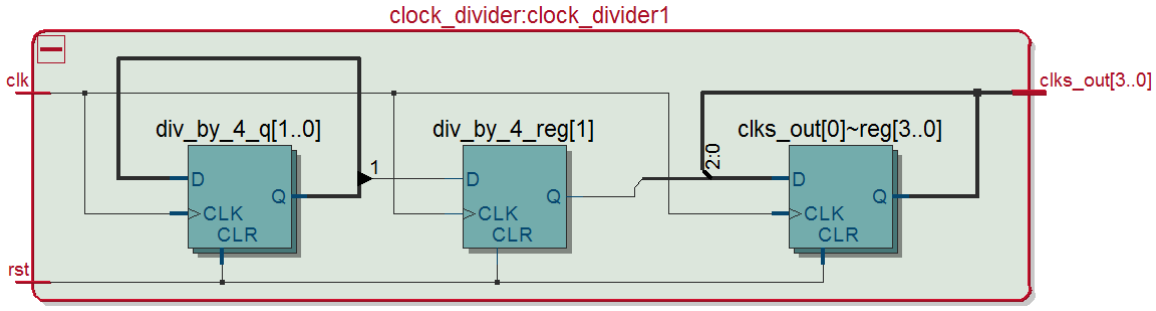

Figure 5 RTL diagram of the clock divider module

2.1.3 Input Buffer

The main purpose of this module is to perform the load balancing requirement set by the analog receiver. It stores in registers the value of the digitalized input signal coming from the analog differential amplifier and equalizer.

This module is connected directly with the analog stage, but the outputs are not connected to any other module.

The input and output ports of this module are the following:

Name	Direction	Width	Description
rst	input	1 bit	Asynchronous active low reset shared with the rest of the chip.
clk	input	1 bit	Reference clock running at 500 MHz.
a_rx	input	1 bit	Serial input received from the analog module. This signal contains the serialized data and embedded clock.
a_rx_diff	input	1 bit	Complementary serialized input signal. This is required in order to meet the load balancing requirements for the analog receiver.
a_rx_buff	output	1 bit	Register that stores the value of the serial input signal.
a_rx_diff_buff	output	1 bit	Register that stores the value of the complementary serial input signal.

The block representation of this module is shown in Fig. 6:

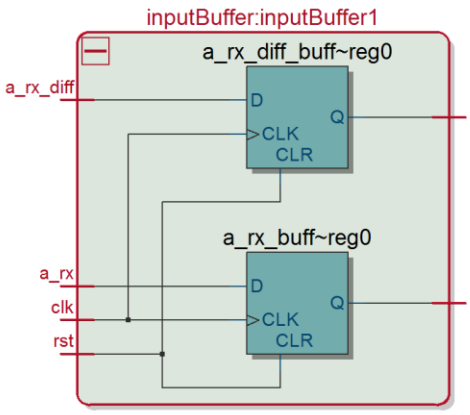

Figure 6 RTL diagram of the input buffer used for load balancing.

2.1.4 Deserializer

This module converts the incoming serial bit stream into an encoded parallel data. It is in charge of recovering the clock signal and data based on the incoming stream using an internal module CDR that will be described in the following section. Once tha data is deserialized, this module detects the start of a frame which is indicated by a reserved character.

It interacts with the analog receiver and injects encoded data into the pipeline decoding module. Its clock comes from the clock divider module.

The input and output ports of this module are the following:

Name	Direction	Width	Description
Rst	Input	1 bit	Asynchronous active low reset shared with the rest of the chip.
clks_in[3:0]	Input	4 bit	Reference clocks running at 125 MHz at four different equidistant phases.
a_rx	Input	1 bit	Serial input received from the analog module. This signal contains the serialized data and embedded clock.
disparity_d	Input	1 bit	Initial disparity. It is tied to '0' for this implementation.
c_parallel_out[9:0]	Output	10 bit	Encoded data after the serial-to-parallel conversion.
clock_out	Output	1 bit	Output clock from the deserializing stage.
disparity_q	Output	1 bit	Resultant disparity obtained from the decoding process.
c_data_valid	Output	1 bit	Data valid signal to indicate that a parallel data is available in the output registers.

Block representation of this module is shown in Fig. 7.

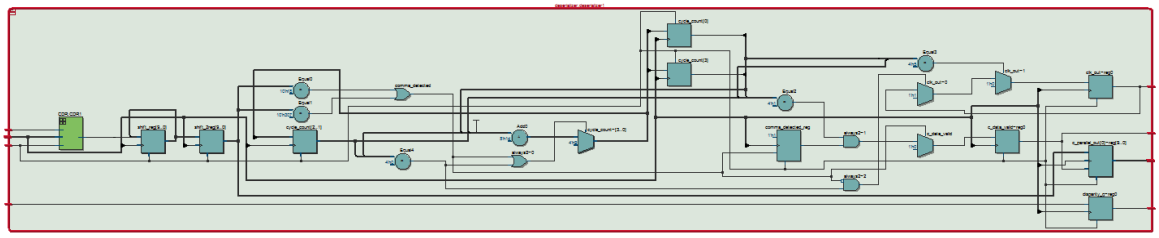

Figure 7 Deserializer block diagram

2.1.5 Clock Divider Recovery (CDR)

This module recovers the embedded clock from the input serial stream. The digital value of the incoming serial signal is obtained applying oversampling methods to add redundancy to the information retrieved. This oversampling technique allows us to reduce errors in the transmission and deserialization of the data by having more than one reference value to determine the phase of the signal with respect to the system clock, and also the logic value of the sampled stream.

The CDR module was modeled in a different entity in order to simplify and enhance its validation due to its complexity. This change allows to implement some improvements into the sampling techniques, catching more bugs and issues as well. It is instantiated inside the deserializer module.

The input and output ports of this module are the following:

Name	Direction	Width	Description
rst	Input	1 bit	Asynchronous active low reset shared with the rest of the chip.
clks_in[3:0]	Input	4 bit	Reference clocks running at 125 MHz at four different equidistant phases.
a_rx	Input	1 bit	Serial input received from the analog module. This signal contains the serialized data and embedded clock.
samp_reg	Output	1 bit	Best sample recovered from the incoming serial stream.

The block diagram of this module is as follows:

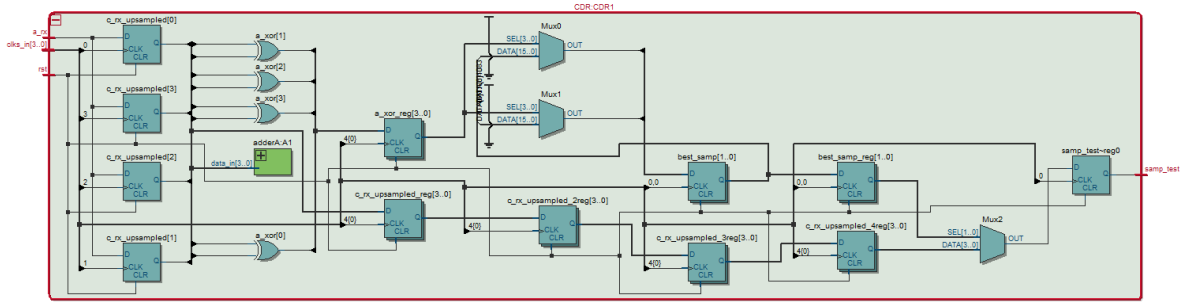

Figure 8 Clock and data recovery module block representation

2.1.6 Pipelined 8B10B decoder

An existing intellectual property (IP) [8] was used to perform the 8B10B decoding stage, required to recover the original data sent by the transmission entity. The code was enhanced and validated by adding pipelining techniques into the module. This was part of the requirements to meet the timing specifications of the system.

This module is the final stage of the digital reception. The input data is retrieved from the deserializer module and the output data and data valid signals are connected to the external connection pads.

The input and output ports of this module are the following:

Name	Direction	Width	Description
reset	input	1 bit	Asynchronous active low reset shared with the rest of the chip.
clock	input	1 bit	Reference clock running at 500 MHz.
data_valid	input	1 bit	Flag to indicate that the incoming data is a valid input for the pipeline.
datain[9:0]	input	10 bit	Encoded data after the serial-to-parallel conversion.
dispin	input	1 bit	Initial disparity. It is tied to '0' for this implementation.
dataout[7:0]	output	8 bit	Decoded parallel data.
data_valid_pipe	output	1 bit	Data valid signal for the output pipelined data.
dispout	output	1 bit	Resultant disparity obtained from the decoding process.
code_err	output	1 bit	Flag to indicate that an error occurred during the decoding process.
disp_err	output	1 bit	Flag to indicate that an error occurred during the disparity calculation process.

A block representation of this module is shown below:

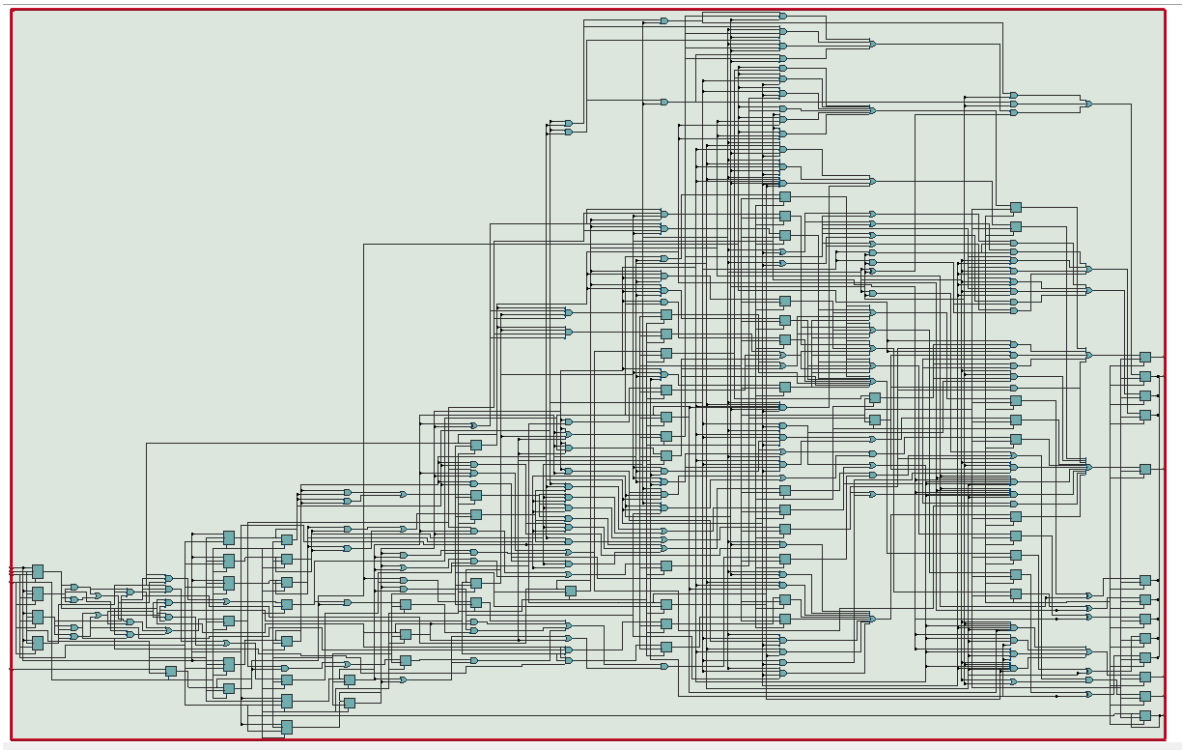

Figure 9 8B10B decoder with pipelined architecture

2.2 RTL fixes and enhancements

There were some design changes and enhancements based on different criteria. This criteria can be either to improve the existing bit recovery algorithm or to achieve the area and timing requirements of the chip. Most of them were applied in order to meet the timing requirements of the system. All these requirements should be met in order to get the expected functionality of the system without compromising the performance of the standard cells.

In order to get the timing analysis a script was run to check what the critical paths of the system are and what can be done at the design stage to face this lack of setup or hold specifications.

Something important to recall is that the system has two different clock domains running at different frequencies. We have a fast clock that can run up to 1000 MHz (f_{max}), and a slow clock that runs at a frequency f_{system} four times slower than the fast clock. These eight phases were defined by the number of samples taken per bit. The fast clock is used to oversample the incoming serial signal to avoid transmission errors adding redundancy to the data acquisition. The rest of the system can work at a slower frequency, so the only unit that needs to run at the maximum frequency is the clock and data recovery module.

$$f_{max} = f_{inputClk} \ : \ Oversampling\ clock\ frequency$$

$$f_{system} = \frac{1}{4} f_{max} \ : \ System\ clock\ frequency$$

$$f_{max} \ : \ Maximum\ system\ frequency$$

$$f_{inputClk} \ : \ Input\ clock\ frequency$$

$$f_{system} \ : \ Digital\ receiver\ frequency$$

In a future implementation is possible to add another clock to perform clock gating techniques, improving the energy consumption as well. This clock will be fed into the decoder module, which performs one decoding every ten serial recovered bits.

$$f_{decoder} = \frac{1}{10} f_{system} \ : \ Output\ data\ frequency$$

$$f_{system} \ : \ Digital\ receiver\ frequency$$

$$f_{decoder} \ : \ Decoder\ module\ frequency$$

Said that, different issues were found that differ over the various receiver stages in which the data is processed.

2.2.1 Edge detector and best sampling enhanced technique

Based on the results and testbench used by previous designer [5] some additional delays were inserted into the environment to validate the robustness of the system with these external changes. Some issues were found in the capabilities of the original CDR when the delay and latency change. To fix this problem we included an edge detector circuit that is capable to identify a transition from 0 to 1 and vice versa.

This circuit consists in an array of XOR gates and comparators with some logic to detect the position of the sample where the transition occurred.

Using the results from the edge detector circuit we improved the sampling methods by aligning the sample index to the middle of the bit. This is accomplished knowing the index of the transition of the signal received and the number of samples per bit.

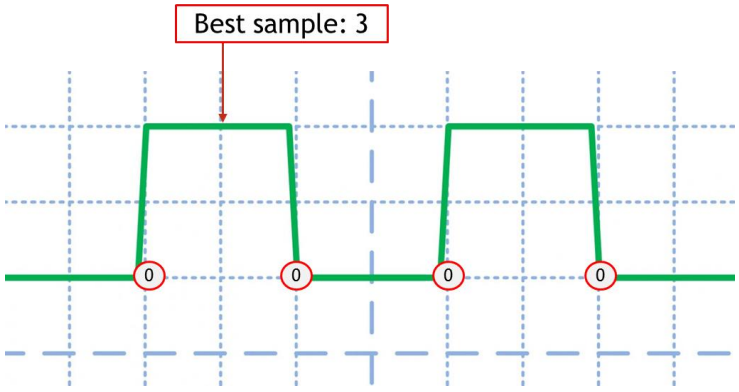

Figure 10 Best sample is aligned to the center of the bit

The results shows that the best sample index is correctly updated depending on the logic transitions of the signal.

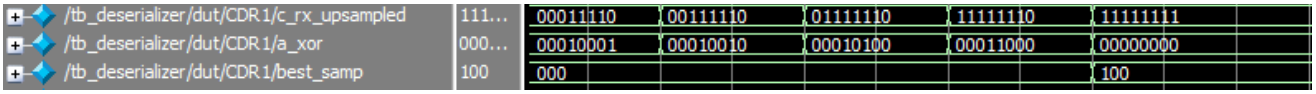

Figure 11Best sample index update

2.2.2 Number of samples per bit reduction

In the fastest section of the circuit we had problems trying to meet the setup and hold timing requirements since the number of samples per bit is larger for the required frequency of 1000 MHz.

```
          Pin                  Type       Fanout Load Slew Delay Arrival
                                             (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock 1000MHz_CLK)           launch                                0 R
                              latency                    +150     150 R
deserializer1
  CDR1
    c_rx_upsampled_reg[2]/CK                              28          150 R
    c_rx_upsampled_reg[2]/Q   DFFRHQX8TS        4 44.5  120  +387     537 F
    g24/A                                                 +0     537
    g24/Y                     INVX8TS           2 26.9   80  +101     638 R
    g1104/A                                               +0     638
    g1104/Y                   NAND2X4TS         2 25.2  128  +114     752 F
    g32/A                                                 +0     752
    g32/Y                     NAND2X6TS         2 28.3  122  +128     880 R
    g31/A1                                                +0     880
    g31/Y                     OAI21X4TS         1 12.6   93  +132    1012 F
    g92/B0                                                +0    1012
    g92/Y                     AOI21X4TS         1 12.5  165  +156    1168 R
    g1141/B0                                              +0    1168
    g1141/Y                   OAI21X4TS         1  6.0   71  +113    1281 F
    best_samp_reg[0]/D        DFFRHQX8TS                    +0    1281
    best_samp_reg[0]/CK       setup                   28  +236    1517 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 1000MHz_CLK)           capture                           1000 R
                              latency                    +150    1150 R
                              uncertainty                 -17    1133 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :    -384ps (TIMING VIOLATION)
Start-point  : deserializer1/CDR1/c_rx_upsampled_reg[2]/CK
End-point    : deserializer1/CDR1/best_samp_reg[0]/D
```

Figure 12 A negative slack was obtained using the fast clock in the sampling section.

Two options were considered to fix this. The first solution was to reduce the frequency of the digital receiver. Doing this the desired timing can be achieved with no RTL modifications without changing the number of valid bits processed per clock.

But this approach implies a significant degradation of the data throughput of the receiver, since the number of words received is reduced to a half of the original throughput. The number of words received using this system in an ideal transmission depends on the clock frequency, number of samples and encoding scheme. The following formula relates these variables and determines the recovered valid 8-bit words:

$$N_{data} = \frac{S_{sampling_{clock}}}{n_{samples}\, m_{encoding}} \quad : \quad Data\ recovered\ equation$$

$$N_{data} \quad : \quad Number\ of\ 8\ bit\ words\ recovered$$

$$S_{sampling_{clock}} \quad : \quad Number\ of\ samples\ per\ second$$

$$n_{samples} \quad : \quad Number\ of\ samples\ per\ bit$$

$$m_{encoding} \quad : \quad Bits\ encoded\ per\ 8\ bit\ word$$

where S_{clock} is the number of samples per second (proportional to the frequency of the input clock), $n_{samples}$ is the number of samples per bit, $m_{encoding}$ is the number of bits required for the decoding scheme (in this case 10 bit since we are using a 10B8B based encoding technique) and N_{data} is the number of 8-bit data recovered in a period of one second.

Then, using the current values of the design we get the actual number of 8-bit words received assuming an ideal transmission with no delays or synchronization special characters:

$$N_{data} = \frac{1 \times 10^6\ \frac{samples}{s}}{8_{\frac{samples}{bit}} \times\ 10\ _{\frac{bit}{word}}} = 12.5\,\frac{kB}{s}$$

With the proposed change in the frequency this number falls to half of the data.

$$N_{S1_{data}} = \frac{S_{1_{sampling\_clock}}}{n_{samples}\, m_{encoding}}$$

$$N_{S1_{data}} = \frac{5 \times 10^5 \ \frac{samples}{s}}{8_{\frac{samples}{bit}} \times \ 10 \ _{\frac{bit}{word}}} = 6.25 \frac{kB}{s}$$

Changes in the frequency is not allowed since it implies a degradation of the throughput as described in this section. So, a second proposal was analyzed to avoid this situation without compromising the overall performance of the SerDes.

In a second approach the clock frequency is reduced but also reducing the number of samples per bit in the oversampling process. At a first glance this can be seen as an increase in the error probability and redundancy since we are reducing the number of samples, thus increasing the number of errors per frame in the system. We compensate this possible degradation by using the edge detector module described in a previous section. A further analysis on the effects of reducing the number of samples can be done as part of a future work.

This allows us to mantain the same throughput, even if a slower frequency is used in the system. In order to perform the edge detector algorithm we need to keep at least three samples. We use four samples per bit in the proposed model which is half of the samples that were used in the previous work [5].

Then the equation for the resultant throughput of the system is as follows:

$$N_{S2_{data}} = \frac{S_{2_{sampling\_clock}}}{n_{samples}\, m_{encoding}}$$

$$N_{S1_{data}} = \frac{5 \times 10^5 \ \frac{samples}{s}}{4_{\frac{samples}{bit}} \times \ 10 \ _{\frac{bit}{word}}} = 12.5 \frac{kB}{s}$$

An additional run of the logic synthesis flow was launched in order to verify these changes. The reports reflected that the enhancements were successful.

```
           Pin                          Type         Fanout Load Slew Delay Arrival
                                                            (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------------------
(clock clock_ph1)                   launch                                  2000 R
deserializer1
  CDR1
    c_rx_upsampled_reg[1]/clk                                    0          2000 R
    c_rx_upsampled_reg[1]/q   (u)  unmapped_d_flop       3 18.6    0  +350    2350 R
    g1/in_0                                                        +0    2350
    g1/z                      (u)  unmapped_xor2         1  6.2    0  +234    2584 R
    a_xor_reg_reg[3]/d             unmapped_d_flop                 +0    2584
    a_xor_reg_reg[3]/clk           setup                     0     +229    2813 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock_ph0)                   capture                                 8000 R
-------------------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :    5187ps
Start-point  : deserializer1/CDR1/c_rx_upsampled_reg[1]/clk
End-point    : deserializer1/CDR1/a_xor_reg_reg[3]/d
```

Figure 13 Positive slack after the sampling phases reduction.

We can appreciate that the timing slack is positive after the changes. The end point signal changed since some intermediate registers were added in the process.

2.2.3 Pipelined 8B10B decoder

An external IP was used to perform the 8B10B decoding process. This provides us a reliable module in which the functional verification was easy to complete. The main issue with this model is that it does not include any sequential logic, thus causing several timing violations in many paths. Actually the critical path in the whole system is inside this module.

```
          Pin                      Type       Fanout Load Slew Delay Arrival
                                                     (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------------
(clock 1000MHz_CLK)              launch                                    0 R
                                 latency                          +150     150 R
deserializer1
  c_parallel_out_reg[1]/CK                                  28              150 R
  c_parallel_out_reg[1]/Q        DFFRHQX8TS          6 57.4  174  +419     569 R
deserializer1/c_parallel_out[1]
decode1/datain[1]
  g14210/A                                                          +0     569
  g14210/Y                       INVX6TS             1 19.4   61  +105     674 F
  g14164/B                                                          +0     674
  g14164/Y                       NAND2X8TS           2 28.4  106  +101     775 R
  g14101/A                                                          +0     775
  g14101/Y                       INVX8TS             3 24.5   50   +75     851 F
  g14079/AN                                                         +0     851
  g14079/Y                       NOR2BX4TS           3 27.1  105  +184    1035 F
  g14032/A                                                          +0    1035
  g14032/Y                       MXI2X4TS            1 12.9  210  +175    1210 R
  g13999/B                                                          +0    1210
  g13999/Y                       NAND3X4TS           1 12.6  122  +171    1381 F
  g13985/B0                                                         +0    1381
  g13985/Y                       AOI21X4TS           1  9.0  141  +151    1532 R
  g13959/A                                                          +0    1532
  g13959/Y                       AND3X6TS            1 25.4  116  +252    1785 R
  g13951/A                                                          +0    1785
  g13951/Y                       NAND4X8TS           1 34.8  166  +147    1932 F
decode1/code_err
pad_code_err/A                                                      +0    1932
pad_code_err/P             (P)   POC4C               1  0.0 1244 +1474    3407 F
code_err                         out port                           +0    3407 F
(ODelay)                         ext delay                        +200    3607 F
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock 1000MHz_CLK)              capture                                  1000 R
                                 latency                          +150    1150 R
                                 uncertainty                       -17    1133 R
-------------------------------------------------------------------------------
Cost Group   : 'C2O' (path_group 'C2O')
Timing slack :   -2474ps (TIMING VIOLATION)
Start-point  : deserializer1/c_parallel_out_reg[1]/CK
End-point    : code_err
```

Figure 14 Critical path showing a negative slack timing violation in the decoding stage.

The first solution was the addition of registers inside the module to break the critical path. But even with that there were some other options.

The first approach was the insertion of flip flops to break only the longest paths adding a blocking mechanism to avoid the injection of new data in the input register when a data is being processed inside the module. This is a good approach if we do not need to achieve a high performance. Only two requirements are needed: the synchronization of the data valid signal and a blocking controller to avoid noise in the decoding stage.

In this case we delay the decoding process by N clock cycles. This number depends on the maximum number of registers that are inserted in a single critical path. So we can have one decoded word every N cycles. The first data has an initial delay of N cycles as well.

$$i_D = N * C \quad : \quad Initial\ delay$$

$$d_D = N\ clock\ cycles \quad : \quad Data\ calculation\ delay$$

$$N \quad : \quad Number\ of\ clock\ cycles\ to\ decode\ one\ 8\ bit\ word$$

$$C \quad : \quad One\ clock\ cycle\ delay$$

Another similar approach is to include a pipeline in the decoding process. Similar to the first proposal, it includes register addition to break the critical path. The difference is that there is no need of any special blocking controller when a data is being processed by the decoder, since more than one input data can be processed by the decoder at the same time.

Pipelining the decoder inserts the same extra initial delay to this stage, but the main difference is that, once the pipeline is full we can obtain one decoded word every single clock cycle.

$$i_D = N * C \quad : \quad Initial\ delay$$

$$d_D = 1\ clock\ cycle \quad : \quad Data\ calculation\ delay$$

A valid signal determines whether the data at the input of the pipeline is valid or not. If this signal is asserted the data enters into the pipeline and it is processed following the 8B10B decoding standards.

The results of this pipelining technique have a positive slack calculation for the signals under observation. This pipeline implementation has five stages, which is the initial clock cycles delay value.

```
        Pin                    Type         Fanout Load Slew Delay Arrival
                                             (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock clock_ph0)           launch                                    0 R
decode1
  gi_4reg_reg/clk                                      0              0 R
  gi_4reg_reg/q        (u)  unmapped_d_flop       7 43.4    0  +409     409 R
  g2629/in_0                                                +0     409
  g2629/z              (u)  unmapped_not          6 37.2    0  +142     550 F
  g2736/in_1                                                +0     550
  g2736/z              (u)  unmapped_nor2         1  6.2    0   +94     645 R
  g2737/in_0                                                +0     645
  g2737/z              (u)  unmapped_not          2 12.4    0   +78     723 F
  g2841/in_1                                                +0     723
  g2841/z              (u)  unmapped_nand2        1  6.2    0   +94     818 R
  g2842/in_0                                                +0     818
  g2842/z              (u)  unmapped_not          1  6.2    0   +62     880 F
  g2920/in_3                                                +0     880
  g2920/z              (u)  unmapped_nor4         1  6.2    0  +167    1047 R
  g2921/in_0                                                +0    1047
  g2921/z              (u)  unmapped_not          1  6.2    0   +62    1110 F
  g2953/in_1                                                +0    1110
  g2953/z              (u)  unmapped_nand2        1  6.2    0   +94    1204 R
  g2954/in_0                                                +0    1204
  g2954/z              (u)  unmapped_not          1  6.2    0   +62    1266 F
  g2971/in_2                                                +0    1266
  g2971/z              (u)  unmapped_nand4        1  6.2    0  +167    1434 R
  code_err_reg_reg/d        unmapped_d_flop                  +0    1434
  code_err_reg_reg/clk      setup                         0  +229    1663 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clock_ph0)           capture                                8000 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :    6337ps
Start-point  : decode1/gi_4reg_reg/clk
End-point    : decode1/code_err_reg_reg/d
```

Figure 15 Timing slack after pipelining techniques.

2.2.4 One bit adder microarchitecture changes

The adders to count the number of ones in the oversampled data were redesigned to achieve a better latency and reduce the number of logic elements improving the area, timing and power consumption of the system. In order to accomplish this goal the design changed from the previous one as described in the following sections.

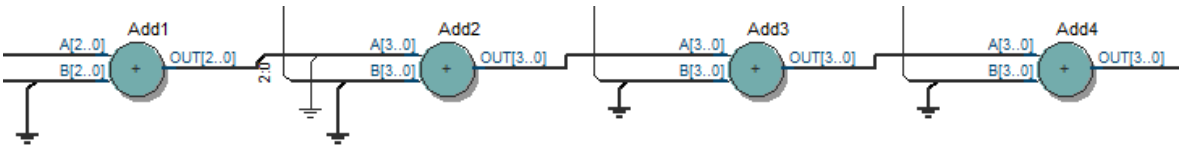

Figure 16 Daisy-chain adder

In the previous work the adders were instantiated in a daisy-chain connection as described in figure 8, having a latency of n stages where n is the number of inputs of the adder. In this case we instantiated a 1-bit 8-input adder, so we had eight stages to obtain the final result.

$$Number\ of\ stages = n$$

With the full custom adder we reduced the number of stages, based on a combinatory gates array simplified for 1-bit adders.

$$Number\ of\ stages = \log_2(n) - 1$$

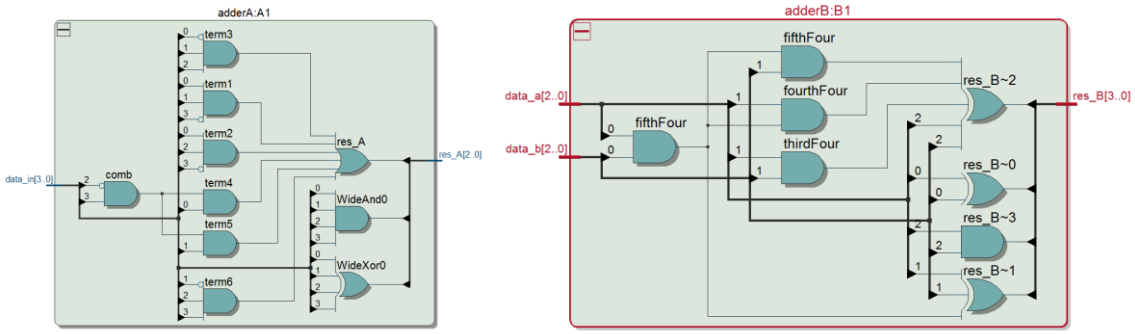

Figure 17 One-bit adder proposed

2.3 Logic Synthesis

A second logic synthesis was launched to verify the correctness of the results using the RTL Compiler tool. The top module signals were aligned to the established specifications set by the SerDes design team. The main reports are attached in the Appendix.

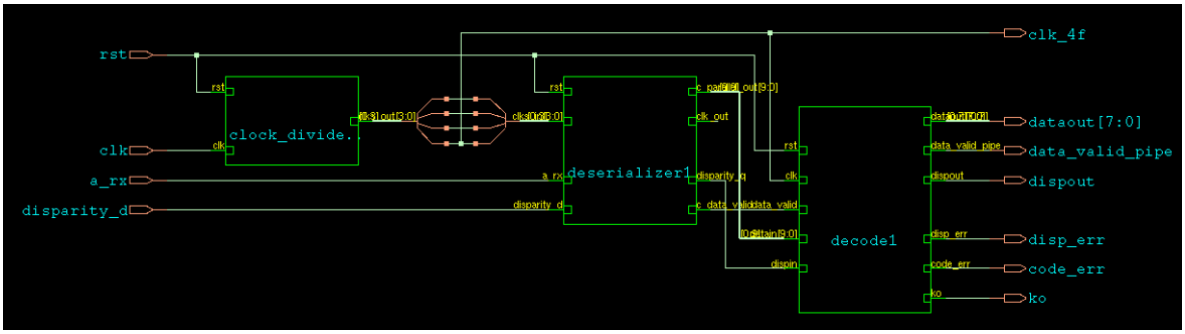

Figure 18 Digital receiver logic synthesis diagram.

2.4 Gate level simulation

At this stage of the chip design process we are able to model the RTL description of the design at gate level. This step is a functional verification of the generated gate level netlist of the module, instead of the behavioral representation used for the design of the system.

The Modelsim tool was used to simulate this netlist, obtaining the standard cells HDL description from the technology library directories. As the system is described using Verilog the same language was selected to model the cells.

Four new files were required to perform the gate level simulation. Two files are the standard cells models, one file is for the input and output pads and the last one was the netlist of the digital receiver module generated by RC compiler.

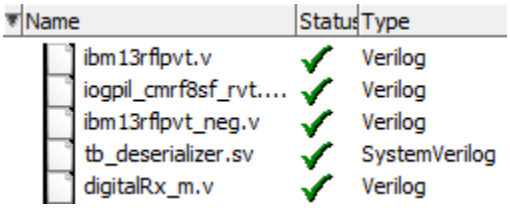

Figure 19 Verilog libraries added for the standard cells and IO pads.

The testbench did not have any modifications, since we are trying to validate the equivalence between the behavioral and structural representations of the architecture.

One possible source of error in this simulation that differs from the behavioral model is the presence of unknown values due to dangling signals, wrong register initialization or latches inferring. Therefore, an additional effort was done to validate that all the inputs are driven correctly by a valid signal and to check that all the internal signals are connected to the corresponding outputs.

The results show the same exact results as in the RTL model.

A diagram of the digitalRx simulation injecting the test values is shown below. The valid signal is now driven by the decoding pipeline.

```
serial_mblb(10'h0f8);   // sync character
// send test patterns
serial_mblb(10'h365);   // 5
serial_mblb(10'h366);   // 6
serial_mblb(10'h347);   // 7
serial_mblb(10'h0a7);   // 8
serial_mblb(10'h369);   // 9
```

Figure 20 Test values injected into the DUT

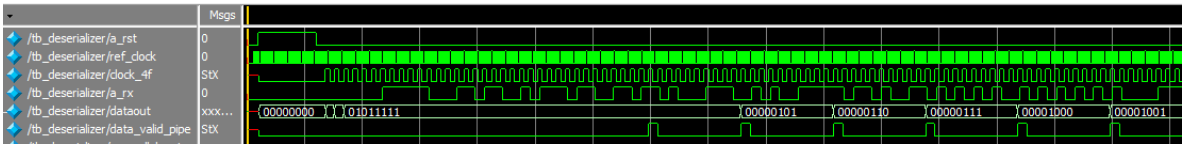

Figure 21 Digital receiver top module simulation.

The same results were obtained for the rest of the internal modules.

Clock divider

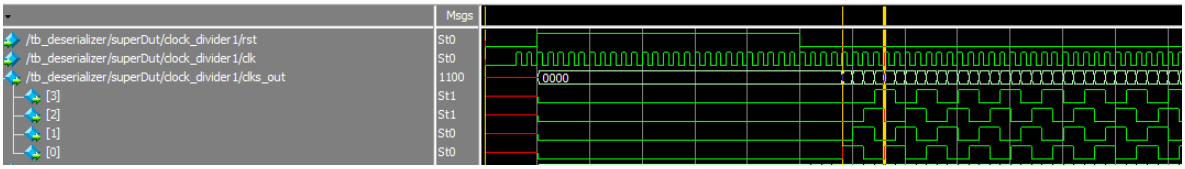

Figure 22 Phases reduction waveforms.

CDR (including oversampling and best sample calculation)

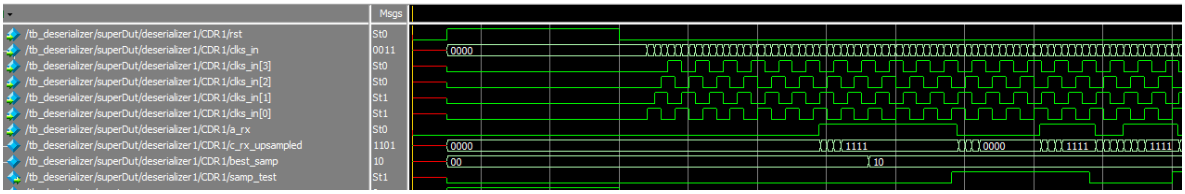

Figure 23 Best sampling technique after oversampling the incoming serial stream.

Pipelined decoder (showing the five pipeline stages delays between the cursors)

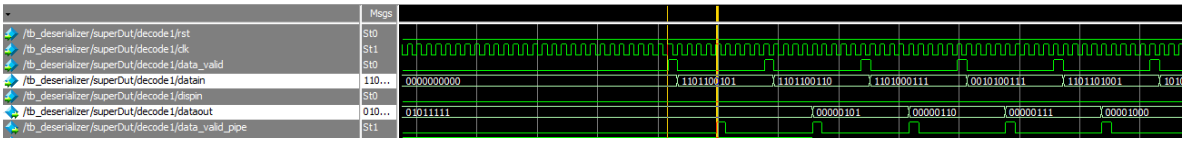

Figure 24 Pipelined output from the 8B10B decoder.

CHAPTER 3. Physical Synthesis

In order to get the manufacturing output files for the digital receiver circuit, a set of steps were performed to obtain a layout representation of the synthesized netlist representation generated at the logic synthesis stage. This section describes the methodology followed to perform the physical synthesis highlighting the critical tasks that are required to satisfy the design rules constrains.

3.1 Basic scripts and layout

In a second run of the Physical synthesis script we obtained the layout of the standard cells using Encounter tool from Cadence.

The floorplan had the following dimensions as shown in Fig. 25.

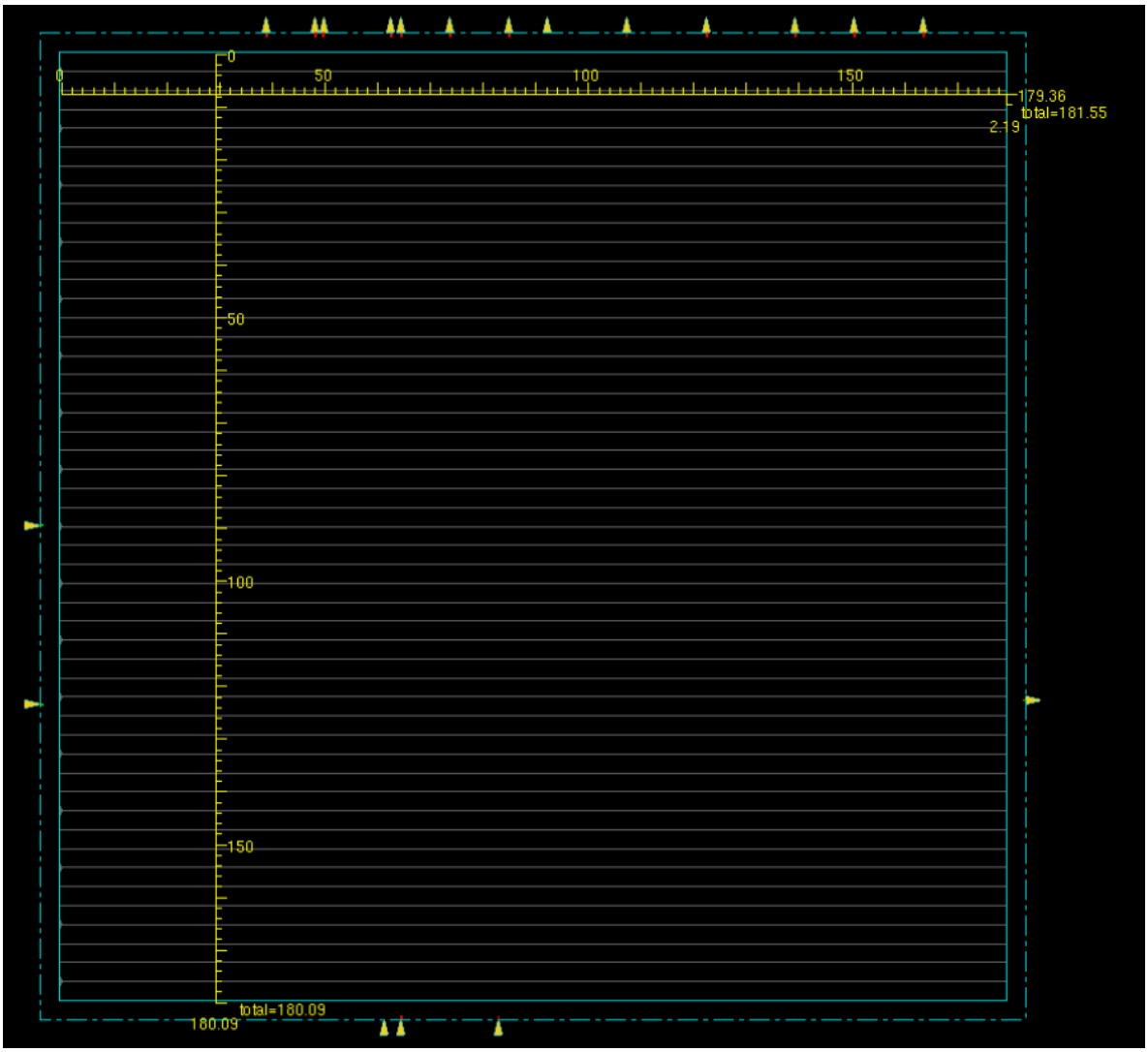

Figure 25 Floorplan dimensions

The layout was obtained as well and it is illustrated in Fig. 26:

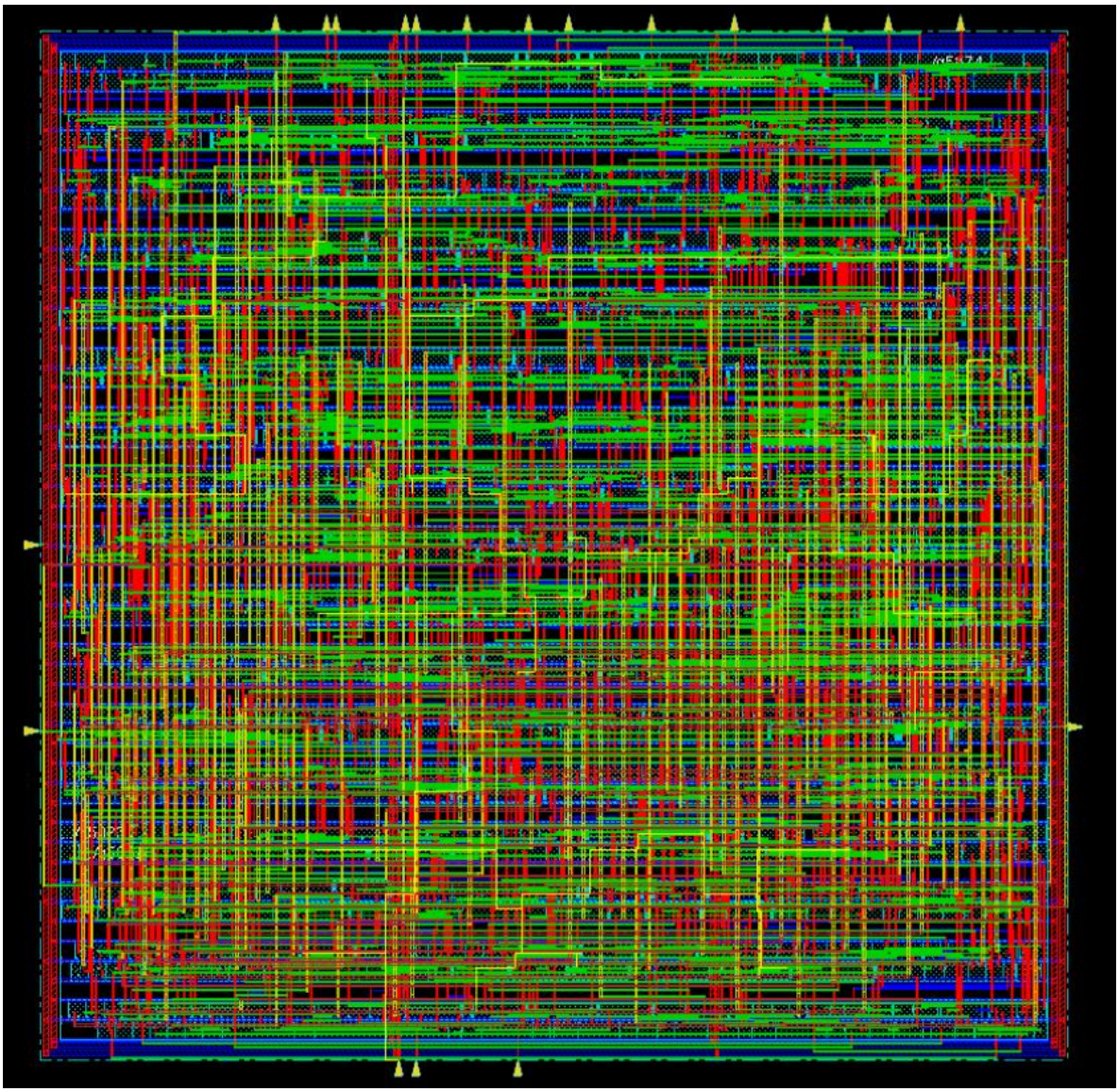

Figure 26 Physical synthesis of the digital receiver

The clock tree was synthesized as well. Fig. 27 shows the clock tree synthesis.

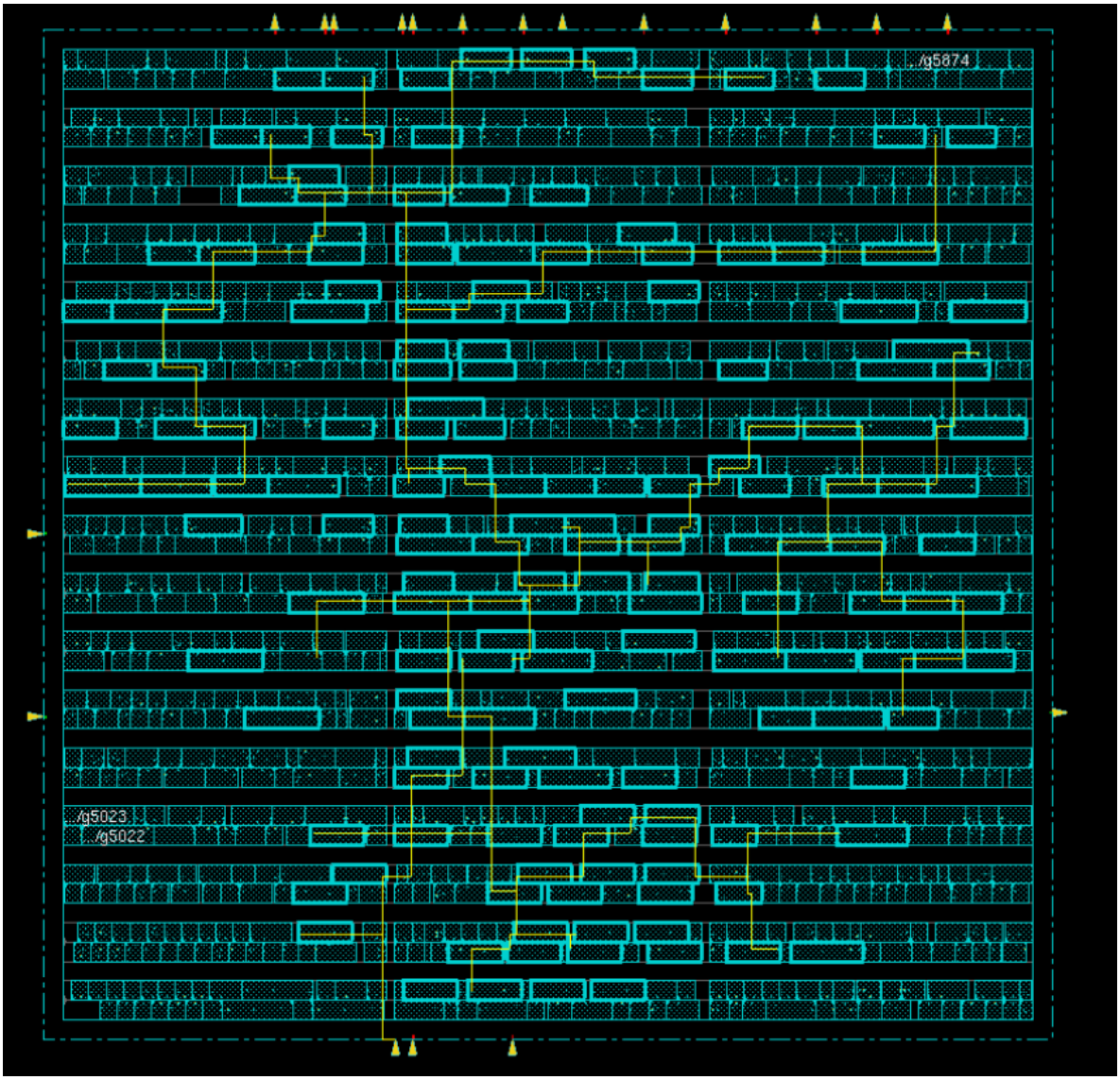

Figure 27 Clock tree physical synthesis

3.2 Timing analysis

3.2.1 Negative slack fixes

A second timing analysis is performed at the synthesis stage. This static analysis verifies every single cell to obtain the setup and hold times and compare them with the reference frequency defined in the constraint file.

The tool creates some report files for all the paths. A summary file contains the timing violations due to negative slack at the top of the document.

A negative slack is detected in the design, even when the logic synthesis report showed a clean path from the timing analysis point of view. Doing a further review of the documents, the critical path was isolated to know what are the registers or clocks that are having a slower response than the minimum required. In this case, the register-to-register file specifies the begin and end point of the timing issue in question.

This particular signal inside the clock divider module is generated by a register array as described in section 2.1.5. The output of the module consists of four different clock phases that enable the oversampling process, but also the bit 0 of the output clocks vector acts as the system clock. This implies that the output of the flip-flop *div_by_4_q_reg* is fed to a large number of modules and registers, thus requiring a buffered topology to maintain the integrity of the signal to every single end-point.

```
// Generated by Cadence Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1

// Verification Directory fv/digitalRx_wrapper

module clock_divider(rst, clk, clks_out);
  input rst, clk;
  output [3:0] clks_out;
  wire rst, clk;
  wire [3:0] clks_out;
  wire [1:0] div_by_4_q;
  wire [1:0] div_by_4_reg;
  wire UNCONNECTED, UNCONNECTED0, UNCONNECTED1, UNCONNECTED2,
       UNCONNECTED3, UNCONNECTED4, n_0, n_1;
  INVX2TS g11(.A (rst), .Y (n_1));
  DFFRX2TS \div_by_4_q_reg[1] (.RN (n_1), .CK (clk), .D
       (div_by_4_q[0]), .Q (div_by_4_q[1]), .QN (n_0));
  DFFRXLTS \clks_out_reg[3] (.RN (n_1), .CK (clk), .D (clks_out[2]), .Q
       (clks_out[3]), .QN (UNCONNECTED));
  DFFRXLTS \clks_out_reg[2] (.RN (n_1), .CK (clk), .D (clks_out[1]), .Q
       (clks_out[2]), .QN (UNCONNECTED0));
  DFFRXLTS \clks_out_reg[1] (.RN (n_1), .CK (clk), .D (clks_out[0]), .Q
       (clks_out[1]), .QN (UNCONNECTED1));
  DFFRXLTS \clks_out_reg[0] (.RN (n_1), .CK (clk), .D
       (div_by_4_reg[1]), .Q (clks_out[0]), .QN (UNCONNECTED2));
  DFFRXLTS \div_by_4_reg_reg[1] (.RN (n_1), .CK (clk), .D
       (div_by_4_q[1]), .Q (div_by_4_reg[1]), .QN (UNCONNECTED3));
  DFFRXLTS \div_by_4_q_reg[0] (.RN (n_1), .CK (clk), .D (n_0), .Q
       (div_by_4_q[0]), .QN (UNCONNECTED4));
endmodule
```

Figure 28 Original tool generated netlist

The tool is aware enough of this requirement in a module-to-module analysis, but in the big picture the system cannot handle this kind of issues. The proposed solution was to increase the size of the cells with a big fanout, replacing the cells DFFRX2TS by DFFRX4TS.

```
// Generated by Cadence Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1
// Modified by Rogelio Rivas
// Verification Directory fv/digitalRx

module clock_divider(rst, clk, clks_out);
  input rst, clk;
  output [3:0] clks_out;
  wire rst, clk;
  wire [3:0] clks_out;
  wire [1:0] div_by_4_reg;
  wire [1:0] div_by_4_q;
  wire n_0, n_1;
  DFFRHQX4TS \clks_out_reg[3] (.RN (n_0), .CK (clk), .D (clks_out[2]),
       .Q (clks_out[3]));
  DFFRHQX4TS \clks_out_reg[2] (.RN (n_0), .CK (clk), .D (clks_out[1]),
       .Q (clks_out[2]));
  DFFRHQX4TS \clks_out_reg[1] (.RN (n_0), .CK (clk), .D (clks_out[0]),
       .Q (clks_out[1]));
  DFFRHQX4TS \clks_out_reg[0] (.RN (n_0), .CK (clk), .D
       (div_by_4_reg[1]), .Q (clks_out[0]));
  DFFRHQX4TS \div_by_4_reg_reg[1] (.RN (n_0), .CK (clk), .D
       (div_by_4_q[1]), .Q (div_by_4_reg[1]));
  INVX2TS g11(.A (rst), .Y (n_0));
  DFFRHQX2TS \div_by_4_q_reg[1] (.RN (n_0), .CK (clk), .D
       (div_by_4_q[0]), .Q (div_by_4_q[1]));
  DFFRHQX4TS \div_by_4_q_reg[0] (.RN (n_0), .CK (clk), .D (n_1), .Q
       (div_by_4_q[0]));
  CLKINVX1TS g8(.A (div_by_4_q[1]), .Y (n_1));
endmodule
```

Figure 29 Modified netlist with bigger cells

3.3 PADS inclusion analysis

An additional synthesis was performed in order to make the integration with other modules easier. This stage consisted in the insertion of pads to connect the signals to the external world. This required a wrapper module to connect the corresponding outputs with the pads.

One of the main purposes of this enhancement was to detect possible design violations and catch the issues at this early stage.

The dimensions of the floorplan are not the final ones. This is only a rough representation of the whole chip. The integrator of the SerDes should set the final dimensions to allocate all the functional digital and analog blocks.

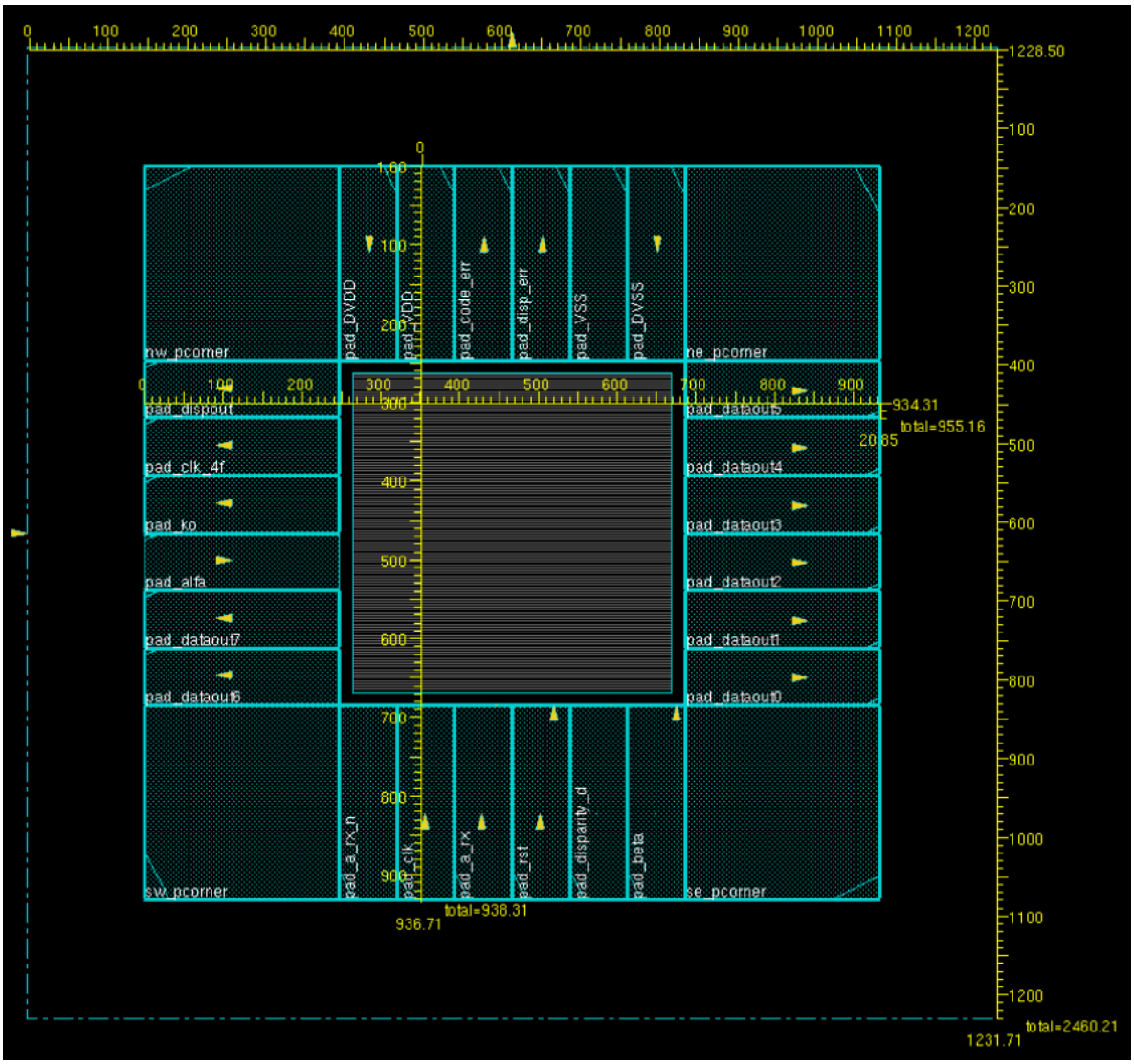

Figure 30 Floorplan dimensions included some pads to emulate the final implementation of the chip.

The resultant layout is shown in Fig. 31.

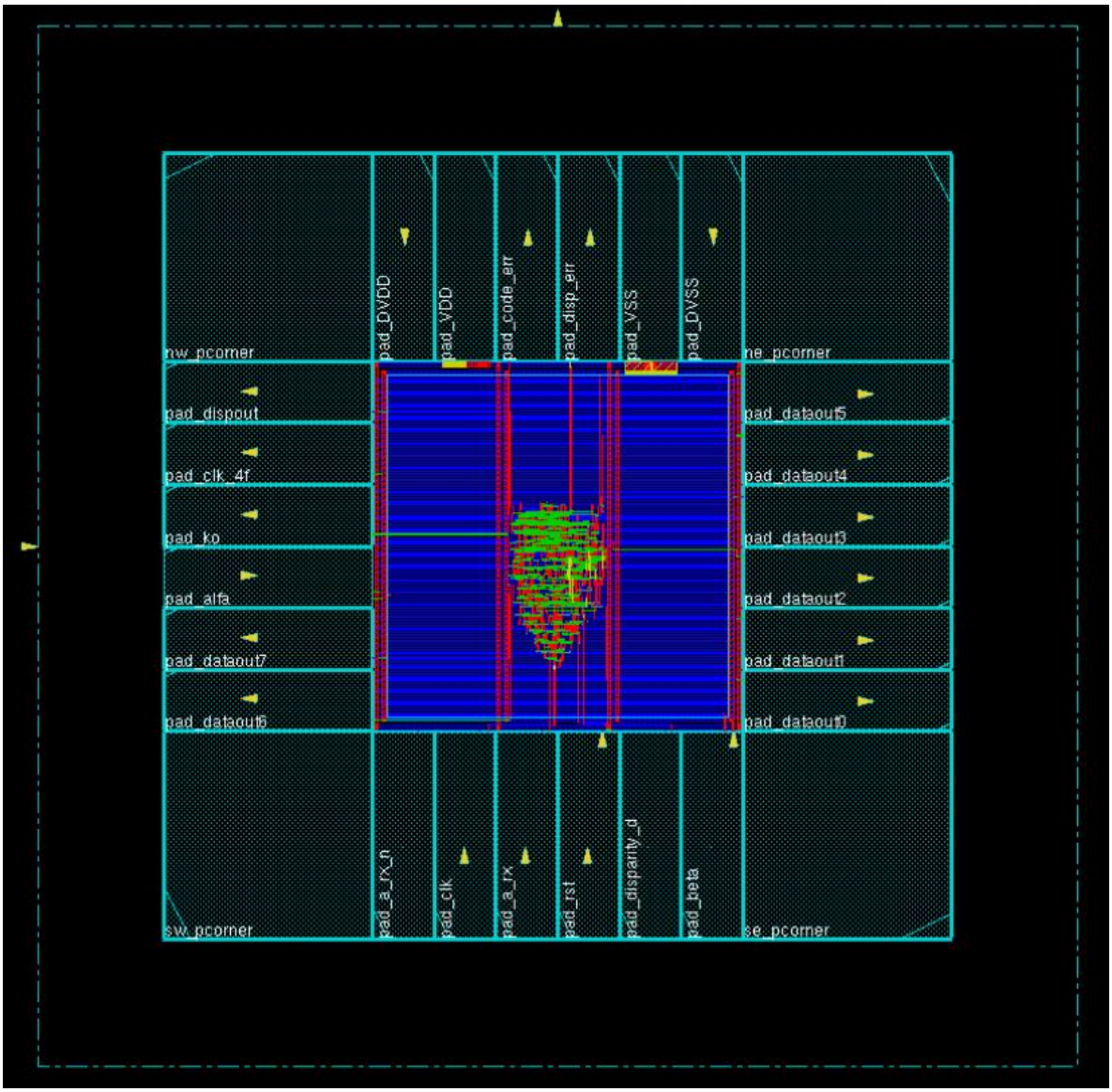

Figure 31 Physical synthesis of the proposed model with IO pads.

The clock tree maps the reference clock and the slow clock, which will be connected to the rest of the modules of the chip. The clock tree synthesized is shown in Fig. 32 and Fig. 33.

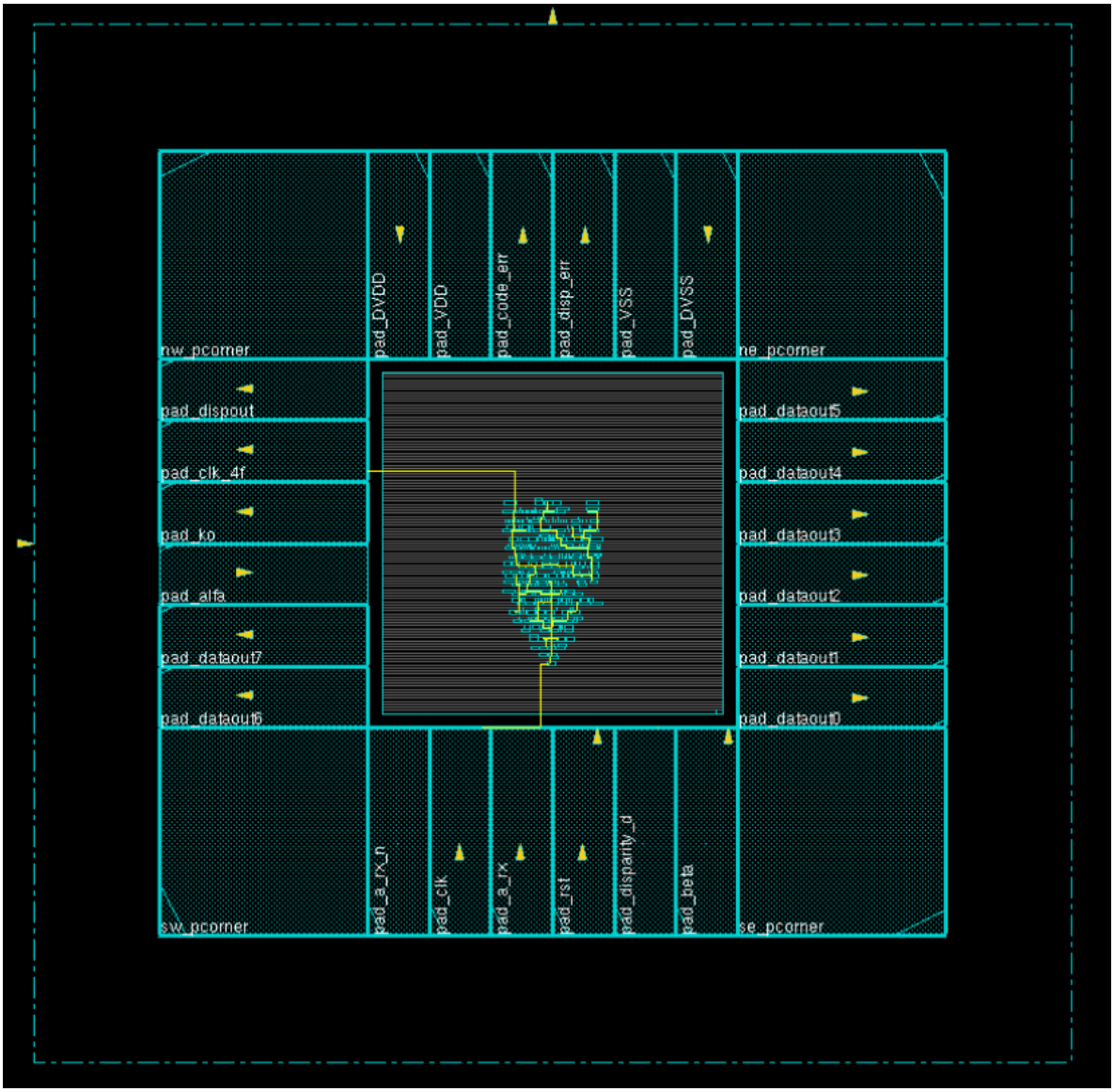

Figure 32 Clock tree synthesis.

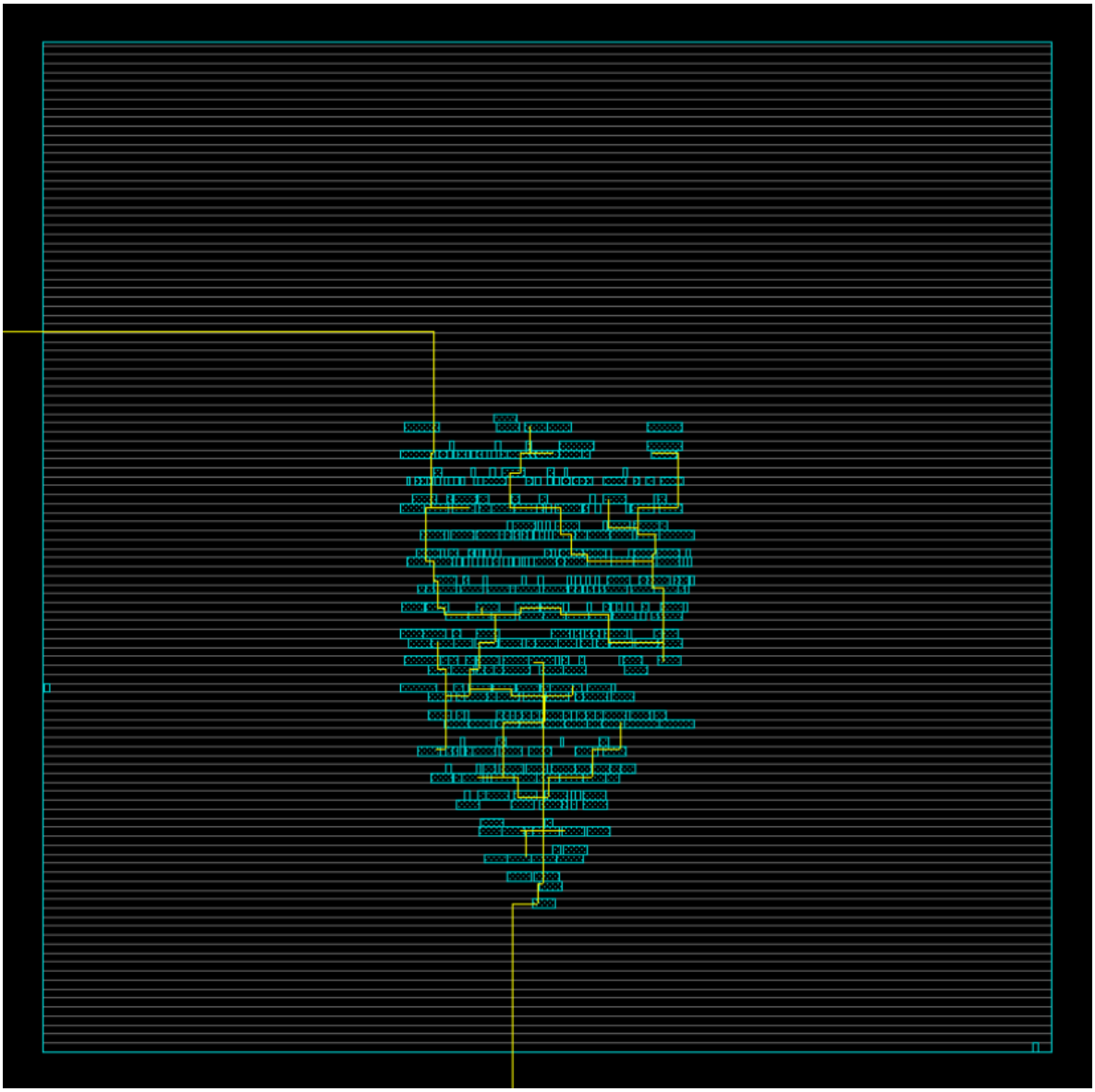

Figure 33 Clock tree synthesis for the input reference clock and the resultant slow clock,

3.4 Geometry, DRC, VLS and equivalence verification

Geometry, connectivity and DRC verifications where performed to review and validate possible issues.

```
encounter 2>  *** Starting Verify Geometry (MEM: 851.3) ***

  VERIFY GEOMETRY ..... Starting Verification
  VERIFY GEOMETRY ..... Initializing
  VERIFY GEOMETRY ..... Deleting Existing Violations
  VERIFY GEOMETRY ..... Creating Sub-Areas
                  ..... bin size: 2560
  VERIFY GEOMETRY ..... SubArea : 1 of 1
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 1 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 1.00
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**********
 *** verify geometry (CPU: 0:00:00.8  MEM: 108.2M)
```

Figure 34 Geometry verification results

```
VERIFY_CONNECTIVITY use new engine.

******** Start: VERIFY CONNECTIVITY ********
Start Time: Sun Jun 19 16:06:43 2016

Design Name: digitalRx
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (187.2000, 187.2000)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Sun Jun 19 16:06:43 2016
Time Elapsed: 0:00:00.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.1  MEM: 0.000M)
```

Figure 35 Connectivity verification results

```
encounter 2>  *** Starting Verify DRC (MEM: 959.5) ***

  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 1
  VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

  Verification Complete : 0 Viols.

 *** End Verify DRC (CPU: 0:00:00.5  ELAPSED TIME: 0.00  MEM: 0.0M) ***
```

Figure 36 DRC verification results

The PADS included design passed the Geometry, DRC and Connectivity verifications.

```
encounter 1>  *** Starting Verify Geometry (MEM: 857.9) ***

  VERIFY GEOMETRY ..... Starting Verification
  VERIFY GEOMETRY ..... Initializing
  VERIFY GEOMETRY ..... Deleting Existing Violations
  VERIFY GEOMETRY ..... Creating Sub-Areas
                  ..... bin size: 2560
  VERIFY GEOMETRY ..... SubArea : 1 of 4
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 1 complete 0 Viols. 0 Wrngs.
  VERIFY GEOMETRY ..... SubArea : 2 of 4
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 2 complete 0 Viols. 0 Wrngs.
  VERIFY GEOMETRY ..... SubArea : 3 of 4
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 3 complete 0 Viols. 0 Wrngs.
  VERIFY GEOMETRY ..... SubArea : 4 of 4
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 4 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 1.00
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**********
 *** verify geometry (CPU: 0:00:00.9  MEM: 80.1M)
```

Figure 37 Geometry verification

```
encounter 1>  *** Starting Verify DRC (MEM: 938.0) ***

  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.

 *** End Verify DRC (CPU: 0:00:00.6  ELAPSED TIME: 1.00  MEM: 0.0M) ***
```

Figure 38 DRC verification

```
VERIFY_CONNECTIVITY use new engine.

******** Start: VERIFY CONNECTIVITY ********
Start Time: Mon Jun 20 09:44:26 2016

Design Name: digitalRx_wrapper
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (1232.0000, 1232.0000)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Mon Jun 20 09:44:26 2016
Time Elapsed: 0:00:00.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.0  MEM: 0.000M)
```

Figure 39 Connectivity verification

3.5 Import layout to Virtuoso

A final stage of the SerDes fabrication process is the layout generation in the Virtuoso environment to merge the analog output designs with the digital section. This tool performs additional set of tests to validate possible bug escapes that might be overseen by the Encounter program.

The first step on this process is the generation of a.gds file, which is compatible with the Virtuoso environment. In Fig. 40 we can see the Virtuoso representation of the circuit.

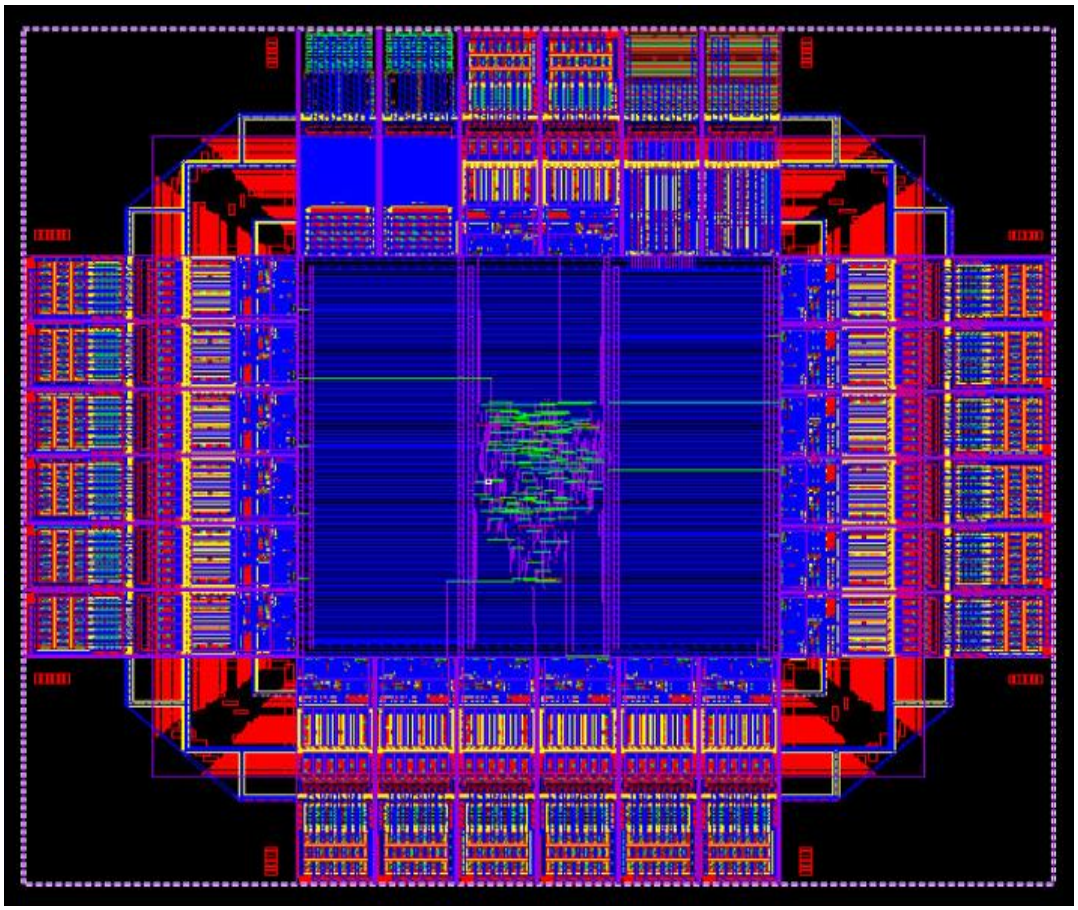

Figure 40 Virtuoso final layout

A second run of the DRC verification was launched to ensure that we are aligning the design to the design constraints defined for this technology. This analysis differs from the Encounter process, so we need to clean additional errors even when the DRC passes in a previous stage.

The following issues were faced during the DRC verification process with Calibre tool:

- GRLOGIC issues.
- N-well connectivity errors.
- Latch up violations.
- Miscellaneous antenna violations.

3.5.1 GRLOGIC issues

A GRLOGIC violation was shown by the DRC verification tool trying to apply the more stringent layout rule checking on the logic standard cell circuit.

Fig. 41 shows more than 1000 violations caused by the same issue.

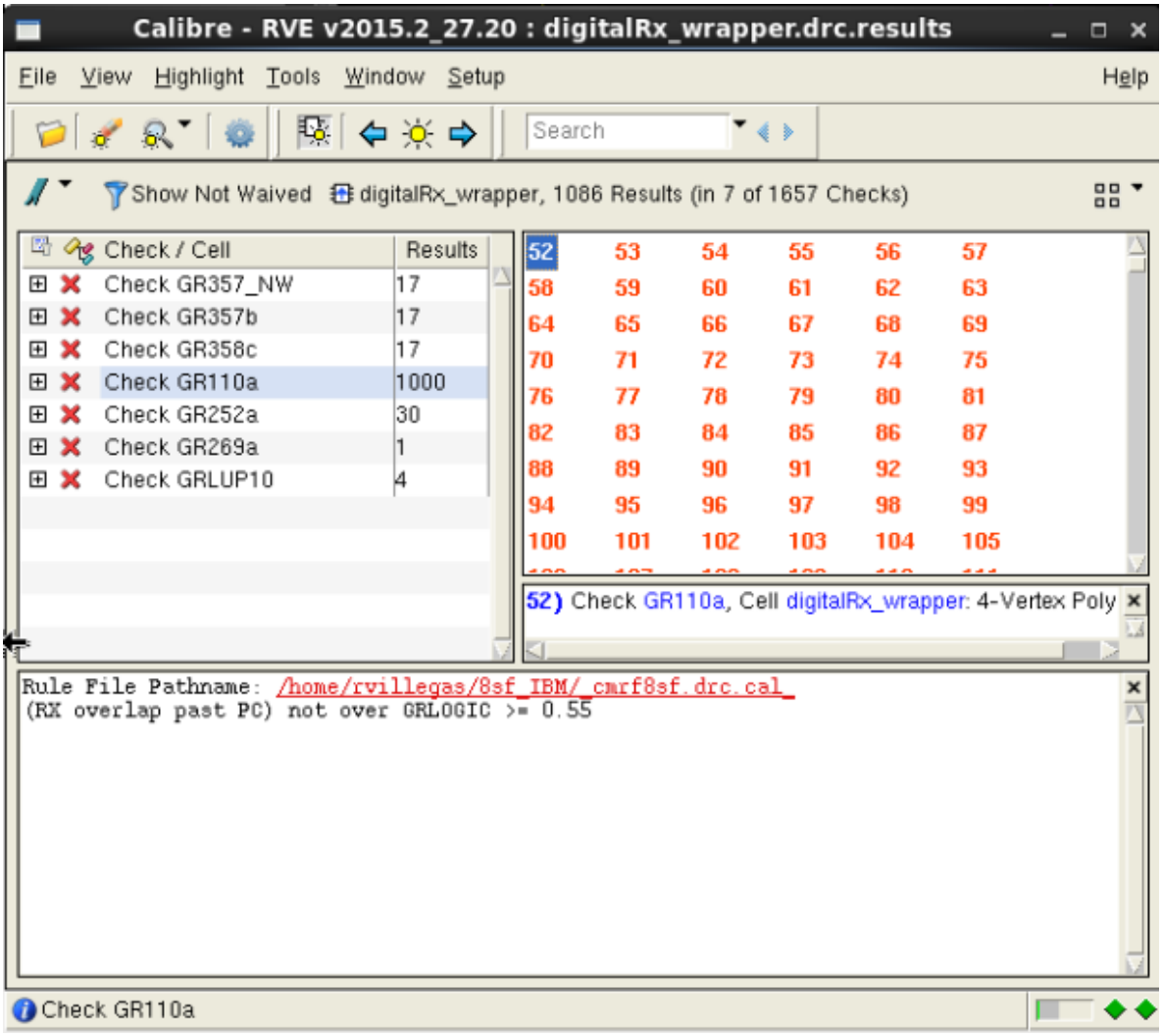

Figure 41 GRLOGIC error signature

In order to solve this issue the circuit was surrounded with a GRLOGIC shape that allows the tool to identify the logic section of the circuit. This techniques isolates the logic cells and exempt them from recommended rule checking for full-custom layout designs.

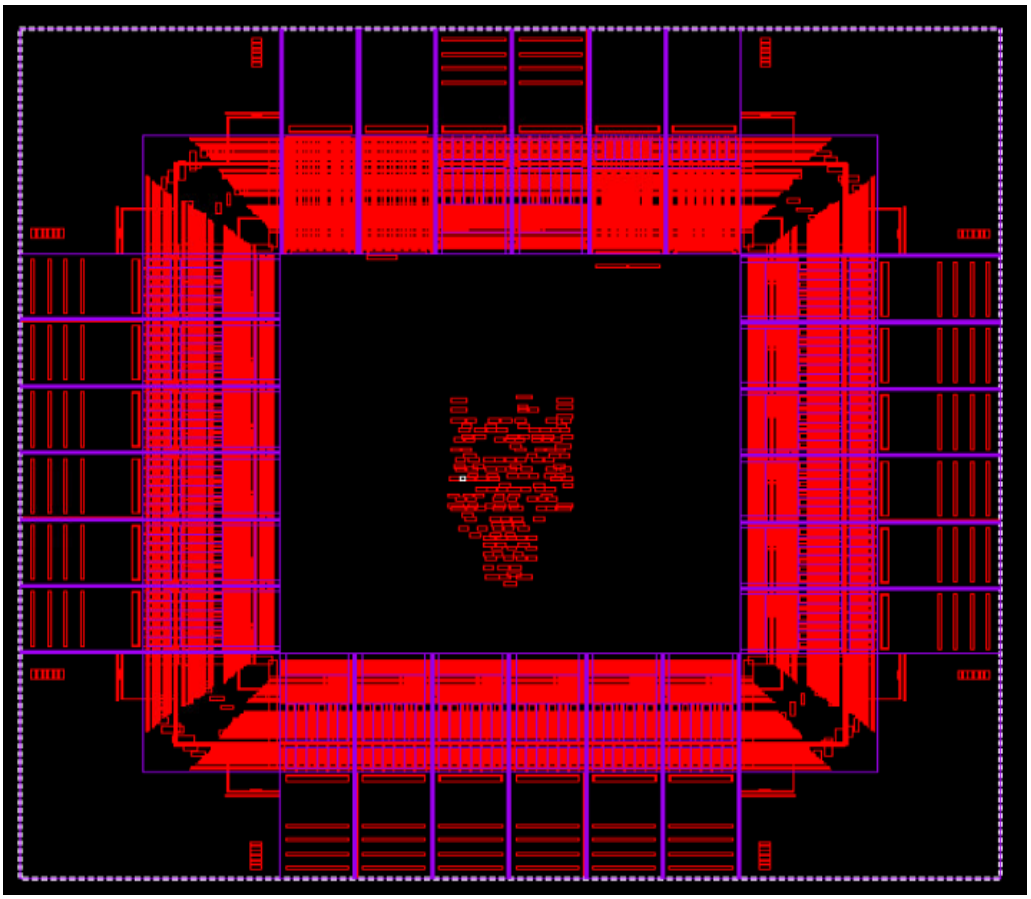

Figure 42 GRLOGIC suround shape

In Fig. 43 the GRLOGIC errors are cleaned, showing successful results due to the addition of the mentioned shape.

3.5.2 N-well connectivity errors

Another limitation of the Encounter database to Virtuoso gds file conversion is a connectivity issue in the n-well section.

In Fig. 43 seventeen N-well spacing errors are shown.

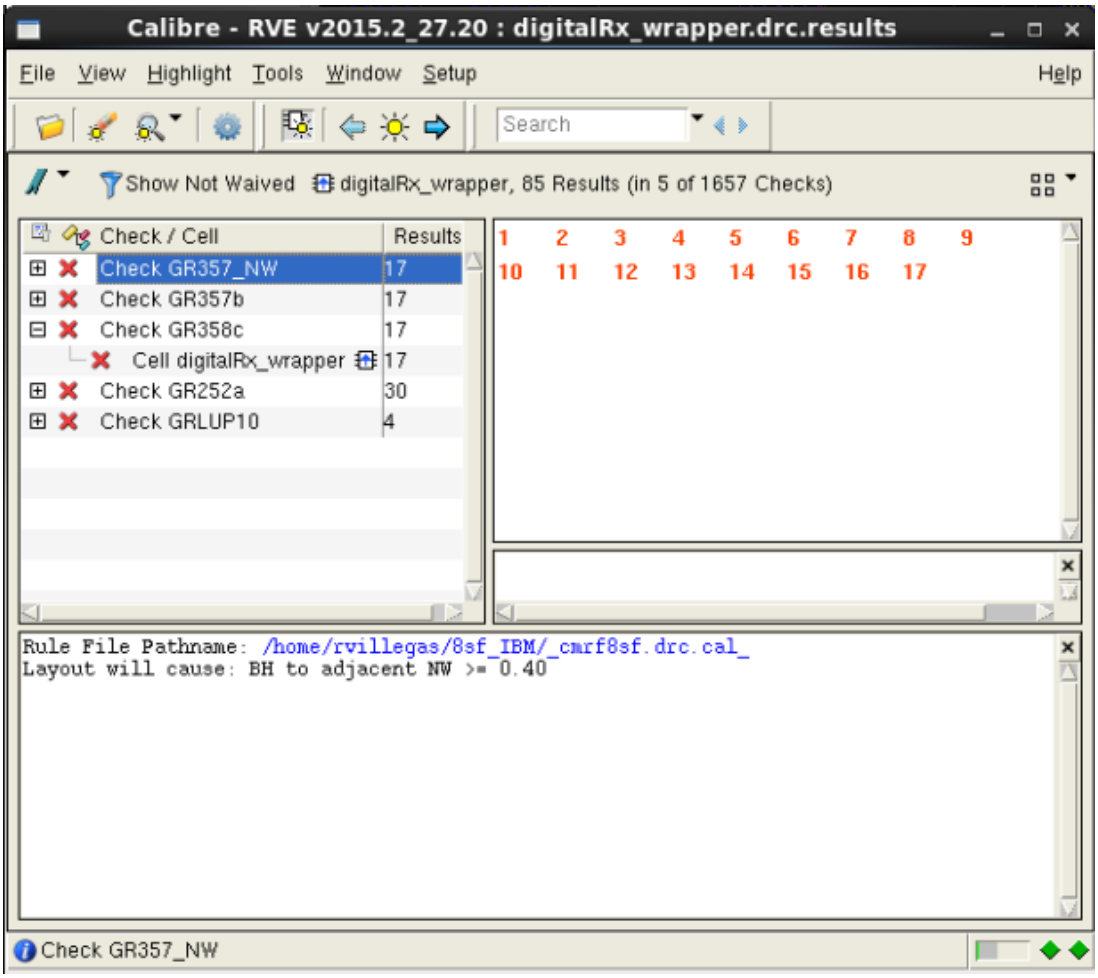

Figure 43 Error signature of the n-well violations

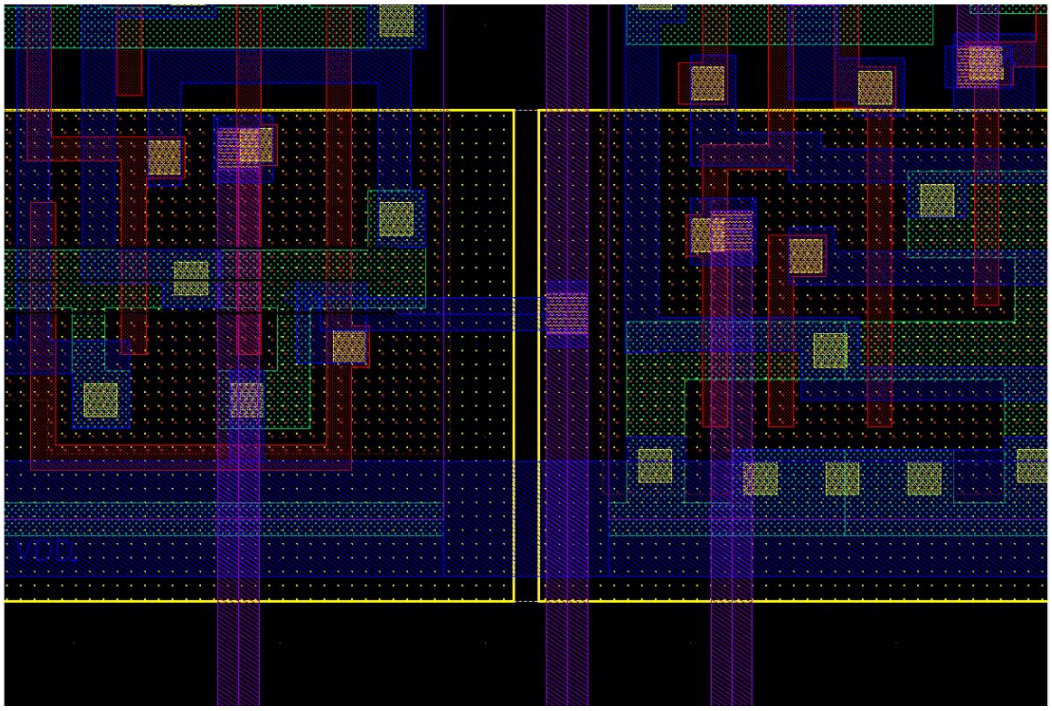

Figure 44 Layout n-well connectivity issue

This issue is fixed by manually adding the missing n-well layers to complete the connection.

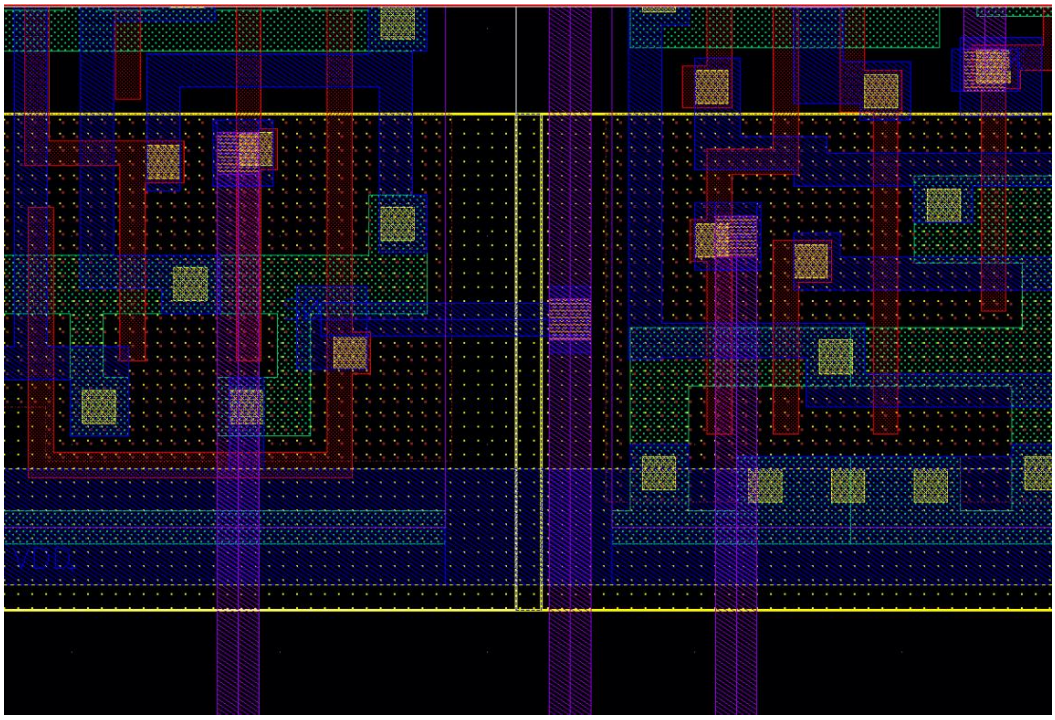

Figure 45 Layout manual n-well fix

Fig. 46 shows that the n-well errors got fixed with the proposed change.

3.5.3 Latch up violations

These violations rise when the RX intersecting n-well area is not big enough proportionally to the area of the PC material intersecting RX.

Three GRLUP10 violations were found in the design as shown in Fig. 46.

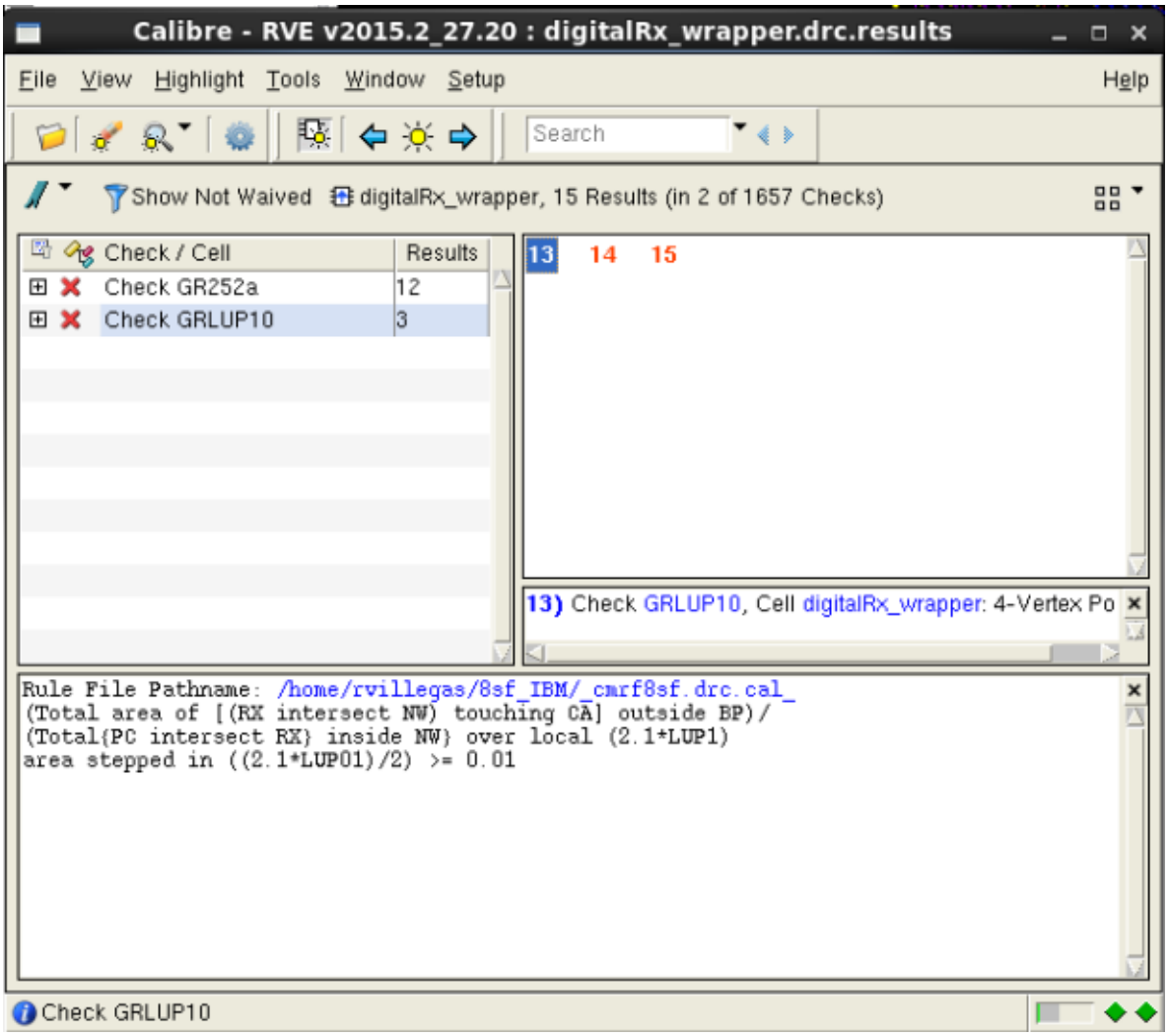

Figure 46 Latchup violation signature

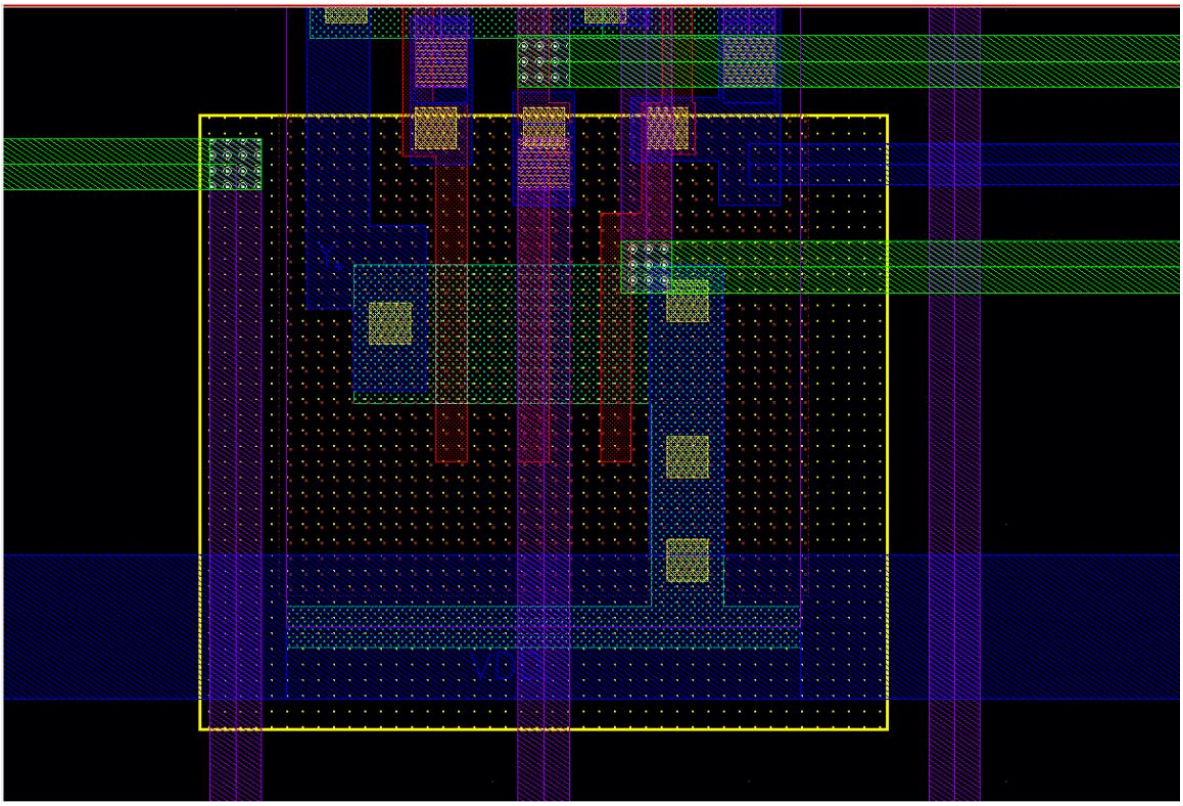

Figure 47 Latchup error layout

Said that, the alternative is to add additional n-well contacts intersecting the RX material and connect it to VDD. Thus, we will need to add an nwCont component and a via to connect from the RX layer to the VDD layer, in this case M1 layer.

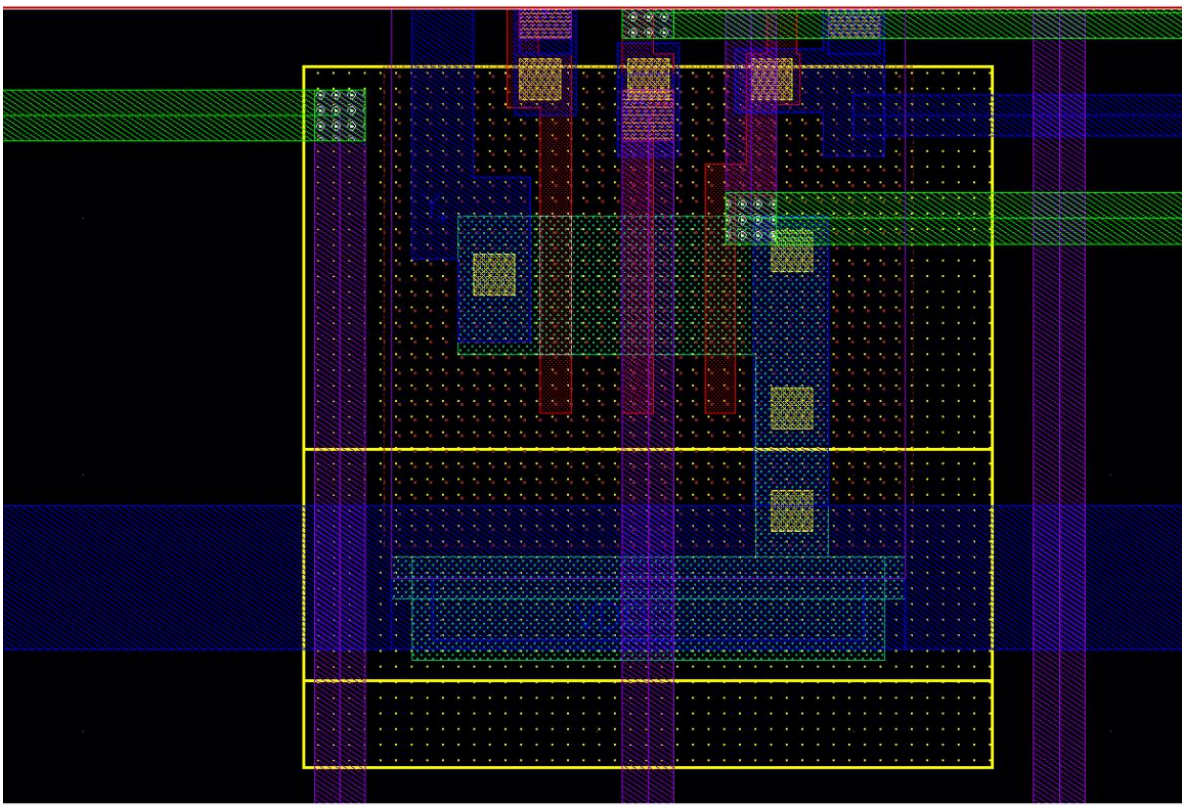

Figure 48 Fixed layout connecting the RX to M1

In Fig. 50 the errors were fixed

3.5.4 Miscellaneous antenna violations

The DRC verification tool also reported several antenna violations in a first run of the analysis.

These violations were caused due to the routing of some paths from the lowest layer to the highest layer, which means that a path is being connected from RX layer to M4 layer.

It is possible to fix this issue by manually adding diodes in the failing areas, thus avoiding the antenna problem. A different approach can be applied in the Encounter tool by limiting the number of layers used to generate the layout. This last approach was used for this system to avoid manual changes in the layout, which can be a potential source of some other rule violations.

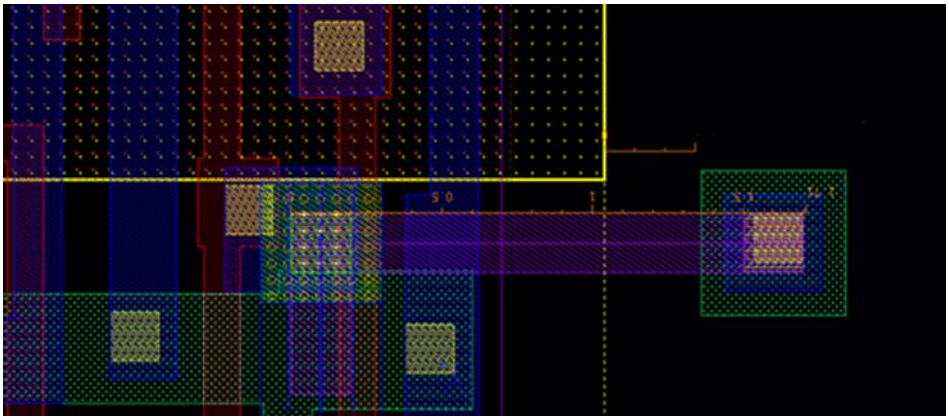

Figure 49 Diode inserted as possible fix for the antenna violation

In a final DRC verification run we can appreciate that all the violations were fixed

DRC Summary Report - digitalRx_wrapper.drc.summary

File Edit Options Windows

```
RULECHECK GRT3WQCAP24a ..... TOTAL Result Count = 0
RULECHECK GRT3WQCAP24b ..... TOTAL Result Count = 0
RULECHECK GR594_M1 ......... TOTAL Result Count = 0
RULECHECK GRT3W594a_M1 ..... TOTAL Result Count = 0
RULECHECK GRT3W594b_M1 ..... TOTAL Result Count = 0
RULECHECK GR595_M1 ......... TOTAL Result Count = 0
RULECHECK GRT3W595_M1 ..... TOTAL Result Count = 0
RULECHECK GR594_M2 ......... TOTAL Result Count = 0
RULECHECK GRT3W594a_M2 ..... TOTAL Result Count = 0
RULECHECK GRT3W594b_M2 ..... TOTAL Result Count = 0
RULECHECK GR595_M2 ......... TOTAL Result Count = 0
RULECHECK GRT3W595_M2 ..... TOTAL Result Count = 0
RULECHECK GR594_M3 ......... TOTAL Result Count = 0
RULECHECK GRT3W594a_M3 ..... TOTAL Result Count = 0
RULECHECK GRT3W594b_M3 ..... TOTAL Result Count = 0
RULECHECK GR595_M3 ......... TOTAL Result Count = 0
RULECHECK GRT3W595_M3 ..... TOTAL Result Count = 0
--------------------------------------------------------------------------------
--- RULECHECK RESULTS STATISTICS (BY CELL)
---
--------------------------------------------------------------------------------
--- SUMMARY
---
TOTAL CPU Time:                  38
TOTAL REAL Time:                 43
TOTAL Original Layer Geometries: 49888 (1340063)
TOTAL DRC RuleChecks Executed:   1655
TOTAL DRC Results Generated:     0 (0)
```

Edit Row 107 Col 1

Figure 50 Final DRC report

3.5.5 LVS analysis

In order to perform the layout v.s. schematic comparison to probe the equivalence of the resultant layout, a Verilog netlist was generated using the Encounter Cadence tool. This file will serve as input for the v2lvs translator.

```
[rvillegas@fv00 LVS]$ v2lvs -v digitalRx_wrapper_prelvs.v -l all_libs.cdl -o digitalRx_wrapper_cdl.cdl
 -s0 VSS -s1 VDD
```

This netlist representation is converted to a Spice friendly file format that can be interpreted by the LVS tool.

During this process we found several issues with the existing technology libraries and their compatibility with the LVS analysis tool. A correct Spice translation was obtained using the v2lvs, but some primitives used by this language were not correctly mapped in the Cadence LVS tool as shown in Fig. 51. A further cleanup of these violations is part of the future job for this project.

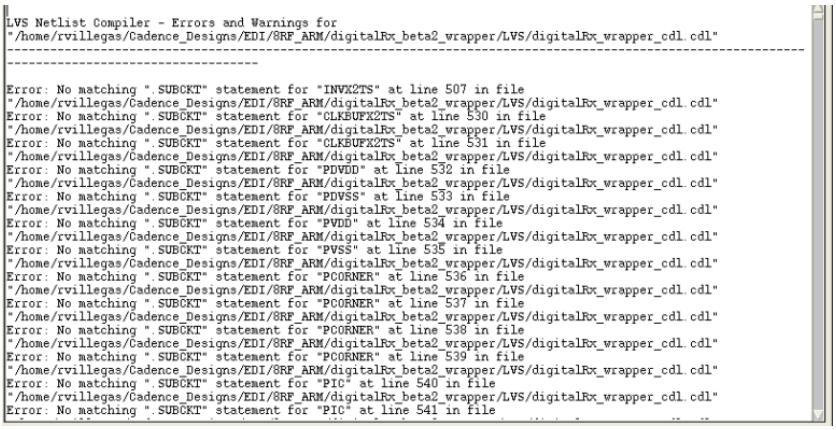

Figure 51 LVS translation issues

Conclusions

This document presented the description of the design and implementation of a digital receiver of a mixed signal SerDes system.

In the first sections an analysis of the previous work developed during the past year in the specialty program was performed, getting some inputs for the enhancement of a PCI Express SerDes system. A research was done in order to define the specs of the project. This allowed us to support and improve the modules, schematics, and HDL code or verification techniques. It also contains a detailed description of the input and output signals of each model, including its RTL representation as well.

Once the environment and previous work were analyzed, some areas of opportunity were identified on early stages of the project i.e. design stage. The fixes and enhancements were described over the document. The module CDR was enhanced to obtain a robust design that handles timing variations using a best sampling technique. This allows us to inject signals with different delays in the testbench. Functional verification was performed and the RTL description was compiled during the logic synthesis process.

As part of the RTL enhancements, an algorithm to obtain the best sampling timing was developed and described using an HDL language. This algorithm fixed an ambiguity issue found in previous designs where the rising or failing-edge of the incoming stream is aligned at the middle of the sampling sweep period. The solution was the insertion edge detector circuit. There are several improvements that can be applied into this technique, as the latency of the best sample index update, which requires a complex FSM for its implementation.

The clean up of the timing negative slack violations included a reiteration of an existing IP to perform the 8B10B decoding process. Some changes to include pipelining capabilities into the system were performed reducing the critical path of the combinatory logic. This IP is required to obtain the final data, but also to get possible errors in the transmission, as the disparity or encoding errors. This contribution will be released as an open source code as it was done by the authors of the original IP. The final maximum frequency was fixed to 500 MHz as reference for the rest of the system, but the deserializer itself can reach a frequency up to 750 MHz.

In the physical synthesis, a layout was generated using the IBM 130nm technology libraries. Timing analysis was launched and cleaned up by replacing standard cells that were not handling correctly the fanout in some of its output cells. Design constraint rules were applied and verified to obtain a clean model that can be used at the manufacturing stage. The results presented in this document can serve as the basis for the fabrication of a System On Chip.

References

[1] A. Patel, “The basics of SerDes (serializers/deserializers) for interfacing”, http://www.planetanalog.com/document.asp?doc_id=528099, 2010

[2] D. Serpanos, “Designing High-speed Communication Systems”, http://www.ercim.eu/publication/Ercim_News/enw37/serpanos.html, n.d.

[3] Chaoscode, “SerDes.png”, Wikimedia Commons, https://commons.wikimedia.org/wiki/File:SerDes.png, June 2011

[4] N. Instruments, "PCI EXPRESS - An Overview of the PCI Express Standard," http://www.ni.com/white-paper/3767/en/, Nov. 05, 2014

[5] D. Warnakulasuriyarachchi, “Design and Simulation of a PCI Express Gen 3.0 Communication Channel.” S.B. Electrical Science and Engineering Theses, MIT, 2009

[6] J. Centeno, “Design and Integration of a Deserializer Module for a SerDes Mixed Signal System on Chip”, Guadalajara: ITESO, 2015.

[7] C. González, “Diseño de modulo Serializador de un sistema SerDes para protocolo de comunicación PCI Express”, Guadalajara: ITESO, 2015.

[8] C. Benz, “Open source 8b10b encoder/decoder”, asic-fpga (at) chuckbenz.com, 2002.

[9] E. Martinez-Guerrero, "Reglas de diseño para layout, course notes," ITESO, Guadalajara Jalisco, Fall 2015.

[10] Department 9G8A, “CMOS8RF Design Manual”, Mixed Signal Technology Development, IBM Microelectronics Division, Fall 2010.

Appendix

A. Verilog source files

a. digtalRx.v

```
module digitalRx(
        input   rst,
  input         clk,
  input         a_rx,
        input   a_rx_n,
  input         disparity_d,
        output  [7:0]    dataout,
        output  dispout,
        output  code_err,
        output  disp_err,
        output  data_valid_pipe,
        output  clk_4f,
        output  ko
);

        wire [3:0]clks_in;
        wire            dispin ;
        wire [9:0] c_parallel_out;
  wire clk_out;
        wire c_data_valid;
  wire  disparity_q;
        wire a_rx_buff;
        wire a_rx_diff_buff;

        assign clk_4f = clks_in[0];

        inputBuffer inputBuffer1(rst, clk, a_rx, a_rx_n, a_rx_buff, a_rx_diff_buff);
        deserializer deserializer1(rst, clks_in, a_rx, c_parallel_out, clk_out, disparity_d,
disparity_q, c_data_valid);
        decodePipe decode1(rst, clks_in[0], c_data_valid, c_parallel_out, disparity_q, dataout,
data_valid_pipe, dispout, disp_err, code_err, ko);
        clock_divider clock_divider1(rst, clk, clks_in);


endmodule
```

b. clock_divider.v

```
module clock_divider(
        input rst,
        input clk,
        output reg [3:0] clks_out
```

```
);

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

        assign clk_div_by_4 = div_by_4_reg[1];

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

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

endmodule
```

c. CDR.v

```
module CDR(rst, clks_in, a_rx, samp_test);
        input rst;
  input [3:0] clks_in;
  input a_rx;
        //output reg c_rx;
        output reg samp_test;

        reg [3:0] c_rx_upsampled;
        reg [3:0] c_rx_upsampled_reg;
        reg [3:0] c_rx_upsampled_2reg;
        reg [3:0] c_rx_upsampled_3reg;
        reg [3:0] c_rx_upsampled_4reg;
        wire [3:0] a_xor;
        reg [3:0] a_xor_reg;
        //wire [3:0] a_and;
        reg [3:0] a_and_reg;
        reg [1:0] best_samp;
        reg [1:0] best_samp_reg;
        reg a_rx_reg;
```

```
        wire [2:0] resA1;
        wire [2:0] resA2;
wire [3:0] num_of_ones_in_rx;
        wire clk;
        //wire edge_detected;

        // Use the phase 0 clk as the system clk
assign clk = clks_in[0];

        // Count the number of "1"s in the upsamples array
        adderA A1(c_rx_upsampled[3:0], resA1);
        //adderA A2(c_rx_upsampled[7:4], resA2);
        //adderB B1(resA1, resA2, num_of_ones_in_rx);

        assign a_xor[3]= c_rx_upsampled[1] ^ c_rx_upsampled[2];
        assign a_xor[0]= c_rx_upsampled[2] ^ c_rx_upsampled[3];
        assign a_xor[1]= c_rx_upsampled[1] ^ c_rx_upsampled[0];
        assign a_xor[2]= c_rx_upsampled[0] ^ c_rx_upsampled[1];

        //assign a_and[0]= a_xor[0] & a_xor[1];
        //assign a_and[1]= a_xor[1] & a_xor[2];
        //assign a_and[2]= a_xor[2] & a_xor[3];
        //assign a_and[3]= a_xor[3] & a_xor[0];

        //assign edge_detected = |(a_xor);


        // CDR
// Up-sample de serial input with clks_in at different
// phases.
genvar index;
generate
                for (index=0; index < 4; index=index+1) begin : gen_upsample
                        always @(posedge clks_in[index], posedge rst) begin
                                if (rst) begin
                                        c_rx_upsampled[index] <= 1'b0;
                                        //a_rx_reg <= 1'b0;
        end else begin
          c_rx_upsampled[index] <= a_rx;
                                        //a_rx_reg <= a_rx;
        end
                        end
  end
 endgenerate

         // Transition tracking
         always@(posedge clk, posedge rst)
         begin
                if(rst) begin
                        a_xor_reg <= 4'h00;
```

```
                        c_rx_upsampled_reg <= 4'h00;
                        c_rx_upsampled_2reg <= 4'h00;
                        c_rx_upsampled_3reg <= 4'h00;
                        c_rx_upsampled_4reg <= 4'h00;
                end else begin
                        a_xor_reg <= a_xor;
                        c_rx_upsampled_reg <= c_rx_upsampled;
                        c_rx_upsampled_2reg <= c_rx_upsampled_reg;
                        c_rx_upsampled_3reg <= c_rx_upsampled_2reg;
                        c_rx_upsampled_4reg <= c_rx_upsampled_3reg;
                end
        end

        always@(posedge clk, posedge rst)
        begin
        if(rst) begin
                best_samp <= 2'h0;
                best_samp_reg <= 2'h0;
        end else begin
                best_samp_reg <= best_samp;
                //if(edge_detected_Xreg) begin
                        casex(a_xor_reg)
                                4'b1xxx: best_samp <= 2'h3 ;
                                4'b01xx: best_samp <= 2'h2 ;
                                4'b001x: best_samp <= 2'h1 ;
                                4'b0001: best_samp <= 2'h0 ;
                                //8'b00001xxx: best_samp <= 3'h3 ;
                                //8'b000001xx: best_samp <= 3'h2 ;
                                //8'b0000001x: best_samp <= 3'h1 ;
                                //8'b00000001: best_samp <= 3'h0 ;
                        default: best_samp <= best_samp ;
                        endcase
                //end
                end
        end

        // Sample only in the best timing according to the
        // transition detector output
        always@(posedge clk, posedge rst)
        begin
                if(rst) begin
                        samp_test <= 1'b0;
                end else begin
                        samp_test <= c_rx_upsampled_4reg[best_samp_reg];
                end
        end

        // Consider a_rx a 1, if a_rx stayed asserted on
// most of the sampling phases.
/*always @(posedge clk, posedge rst)
begin
```

```
        if (rst)
            c_rx <= 1'b0;
        else
            c_rx <= (num_of_ones_in_rx > 4'd4) ? 1'b1 : 1'b0;
    end*/

endmodule
```

d. deserializer.v

```
module deserializer(rst, clks_in, a_rx, c_parallel_out, clk_out, disparity_d, disparity_q,
c_data_valid);
    input rst;
    input [3:0] clks_in;
    input a_rx;
    input disparity_d ;
    output reg [9:0] c_parallel_out;
    output reg clk_out;
    output reg  disparity_q;
            output reg c_data_valid;

    //reg [7:0] c_rx_upsampled;
            // wire [2:0] resA1;
            // wire [2:0] resA2;
    //wire [3:0] num_of_ones_in_rx;
    wire c_rx;
            reg comma_detected_reg;
    wire comma_detected;
    wire clk;
    reg [9:0] shift_reg;
            reg [9:0] shift_2reg;
    reg [3:0] cycle_count;
            wire samp_test;
    integer i;

    // Use the phase 0 clk as the system clk
    assign clk = clks_in[0];

    //adderA A1(c_rx_upsampled[3:0], resA1);
            //adderA A2(c_rx_upsampled[7:4], resA2);
            //adderB B1(resA1, resA2, num_of_ones_in_rx);

            //CDR
            CDR CDR1(
                    .rst(rst),
                    .clks_in(clks_in),
                    .a_rx(a_rx),
                    //.c_rx(c_rx),
                    .samp_test(samp_test)
```

```
                    );


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

    // sipo 10-bit buffer
    // 10 bit shift register
    always@(posedge clk, posedge rst) begin
        if (rst) begin
            shift_reg <= 10'h000;
                                    shift_2reg <= 10'h000;
        end else begin
            shift_reg <= {shift_reg[8:0], samp_test};  // TODO use rst
                                    shift_2reg <= shift_reg;
                    end
    end

    // Look for comma symbol
    assign comma_detected = (shift_2reg == 10'b0011111000) || (shift_2reg ==
10'b1100000111);

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

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

```
            c_data_valid <= 1'b0;

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

           always@(posedge clk, posedge rst) begin
                  if(rst) begin
                         comma_detected_reg <= 1'b0;
                  end else begin
                         if(comma_detected) begin
                                comma_detected_reg <= 1'b1;
                         end else begin
                                comma_detected_reg <= comma_detected_reg;
                         end
                  end
           end

    // reg outputs
    always @(posedge clk, posedge rst) begin
       if(rst) begin
          c_parallel_out <= 10'd0;
       end else begin
          if (c_data_valid) begin
             c_parallel_out <= shift_2reg;
                                end
                   end
    end

endmodule
```

e. inputBuffer.v

```
module inputBuffer(
  input rst,
  input clk,
  input a_rx,
  input a_rx_diff,
  output reg a_rx_buff,
  output reg a_rx_diff_buff
);

always@(posedge clk, posedge rst) begin
  if(rst) begin
```

```
    a_rx_buff <= 1'b0;
          a_rx_diff_buff <= 1'b0;
  end else begin
    a_rx_buff <= a_rx;
          a_rx_diff_buff <= a_rx_diff;
  end
end


endmodule
```

f. decodePipe.v

```
// Chuck Benz, Hollis, NH   Copyright (c)2002
//
// The information and description contained herein is the
// property of Chuck Benz.
//
// Permission is granted for any reuse of this information
// and description as long as this copyright notice is
// preserved.  Modifications may be made as long as this
// notice is preserved.

// per Widmer and Franaszek

// Pipelining techniques added by Rogelio Rivas

module decodePipe (rst, clk, data_valid, datain, dispin, dataout, data_valid_pipe, dispout,
disp_err, code_err, ko) ;
  input rst;
  input clk;
  input data_valid;
  input [9:0]   datain ;
  input         dispin ;
  output [7:0]  dataout ;
  output        dispout ;
  output        code_err ;
  output        disp_err ;
  output ko ;
  output data_valid_pipe ;

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

```
wire hi = datain[8] ;
wire ji = datain[9] ;

reg ai_reg ;
reg bi_reg ;
reg ci_reg ;
reg di_reg ;
reg ei_reg ;
reg ii_reg ;
reg fi_reg ;
reg gi_reg ;
reg hi_reg ;
reg ji_reg ;
reg dispin_reg ;

reg ai_2reg ;
reg bi_2reg ;
reg ci_2reg ;
reg di_2reg ;
reg ei_2reg ;
reg ii_2reg ;
reg fi_2reg ;
reg gi_2reg ;
reg hi_2reg ;
reg ji_2reg ;
//reg aeqb_reg  ;
//reg ceqd_reg ;
reg p22_reg ;
reg p13_reg ;
reg p31_reg ;
reg p40_reg ;
reg p04_reg ;
reg dispin_2reg ;

reg p22bceeqi_reg ;
reg p22bncneeqi_reg ;
reg p13in_reg ;
reg p31i_reg ;
reg p13dei_reg ;
reg p22aceeqi_reg ;
reg p22ancneeqi_reg ;
reg p13en_reg ;
reg anbnenin_reg ;
reg abei_reg ;
//reg cdei_reg ;
reg cndnenin_reg ;
reg disp6a_reg ; // pos disp if p22 and was pos, or p31.
reg disp6a2_reg ;  // disp is ++ after 4 bits
reg disp6a0_reg ; // -- disp after 4 bits
reg feqg_reg ;
reg heqj_reg ;
```

```
// non-zero disparity cases:
//reg p22enin_reg ;
//reg p22ei_reg ;
//wire p13in = p12 & !ii ;
//wire p31i = p31 & ii ;
//reg p31dnenin_reg ;
//wire p13dei = p13 & di & ei & ii ;
//reg p31e_reg ;

reg ai_3reg ;
reg bi_3reg ;
reg ci_3reg ;
reg di_3reg ;
reg ei_3reg ;
reg ii_3reg ;
reg fi_3reg ;
reg gi_3reg ;
reg hi_3reg ;
reg ji_3reg ;
reg dispin_3reg ;
reg disp6p_reg ;
reg disp6n_reg ;
reg p40_2reg ;
reg p04_2reg  ;
reg p31_2reg ;
reg p13_2reg ;

reg ai_4reg ;
reg bi_4reg ;
reg ci_4reg ;
reg di_4reg ;
reg ei_4reg ;
reg ii_4reg ;
reg fi_4reg ;
reg gi_4reg ;
reg hi_4reg ;
reg ji_4reg ;
reg compa_reg ;
reg compb_reg ;
reg compc_reg ;
reg compd_reg ;
reg compe_reg ;
//reg k28_reg ;
reg k28p_reg ;
reg dispin_4reg ;
reg disp6p_2reg ;
reg disp6n_2reg ;
reg dispaux_a_reg ;
reg code_aux_a_reg ;
reg code_aux_b_reg ;
```

```
  reg code_aux_c_reg ;
  reg code_aux_d_reg ;
  reg p13_3reg ;
  reg p31_3reg ;

  reg disp6b_reg ;
  reg fghj22_reg ;
  reg fghjp13_reg ;
  reg fghjp31_reg ;

  reg ao_reg ;
  reg bo_reg ;
  reg co_reg ;
  reg do__reg ;
  reg eo_reg ;
  reg fo_reg ;
  reg go_reg ;
  reg ho_reg ;

  reg ko_reg ;

  reg dispout_reg;
  reg disp_err_reg;
  reg code_err_reg;

  reg data_valid_1pipe;
  reg data_valid_2pipe;
  reg data_valid_3pipe;
  reg data_valid_4pipe;
  reg data_valid_5pipe;

    wire aeqb = (ai_reg & bi_reg) | (!ai_reg & !bi_reg) ;
  wire ceqd = (ci_reg & di_reg) | (!ci_reg & !di_reg) ;
  wire p22 = (ai_reg & bi_reg & !ci_reg & !di_reg) |
             (ci_reg & di_reg & !ai_reg & !bi_reg) |
             ( !aeqb & !ceqd) ;
  wire p13 = ( !aeqb & !ci_reg & !di_reg) |
             ( !ceqd & !ai_reg & !bi_reg) ;
  wire p31 = ( !aeqb & ci_reg & di_reg) |
             ( !ceqd & ai_reg & bi_reg) ;

  wire p40 = ai_reg & bi_reg & ci_reg & di_reg ;
  wire p04 = !ai_reg & !bi_reg & !ci_reg & !di_reg ;

  wire disp6a = p31_reg | (p22_reg & dispin_2reg) ; // pos disp if p22 and was pos, or p31.
   wire disp6a2 = p31_reg & dispin_2reg ;  // disp is ++ after 4 bits
   wire disp6a0 = p13_reg & ! dispin_2reg ; // -- disp after 4 bits

  wire disp6b = (((ei_3reg & ii_3reg & ! disp6a0_reg) | (disp6a_reg & (ei_3reg | ii_3reg)) |
disp6a2_reg |
                    (ei_3reg & ii_3reg & di_3reg)) & (ei_3reg | ii_3reg | di_3reg)) ;
```

```
// The 5B/6B decoding special cases where ABCDE != abcde

wire p22bceeqi = p22_reg & bi_2reg & ci_2reg & (ei_2reg == ii_2reg) ;
wire p22bncneeqi = p22_reg & !bi_2reg & !ci_2reg & (ei_2reg == ii_2reg) ;
wire p13in = p13_reg & !ii_2reg ;
wire p31i = p31_reg & ii_2reg ;
wire p13dei = p13_reg & di_2reg & ei_2reg & ii_2reg ;
wire p22aceeqi = p22_reg & ai_2reg & ci_2reg & (ei_2reg == ii_2reg) ;
wire p22ancneeqi = p22_reg & !ai_2reg & !ci_2reg & (ei_2reg == ii_2reg) ;
wire p13en = p13_reg & !ei_2reg ;
wire anbnenin = !ai_2reg & !bi_2reg & !ei_2reg & !ii_2reg ;
wire abei = ai_2reg & bi_2reg & ei_2reg & ii_2reg ;
//wire cdei = ci_2reg & di_2reg & ei_2reg & ii_2reg ;
wire cndnenin = !ci_2reg & !di_2reg & !ei_2reg & !ii_2reg ;

// non-zero disparity cases:
//wire p22enin = p22_reg & !ei_2reg & !ii_2reg ;
//wire p22ei = p22_reg & ei_2reg & ii_2reg ;
//wire p13in = p12 & !ii ;
//wire p31i = p31 & ii ;
//wire p31dnenin = p31_reg & !di_2reg & !ei_2reg & !ii_2reg ;
//wire p13dei = p13 & di & ei & ii ;
//wire p31e = p31_reg & ei_2reg ;

wire compa = p22bncneeqi_reg | p31i_reg | p13dei_reg | p22ancneeqi_reg |
             p13en_reg | abei_reg | cndnenin_reg ;
wire compb = p22bceeqi_reg | p31i_reg | p13dei_reg | p22aceeqi_reg |
             p13en_reg | abei_reg | cndnenin_reg ;
wire compc = p22bceeqi_reg | p31i_reg | p13dei_reg | p22ancneeqi_reg |
             p13en_reg | anbnenin_reg | cndnenin_reg ;
wire compd = p22bncneeqi_reg | p31i_reg | p13dei_reg | p22aceeqi_reg |
             p13en_reg | abei_reg | cndnenin_reg ;
wire compe = p22bncneeqi_reg | p13in_reg | p13dei_reg | p22ancneeqi_reg |
             p13en_reg | anbnenin_reg | cndnenin_reg ;

wire ao = ai_4reg ^ compa_reg ;
wire bo = bi_4reg ^ compb_reg ;
wire co = ci_4reg ^ compc_reg ;
wire do_ = di_4reg ^ compd_reg ;
wire eo = ei_4reg ^ compe_reg ;

wire feqg = (fi_2reg & gi_2reg) | (!fi_2reg & !gi_2reg) ;
wire heqj = (hi_2reg & ji_2reg) | (!hi_2reg & !ji_2reg) ;

wire fghj22 = (fi_3reg & gi_3reg & !hi_3reg & !ji_3reg) |
              (!fi_3reg & !gi_3reg & hi_3reg & ji_3reg) |
              ( !feqg_reg & !heqj_reg) ;
wire fghjp13 = ( !feqg_reg & !hi_3reg & !ji_3reg) |
               ( !heqj_reg & !fi_3reg & !gi_3reg) ;
wire fghjp31 = ( (!feqg_reg) & hi_3reg & ji_3reg) |
```

```
                    ( !heqj_reg & fi & gi_3reg) ;

  wire dispout_pre = (fghjp31_reg | (disp6b_reg & fghj22_reg) | (hi_4reg & ji_4reg)) & (hi_4reg
| ji_4reg) ;
  wire ko_pre = ( (ci_4reg & di_4reg & ei_4reg & ii_4reg) | ( !ci_4reg & !di_4reg & !ei_4reg &
!ii_4reg) |
                  (p13_3reg & !ei_4reg & ii_4reg & gi_4reg & hi_4reg & ji_4reg) |
                  (p31_3reg & ei_4reg & !ii_4reg & !gi_4reg & !hi_4reg & !ji_4reg)) ;

  /*wire alt7 =   (fi & !gi & !hi & // 1000 cases, where disp6b is 1
                  ((dispin & ci & di & !ei & !ii) | ko |
                   (dispin & !ci & di & !ei & !ii))) |
                 (!fi & gi & hi & // 0111 cases, where disp6b is 0
                  (( !dispin & !ci & !di & ei & ii) | ko |
                   ( !dispin & ci & !di & ei & ii))) ;*/

  //wire k28 = (ci_3reg & di_3reg & ei_3reg & ii_3reg) | ! (ci_3reg | di_3reg | ei_3reg | ii_3reg) ;
  // k28 with positive disp into fghi - .1, .2, .5, and .6 special cases
  wire k28p = ! (ci_3reg | di_3reg | ei_3reg | ii_3reg) ;
  wire fo = (ji_4reg & !fi_4reg & (hi_4reg | !gi_4reg | k28p_reg)) |
            (fi_4reg & !ji_4reg & (!hi_4reg | gi_4reg | !k28p_reg)) |
            (k28p_reg & gi_4reg & hi_4reg) |
            (!k28p_reg & !gi_4reg & !hi_4reg) ;
  wire go = (ji_4reg & !fi_4reg & (hi_4reg | !gi_4reg | !k28p_reg)) |
            (fi_4reg & !ji_4reg & (!hi_4reg | gi_4reg |k28p_reg)) |
            (!k28p_reg & gi_4reg & hi_4reg) |
            (k28p_reg & !gi_4reg & !hi_4reg) ;
  wire ho = ((ji_4reg ^ hi_4reg) & ! ((!fi_4reg & gi_4reg & !hi_4reg & ji_4reg & !k28p_reg) |
(!fi_4reg & gi_4reg & hi_4reg & !ji_4reg & k28p_reg) |
                                (fi_4reg & !gi_4reg & !hi_4reg & ji_4reg & !k28p_reg) | (fi_4reg &
!gi_4reg & hi_4reg & !ji_4reg & k28p_reg))) |
            (!fi_4reg & gi_4reg & hi_4reg & ji_4reg) | (fi_4reg & !gi_4reg & !hi_4reg & !ji_4reg) ;

  wire disp6p = (p31_reg & (ei_2reg | ii_2reg)) | (p22_reg & ei_2reg & ii_2reg) ;
  wire disp6n = (p13_reg & ! (ei_2reg & ii_2reg)) | (p22_reg & !ei_2reg & !ii_2reg) ;
  /*
  wire disp4p = fghjp31 ;
  wire disp4n = fghjp13 ;*/

  wire code_aux_a = p40_2reg | p04_2reg | (fi_3reg & gi_3reg & hi_3reg & ji_3reg) | (!fi_3reg
& !gi_3reg & !hi_3reg & !ji_3reg) |
                    (p13_2reg & !ei_3reg & !ii_3reg) | (p31_2reg & ei_3reg & ii_3reg) |
                    (ei_3reg & ii_3reg & fi_3reg & gi_3reg & hi_3reg) | (!ei_3reg & !ii_3reg &
!fi_3reg & !gi_3reg & !hi_3reg) ;

  wire code_aux_b = (ei_3reg & !ii_3reg & gi_3reg & hi_3reg & ji_3reg) | (!ei_3reg & ii_3reg &
!gi_3reg & !hi_3reg & !ji_3reg) |
                    (!p31_2reg & ei_3reg & !ii_3reg & !gi_3reg & !hi_3reg & !ji_3reg) |
                    (!p13_2reg & !ei_3reg & ii_3reg & gi_3reg & hi_3reg & ji_3reg) ;

  wire code_aux_c = (((ei_3reg & ii_3reg & !gi_3reg & !hi_3reg & !ji_3reg) |
```

```
                        (!ei_3reg & !ii_3reg & gi_3reg & hi_3reg & ji_3reg)) &
                       ! ((ci_3reg & di_3reg & ei_3reg) | (!ci_3reg & !di_3reg & !ei_3reg))) ;

  wire code_aux_d = (ci_3reg & di_3reg & ei_3reg & ii_3reg & !fi_3reg & !gi_3reg & !hi_3reg) |
                    (!ci_3reg & !di_3reg & !ei_3reg & !ii_3reg & fi_3reg & gi_3reg & hi_3reg) ;

  wire code_err_pre = code_aux_a_reg |
                    code_aux_b_reg |
                    code_aux_c_reg |
                    (disp6p_2reg & fghjp31_reg) | (disp6n & fghjp13_reg) |
                    (ai_4reg & bi_4reg & ci_4reg & !ei_4reg & !ii_4reg & ((!fi_4reg & !gi_4reg) |
fghjp13_reg)) |
                    (!ai_4reg & !bi_4reg & !ci_4reg & ei_4reg & ii_4reg & ((fi_4reg & gi_4reg) |
fghjp31_reg)) |
                    (fi_4reg & gi_4reg & !hi_4reg & !ji_4reg & disp6p_2reg) |
                    (!fi_4reg & !gi_4reg & hi_4reg & ji_4reg & disp6n_2reg) |
                    code_aux_d_reg ;

  assign dataout = {ho_reg, go_reg, fo_reg, eo_reg, do__reg, co_reg, bo_reg, ao_reg} ;
  assign data_valid_pipe = data_valid_5pipe ;
  assign dispout = dispout_reg ;
  assign disp_err = disp_err_reg ;
  assign code_err = code_err_reg ;
  assign ko = ko_reg ;

  // my disp err fires for any legal codes that violate disparity, may fire for illegal codes
  wire dispaux_a = (dispin_3reg & disp6p_reg) | (disp6n_reg & !dispin_3reg) |
                    (dispin_3reg & !disp6n_reg & fi_3reg & gi_3reg) |
                    (dispin_3reg & ai_3reg & bi_3reg & ci_3reg) |
                                (!dispin_3reg & !disp6p_reg & !fi_3reg & !gi_3reg) |
                    (!dispin_3reg & !ai_3reg & !bi_3reg & !ci_3reg);
  wire disp_err_pre = ( dispaux_a_reg |
                    (dispin_4reg & !disp6n_2reg & fghjp31_reg) |
                    (!dispin_4reg & !disp6p_2reg & fghjp13_reg) |
                    (disp6p_2reg & fghjp31_reg) | (disp6n_2reg & fghjp13_reg)) ;

  always@(posedge clk, posedge rst) begin   // First Pipe Stage
        if(rst) begin
                ai_reg <= 0 ;
                bi_reg <= 0 ;
                ci_reg <= 0 ;
                di_reg <= 0 ;
                ei_reg <= 0 ;
                ii_reg <= 0 ;
                fi_reg <= 0 ;
                gi_reg <= 0 ;
                hi_reg <= 0 ;
                ji_reg <= 0 ;
                data_valid_1pipe <= 0 ;
                dispin_reg <= 0 ;
        end else begin
```

```
                data_valid_1pipe <= data_valid ;
                dispin_reg <= dispin ;
                if (data_valid) begin
                        ai_reg <= ai ;
                        bi_reg <= bi ;
                        ci_reg <= ci ;
                        di_reg <= di ;
                        ei_reg <= ei ;
                        ii_reg <= ii ;
                        fi_reg <= fi ;
                        gi_reg <= gi ;
                        hi_reg <= hi ;
                        ji_reg <= ji ;
                end else begin
                        ai_reg <= ai_reg ;
                        bi_reg <= bi_reg ;
                        ci_reg <= ci_reg ;
                        di_reg <= di_reg ;
                        ei_reg <= ei_reg ;
                        ii_reg <= ii_reg ;
                        fi_reg <= fi_reg ;
                        gi_reg <= gi_reg ;
                        hi_reg <= hi_reg ;
                        ji_reg <= ji_reg ;
                end
        end
end

always@(posedge clk, posedge rst) begin   // Second Pipe Stage
        if(rst) begin
                ai_2reg <= 0 ;
                bi_2reg <= 0 ;
                ci_2reg <= 0 ;
                di_2reg <= 0 ;
                ei_2reg <= 0 ;
                ii_2reg <= 0 ;
                fi_2reg <= 0 ;
                gi_2reg <= 0 ;
                hi_2reg <= 0 ;
                ji_2reg <= 0 ;
                //aeqb_reg <= 0 ;
                //ceqd_reg <= 0 ;
                p22_reg <= 0 ;
                p13_reg <= 0 ;
                p31_reg <= 0 ;
                p40_reg <= 0 ;
                p04_reg <= 0 ;
                data_valid_2pipe <= 0 ;
                dispin_2reg <= 0 ;
        end else begin
          data_valid_2pipe <= data_valid_1pipe ;
```

```
                ai_2reg <= ai_reg ;
                bi_2reg <= bi_reg ;
                ci_2reg <= ci_reg ;
                di_2reg <= di_reg ;
                ei_2reg <= ei_reg ;
                ii_2reg <= ii_reg ;
                fi_2reg <= fi_reg ;
                gi_2reg <= gi_reg ;
                hi_2reg <= hi_reg ;
                ji_2reg <= ji_reg ;
                //aeqb_reg <= aeqb ;
                //ceqd_reg <= ceqd ;
                p22_reg <= p22 ;
                p13_reg <= p13 ;
                p31_reg <= p31 ;
                p40_reg <= p40 ;
                p04_reg <= p04 ;
                dispin_2reg <= dispin_reg ;
        end
end

  always@(posedge clk, posedge rst) begin   // Third Pipe Stage
        if(rst) begin
                ai_3reg <= 0 ;
                bi_3reg <= 0 ;
                ci_3reg <= 0 ;
                di_3reg <= 0 ;
                ei_3reg <= 0 ;
                ii_3reg <= 0 ;
                fi_3reg <= 0 ;
                gi_3reg <= 0 ;
                hi_3reg <= 0 ;
                ji_3reg <= 0 ;
                p22bceeqi_reg <= 0 ;
                p22bncneeqi_reg <= 0 ;
                p13in_reg <= 0 ;
                p31i_reg <= 0 ;
                p13dei_reg <= 0 ;
                p22aceeqi_reg <= 0 ;
                p22ancneeqi_reg <= 0 ;
                p13en_reg <= 0 ;
                anbnenin_reg <= 0 ;
                abei_reg <= 0 ;
                //cdei_reg <= 0 ;
                cndnenin_reg <= 0 ;
                //p22enin_reg <= 0 ;
                //p22ei_reg <= 0 ;
                //p31dnenin_reg <= 0 ;
                //p31e_reg <= 0 ;
                data_valid_3pipe <= 0 ;
                feqg_reg <= 0 ;
```

```
                heqj_reg <= 0 ;
                disp6a_reg <= 0 ;  // pos disp if p22 and was pos, or p31.
                disp6a2_reg <= 0 ;  // disp is ++ after 4 bits
                disp6a0_reg <= 0 ;
                disp6p_reg <= 0 ;
                disp6n_reg <= 0 ;
                dispin_3reg <= 0 ;
                p40_2reg <= 0 ;
                p04_2reg <= 0 ;
                p31_2reg <= 0 ;
                p13_2reg <= 0 ;
        end else begin
                data_valid_3pipe <= data_valid_2pipe ;
                ai_3reg <= ai_2reg ;
                bi_3reg <= bi_2reg ;
                ci_3reg <= ci_2reg ;
                di_3reg <= di_2reg ;
                ei_3reg <= ei_2reg ;
                ii_3reg <= ii_2reg ;
                fi_3reg <= fi_2reg ;
                gi_3reg <= gi_2reg ;
                hi_3reg <= hi_2reg ;
                ji_3reg <= ji_2reg ;
                p22bceeqi_reg <= p22bceeqi ;
                p22bncneeqi_reg <= p22bncneeqi ;
                p13in_reg <= p13in ;
                p31i_reg <= p31i ;
                p13dei_reg <= p13dei ;
                p22aceeqi_reg <= p22aceeqi ;
                p22ancneeqi_reg <= p22ancneeqi ;
                p13en_reg <= p13en ;
                anbnenin_reg <= anbnenin ;
                abei_reg <= abei ;
                //cdei_reg <= cdei ;
                cndnenin_reg <= cndnenin ;
                //p22enin_reg <= p22enin ;
                //p22ei_reg <= p22ei ;
                //p31dnenin_reg <= p31dnenin ;
                //p31e_reg <= p31e ;
                feqg_reg <= feqg ;
                heqj_reg <= heqj ;
                disp6a_reg <= disp6a; // pos disp if p22 and was pos, or p31.
                disp6a2_reg <= disp6a2;  // disp is ++ after 4 bits
                disp6a0_reg <= disp6a0;
                disp6p_reg <= disp6p ;
                disp6n_reg <= disp6n ;
                dispin_3reg <= dispin_2reg ;
                p40_2reg <= p40_reg ;
                p04_2reg <= p04_reg ;
                p31_2reg <= p31_reg ;
                p13_2reg <= p13_reg ;
```

```
        end
        end

    always@(posedge clk, posedge rst) begin   // Fourth Pipe Stage
        if(rst) begin
                ai_4reg <= 0 ;
                bi_4reg <= 0 ;
                ci_4reg <= 0 ;
                di_4reg <= 0 ;
                ei_4reg <= 0 ;
                ii_4reg <= 0 ;
                fi_4reg <= 0 ;
                gi_4reg <= 0 ;
                hi_4reg <= 0 ;
                ji_4reg <= 0 ;
                compa_reg <= 0 ;
                compb_reg <= 0 ;
                compc_reg <= 0 ;
                compd_reg <= 0 ;
                compe_reg <= 0 ;
                //k28_reg <= 0 ;
                k28p_reg <= 0 ;
                data_valid_4pipe <= 0 ;
                disp6b_reg <= 0;
                fghj22_reg <= 0 ;
                fghjp13_reg <= 0 ;
                fghjp31_reg <= 0 ;
                disp6p_2reg <= 0 ;
                disp6n_2reg <= 0 ;
                dispin_4reg <= 0 ;
                dispaux_a_reg <= 0 ;
                code_aux_a_reg <= 0 ;
                code_aux_b_reg <= 0 ;
                code_aux_c_reg <= 0 ;
                code_aux_d_reg <= 0 ;
                p31_3reg <= 0 ;
                p13_3reg <= 0 ;
        end else begin
                data_valid_4pipe <= data_valid_3pipe ;
                ai_4reg <= ai_3reg ;
                bi_4reg <= bi_3reg ;
                ci_4reg <= ci_3reg ;
                di_4reg <= di_3reg ;
                ei_4reg <= ei_3reg ;
                ii_4reg <= ii_3reg ;
                fi_4reg <= fi_3reg ;
                gi_4reg <= gi_3reg ;
                hi_4reg <= hi_3reg ;
                ji_4reg <= ji_3reg ;
                compa_reg <= compa ;
                compb_reg <= compb ;
```

```
                compc_reg <= compc ;
                compd_reg <= compd ;
                compe_reg <= compe ;
                //k28_reg <= k28 ;
                k28p_reg <= k28p ;
                disp6b_reg <= disp6b ;
                fghj22_reg <= fghj22 ;
                fghjp13_reg <= fghjp13 ;
                fghjp31_reg <= fghjp31 ;
                disp6p_2reg <= disp6p_reg ;
                disp6n_2reg <= disp6n_reg ;
                dispin_4reg <= dispin_3reg ;
                dispaux_a_reg <= dispaux_a ;
                code_aux_a_reg <= code_aux_a ;
                code_aux_b_reg <= code_aux_b ;
                code_aux_c_reg <= code_aux_c ;
                code_aux_d_reg <= code_aux_d ;
                p31_3reg <= p31_2reg ;
                p13_3reg <= p13_2reg ;
        end
end

    always@(posedge clk, posedge rst) begin   // Fifth Pipe Stage
        if(rst) begin
                ao_reg <= 0 ;
                bo_reg <= 0 ;
                co_reg <= 0 ;
                do__reg <= 0 ;
                eo_reg <= 0 ;
                fo_reg <= 0 ;
                go_reg <= 0 ;
                ho_reg <= 0 ;
                data_valid_5pipe <= 0 ;
                dispout_reg <= 0 ;
                disp_err_reg <= 0 ;
                code_err_reg <= 0 ;
                ko_reg <= 0 ;
        end else begin
                data_valid_5pipe <= data_valid_4pipe ;
                ao_reg <= ao ;
                bo_reg <= bo ;
                co_reg <= co ;
                do__reg <= do_ ;
                eo_reg <= eo ;
                fo_reg <= fo ;
                go_reg <= go ;
                ho_reg <= ho ;
                dispout_reg <= dispout_pre ;
                disp_err_reg <= disp_err_pre ;
                code_err_reg <= code_err_pre ;
                ko_reg <= ko_pre;
```

```
        end
  end

endmodule
```

g. adderA.v

```
module adderA(data_in, res_A);
// This modules calculates the sum of four 1-bit length data.
// A combinatory equation describes each bit of the output value res_A.
// Signals:
        input [3:0] data_in;                    // Input sequence. This data is obtained from the
XOR comparison.
        output [2:0] res_A;                     // Output result. Positive value to store the
number of 1's of data_in

        wire term1 = (~data_in[3]&data_in[1]&data_in[0]);
        wire term2 = (~data_in[3]&data_in[2]&data_in[0]);
        wire term3 = (data_in[2]&data_in[1]&~data_in[0]);
        wire term4 = (data_in[3]&~data_in[2]&data_in[0]);
        wire term5 = (data_in[3]&~data_in[2]&data_in[1]);
        wire term6 = (data_in[3]&data_in[2]&~data_in[1]);


        assign res_A[0] = ^(data_in);
        assign res_A[1] = term1|term2|term3|term4|term5|term6;
        assign res_A[2] = &(data_in);

endmodule
```

h. adderB.v

```
module adderB (data_a, data_b, res_B);
// This module calculates the number of 1's in an 8-bit sequence.
// Signals:
        input [2:0] data_a;
        input [2:0] data_b;
        output [3:0] res_B;

        wire thirdFour;
        wire fourthFour;
        wire fifthFour;

        assign thirdFour = data_a[1]&data_b[1];
        assign fourthFour = data_a[1]&(data_a[0]&data_b[0]);
        assign fifthFour = data_b[1]&(data_a[0]&data_b[0]);

        assign res_B[0] = data_a[0]^data_b[0];
```

```
        assign res_B[1] = data_a[1]^data_b[1]^(data_a[0]&data_b[0]);
        assign res_B[2] = data_a[2]^data_b[2]^thirdFour^fourthFour^fifthFour;
        assign res_B[3] = data_a[2]&data_b[2];

endmodule
```

i. deserializerTB.sv

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

`timescale 1ns / 100ps
module tb_deserializer;

parameter CLOCK_PERIOD = 100;
parameter MAX_JITTER = 10;
parameter DELTA_JITTER = 17 ;

wire      clk;
reg        ref_clock;
reg       a_rst;
reg       a_rx;

reg  [3:0] clocks_in;
wire [9:0] c_parallel_out;
reg disparity_in;
wire disparity_out;
wire clock_out;
wire clock_4f;

// Digital Rx
wire [7:0] dataout;
wire dispout;
wire code_err;
wire disp_err;
wire data_valid_pipe;
wire ko;

reg[7:0] data_to_encode;

wire [9:0] data_encoded;
reg [7:0] test_values;
reg [9:0] data_encoded_reg;
```

```
wire [3:0] clock_phases;

integer clock_period;
integer max_jitter;
integer delta_jitter;
integer noisy_clock;

integer i,j;

// clock signal generation (period = 20)
initial
begin
        ref_clock = 0;
        clock_period = 100;
        max_jitter = 10;
        delta_jitter = $random % max_jitter ;
        noisy_clock = ($random % 2) ? clock_period + delta_jitter : clock_period - delta_jitter;
        $display("The noisy clock value is %d ",noisy_clock);
        $display("Delta jitter is %d",DELTA_JITTER);
   clocks_in = 8'h0f;
        disparity_in = 1'b0;
   #50;
   forever begin
      for (i=0; i < 4; i=i+1) begin
         #10 clocks_in[i] = ~clocks_in[i];
                        ref_clock = ~ref_clock;
         clocks_in[4+i] = ~clocks_in[4+i];
      end
              //#10 ref_clock = ~ref_clock;
   end
end

assign clk = clocks_in[0];


// serial input generation
initial
begin
   // reset
        test_values = 8'h00;
   a_rst    = 1'b0;
   a_rx = 1'b0;
        data_to_encode = 0;
   #100  a_rst = 1'b1;
   #500 a_rst = 1'b0;

   @(posedge clk);
   // send comma
        #327;
   serial_mblb(10'h0f8);
   // send test patterns
```

```
        serial_mblb(10'h365); // 5
        serial_mblb(10'h366); // 6
        serial_mblb(10'h347); // 7
        serial_mblb(10'h0a7); // 8
        serial_mblb(10'h369); // 9
        //test_values = 8'hfd;
        //
        //consecutive_frame(5);

   serial_mblb(10'h2aa);
   serial_mblb(10'h155);
   serial_mblb(10'h10f);
   serial_mblb(10'h000);
        #2000;
        $finish;
end

always@(negedge clk) begin
        test_values = test_values + 1 ;

end


task serial_mblb(input[9:0] data);
begin
   for (j=9; j >= 0; j=j-1) begin
      #DELTA_JITTER;
                //#17;
                a_rx = data[j];
                //#10;
      @(posedge clk);
   end
end
endtask

task consecutive_frame(input[7:0] init_value);
begin
        data_to_encode <= init_value;
        @(posedge clk);
        //data_encoded_reg = init_value;
   for (j=10; j >= 0; j=j-1) begin
                data_to_encode <= data_to_encode + 1;
                #50;
      @(posedge clk);
   end
end
endtask

//encode encode1( {1'b0 , data_to_encode} , 1'b0, data_encoded, dispout_aux);
digitalRx superDut(a_rst, ref_clock, a_rx, ~a_rx, disparity_in, dataout, dispout, code_err,
disp_err, data_valid_pipe, clock_4f, ko);
```

```
//deserializer dut(a_rst, clocks_in[3:0], a_rx, c_parallel_out, clock_out, disparity_in,
disparity_out);
clock_divider clock_divider_dut(a_rst, ref_clock, clock_phases);


endmodule
```

B. Logic synthesis files

a. Netlist

```
// Generated by Cadence Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1
// Modified by Rogelio Rivas
// Verification Directory fv/digitalRx

module clock_divider(rst, clk, clks_out);
  input rst, clk;
  output [3:0] clks_out;
  wire rst, clk;
  wire [3:0] clks_out;
  wire [1:0] div_by_4_reg;
  wire [1:0] div_by_4_q;
  wire n_0, n_1;
  DFFRHQX4TS \clks_out_reg[3] (.RN (n_0), .CK (clk), .D (clks_out[2]),
       .Q (clks_out[3]));
  DFFRHQX4TS \clks_out_reg[2] (.RN (n_0), .CK (clk), .D (clks_out[1]),
       .Q (clks_out[2]));
  DFFRHQX4TS \clks_out_reg[1] (.RN (n_0), .CK (clk), .D (clks_out[0]),
       .Q (clks_out[1]));
  DFFRHQX4TS \clks_out_reg[0] (.RN (n_0), .CK (clk), .D
       (div_by_4_reg[1]), .Q (clks_out[0]));
  DFFRHQX4TS \div_by_4_reg_reg[1] (.RN (n_0), .CK (clk), .D
       (div_by_4_q[1]), .Q (div_by_4_reg[1]));
  INVX2TS g11(.A (rst), .Y (n_0));
  DFFRHQX2TS \div_by_4_q_reg[1] (.RN (n_0), .CK (clk), .D
       (div_by_4_q[0]), .Q (div_by_4_q[1]));
  DFFRHQX4TS \div_by_4_q_reg[0] (.RN (n_0), .CK (clk), .D (n_1), .Q
       (div_by_4_q[0]));
  CLKINVX1TS g8(.A (div_by_4_q[1]), .Y (n_1));
endmodule

module decodePipe(rst, clk, data_valid, datain, dispin, dataout,
    data_valid_pipe, dispout, disp_err, code_err, ko);
  input rst, clk, data_valid, dispin;
  input [9:0] datain;
```

```
output [7:0] dataout;
output data_valid_pipe, dispout, disp_err, code_err, ko;
wire rst, clk, data_valid, dispin;
wire [9:0] datain;
wire [7:0] dataout;
wire data_valid_pipe, dispout, disp_err, code_err, ko;
wire UNCONNECTED, UNCONNECTED0, UNCONNECTED1, UNCONNECTED2,
    UNCONNECTED3, UNCONNECTED4, UNCONNECTED5, UNCONNECTED6;
wire UNCONNECTED7, UNCONNECTED8, UNCONNECTED9, UNCONNECTED10,
    UNCONNECTED11, UNCONNECTED12, UNCONNECTED13, UNCONNECTED14;
wire UNCONNECTED15, UNCONNECTED16, UNCONNECTED17, UNCONNECTED18,
    UNCONNECTED19, UNCONNECTED20, UNCONNECTED21, UNCONNECTED22;
wire UNCONNECTED23, UNCONNECTED24, UNCONNECTED25, UNCONNECTED26,
    UNCONNECTED27, UNCONNECTED28, UNCONNECTED29, UNCONNECTED30;
wire UNCONNECTED31, UNCONNECTED32, UNCONNECTED33, UNCONNECTED34,
    UNCONNECTED35, UNCONNECTED36, UNCONNECTED37, UNCONNECTED38;
wire UNCONNECTED39, UNCONNECTED40, UNCONNECTED41, UNCONNECTED42,
    UNCONNECTED43, UNCONNECTED44, UNCONNECTED45, UNCONNECTED46;
wire UNCONNECTED47, UNCONNECTED48, UNCONNECTED49, UNCONNECTED50,
    UNCONNECTED51, UNCONNECTED52, UNCONNECTED53, UNCONNECTED54;
wire UNCONNECTED55, UNCONNECTED56, UNCONNECTED57, UNCONNECTED58,
    UNCONNECTED59, UNCONNECTED60, UNCONNECTED61, UNCONNECTED62;
wire UNCONNECTED63, UNCONNECTED64, UNCONNECTED65, UNCONNECTED66,
    UNCONNECTED67, UNCONNECTED68, UNCONNECTED69, abei_reg;
wire ai_4reg, ai_reg, anbnenin_reg, bi_4reg, bi_reg, ci_2reg,
    ci_4reg, ci_reg;
wire cndnenin_reg, code_aux_a_reg, code_aux_b_reg, code_aux_c_reg,
    code_aux_d_reg, compa_reg, compb_reg, compc_reg;
wire compd_reg, compe_reg, data_valid_1pipe, data_valid_2pipe,
    data_valid_3pipe, data_valid_4pipe, di_2reg, di_reg;
wire disp6a0_reg, disp6a2_reg, disp6a_reg, disp6b_reg, dispaux_a_reg,
    dispin_3reg, dispin_4reg, dispin_reg;
wire ei_3reg, ei_reg, fghj22_reg, fghjp31_reg, fi_reg, gi_reg,
    hi_3reg, hi_4reg;
wire hi_reg, ii_3reg, ii_reg, ji_3reg, ji_reg, n_0, n_1, n_2;
wire n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_10;
wire n_11, n_12, n_13, n_21, n_22, n_23, n_24, n_25;
wire n_26, n_27, n_28, n_29, n_30, n_31, n_32, n_33;
wire n_34, n_35, n_36, n_37, n_38, n_39, n_40, n_41;
wire n_42, n_43, n_44, n_45, n_46, n_47, n_48, n_51;
wire n_52, n_53, n_54, n_55, n_56, n_57, n_58, n_59;
wire n_60, n_61, n_62, n_63, n_64, n_65, n_66, n_67;
wire n_68, n_69, n_70, n_71, n_72, n_73, n_74, n_75;
wire n_76, n_77, n_78, n_79, n_80, n_81, n_82, n_83;
wire n_84, n_85, n_86, n_87, n_88, n_89, n_90, n_91;
wire n_92, n_93, n_94, n_95, n_96, n_97, n_98, n_99;
wire n_100, n_101, n_102, n_103, n_104, n_105, n_106, n_107;
wire n_108, n_109, n_110, n_111, n_112, n_113, n_114, n_115;
wire n_116, n_118, n_119, n_120, n_121, n_122, n_123, n_124;
wire n_125, n_126, n_127, n_128, n_129, n_130, n_131, n_132;
wire n_133, n_134, n_135, n_136, n_137, n_138, n_139, n_140;
```

```
  wire n_141, n_143, n_144, n_145, n_146, n_147, n_148, n_149;
  wire n_150, n_151, n_152, n_153, n_154, n_155, n_156, n_157;
  wire n_158, n_159, n_161, n_162, n_164, n_165, n_166, n_167;
  wire n_168, n_169, n_170, n_171, n_172, n_173, n_174, n_175;
  wire n_176, n_178, n_180, n_181, n_182, n_183, n_184, n_186;
  wire n_187, n_188, n_191, n_193, n_194, n_195, n_196, n_197;
  wire n_198, n_201, n_202, n_203, n_204, n_205, n_206, n_207;
  wire n_208, n_209, n_210, n_213, n_214, n_215, n_216, n_217;
  wire n_219, n_221, n_222, n_223, n_224, n_227, n_229, n_230;
  wire n_231, n_233, n_234, n_236, n_238, n_239, n_240, n_243;
  wire n_244, n_245, n_246, n_247, n_248, n_249, n_251, n_254;
  wire n_255, n_256, n_257, n_258, n_259, n_260, n_261, n_266;
  wire n_267, n_268, n_269, n_270, n_271, n_272, n_273, n_274;
  wire n_276, n_278, n_279, n_280, n_281, n_282, n_283, n_284;
  wire n_285, n_286, n_287, n_288, n_289, n_290, n_291, n_292;
  wire n_293, n_294, n_295, n_296, n_297, n_298, n_299, n_300;
  wire n_301, n_302, n_304, n_305, n_307, n_308, n_352, n_363;
  wire n_364, n_371, n_372, n_373, n_376, n_377, n_378, n_379;
  wire n_380, n_381, n_382, n_383, n_384, n_385, n_386, n_387;
  wire n_388, p04_2reg, p04_reg, p13_2reg, p13_3reg, p13_reg,
      p13dei_reg, p13en_reg;
  wire p13in_reg, p22_reg, p22aceeqi_reg, p22ancneeqi_reg,
      p22bceeqi_reg, p31_3reg, p31i_reg, p40_2reg;
  wire p40_reg;
  DFFRHQX2TS ai_3reg_reg(.RN (n_2), .CK (clk), .D (n_32), .Q (n_42));
  DFFRHQX2TS anbnenin_reg_reg(.RN (n_2), .CK (clk), .D (n_141), .Q
      (anbnenin_reg));
  DFFRHQX2TS bi_3reg_reg(.RN (n_2), .CK (clk), .D (n_30), .Q (n_43));
  DFFRHQX2TS cndnenin_reg_reg(.RN (n_2), .CK (clk), .D (n_140), .Q
      (cndnenin_reg));
  DFFRHQX2TS di_2reg_reg(.RN (n_2), .CK (clk), .D (di_reg), .Q
      (di_2reg));
  DFFRHQX2TS di_3reg_reg(.RN (n_2), .CK (clk), .D (di_2reg), .Q (n_35));
  DFFRHQX2TS disp6a2_reg_reg(.RN (n_2), .CK (clk), .D (n_204), .Q
      (disp6a2_reg));
  DFFRHQX8TS ei_2reg_reg(.RN (n_2), .CK (clk), .D (ei_reg), .Q (n_80));
  DFFRX4TS ei_4reg_reg(.RN (n_2), .CK (clk), .D (n_87), .Q (n_104), .QN
      (n_103));
  DFFRHQX8TS fi_3reg_reg(.RN (n_2), .CK (clk), .D (n_57), .Q (n_84));
  DFFRHQX8TS fi_4reg_reg(.RN (n_2), .CK (clk), .D (n_84), .Q (n_98));
  DFFRHQX4TS gi_4reg_reg(.RN (n_2), .CK (clk), .D (n_29), .Q (n_100));
  DFFRHQX8TS hi_3reg_reg(.RN (n_2), .CK (clk), .D (n_58), .Q (hi_3reg));
  DFFRHQX4TS hi_4reg_reg(.RN (n_2), .CK (clk), .D (n_95), .Q (hi_4reg));
  DFFRHQX2TS ii_2reg_reg(.RN (n_2), .CK (clk), .D (ii_reg), .Q (n_54));
  DFFRHQX8TS ji_3reg_reg(.RN (n_2), .CK (clk), .D (n_56), .Q (ji_3reg));
  DFFRHQX8TS ji_4reg_reg(.RN (n_2), .CK (clk), .D (ji_3reg), .Q (n_82));
  DFFRHQX4TS k28p_reg_reg(.RN (n_2), .CK (clk), .D (n_171), .Q (n_81));
  DFFRHQX4TS p22_reg_reg(.RN (n_2), .CK (clk), .D (n_386), .Q
      (p22_reg));
  BUFX8TS g4747(.A (n_116), .Y (dispout));
  BUFX8TS g4749(.A (n_115), .Y (code_err));
```

```
  NAND4X2TS g4750(.A (n_293), .B (n_305), .C (n_254), .D (n_251), .Y
      (n_308));
  OR2X1TS g4751(.A (n_371), .B (n_388), .Y (n_307));
  BUFX8TS g4754(.A (n_114), .Y (disp_err));
  AND3X2TS g4755(.A (n_285), .B (n_296), .C (n_268), .Y (n_305));
  NAND3BX2TS g4756(.AN (dispaux_a_reg), .B (n_291), .C (n_290), .Y
      (n_304));
  BUFX8TS g4759(.A (n_113), .Y (dataout[0]));
  BUFX8TS g4761(.A (n_112), .Y (dataout[4]));
  BUFX8TS g4763(.A (n_111), .Y (dataout[3]));
  BUFX8TS g4765(.A (n_110), .Y (dataout[2]));
  BUFX8TS g4767(.A (n_109), .Y (dataout[1]));
  CLKXOR2X2TS g4768(.A (n_72), .B (compa_reg), .Y (n_302));
  CLKXOR2X2TS g4769(.A (n_69), .B (compb_reg), .Y (n_301));
  CLKXOR2X2TS g4770(.A (n_104), .B (compe_reg), .Y (n_300));
  CLKXOR2X2TS g4771(.A (ci_4reg), .B (compc_reg), .Y (n_299));
  CLKXOR2X2TS g4772(.A (compd_reg), .B (n_61), .Y (n_298));
  CLKAND2X2TS g4773(.A (disp6b_reg), .B (fghj22_reg), .Y (n_297));
  CLKAND2X2TS g4774(.A (n_216), .B (n_289), .Y (n_296));
  BUFX8TS g4776(.A (n_108), .Y (ko));
  MX2X1TS g4779(.S0 (n_65), .B (n_276), .A (n_373), .Y (n_295));
  MX2X1TS g4780(.S0 (dispin_3reg), .B (n_45), .A (n_274), .Y (n_294));
  CLKAND2X2TS g4781(.A (n_240), .B (n_283), .Y (n_293));
  OAI2BB1X2TS g4782(.A0N (n_105), .A1N (n_231), .B0 (n_40), .Y (n_292));
  NAND2X2TS g4783(.A (n_280), .B (fghjp31_reg), .Y (n_291));
  NAND2BX2TS g4784(.AN (n_279), .B (n_73), .Y (n_290));
  NOR2X2TS g4785(.A (code_aux_b_reg), .B (code_aux_a_reg), .Y (n_289));
  NAND2X1TS g4791(.A (n_272), .B (n_260), .Y (n_288));
  NAND2X1TS g4792(.A (n_272), .B (n_258), .Y (n_287));
  NAND2X1TS g4793(.A (n_272), .B (n_256), .Y (n_286));
  NAND3X2TS g4794(.A (n_27), .B (n_388), .C (n_76), .Y (n_285));
  AOI21X2TS g4795(.A0 (n_92), .A1 (disp6a2_reg), .B0 (n_271), .Y
      (n_284));
  NAND3BX2TS g4796(.AN (n_157), .B (n_156), .C (n_83), .Y (n_283));
  NAND2X1TS g4797(.A (n_270), .B (n_34), .Y (n_282));
  NAND2X1TS g4798(.A (n_269), .B (n_34), .Y (n_281));
  OR2X2TS g4799(.A (n_267), .B (n_83), .Y (n_280));
  AOI2BB1X2TS g4800(.A0N (n_83), .A1N (dispin_4reg), .B0 (n_76), .Y
      (n_279));
  OR2X1TS g4803(.A (n_195), .B (n_382), .Y (n_278));
  NAND2X2TS g4805(.A (n_47), .B (n_176), .Y (n_276));
  OR3X2TS g4807(.A (n_161), .B (n_261), .C (n_53), .Y (n_274));
  OAI21X2TS g4808(.A0 (n_145), .A1 (n_53), .B0 (n_48), .Y (n_273));
  CLKAND2X3TS g4809(.A (n_266), .B (n_230), .Y (n_272));
  OA21X2TS g4810(.A0 (disp6a2_reg), .A1 (disp6a_reg), .B0 (n_135), .Y
      (n_271));
  CLKAND2X2TS g4811(.A (n_257), .B (n_259), .Y (n_270));
  CLKAND2X2TS g4812(.A (n_259), .B (n_255), .Y (n_269));
  NAND2X2TS g4813(.A (n_83), .B (fghjp31_reg), .Y (n_268));
  NOR2BX2TS g4814(.AN (dispin_4reg), .B (n_76), .Y (n_267));
  NAND4X2TS g4818(.A (n_156), .B (n_101), .C (n_51), .D (p31_3reg), .Y
```

```
    (n_46));
  NOR2X2TS g4820(.A (n_134), .B (n_64), .Y (n_261));
  NOR2X4TS g4821(.A (n_249), .B (cndnenin_reg), .Y (n_266));
  NOR2X2TS g4822(.A (n_62), .B (n_75), .Y (n_260));
  NOR2X2TS g4823(.A (p22aceeqi_reg), .B (p22bceeqi_reg), .Y (n_258));
  NOR2X2TS g4824(.A (p31i_reg), .B (p22bceeqi_reg), .Y (n_257));
  NOR2X2TS g4825(.A (p22aceeqi_reg), .B (n_75), .Y (n_256));
  NOR2X4TS g4826(.A (n_62), .B (anbnenin_reg), .Y (n_259));
  NOR2X2TS g4827(.A (n_75), .B (p13in_reg), .Y (n_255));
  NAND2BX2TS g4832(.AN (n_238), .B (n_73), .Y (n_254));
  OR2X2TS g4835(.A (n_191), .B (n_234), .Y (n_251));
  OR2X2TS g4837(.A (p13en_reg), .B (p13dei_reg), .Y (n_249));
  BUFX8TS g4839(.A (n_107), .Y (dataout[6]));
  NAND2X1TS g4849(.A (n_229), .B (n_44), .Y (n_248));
  NAND2X1TS g4850(.A (n_229), .B (n_214), .Y (n_247));
  AND3X1TS g4851(.A (n_227), .B (n_32), .C (n_21), .Y (n_246));
  NOR2X2TS g4852(.A (n_380), .B (n_123), .Y (n_245));
  AND3X1TS g4853(.A (n_227), .B (n_30), .C (n_86), .Y (n_244));
  NOR2X2TS g4854(.A (n_380), .B (n_122), .Y (n_243));
  OR4X2TS g4857(.A (n_38), .B (n_173), .C (n_99), .D (n_65), .Y
    (n_240));
  INVX1TS g4858(.A (n_238), .Y (n_239));
  AND2X2TS g4859(.A (n_222), .B (n_207), .Y (n_238));
  OAI2BB1X2TS g4861(.A0N (n_120), .A1N (n_93), .B0 (n_219), .Y (n_236));
  NOR2BX2TS g4863(.AN (n_157), .B (fghjp31_reg), .Y (n_234));
  AO21X2TS g4864(.A0 (p22_reg), .A1 (n_68), .B0 (n_70), .Y (n_233));
  NAND2BX2TS g4866(.AN (n_92), .B (disp6a0_reg), .Y (n_231));
  NOR2X2TS g4867(.A (p31i_reg), .B (abei_reg), .Y (n_230));
  CLKINVX2TS g4869(.A (n_380), .Y (n_227));
  AND2X2TS g4870(.A (n_210), .B (n_208), .Y (n_229));
  NOR2X2TS g4874(.A (n_170), .B (n_63), .Y (n_224));
  NOR2X2TS g4875(.A (n_27), .B (n_73), .Y (n_223));
  NAND2BX2TS g4876(.AN (n_125), .B (p22_reg), .Y (n_222));
  AND3X1TS g4877(.A (n_127), .B (di_2reg), .C (p13_reg), .Y (n_221));
  NAND2BX2TS g4879(.AN (n_41), .B (n_144), .Y (n_219));
  NOR2X2TS g4881(.A (n_87), .B (n_63), .Y (n_217));
  NOR2X2TS g4882(.A (code_aux_c_reg), .B (code_aux_d_reg), .Y (n_216));
  OAI21X4TS g4893(.A0 (n_146), .A1 (n_66), .B0 (n_198), .Y (n_215));
  CLKMX2X2TS g4894(.S0 (n_81), .B (n_178), .A (n_188), .Y (n_214));
  MX2X1TS g4895(.S0 (n_81), .B (n_188), .A (n_178), .Y (n_213));
  NAND2X2TS g4898(.A (n_156), .B (n_59), .Y (n_210));
  CLKMX2X2TS g4899(.S0 (n_81), .B (n_82), .A (hi_4reg), .Y (n_209));
  NAND2X2TS g4900(.A (n_388), .B (n_133), .Y (n_208));
  NAND2BX2TS g4901(.AN (n_127), .B (p13_reg), .Y (n_207));
  MX2X1TS g4902(.S0 (n_81), .B (n_74), .A (n_132), .Y (n_206));
  AND2X1TS g4903(.A (n_89), .B (n_71), .Y (n_205));
  CLKAND2X2TS g4904(.A (n_71), .B (n_68), .Y (n_204));
  NOR2BX2TS g4905(.AN (n_193), .B (n_89), .Y (n_203));
  NOR2BX2TS g4906(.AN (n_193), .B (n_80), .Y (n_202));
  NOR2BX2TS g4907(.AN (n_193), .B (n_68), .Y (n_201));
  NAND3BX4TS g4917(.AN (n_67), .B (datain[6]), .C (n_29), .Y (n_198));
```

```
OR2X1TS g4918(.A (n_28), .B (n_182), .Y (n_197));
NAND2X1TS g4919(.A (n_184), .B (n_183), .Y (n_196));
NOR2X2TS g4920(.A (n_180), .B (n_145), .Y (n_195));
OAI21X2TS g4921(.A0 (n_134), .A1 (n_67), .B0 (n_175), .Y (n_194));
BUFX3TS g4922(.A (p13_reg), .Y (n_193));
NAND3X2TS g4925(.A (n_174), .B (n_99), .C (n_65), .Y (n_191));
OAI2BB1X2TS g4931(.A0N (n_130), .A1N (n_60), .B0 (n_166), .Y (n_187));
OAI2BB1X2TS g4932(.A0N (n_119), .A1N (n_60), .B0 (n_167), .Y (n_186));
NAND3BX2TS g4935(.AN (n_135), .B (n_168), .C (n_148), .Y (n_184));
NAND3X2TS g4936(.A (n_169), .B (n_105), .C (n_33), .Y (n_183));
AND3X2TS g4937(.A (n_170), .B (n_150), .C (n_105), .Y (n_182));
NAND2BX2TS g4938(.AN (n_84), .B (n_169), .Y (n_181));
CLKAND2X2TS g4939(.A (n_155), .B (n_146), .Y (n_180));
OA21X2TS g4941(.A0 (n_132), .A1 (n_98), .B0 (n_153), .Y (n_188));
OA21X2TS g4942(.A0 (n_352), .A1 (n_100), .B0 (n_152), .Y (n_178));
NAND3X2TS g4944(.A (n_61), .B (ci_4reg), .C (n_51), .Y (n_176));
NAND2BX2TS g4945(.AN (n_66), .B (n_143), .Y (n_175));
CLKAND2X2TS g4946(.A (n_151), .B (n_104), .Y (n_174));
NAND3X2TS g4947(.A (n_103), .B (n_69), .C (n_72), .Y (n_173));
NOR2X2TS g4948(.A (n_67), .B (n_66), .Y (n_172));
BUFX8TS g4951(.A (n_106), .Y (data_valid_pipe));
NAND2X2TS g4955(.A (n_79), .B (n_52), .Y (n_167));
NAND2X2TS g4956(.A (n_79), .B (n_128), .Y (n_166));
AND2X1TS g4957(.A (n_119), .B (n_52), .Y (n_165));
NAND2X2TS g4958(.A (n_118), .B (n_130), .Y (n_164));
AND2X1TS g4960(.A (n_130), .B (n_128), .Y (n_162));
NOR2X4TS g4961(.A (n_135), .B (n_148), .Y (n_171));
NOR2X2TS g4962(.A (n_136), .B (n_31), .Y (n_161));
NOR2BX2TS g4964(.AN (n_143), .B (n_145), .Y (n_159));
NOR2X4TS g4965(.A (n_134), .B (hi_3reg), .Y (n_170));
CLKAND2X4TS g4966(.A (n_143), .B (n_96), .Y (n_169));
NOR2X2TS g4967(.A (n_134), .B (n_146), .Y (n_158));
NOR2X4TS g4968(.A (n_146), .B (n_96), .Y (n_168));
NAND2X2TS g4969(.A (n_105), .B (hi_3reg), .Y (n_155));
NAND2BX2TS g4970(.AN (n_145), .B (n_95), .Y (n_154));
NAND2X2TS g4971(.A (n_352), .B (n_102), .Y (n_153));
NAND2X4TS g4972(.A (n_59), .B (n_102), .Y (n_157));
AND2X4TS g4973(.A (n_74), .B (n_132), .Y (n_156));
NAND2X2TS g4974(.A (n_132), .B (n_98), .Y (n_152));
NOR2X2TS g4976(.A (n_72), .B (n_69), .Y (n_151));
INVX1TS g4983(.A (n_149), .Y (n_150));
BUFX4TS g4986(.A (n_147), .Y (n_105));
NOR2X2TS g4988(.A (n_97), .B (n_124), .Y (n_141));
NOR2X2TS g4989(.A (n_97), .B (n_121), .Y (n_140));
AND3X1TS g4990(.A (n_127), .B (n_32), .C (n_30), .Y (n_139));
CLKXOR2X2TS g4991(.A (n_90), .B (n_55), .Y (n_138));
CLKXOR2X2TS g4992(.A (n_88), .B (n_56), .Y (n_137));
NAND2X2TS g4993(.A (n_35), .B (n_91), .Y (n_149));
OR2X4TS g4994(.A (n_91), .B (n_36), .Y (n_148));
OR2X2TS g4995(.A (n_42), .B (n_43), .Y (n_136));
AND2X2TS g4996(.A (n_87), .B (ii_3reg), .Y (n_147));
```

```
  NAND2X6TS g4997(.A (n_95), .B (ji_3reg), .Y (n_146));
  NAND2X4TS g4998(.A (n_84), .B (n_29), .Y (n_145));
  NOR2X4TS g4999(.A (n_93), .B (n_120), .Y (n_144));
  NOR2X6TS g5000(.A (hi_3reg), .B (ji_3reg), .Y (n_143));
  CLKINVX3TS g5008(.A (n_102), .Y (n_101));
  INVX2TS g5012(.A (n_59), .Y (n_133));
  INVX3TS g5015(.A (n_82), .Y (n_132));
  OR2X4TS g5016(.A (n_87), .B (ii_3reg), .Y (n_135));
  OR2X6TS g5017(.A (n_84), .B (n_94), .Y (n_134));
  ADDHX4TS g5031(.A (bi_reg), .B (ai_reg), .S (n_131), .CO (n_130));
  ADDHX4TS g5032(.A (di_reg), .B (ci_reg), .S (n_129), .CO (n_128));
  CLKINVX3TS g5033(.A (n_126), .Y (n_127));
  BUFX4TS g5038(.A (hi_3reg), .Y (n_95));
  NAND2X4TS g5039(.A (n_89), .B (n_80), .Y (n_126));
  OR2X2TS g5040(.A (n_32), .B (n_30), .Y (n_124));
  OR2X2TS g5041(.A (n_78), .B (n_86), .Y (n_123));
  OR2X2TS g5042(.A (n_77), .B (n_21), .Y (n_122));
  OR2X2TS g5043(.A (n_86), .B (di_2reg), .Y (n_121));
  OR2X4TS g5044(.A (n_80), .B (n_54), .Y (n_125));
  BUFX4TS g5051(.A (ii_3reg), .Y (n_120));
  NOR2X4TS g5066(.A (ai_reg), .B (bi_reg), .Y (n_119));
  NOR2X4TS g5067(.A (ci_reg), .B (di_reg), .Y (n_118));
  INVX2TS g5084(.A (rst), .Y (n_2));
  DFFRX2TS abei_reg_reg(.RN (n_2), .CK (clk), .D (n_139), .Q
      (abei_reg), .QN (UNCONNECTED));
  DFFRX2TS ai_2reg_reg(.RN (n_2), .CK (clk), .D (ai_reg), .Q (n_78),
      .QN (UNCONNECTED0));
  DFFRX2TS ai_4reg_reg(.RN (n_2), .CK (clk), .D (n_42), .Q (ai_4reg),
      .QN (UNCONNECTED1));
  DFFRX2TS ao_reg_reg(.RN (n_2), .CK (clk), .D (n_302), .Q (n_113), .QN
      (UNCONNECTED2));
  DFFRX2TS bi_2reg_reg(.RN (n_2), .CK (clk), .D (bi_reg), .Q (n_77),
      .QN (UNCONNECTED3));
  DFFRX2TS bi_4reg_reg(.RN (n_2), .CK (clk), .D (n_43), .Q (bi_4reg),
      .QN (UNCONNECTED4));
  DFFRX2TS bo_reg_reg(.RN (n_2), .CK (clk), .D (n_301), .Q (n_109), .QN
      (UNCONNECTED5));
  DFFRX2TS ci_2reg_reg(.RN (n_2), .CK (clk), .D (ci_reg), .Q (ci_2reg),
      .QN (n_85));
  DFFRX4TS ci_3reg_reg(.RN (n_2), .CK (clk), .D (n_21), .Q (n_91), .QN
      (UNCONNECTED6));
  DFFRX4TS ci_4reg_reg(.RN (n_2), .CK (clk), .D (n_31), .Q (ci_4reg),
      .QN (n_99));
  DFFRX2TS co_reg_reg(.RN (n_2), .CK (clk), .D (n_299), .Q (n_110), .QN
      (UNCONNECTED7));
  DFFRX2TS code_aux_a_reg_reg(.RN (n_2), .CK (clk), .D (n_278), .Q
      (code_aux_a_reg), .QN (UNCONNECTED8));
  DFFRX2TS code_aux_b_reg_reg(.RN (n_2), .CK (clk), .D (n_372), .Q
      (code_aux_b_reg), .QN (UNCONNECTED9));
  DFFRX2TS code_aux_c_reg_reg(.RN (n_2), .CK (clk), .D (n_196), .Q
      (code_aux_c_reg), .QN (UNCONNECTED10));
```

```
DFFRX2TS code_aux_d_reg_reg(.RN (n_2), .CK (clk), .D (n_197), .Q
    (code_aux_d_reg), .QN (UNCONNECTED11));
DFFRX2TS code_err_reg_reg(.RN (n_2), .CK (clk), .D (n_308), .Q
    (n_115), .QN (UNCONNECTED12));
DFFRX2TS compa_reg_reg(.RN (n_2), .CK (clk), .D (n_288), .Q
    (compa_reg), .QN (UNCONNECTED13));
DFFRX2TS compb_reg_reg(.RN (n_2), .CK (clk), .D (n_287), .Q
    (compb_reg), .QN (UNCONNECTED14));
DFFRX2TS compc_reg_reg(.RN (n_2), .CK (clk), .D (n_282), .Q
    (compc_reg), .QN (UNCONNECTED15));
DFFRX2TS compd_reg_reg(.RN (n_2), .CK (clk), .D (n_286), .Q
    (compd_reg), .QN (UNCONNECTED16));
DFFRX2TS compe_reg_reg(.RN (n_2), .CK (clk), .D (n_281), .Q
    (compe_reg), .QN (UNCONNECTED17));
DFFRX1TS data_valid_1pipe_reg(.RN (n_2), .CK (clk), .D (n_363), .Q
    (data_valid_1pipe), .QN (UNCONNECTED18));
DFFRX1TS data_valid_2pipe_reg(.RN (n_2), .CK (clk), .D
    (data_valid_1pipe), .Q (data_valid_2pipe), .QN (UNCONNECTED19));
DFFRX1TS data_valid_3pipe_reg(.RN (n_2), .CK (clk), .D
    (data_valid_2pipe), .Q (data_valid_3pipe), .QN (UNCONNECTED20));
DFFRX1TS data_valid_4pipe_reg(.RN (n_2), .CK (clk), .D
    (data_valid_3pipe), .Q (data_valid_4pipe), .QN (UNCONNECTED21));
DFFRX2TS data_valid_5pipe_reg(.RN (n_2), .CK (clk), .D
    (data_valid_4pipe), .Q (n_106), .QN (UNCONNECTED22));
DFFRX4TS di_4reg_reg(.RN (n_2), .CK (clk), .D (n_92), .Q (n_61), .QN
    (UNCONNECTED23));
DFFRX2TS disp6a0_reg_reg(.RN (n_2), .CK (clk), .D (n_201), .Q
    (disp6a0_reg), .QN (UNCONNECTED24));
DFFRX2TS disp6a_reg_reg(.RN (n_2), .CK (clk), .D (n_233), .Q
    (disp6a_reg), .QN (UNCONNECTED25));
DFFRX2TS disp6b_reg_reg(.RN (n_2), .CK (clk), .D (n_292), .Q
    (disp6b_reg), .QN (UNCONNECTED26));
DFFRX4TS disp6n_2reg_reg(.RN (n_2), .CK (clk), .D (n_53), .Q (n_76),
    .QN (UNCONNECTED27));
DFFRX4TS disp6n_reg_reg(.RN (n_2), .CK (clk), .D (n_239), .Q (n_53),
    .QN (UNCONNECTED28));
DFFRX4TS disp6p_2reg_reg(.RN (n_2), .CK (clk), .D (n_64), .Q (n_83),
    .QN (UNCONNECTED29));
DFFRX4TS disp6p_reg_reg(.RN (n_2), .CK (clk), .D (n_378), .Q (n_64),
    .QN (UNCONNECTED30));
DFFRX2TS disp_err_reg_reg(.RN (n_2), .CK (clk), .D (n_304), .Q
    (n_114), .QN (UNCONNECTED31));
DFFRX2TS dispaux_a_reg_reg(.RN (n_2), .CK (clk), .D (n_294), .Q
    (dispaux_a_reg), .QN (UNCONNECTED32));
DFFRX4TS dispin_2reg_reg(.RN (n_2), .CK (clk), .D (dispin_reg), .Q
    (n_68), .QN (UNCONNECTED33));
DFFRX2TS dispin_3reg_reg(.RN (n_2), .CK (clk), .D (n_68), .Q
    (dispin_3reg), .QN (UNCONNECTED34));
DFFRX2TS dispin_4reg_reg(.RN (n_2), .CK (clk), .D (dispin_3reg), .Q
    (dispin_4reg), .QN (UNCONNECTED35));
DFFRX2TS dispout_reg_reg(.RN (n_2), .CK (clk), .D (n_307), .Q
```

```
    (n_116), .QN (UNCONNECTED36));
  DFFRX2TS do__reg_reg(.RN (n_2), .CK (clk), .D (n_298), .Q (n_111),
    .QN (UNCONNECTED37));
  DFFRX4TS ei_3reg_reg(.RN (n_2), .CK (clk), .D (n_80), .Q (ei_3reg),
    .QN (n_93));
  DFFRX2TS eo_reg_reg(.RN (n_2), .CK (clk), .D (n_300), .Q (n_112), .QN
    (UNCONNECTED38));
  DFFRX4TS feqg_reg_reg(.RN (n_2), .CK (clk), .D (n_138), .Q (n_66),
    .QN (UNCONNECTED39));
  DFFRX2TS fghj22_reg_reg(.RN (n_2), .CK (clk), .D (n_381), .Q
    (fghj22_reg), .QN (UNCONNECTED40));
  DFFRX2TS fghjp13_reg_reg(.RN (n_2), .CK (clk), .D (n_194), .Q (n_26),
    .QN (UNCONNECTED41));
  DFFRX2TS fghjp31_reg_reg(.RN (n_2), .CK (clk), .D (n_215), .Q (n_25),
    .QN (UNCONNECTED42));
  DFFRX2TS fi_2reg_reg(.RN (n_2), .CK (clk), .D (fi_reg), .Q (n_57),
    .QN (n_90));
  DFFRX2TS fo_reg_reg(.RN (n_2), .CK (clk), .D (n_248), .Q (n_24), .QN
    (UNCONNECTED43));
  DFFRX2TS gi_2reg_reg(.RN (n_2), .CK (clk), .D (gi_reg), .Q (n_55),
    .QN (UNCONNECTED44));
  DFFRX4TS gi_3reg_reg(.RN (n_2), .CK (clk), .D (n_55), .Q (n_94), .QN
    (n_96));
  DFFRX2TS go_reg_reg(.RN (n_2), .CK (clk), .D (n_247), .Q (n_107), .QN
    (UNCONNECTED45));
  DFFRX4TS heqj_reg_reg(.RN (n_2), .CK (clk), .D (n_137), .Q (n_67),
    .QN (UNCONNECTED46));
  DFFRX2TS hi_2reg_reg(.RN (n_2), .CK (clk), .D (hi_reg), .Q (n_58),
    .QN (n_88));
  DFFRX2TS ho_reg_reg(.RN (n_2), .CK (clk), .D (n_384), .Q (n_23), .QN
    (UNCONNECTED47));
  DFFRX4TS ii_3reg_reg(.RN (n_2), .CK (clk), .D (n_89), .Q (ii_3reg),
    .QN (UNCONNECTED48));
  DFFRX4TS ii_4reg_reg(.RN (n_2), .CK (clk), .D (n_120), .Q (n_65), .QN
    (UNCONNECTED49));
  DFFRX2TS ji_2reg_reg(.RN (n_2), .CK (clk), .D (ji_reg), .Q (n_56),
    .QN (UNCONNECTED50));
  DFFRX2TS ko_reg_reg(.RN (n_2), .CK (clk), .D (n_295), .Q (n_108), .QN
    (UNCONNECTED51));
  DFFRX2TS p04_2reg_reg(.RN (n_2), .CK (clk), .D (p04_reg), .Q
    (p04_2reg), .QN (UNCONNECTED52));
  DFFRX1TS p04_reg_reg(.RN (n_2), .CK (clk), .D (n_165), .Q (p04_reg),
    .QN (UNCONNECTED53));
  DFFRX2TS p13_2reg_reg(.RN (n_2), .CK (clk), .D (n_193), .Q
    (p13_2reg), .QN (UNCONNECTED54));
  DFFRX2TS p13_3reg_reg(.RN (n_2), .CK (clk), .D (n_63), .Q (p13_3reg),
    .QN (UNCONNECTED55));
  DFFRX4TS p13_reg_reg(.RN (n_2), .CK (clk), .D (n_186), .Q (p13_reg),
    .QN (UNCONNECTED56));
  DFFRX2TS p13dei_reg_reg(.RN (n_2), .CK (clk), .D (n_221), .Q
    (p13dei_reg), .QN (UNCONNECTED57));
```

```
DFFRX2TS p13en_reg_reg(.RN (n_2), .CK (clk), .D (n_202), .Q
    (p13en_reg), .QN (UNCONNECTED58));
DFFRX2TS p13in_reg_reg(.RN (n_2), .CK (clk), .D (n_203), .Q
    (p13in_reg), .QN (UNCONNECTED59));
DFFRX4TS p22aceeqi_reg_reg(.RN (n_2), .CK (clk), .D (n_246), .Q
    (p22aceeqi_reg), .QN (UNCONNECTED60));
DFFRX2TS p22ancneeqi_reg_reg(.RN (n_2), .CK (clk), .D (n_245), .Q
    (p22ancneeqi_reg), .QN (UNCONNECTED61));
DFFRX4TS p22bceeqi_reg_reg(.RN (n_2), .CK (clk), .D (n_244), .Q
    (p22bceeqi_reg), .QN (UNCONNECTED62));
DFFRX2TS p22bncneeqi_reg_reg(.RN (n_2), .CK (clk), .D (n_243), .Q
    (n_37), .QN (UNCONNECTED63));
DFFRX4TS p31_2reg_reg(.RN (n_2), .CK (clk), .D (n_71), .Q (n_41), .QN
    (UNCONNECTED64));
DFFRX2TS p31_3reg_reg(.RN (n_2), .CK (clk), .D (n_41), .Q (p31_3reg),
    .QN (UNCONNECTED65));
DFFRX2TS p31_reg_reg(.RN (n_2), .CK (clk), .D (n_187), .Q (n_70), .QN
    (UNCONNECTED66));
DFFRX4TS p31i_reg_reg(.RN (n_2), .CK (clk), .D (n_205), .Q
    (p31i_reg), .QN (UNCONNECTED67));
DFFRX2TS p40_2reg_reg(.RN (n_2), .CK (clk), .D (p40_reg), .Q
    (p40_2reg), .QN (UNCONNECTED68));
DFFRX1TS p40_reg_reg(.RN (n_2), .CK (clk), .D (n_162), .Q (p40_reg),
    .QN (UNCONNECTED69));
INVX3TS drc_bufs(.A (n_93), .Y (n_87));
INVX2TS drc_bufs5088(.A (n_85), .Y (n_86));
INVX2TS drc_bufs5089(.A (n_85), .Y (n_21));
CLKBUFX4TS drc5129(.A (n_131), .Y (n_79));
INVX2TS drc_bufs5152(.A (n_352), .Y (n_74));
BUFX3TS fopt5168(.A (n_70), .Y (n_71));
BUFX3TS drc5204(.A (p13_2reg), .Y (n_63));
BUFX3TS drc_bufs5214(.A (n_36), .Y (n_92));
BUFX2TS drc5223(.A (n_129), .Y (n_60));
BUFX2TS fopt5310(.A (n_118), .Y (n_52));
BUFX3TS fopt5323(.A (n_104), .Y (n_51));
CLKBUFX2TS drc_bufs5343(.A (n_22), .Y (n_47));
CLKBUFX2TS drc_bufs5345(.A (n_273), .Y (n_45));
CLKBUFX2TS drc_bufs5389(.A (n_213), .Y (n_44));
CLKBUFX2TS drc_bufs5420(.A (n_284), .Y (n_40));
CLKBUFX2TS drc_bufs5430(.A (n_206), .Y (n_39));
CLKBUFX2TS drc_bufs5455(.A (n_223), .Y (n_38));
BUFX4TS fopt5483(.A (n_100), .Y (n_102));
BUFX4TS drc5510(.A (ai_4reg), .Y (n_72));
BUFX4TS drc5527(.A (bi_4reg), .Y (n_69));
BUFX4TS fopt5560(.A (n_98), .Y (n_59));
BUFX2TS fopt5565(.A (n_35), .Y (n_36));
BUFX4TS fopt5571(.A (n_54), .Y (n_89));
BUFX3TS drc5576(.A (p22ancneeqi_reg), .Y (n_62));
BUFX3TS fopt5588(.A (n_125), .Y (n_97));
BUFX2TS fopt5627(.A (n_266), .Y (n_34));
CLKBUFX2TS fopt5647(.A (n_149), .Y (n_33));
```

```
  BUFX3TS fopt5652(.A (n_78), .Y (n_32));
  BUFX4TS drc5685(.A (n_37), .Y (n_75));
  BUFX2TS fopt5734(.A (n_91), .Y (n_31));
  BUFX3TS fopt5739(.A (n_77), .Y (n_30));
  BUFX4TS fopt5749(.A (n_94), .Y (n_29));
  NOR2BX2TS g2(.AN (n_171), .B (n_154), .Y (n_28));
  NOR2X6TS g5751(.A (n_98), .B (n_100), .Y (n_27));
  BUFX4TS g5752(.A (n_26), .Y (n_73));
  BUFX4TS g5753(.A (n_25), .Y (fghjp31_reg));
  BUFX8TS g5754(.A (n_24), .Y (dataout[5]));
  BUFX8TS g5755(.A (n_23), .Y (dataout[7]));
  NAND4BX2TS g5764(.AN (n_101), .B (n_388), .C (n_103), .D (p13_3reg),
      .Y (n_22));
  DFFRHQX8TS ai_reg_reg(.RN (n_2), .CK (clk), .D (n_4), .Q (ai_reg));
  DFFRHQX8TS bi_reg_reg(.RN (n_2), .CK (clk), .D (n_3), .Q (bi_reg));
  DFFRHQX8TS ci_reg_reg(.RN (n_2), .CK (clk), .D (n_6), .Q (ci_reg));
  DFFRHQX2TS ii_reg_reg(.RN (n_2), .CK (clk), .D (n_7), .Q (ii_reg));
  DFFRHQX2TS ji_reg_reg(.RN (n_2), .CK (clk), .D (n_13), .Q (ji_reg));
  DFFRHQX8TS di_reg_reg(.RN (n_2), .CK (clk), .D (n_5), .Q (di_reg));
  DFFRHQX2TS dispin_reg_reg(.RN (n_2), .CK (clk), .D (n_12), .Q
      (dispin_reg));
  DFFRHQX2TS ei_reg_reg(.RN (n_2), .CK (clk), .D (n_11), .Q (ei_reg));
  DFFRHQX2TS fi_reg_reg(.RN (n_2), .CK (clk), .D (n_10), .Q (fi_reg));
  DFFRHQX2TS gi_reg_reg(.RN (n_2), .CK (clk), .D (n_9), .Q (gi_reg));
  DFFRHQX2TS hi_reg_reg(.RN (n_2), .CK (clk), .D (n_8), .Q (hi_reg));
  MX2X1TS g3314(.S0 (n_363), .B (datain[9]), .A (ji_reg), .Y (n_13));
  MX2X1TS g3315(.S0 (n_0), .B (dispin), .A (dispin_reg), .Y (n_12));
  MX2X1TS g3316(.S0 (n_1), .B (datain[4]), .A (ei_reg), .Y (n_11));
  MX2X1TS g3317(.S0 (n_364), .B (datain[6]), .A (fi_reg), .Y (n_10));
  MX2X1TS g3318(.S0 (n_364), .B (datain[7]), .A (gi_reg), .Y (n_9));
  MX2X1TS g3319(.S0 (n_1), .B (datain[8]), .A (hi_reg), .Y (n_8));
  MX2X1TS g3320(.S0 (n_1), .B (datain[5]), .A (ii_reg), .Y (n_7));
  CLKMX2X2TS g3321(.S0 (n_1), .B (datain[2]), .A (ci_reg), .Y (n_6));
  CLKMX2X2TS g3322(.S0 (n_364), .B (datain[3]), .A (di_reg), .Y (n_5));
  CLKMX2X2TS g3323(.S0 (n_363), .B (datain[0]), .A (ai_reg), .Y (n_4));
  CLKMX2X2TS g3324(.S0 (n_363), .B (datain[1]), .A (bi_reg), .Y (n_3));
  BUFX4TS drc_bufs3364(.A (n_0), .Y (n_1));
  BUFX2TS drc_bufs3369(.A (n_364), .Y (n_0));
  BUFX3TS drc_bufs5921(.A (hi_4reg), .Y (n_352));
  BUFX4TS drc_bufs5931(.A (data_valid), .Y (n_363));
  BUFX4TS drc_bufs5932(.A (data_valid), .Y (n_364));
  AOI2BB1X2TS g5939(.A0N (n_297), .A1N (fghjp31_reg), .B0 (n_156), .Y
      (n_371));
  AO22X2TS g5940(.A0 (n_169), .A1 (n_236), .B0 (n_168), .B1 (n_376), .Y
      (n_372));
  OAI31X4TS g5941(.A0 (n_104), .A1 (ci_4reg), .A2 (n_61), .B0 (n_46),
      .Y (n_373));
  AOI31X4TS g5943(.A0 (n_42), .A1 (n_91), .A2 (n_43), .B0 (n_64), .Y
      (n_48));
  AO21X2TS g5944(.A0 (n_217), .A1 (n_120), .B0 (n_144), .Y (n_376));
  AOI33X4TS g5945(.A0 (n_39), .A1 (n_101), .A2 (n_59), .B0 (n_209), .B1
```

```
      (n_133), .B2 (n_102), .Y (n_377));
  AO22X2TS g5946(.A0 (n_71), .A1 (n_97), .B0 (n_127), .B1 (p22_reg), .Y
      (n_378));
  AOI211X4TS g5947(.A0 (n_147), .A1 (n_41), .B0 (p40_2reg), .C0
      (p04_2reg), .Y (n_379));
  OAI2BB1X4TS g5948(.A0N (n_125), .A1N (n_126), .B0 (p22_reg), .Y
      (n_380));
  OR3X2TS g5949(.A (n_159), .B (n_172), .C (n_158), .Y (n_381));
  OAI211X4TS g5950(.A0 (n_224), .A1 (n_135), .B0 (n_181), .C0 (n_379),
      .Y (n_382));
  OAI2BB1X2TS g5951(.A0N (n_383), .A1N (n_387), .B0 (n_377), .Y
      (n_384));
  NAND2BX2TS g3(.AN (n_27), .B (n_157), .Y (n_383));
  AO21X2TS g5952(.A0 (n_119), .A1 (n_128), .B0 (n_385), .Y (n_386));
  OAI2BB1X2TS g5953(.A0N (n_60), .A1N (n_79), .B0 (n_164), .Y (n_385));
  ADDHX4TS g5954(.A (n_82), .B (hi_4reg), .S (n_387), .CO (n_388));
endmodule

module CDR(rst, clks_in, a_rx, samp_test);
  input rst, a_rx;
  input [3:0] clks_in;
  output samp_test;
  wire rst, a_rx;
  wire [3:0] clks_in;
  wire samp_test;
  wire [3:0] c_rx_upsampled_4reg;
  wire [3:0] c_rx_upsampled_3reg;
  wire [3:0] c_rx_upsampled_2reg;
  wire [3:0] c_rx_upsampled;
  wire [1:0] best_samp;
  wire [3:0] a_xor_reg;
  wire \best_samp_reg[0]_296 , \best_samp_reg[1]_297 ,
      \c_rx_upsampled_reg[0]_244 , \c_rx_upsampled_reg[1]_247 ,
      \c_rx_upsampled_reg[2]_250 , \c_rx_upsampled_reg[3]_253 , n_4,
      n_5;
  wire n_6, n_7, n_8, n_9, n_10, n_11, n_12, n_15;
  wire n_16, n_18, n_19, n_20;
  DFFRHQX1TS samp_test_reg(.RN (n_4), .CK (clks_in[0]), .D (n_20), .Q
      (samp_test));
  MX2X1TS g137(.S0 (\best_samp_reg[1]_297 ), .B (n_15), .A (n_16), .Y
      (n_20));
  MX2X1TS g138(.S0 (\best_samp_reg[0]_296 ), .B
      (c_rx_upsampled_4reg[3]), .A (c_rx_upsampled_4reg[2]), .Y
      (n_19));
  MX2X1TS g139(.S0 (\best_samp_reg[0]_296 ), .B
      (c_rx_upsampled_4reg[1]), .A (c_rx_upsampled_4reg[0]), .Y
      (n_18));
  DFFRHQX1TS \c_rx_upsampled_4reg_reg[3] (.RN (n_4), .CK (clks_in[0]),
      .D (c_rx_upsampled_3reg[3]), .Q (c_rx_upsampled_4reg[3]));
  DFFRHQX1TS \c_rx_upsampled_4reg_reg[0] (.RN (n_4), .CK (clks_in[0]),
      .D (c_rx_upsampled_3reg[0]), .Q (c_rx_upsampled_4reg[0]));
```

```
DFFRHQX1TS \c_rx_upsampled_4reg_reg[1] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled_3reg[1]), .Q (c_rx_upsampled_4reg[1]));
DFFRHQX1TS \c_rx_upsampled_4reg_reg[2] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled_3reg[2]), .Q (c_rx_upsampled_4reg[2]));
DFFRHQX1TS \c_rx_upsampled_3reg_reg[1] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled_2reg[1]), .Q (c_rx_upsampled_3reg[1]));
DFFRHQX1TS \c_rx_upsampled_3reg_reg[2] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled_2reg[2]), .Q (c_rx_upsampled_3reg[2]));
DFFRHQX1TS \c_rx_upsampled_3reg_reg[3] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled_2reg[3]), .Q (c_rx_upsampled_3reg[3]));
DFFRHQX1TS \c_rx_upsampled_3reg_reg[0] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled_2reg[0]), .Q (c_rx_upsampled_3reg[0]));
DFFRHQX1TS \c_rx_upsampled_2reg_reg[0] (.RN (n_4), .CK (clks_in[0]),
    .D (\c_rx_upsampled_reg[0]_244 ), .Q (c_rx_upsampled_2reg[0]));
DFFRHQX1TS \c_rx_upsampled_2reg_reg[1] (.RN (n_4), .CK (clks_in[0]),
    .D (\c_rx_upsampled_reg[1]_247 ), .Q (c_rx_upsampled_2reg[1]));
DFFRHQX1TS \c_rx_upsampled_2reg_reg[2] (.RN (n_4), .CK (clks_in[0]),
    .D (\c_rx_upsampled_reg[2]_250 ), .Q (c_rx_upsampled_2reg[2]));
DFFRHQX1TS \c_rx_upsampled_2reg_reg[3] (.RN (n_4), .CK (clks_in[0]),
    .D (\c_rx_upsampled_reg[3]_253 ), .Q (c_rx_upsampled_2reg[3]));
DFFRHQX1TS \c_rx_upsampled_reg_reg[0] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled[0]), .Q (\c_rx_upsampled_reg[0]_244 ));
DFFRHQX1TS \c_rx_upsampled_reg_reg[1] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled[1]), .Q (\c_rx_upsampled_reg[1]_247 ));
DFFRHQX1TS \c_rx_upsampled_reg_reg[2] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled[2]), .Q (\c_rx_upsampled_reg[2]_250 ));
DFFRHQX1TS \c_rx_upsampled_reg_reg[3] (.RN (n_4), .CK (clks_in[0]),
    .D (c_rx_upsampled[3]), .Q (\c_rx_upsampled_reg[3]_253 ));
DFFRHQX4TS \c_rx_upsampled_reg[2] (.RN (n_4), .CK (clks_in[2]), .D
    (a_rx), .Q (c_rx_upsampled[2]));
DFFRHQX2TS \best_samp_reg_reg[1] (.RN (n_4), .CK (clks_in[0]), .D
    (best_samp[1]), .Q (\best_samp_reg[1]_297 ));
DFFRHQX2TS \c_rx_upsampled_reg[0] (.RN (n_4), .CK (clks_in[0]), .D
    (a_rx), .Q (c_rx_upsampled[0]));
DFFRHQX2TS \c_rx_upsampled_reg[3] (.RN (n_4), .CK (clks_in[3]), .D
    (a_rx), .Q (c_rx_upsampled[3]));
DFFRHQX4TS \c_rx_upsampled_reg[1] (.RN (n_4), .CK (clks_in[1]), .D
    (a_rx), .Q (c_rx_upsampled[1]));
DFFRHQX2TS \best_samp_reg_reg[0] (.RN (n_4), .CK (clks_in[0]), .D
    (best_samp[0]), .Q (\best_samp_reg[0]_296 ));
INVX2TS g162(.A (rst), .Y (n_4));
CLKBUFX2TS drc_bufs176(.A (n_18), .Y (n_16));
CLKBUFX2TS drc_bufs177(.A (n_19), .Y (n_15));
DFFRHQX2TS \best_samp_reg[0] (.RN (n_4), .CK (clks_in[0]), .D (n_12),
    .Q (best_samp[0]));
OR2X1TS g146(.A (n_10), .B (a_xor_reg[3]), .Y (n_12));
DFFRHQX2TS \best_samp_reg[1] (.RN (n_4), .CK (clks_in[0]), .D (n_11),
    .Q (best_samp[1]));
OR3X2TS g148(.A (n_9), .B (a_xor_reg[3]), .C (a_xor_reg[1]), .Y
    (n_11));
NOR2X2TS g149(.A (n_8), .B (a_xor_reg[1]), .Y (n_10));
```

```
  NOR2BX2TS g150(.AN (best_samp[1]), .B (a_xor_reg[0]), .Y (n_9));
  NAND2BX2TS g151(.AN (a_xor_reg[0]), .B (best_samp[0]), .Y (n_8));
  DFFRHQX2TS \a_xor_reg_reg[0] (.RN (n_4), .CK (clks_in[0]), .D (n_6),
       .Q (a_xor_reg[0]));
  DFFRHQX2TS \a_xor_reg_reg[1] (.RN (n_4), .CK (clks_in[0]), .D (n_5),
       .Q (a_xor_reg[1]));
  DFFRHQX2TS \a_xor_reg_reg[3] (.RN (n_4), .CK (clks_in[0]), .D (n_7),
       .Q (a_xor_reg[3]));
  CLKXOR2X2TS g155(.A (c_rx_upsampled[1]), .B (c_rx_upsampled[2]), .Y
       (n_7));
  CLKXOR2X2TS g156(.A (c_rx_upsampled[2]), .B (c_rx_upsampled[3]), .Y
       (n_6));
  CLKXOR2X2TS g157(.A (c_rx_upsampled[1]), .B (c_rx_upsampled[0]), .Y
       (n_5));
endmodule

module deserializer(rst, clks_in, a_rx, c_parallel_out, clk_out,
    disparity_d, disparity_q, c_data_valid);
  input rst, a_rx, disparity_d;
  input [3:0] clks_in;
  output [9:0] c_parallel_out;
  output clk_out, disparity_q, c_data_valid;
  wire rst, a_rx, disparity_d;
  wire [3:0] clks_in;
  wire [9:0] c_parallel_out;
  wire clk_out, disparity_q, c_data_valid;
  wire [9:0] shift_2reg;
  wire [3:0] cycle_count;
  wire [9:0] shift_reg;
  wire UNCONNECTED70, UNCONNECTED71, comma_detected,
       comma_detected_reg, n_0, n_1, n_2, n_3;
  wire n_4, n_5, n_6, n_7, n_8, n_9, n_10, n_11;
  wire n_12, n_13, n_14, n_15, n_16, n_17, n_18, n_19;
  wire n_20, n_21, n_22, n_23, n_24, n_26, n_29, n_36;
  wire n_42, n_43, n_48, n_49, n_51, n_52, n_53, n_55;
  wire n_56, n_57, n_58, n_59, n_60, n_61, n_62, n_63;
  wire n_74, n_125, n_149, n_150, n_154, samp_test;
  CDR CDR1(.rst (rst), .clks_in (clks_in), .a_rx (a_rx), .samp_test
       (samp_test));
  DFFRHQX4TS c_data_valid_reg(.RN (n_2), .CK (clks_in[0]), .D (n_154),
       .Q (c_data_valid));
  NAND2X4TS g298(.A (n_63), .B (n_62), .Y (comma_detected));
  OR2X2TS g299(.A (n_61), .B (n_42), .Y (n_63));
  NAND2X2TS g300(.A (n_60), .B (n_55), .Y (n_62));
  NAND3BX2TS g301(.AN (shift_2reg[8]), .B (n_59), .C (shift_2reg[7]),
       .Y (n_61));
  CLKAND2X2TS g302(.A (n_58), .B (n_56), .Y (n_60));
  CLKAND2X2TS g303(.A (n_51), .B (n_57), .Y (n_59));
  AND4X2TS g304(.A (shift_2reg[1]), .B (shift_2reg[2]), .C
       (shift_2reg[0]), .D (shift_2reg[9]), .Y (n_58));
  NOR2X2TS g305(.A (shift_2reg[0]), .B (shift_2reg[9]), .Y (n_57));
```

```
NOR2BX2TS g306(.AN (shift_2reg[8]), .B (shift_2reg[7]), .Y (n_56));
DFFRHQX4TS \shift_2reg_reg[9] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[8]), .Q (shift_2reg[9]));
CLKAND2X2TS g308(.A (n_52), .B (n_53), .Y (n_55));
NAND4X2TS g309(.A (shift_2reg[6]), .B (shift_2reg[4]), .C
    (shift_2reg[5]), .D (shift_2reg[3]), .Y (n_42));
DFFRHQX4TS \shift_reg_reg[9] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[7]), .Q (shift_2reg[8]));
NOR2X2TS g311(.A (shift_2reg[5]), .B (shift_2reg[6]), .Y (n_53));
DFFRHQX4TS \shift_reg_reg[8] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[6]), .Q (shift_2reg[7]));
DFFRHQX4TS \shift_reg_reg[7] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[5]), .Q (shift_2reg[6]));
NOR2X2TS g314(.A (shift_2reg[3]), .B (shift_2reg[4]), .Y (n_52));
DFFRHQX4TS \shift_reg_reg[6] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[4]), .Q (shift_2reg[5]));
DFFRHQX4TS \shift_reg_reg[5] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[3]), .Q (shift_2reg[4]));
NOR2X2TS g317(.A (shift_2reg[1]), .B (shift_2reg[2]), .Y (n_51));
DFFRHQX4TS \shift_reg_reg[4] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[2]), .Q (shift_2reg[3]));
DFFRHQX4TS \shift_reg_reg[3] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[1]), .Q (shift_2reg[2]));
AND3X2TS g320(.A (n_49), .B (cycle_count[0]), .C
    (comma_detected_reg), .Y (n_43));
DFFRHQX4TS \shift_reg_reg[2] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_2reg[0]), .Q (shift_2reg[1]));
CLKINVX3TS g322(.A (n_49), .Y (n_1));
NOR2X4TS g323(.A (n_48), .B (cycle_count[3]), .Y (n_49));
DFFRHQX4TS \shift_reg_reg[1] (.RN (n_2), .CK (clks_in[0]), .D
    (shift_reg[0]), .Q (shift_2reg[0]));
OR2X2TS g325(.A (cycle_count[2]), .B (cycle_count[1]), .Y (n_48));
DFFRHQX1TS \shift_reg_reg[0] (.RN (n_2), .CK (clks_in[0]), .D
    (samp_test), .Q (shift_reg[0]));
DFFRHQX1TS disparity_q_reg(.RN (n_2), .CK (clks_in[0]), .D
    (disparity_d), .Q (disparity_q));
CLKINVX2TS g328(.A (cycle_count[0]), .Y (n_74));
INVX2TS g329(.A (rst), .Y (n_2));
DFFSX2TS \cycle_count_reg[3] (.SN (n_2), .CK (clks_in[0]), .D (n_24),
    .Q (n_29), .QN (UNCONNECTED70));
NAND2X1TS g224(.A (n_23), .B (n_18), .Y (n_24));
CLKXOR2X2TS g225(.A (n_17), .B (cycle_count[3]), .Y (n_23));
DFFRHQX4TS \cycle_count_reg[2] (.RN (n_2), .CK (clks_in[0]), .D
    (n_22), .Q (cycle_count[2]));
CLKAND2X2TS g227(.A (n_19), .B (n_18), .Y (n_22));
DFFSX2TS \cycle_count_reg[0] (.SN (n_2), .CK (clks_in[0]), .D (n_20),
    .Q (n_26), .QN (UNCONNECTED71));
DFFRHQX4TS \cycle_count_reg[1] (.RN (n_2), .CK (clks_in[0]), .D
    (n_21), .Q (cycle_count[1]));
CLKAND2X2TS g230(.A (n_18), .B (n_16), .Y (n_21));
NAND2BX1TS g231(.AN (n_74), .B (n_18), .Y (n_20));
```

```
OAI2BB1X2TS g232(.A0N (cycle_count[2]), .A1N (n_4), .B0 (n_17), .Y
    (n_19));
DFFRHQX2TS \c_parallel_out_reg[8] (.RN (n_2), .CK (clks_in[0]), .D
    (n_15), .Q (c_parallel_out[8]));
DFFRHQX2TS \c_parallel_out_reg[0] (.RN (n_2), .CK (clks_in[0]), .D
    (n_10), .Q (c_parallel_out[0]));
DFFRHQX2TS \c_parallel_out_reg[9] (.RN (n_2), .CK (clks_in[0]), .D
    (n_13), .Q (c_parallel_out[9]));
DFFRHQX2TS \c_parallel_out_reg[1] (.RN (n_2), .CK (clks_in[0]), .D
    (n_14), .Q (c_parallel_out[1]));
DFFRHQX2TS \c_parallel_out_reg[7] (.RN (n_2), .CK (clks_in[0]), .D
    (n_6), .Q (c_parallel_out[7]));
DFFRHQX2TS \c_parallel_out_reg[2] (.RN (n_2), .CK (clks_in[0]), .D
    (n_12), .Q (c_parallel_out[2]));
DFFRHQX2TS \c_parallel_out_reg[3] (.RN (n_2), .CK (clks_in[0]), .D
    (n_11), .Q (c_parallel_out[3]));
DFFRHQX2TS \c_parallel_out_reg[4] (.RN (n_2), .CK (clks_in[0]), .D
    (n_9), .Q (c_parallel_out[4]));
DFFRHQX2TS \c_parallel_out_reg[5] (.RN (n_2), .CK (clks_in[0]), .D
    (n_8), .Q (c_parallel_out[5]));
DFFRHQX2TS \c_parallel_out_reg[6] (.RN (n_2), .CK (clks_in[0]), .D
    (n_7), .Q (n_36));
NOR2X6TS g243(.A (n_5), .B (comma_detected), .Y (n_18));
OR2X2TS g244(.A (n_4), .B (cycle_count[2]), .Y (n_17));
CLKXOR2X2TS g245(.A (cycle_count[1]), .B (n_74), .Y (n_16));
MX2X1TS g246(.S0 (n_125), .B (shift_2reg[8]), .A (c_parallel_out[8]),
    .Y (n_15));
MX2X1TS g247(.S0 (n_0), .B (shift_2reg[1]), .A (c_parallel_out[1]),
    .Y (n_14));
MX2X1TS g248(.S0 (n_149), .B (shift_2reg[9]), .A (c_parallel_out[9]),
    .Y (n_13));
MX2X1TS g249(.S0 (n_0), .B (shift_2reg[2]), .A (c_parallel_out[2]),
    .Y (n_12));
MX2X1TS g250(.S0 (n_150), .B (shift_2reg[3]), .A (c_parallel_out[3]),
    .Y (n_11));
MX2X1TS g251(.S0 (n_149), .B (shift_2reg[0]), .A (c_parallel_out[0]),
    .Y (n_10));
MX2X1TS g252(.S0 (n_149), .B (shift_2reg[4]), .A (c_parallel_out[4]),
    .Y (n_9));
MX2X1TS g253(.S0 (n_150), .B (shift_2reg[5]), .A (c_parallel_out[5]),
    .Y (n_8));
MX2X1TS g254(.S0 (n_0), .B (shift_2reg[6]), .A (n_36), .Y (n_7));
MX2X1TS g255(.S0 (n_0), .B (shift_2reg[7]), .A (c_parallel_out[7]),
    .Y (n_6));
DFFRHQX2TS comma_detected_reg_reg(.RN (n_2), .CK (clks_in[0]), .D
    (n_3), .Q (comma_detected_reg));
NOR2X6TS g257(.A (n_1), .B (cycle_count[0]), .Y (n_5));
OR2X2TS g258(.A (cycle_count[0]), .B (cycle_count[1]), .Y (n_4));
OR2X1TS g259(.A (comma_detected_reg), .B (comma_detected), .Y (n_3));
BUFX3TS drc_bufs267(.A (n_29), .Y (cycle_count[3]));
BUFX3TS drc_bufs273(.A (n_36), .Y (c_parallel_out[6]));
```

```
  BUFX4TS drc_bufs276(.A (n_26), .Y (cycle_count[0]));
  BUFX4TS drc_bufs345(.A (n_125), .Y (n_0));
  BUFX3TS fopt553(.A (c_data_valid), .Y (n_125));
  BUFX3TS drc_bufs559(.A (n_125), .Y (n_149));
  CLKBUFX3TS drc_bufs560(.A (n_125), .Y (n_150));
  NOR2BX2TS g2(.AN (n_43), .B (comma_detected), .Y (n_154));
endmodule

module digitalRx(rst, clk, a_rx, a_rx_n, disparity_d, dataout, dispout,
    code_err, disp_err, data_valid_pipe, clk_4f, ko);
  input rst, clk, a_rx, a_rx_n, disparity_d;
  output [7:0] dataout;
  output dispout, code_err, disp_err, data_valid_pipe, clk_4f, ko;
  wire rst, clk, a_rx, a_rx_n, disparity_d;
  wire [7:0] dataout;
  wire dispout, code_err, disp_err, data_valid_pipe, clk_4f, ko;
  wire [3:0] clks_in;
  wire [9:0] c_parallel_out;
  wire c_data_valid, clk_out, disparity_q;
  clock_divider clock_divider1(rst, clk, {clks_in[3:1], clk_4f});
  decodePipe decode1(rst, clk_4f, c_data_valid, c_parallel_out,
      disparity_q, dataout, data_valid_pipe, dispout, disp_err,
      code_err, ko);
  deserializer deserializer1(rst, {clks_in[3:1], clk_4f}, a_rx,
      c_parallel_out, clk_out, disparity_d, disparity_q, c_data_valid);
endmodule
```

b. Netlist with pads

```
// Generated by Cadence Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1

// Verification Directory fv/digitalRx_wrapper

module clock_divider(rst, clk, clks_out);
  input rst, clk;
  output [3:0] clks_out;
  wire rst, clk;
  wire [3:0] clks_out;
  wire [1:0] div_by_4_q;
  wire [1:0] div_by_4_reg;
  wire UNCONNECTED, UNCONNECTED0, UNCONNECTED1, UNCONNECTED2,
      UNCONNECTED3, UNCONNECTED4, n_0, n_1;
  INVX2TS g11(.A (rst), .Y (n_1));
  DFFRX2TS \div_by_4_q_reg[1] (.RN (n_1), .CK (clk), .D
      (div_by_4_q[0]), .Q (div_by_4_q[1]), .QN (n_0));
  DFFRXLTS \clks_out_reg[3] (.RN (n_1), .CK (clk), .D (clks_out[2]), .Q
      (clks_out[3]), .QN (UNCONNECTED));
  DFFRXLTS \clks_out_reg[2] (.RN (n_1), .CK (clk), .D (clks_out[1]), .Q
```

```
      (clks_out[2]), .QN (UNCONNECTED0));
  DFFRXLTS \clks_out_reg[1] (.RN (n_1), .CK (clk), .D (clks_out[0]), .Q
      (clks_out[1]), .QN (UNCONNECTED1));
  DFFRXLTS \clks_out_reg[0] (.RN (n_1), .CK (clk), .D
      (div_by_4_reg[1]), .Q (clks_out[0]), .QN (UNCONNECTED2));
  DFFRXLTS \div_by_4_reg_reg[1] (.RN (n_1), .CK (clk), .D
      (div_by_4_q[1]), .Q (div_by_4_reg[1]), .QN (UNCONNECTED3));
  DFFRXLTS \div_by_4_q_reg[0] (.RN (n_1), .CK (clk), .D (n_0), .Q
      (div_by_4_q[0]), .QN (UNCONNECTED4));
endmodule

module decodePipe(rst, clk, data_valid, datain, dispin, dataout,
    data_valid_pipe, dispout, disp_err, code_err, ko);
  input rst, clk, data_valid, dispin;
  input [9:0] datain;
  output [7:0] dataout;
  output data_valid_pipe, dispout, disp_err, code_err, ko;
  wire rst, clk, data_valid, dispin;
  wire [9:0] datain;
  wire [7:0] dataout;
  wire data_valid_pipe, dispout, disp_err, code_err, ko;
  wire UNCONNECTED5, UNCONNECTED6, UNCONNECTED7, UNCONNECTED8,
      UNCONNECTED9, UNCONNECTED10, UNCONNECTED11, UNCONNECTED12;
  wire UNCONNECTED13, UNCONNECTED14, UNCONNECTED15, UNCONNECTED16,
      UNCONNECTED17, UNCONNECTED18, UNCONNECTED19, UNCONNECTED20;
  wire UNCONNECTED21, UNCONNECTED22, UNCONNECTED23, UNCONNECTED24,
      UNCONNECTED25, UNCONNECTED26, UNCONNECTED27, UNCONNECTED28;
  wire UNCONNECTED29, UNCONNECTED30, UNCONNECTED31, UNCONNECTED32,
      UNCONNECTED33, UNCONNECTED34, UNCONNECTED35, UNCONNECTED36;
  wire UNCONNECTED37, UNCONNECTED38, UNCONNECTED39, UNCONNECTED40,
      UNCONNECTED41, UNCONNECTED42, UNCONNECTED43, UNCONNECTED44;
  wire UNCONNECTED45, UNCONNECTED46, UNCONNECTED47, UNCONNECTED48,
      UNCONNECTED49, UNCONNECTED50, UNCONNECTED51, UNCONNECTED52;
  wire UNCONNECTED53, UNCONNECTED54, UNCONNECTED55, UNCONNECTED56,
      UNCONNECTED57, UNCONNECTED58, UNCONNECTED59, UNCONNECTED60;
  wire UNCONNECTED61, UNCONNECTED62, UNCONNECTED63, UNCONNECTED64,
      UNCONNECTED65, UNCONNECTED66, UNCONNECTED67, UNCONNECTED68;
  wire UNCONNECTED69, UNCONNECTED70, UNCONNECTED71, UNCONNECTED72,
      UNCONNECTED73, UNCONNECTED74, UNCONNECTED75, UNCONNECTED76;
  wire UNCONNECTED77, UNCONNECTED78, UNCONNECTED79, UNCONNECTED80,
      UNCONNECTED81, UNCONNECTED82, abei_reg, ai_3reg;
  wire ai_4reg, ai_reg, anbnenin_reg, bi_2reg, bi_3reg, bi_4reg,
      bi_reg, ci_2reg;
  wire ci_reg, cndnenin_reg, code_aux_a_reg, code_aux_b_reg,
      code_aux_c_reg, code_aux_d_reg, compa_reg, compb_reg;
  wire compc_reg, compd_reg, compe_reg, di_2reg, di_4reg, di_reg,
      disp6a0_reg, disp6a2_reg;
  wire disp6a_reg, disp6b_reg, disp6n_2reg, disp6n_reg, disp6p_2reg,
      disp6p_reg, dispaux_a_reg, dispin_2reg;
  wire dispin_3reg, dispin_4reg, dispin_reg, ei_2reg, ei_4reg, ei_reg,
      feqg_reg, fghj22_reg;
```

```
wire fghjp13_reg, fghjp31_reg, fi_2reg, fi_3reg, fi_4reg, fi_reg,
    gi_2reg, gi_4reg;
wire gi_reg, heqj_reg, hi_2reg, hi_4reg, hi_reg, ii_3reg, ii_4reg,
    ii_reg;
wire ji_2reg, ji_3reg, ji_4reg, ji_reg, k28p_reg, n_1, n_2, n_3;
wire n_4, n_5, n_6, n_7, n_8, n_9, n_10, n_11;
wire n_12, n_13, n_14, n_15, n_16, n_17, n_18, n_19;
wire n_20, n_21, n_22, n_23, n_24, n_25, n_26, n_28;
wire n_30, n_32, n_34, n_35, n_36, n_37, n_38, n_39;
wire n_40, n_41, n_42, n_43, n_44, n_45, n_46, n_47;
wire n_48, n_49, n_50, n_51, n_52, n_53, n_55, n_56;
wire n_57, n_58, n_59, n_60, n_61, n_62, n_63, n_64;
wire n_65, n_66, n_67, n_68, n_69, n_70, n_71, n_72;
wire n_73, n_74, n_75, n_76, n_77, n_78, n_79, n_80;
wire n_81, n_82, n_83, n_84, n_87, n_88, n_89, n_92;
wire n_93, n_95, n_96, n_99, n_100, n_101, n_102, n_103;
wire n_104, n_105, n_106, n_107, n_108, n_109, n_110, n_111;
wire n_112, n_113, n_114, n_117, n_118, n_119, n_120, n_122;
wire n_123, n_124, n_125, n_126, n_127, n_128, n_129, n_130;
wire n_131, n_132, n_133, n_134, n_136, n_137, n_138, n_139;
wire n_140, n_141, n_142, n_143, n_144, n_145, n_146, n_147;
wire n_148, n_149, n_150, n_151, n_152, n_153, n_154, n_155;
wire n_156, n_157, n_158, n_159, n_160, n_161, n_162, n_163;
wire n_164, n_165, n_166, n_167, n_168, n_169, n_170, n_171;
wire n_172, n_173, n_174, n_175, n_176, n_177, n_178, n_179;
wire n_180, n_181, n_182, n_183, n_184, n_185, n_186, n_187;
wire n_188, n_189, n_190, n_191, n_192, n_193, n_194, n_195;
wire n_196, n_197, n_198, n_199, n_200, n_201, n_202, n_203;
wire n_204, n_205, n_206, n_207, n_208, n_209, n_210, n_220;
wire n_221, n_222, n_223, n_224, n_225, n_226, p04_2reg, p04_reg;
wire p13_2reg, p13_3reg, p13_reg, p13dei_reg, p13en_reg, p13in_reg,
    p22_reg, p22aceeqi_reg;
wire p22ancneeqi_reg, p22bceeqi_reg, p22bncneeqi_reg, p31_2reg,
    p31_3reg, p31i_reg, p40_2reg, p40_reg;
DFFRHQX2TS ao_reg_reg(.RN (n_5), .CK (clk), .D (n_206), .Q
    (dataout[0]));
DFFRHQX2TS bo_reg_reg(.RN (n_5), .CK (clk), .D (n_205), .Q
    (dataout[1]));
DFFRHQX2TS co_reg_reg(.RN (n_5), .CK (clk), .D (n_204), .Q
    (dataout[2]));
DFFRHQX2TS code_err_reg_reg(.RN (n_5), .CK (clk), .D (n_210), .Q
    (code_err));
DFFRHQX2TS disp_err_reg_reg(.RN (n_5), .CK (clk), .D (n_208), .Q
    (disp_err));
DFFRHQX2TS dispout_reg_reg(.RN (n_5), .CK (clk), .D (n_209), .Q
    (dispout));
DFFRHQX2TS do__reg_reg(.RN (n_5), .CK (clk), .D (n_203), .Q
    (dataout[3]));
DFFRHQX2TS eo_reg_reg(.RN (n_5), .CK (clk), .D (n_207), .Q
    (dataout[4]));
DFFRHQX2TS fo_reg_reg(.RN (n_5), .CK (clk), .D (n_147), .Q
```

```
    (dataout[5]));
DFFRHQX2TS go_reg_reg(.RN (n_5), .CK (clk), .D (n_146), .Q
    (dataout[6]));
DFFRHQX2TS ho_reg_reg(.RN (n_5), .CK (clk), .D (n_163), .Q
    (dataout[7]));
DFFRHQX2TS ko_reg_reg(.RN (n_5), .CK (clk), .D (n_199), .Q (ko));
NAND4XLTS g4856(.A (n_201), .B (n_187), .C (n_157), .D (n_200), .Y
    (n_210));
OAI21XLTS g4857(.A0 (n_224), .A1 (n_202), .B0 (n_223), .Y (n_209));
NAND3BXLTS g4859(.AN (dispaux_a_reg), .B (n_197), .C (n_196), .Y
    (n_208));
XOR2XLTS g4865(.A (n_64), .B (compe_reg), .Y (n_207));
XOR2XLTS g4866(.A (ai_4reg), .B (compa_reg), .Y (n_206));
XOR2XLTS g4867(.A (bi_4reg), .B (compb_reg), .Y (n_205));
XOR2XLTS g4868(.A (n_25), .B (compc_reg), .Y (n_204));
XOR2XLTS g4869(.A (di_4reg), .B (compd_reg), .Y (n_203));
AOI21XLTS g4870(.A0 (disp6b_reg), .A1 (fghj22_reg), .B0
    (fghjp31_reg), .Y (n_202));
AND3XLTS g4871(.A (n_167), .B (n_166), .C (n_191), .Y (n_201));
NOR4XLTS g4872(.A (code_aux_b_reg), .B (code_aux_a_reg), .C
    (code_aux_c_reg), .D (code_aux_d_reg), .Y (n_200));
MXI2XLTS g4875(.S0 (ii_4reg), .B (n_185), .A (n_181), .Y (n_199));
MXI2XLTS g4876(.S0 (dispin_3reg), .B (n_184), .A (n_183), .Y (n_198));
OAI21XLTS g4877(.A0 (n_178), .A1 (disp6p_2reg), .B0 (fghjp31_reg), .Y
    (n_197));
OAI21XLTS g4878(.A0 (n_177), .A1 (disp6n_2reg), .B0 (fghjp13_reg), .Y
    (n_196));
OAI211XLTS g4885(.A0 (n_149), .A1 (n_83), .B0 (n_179), .C0 (n_150),
    .Y (n_195));
NAND2X1TS g4886(.A (n_182), .B (n_170), .Y (n_194));
NAND2X1TS g4887(.A (n_182), .B (n_169), .Y (n_193));
NAND2X1TS g4888(.A (n_182), .B (n_168), .Y (n_192));
NAND3BXLTS g4889(.AN (n_69), .B (n_107), .C (disp6n_2reg), .Y
    (n_191));
NAND4BBXLTS g4890(.AN (anbnenin_reg), .BN (p13in_reg), .C (n_171), .D
    (n_170), .Y (n_190));
NAND2XLTS g4891(.A (n_180), .B (n_171), .Y (n_189));
NAND4XLTS g4894(.A (n_124), .B (n_158), .C (n_111), .D (n_155), .Y
    (n_188));
OAI21XLTS g4895(.A0 (n_123), .A1 (fghjp31_reg), .B0 (disp6p_2reg), .Y
    (n_187));
OAI21XLTS g4896(.A0 (n_154), .A1 (n_101), .B0 (n_176), .Y (n_186));
AOI31XLTS g4897(.A0 (n_64), .A1 (di_4reg), .A2 (n_25), .B0 (n_175),
    .Y (n_185));
OA21XLTS g4898(.A0 (n_82), .A1 (disp6n_reg), .B0 (n_173), .Y (n_184));
NOR3XLTS g4899(.A (n_172), .B (n_93), .C (disp6n_reg), .Y (n_183));
NOR2BXLTS g4900(.AN (n_174), .B (n_104), .Y (n_181));
NOR3BX1TS g4901(.AN (n_171), .B (p31i_reg), .C (abei_reg), .Y
    (n_182));
NOR4XLTS g4902(.A (p22bceeqi_reg), .B (anbnenin_reg), .C
    (p22ancneeqi_reg), .D (p31i_reg), .Y (n_180));
```

```
OAI21XLTS g4903(.A0 (disp6a2_reg), .A1 (disp6a_reg), .B0 (n_79), .Y
    (n_179));
NOR2BXLTS g4904(.AN (dispin_4reg), .B (disp6n_2reg), .Y (n_178));
NOR2XLTS g4905(.A (dispin_4reg), .B (disp6p_2reg), .Y (n_177));
OAI21XLTS g4906(.A0 (n_151), .A1 (n_81), .B0 (n_102), .Y (n_176));
NOR4BXLTS g4907(.AN (p13_3reg), .B (n_223), .C (n_68), .D (ei_4reg),
    .Y (n_175));
NAND4BXLTS g4908(.AN (n_69), .B (n_224), .C (p31_3reg), .D (ei_4reg),
    .Y (n_174));
AOI31XLTS g4909(.A0 (bi_3reg), .A1 (n_32), .A2 (ai_3reg), .B0
    (disp6p_reg), .Y (n_173));
NOR2XLTS g4910(.A (n_18), .B (disp6p_reg), .Y (n_172));
NOR3X1TS g4911(.A (p13dei_reg), .B (p13en_reg), .C (cndnenin_reg), .Y
    (n_171));
NOR2XLTS g4912(.A (p22ancneeqi_reg), .B (p22bncneeqi_reg), .Y
    (n_170));
NOR2XLTS g4913(.A (p22aceeqi_reg), .B (p22bceeqi_reg), .Y (n_169));
NOR2XLTS g4914(.A (p22aceeqi_reg), .B (p22bncneeqi_reg), .Y (n_168));
NAND2BXLTS g4917(.AN (n_164), .B (fghjp13_reg), .Y (n_167));
NAND4BXLTS g4918(.AN (n_25), .B (n_153), .C (ii_4reg), .D (n_105), .Y
    (n_166));
INVX2TS g4927(.A (n_164), .Y (n_165));
NAND3BXLTS g4928(.AN (n_138), .B (n_136), .C (n_137), .Y (n_163));
NOR3XLTS g4929(.A (n_148), .B (n_30), .C (n_21), .Y (n_162));
NOR3BXLTS g4930(.AN (n_34), .B (n_148), .C (n_52), .Y (n_161));
NOR3XLTS g4931(.A (n_148), .B (n_30), .C (n_34), .Y (n_160));
NOR3BXLTS g4932(.AN (n_20), .B (n_148), .C (n_52), .Y (n_159));
OR2XLTS g4933(.A (n_83), .B (n_145), .Y (n_158));
OAI21XLTS g4934(.A0 (n_92), .A1 (fghjp13_reg), .B0 (n_119), .Y
    (n_157));
OAI2BB1XLTS g4935(.A0N (n_40), .A1N (n_58), .B0 (n_143), .Y (n_156));
AOI22XLTS g4936(.A0 (n_57), .A1 (p22_reg), .B0 (n_59), .B1 (p13_reg),
    .Y (n_164));
AOI211XLTS g4937(.A0 (n_80), .A1 (p13_2reg), .B0 (p40_2reg), .C0
    (p04_2reg), .Y (n_155));
AOI21XLTS g4938(.A0 (n_53), .A1 (n_49), .B0 (n_142), .Y (n_154));
NAND2BXLTS g4939(.AN (fghjp31_reg), .B (n_99), .Y (n_153));
AO21XLTS g4940(.A0 (p22_reg), .A1 (dispin_2reg), .B0 (n_41), .Y
    (n_152));
NOR3BXLTS g4941(.AN (n_50), .B (n_38), .C (p31_2reg), .Y (n_151));
NAND2X1TS g4942(.A (n_28), .B (disp6a2_reg), .Y (n_150));
NOR2BXLTS g4943(.AN (disp6a0_reg), .B (n_28), .Y (n_149));
NAND2X1TS g4947(.A (n_129), .B (n_140), .Y (n_147));
NAND2X1TS g4948(.A (n_129), .B (n_139), .Y (n_146));
OAI2BB1X1TS g4949(.A0N (n_23), .A1N (n_58), .B0 (p22_reg), .Y
    (n_148));
NOR2XLTS g4950(.A (n_100), .B (p31_2reg), .Y (n_145));
NOR3BXLTS g4951(.AN (di_2reg), .B (n_23), .C (n_122), .Y (n_144));
NAND2BXLTS g4952(.AN (n_59), .B (p22_reg), .Y (n_143));
NOR2BXLTS g4953(.AN (n_81), .B (p13_2reg), .Y (n_142));
OAI21XLTS g4963(.A0 (n_84), .A1 (feqg_reg), .B0 (n_127), .Y (n_141));
```

```
MXI2XLTS g4964(.S0 (k28p_reg), .B (n_106), .A (n_108), .Y (n_140));
MXI2XLTS g4965(.S0 (k28p_reg), .B (n_108), .A (n_106), .Y (n_139));
AOI211XLTS g4966(.A0 (n_117), .A1 (k28p_reg), .B0 (n_65), .C0 (n_67),
    .Y (n_138));
OAI211XLTS g4967(.A0 (n_118), .A1 (k28p_reg), .B0 (n_65), .C0 (n_67),
    .Y (n_137));
MXI2XLTS g4968(.S0 (n_70), .B (n_107), .A (n_109), .Y (n_136));
AND2XLTS g4970(.A (n_41), .B (n_36), .Y (n_134));
AND2XLTS g4971(.A (dispin_2reg), .B (n_40), .Y (n_133));
NOR2XLTS g4972(.A (n_122), .B (ei_2reg), .Y (n_132));
NOR2XLTS g4973(.A (n_122), .B (n_37), .Y (n_131));
NOR2XLTS g4974(.A (n_122), .B (dispin_2reg), .Y (n_130));
OAI2BB1XLTS g4981(.A0N (n_63), .A1N (n_61), .B0 (n_114), .Y (n_128));
NAND3BXLTS g4982(.AN (heqj_reg), .B (datain[6]), .C (n_46), .Y
    (n_127));
OAI31X1TS g4983(.A0 (n_101), .A1 (n_226), .A2 (n_79), .B0 (n_113), .Y
    (n_126));
AO21XLTS g4984(.A0 (n_103), .A1 (n_100), .B0 (n_112), .Y (n_125));
NAND2BXLTS g4985(.AN (n_18), .B (n_110), .Y (n_124));
NOR2XLTS g4986(.A (n_107), .B (n_109), .Y (n_129));
AND2XLTS g4987(.A (n_70), .B (n_109), .Y (n_123));
OAI22XLTS g4990(.A0 (n_73), .A1 (heqj_reg), .B0 (n_72), .B1
    (feqg_reg), .Y (n_120));
NOR4BXLTS g4991(.AN (n_26), .B (n_89), .C (n_64), .D (ii_4reg), .Y
    (n_119));
INVX2TS g4996(.A (n_117), .Y (n_118));
AOI22XLTS g4999(.A0 (n_55), .A1 (n_60), .B0 (n_56), .B1 (n_62), .Y
    (n_114));
NAND3BXLTS g5000(.AN (n_83), .B (n_102), .C (n_225), .Y (n_113));
NOR4XLTS g5001(.A (n_73), .B (n_225), .C (n_83), .D (n_44), .Y
    (n_112));
NAND2BXLTS g5002(.AN (n_101), .B (fi_3reg), .Y (n_111));
OAI21XLTS g5003(.A0 (n_79), .A1 (n_43), .B0 (n_72), .Y (n_110));
NOR2BXLTS g5004(.AN (n_99), .B (n_92), .Y (n_117));
AND2XLTS g5005(.A (n_224), .B (fi_4reg), .Y (n_109));
OAI21XLTS g5006(.A0 (n_70), .A1 (hi_4reg), .B0 (n_88), .Y (n_108));
NOR2XLTS g5007(.A (n_223), .B (n_66), .Y (n_107));
OAI21XLTS g5008(.A0 (n_71), .A1 (n_65), .B0 (n_87), .Y (n_106));
NOR3BXLTS g5009(.AN (ei_4reg), .B (ai_4reg), .C (bi_4reg), .Y
    (n_105));
NOR3XLTS g5010(.A (n_64), .B (di_4reg), .C (n_26), .Y (n_104));
AND2XLTS g5016(.A (n_55), .B (n_56), .Y (n_96));
AND2XLTS g5017(.A (n_62), .B (n_60), .Y (n_95));
AND2XLTS g5018(.A (n_80), .B (n_226), .Y (n_103));
NOR2XLTS g5020(.A (n_72), .B (n_46), .Y (n_102));
NAND2BXLTS g5021(.AN (n_84), .B (n_47), .Y (n_101));
NOR2BXLTS g5022(.AN (n_43), .B (n_82), .Y (n_100));
NOR3XLTS g5023(.A (n_32), .B (bi_3reg), .C (ai_3reg), .Y (n_93));
NAND2XLTS g5024(.A (gi_4reg), .B (fi_4reg), .Y (n_99));
NAND2XLTS g5025(.A (bi_4reg), .B (ai_4reg), .Y (n_89));
NAND2BXLTS g5026(.AN (n_67), .B (n_66), .Y (n_88));
```

```
NOR2XLTS g5027(.A (n_66), .B (gi_4reg), .Y (n_92));
NAND2BXLTS g5028(.AN (n_66), .B (n_67), .Y (n_87));
INVX1TS g5036(.A (n_80), .Y (n_79));
NOR3XLTS g5037(.A (n_58), .B (bi_2reg), .C (n_20), .Y (n_78));
NOR3XLTS g5038(.A (n_58), .B (ci_2reg), .C (di_2reg), .Y (n_77));
NOR3BXLTS g5039(.AN (n_21), .B (n_23), .C (n_51), .Y (n_76));
XNOR2XLTS g5040(.A (hi_2reg), .B (ji_2reg), .Y (n_75));
XNOR2XLTS g5041(.A (fi_2reg), .B (gi_2reg), .Y (n_74));
NAND2X1TS g5044(.A (ji_3reg), .B (n_43), .Y (n_84));
NAND2X1TS g5045(.A (n_38), .B (n_49), .Y (n_83));
NAND2XLTS g5046(.A (n_47), .B (fi_3reg), .Y (n_82));
NOR2XLTS g5047(.A (n_53), .B (n_50), .Y (n_81));
NOR2XLTS g5048(.A (n_50), .B (ii_3reg), .Y (n_80));
INVX2TS g5049(.A (n_70), .Y (n_71));
CLKBUFX2TS g5050(.A (n_69), .Y (n_70));
INVX1TS g5051(.A (n_69), .Y (n_68));
CLKBUFX2TS g5052(.A (gi_4reg), .Y (n_69));
CLKBUFX2TS g5053(.A (ji_4reg), .Y (n_67));
CLKBUFX2TS g5054(.A (fi_4reg), .Y (n_66));
CLKBUFX2TS g5056(.A (ei_4reg), .Y (n_64));
OR2XLTS g5057(.A (fi_3reg), .B (n_47), .Y (n_73));
OR2XLTS g5058(.A (n_44), .B (ji_3reg), .Y (n_72));
ADDHXLTS g5070(.A (di_reg), .B (ai_reg), .S (n_63), .CO (n_62));
ADDHXLTS g5071(.A (ci_reg), .B (bi_reg), .S (n_61), .CO (n_60));
INVX1TS g5072(.A (n_57), .Y (n_58));
NAND2X1TS g5073(.A (n_36), .B (ei_2reg), .Y (n_59));
NOR2XLTS g5074(.A (ei_2reg), .B (n_37), .Y (n_57));
NOR2XLTS g5088(.A (ci_reg), .B (bi_reg), .Y (n_56));
NOR2XLTS g5089(.A (di_reg), .B (ai_reg), .Y (n_55));
INVX2TS g5102(.A (rst), .Y (n_5));
INVX1TS drc_bufs5103(.A (n_48), .Y (n_49));
INVX1TS drc_bufs5107(.A (n_45), .Y (n_46));
INVX1TS drc_bufs5111(.A (n_42), .Y (n_43));
INVX1TS drc_bufs5115(.A (n_39), .Y (n_40));
INVX1TS drc_bufs5119(.A (n_53), .Y (n_38));
INVX1TS drc_bufs5123(.A (n_35), .Y (n_36));
INVX1TS drc_bufs5127(.A (n_51), .Y (n_34));
INVX1TS drc_bufs5135(.A (n_52), .Y (n_30));
INVX1TS drc_bufs5143(.A (n_24), .Y (n_25));
INVX2TS drc_bufs5147(.A (n_22), .Y (n_23));
INVX2TS drc_bufs5148(.A (n_59), .Y (n_22));
INVX1TS drc_bufs5151(.A (n_19), .Y (n_20));
INVX2TS drc_bufs5155(.A (n_17), .Y (n_18));
INVX1TS drc_bufs5156(.A (n_73), .Y (n_17));
MX2XLTS g3314(.S0 (data_valid), .B (datain[9]), .A (ji_reg), .Y
    (n_16));
MX2XLTS g3315(.S0 (n_3), .B (dispin), .A (dispin_reg), .Y (n_15));
MX2XLTS g3316(.S0 (n_4), .B (datain[4]), .A (ei_reg), .Y (n_14));
MX2XLTS g3317(.S0 (n_4), .B (datain[6]), .A (fi_reg), .Y (n_13));
MX2XLTS g3318(.S0 (n_1), .B (datain[7]), .A (gi_reg), .Y (n_12));
MX2XLTS g3319(.S0 (n_1), .B (datain[8]), .A (hi_reg), .Y (n_11));
```

```
MX2XLTS g3320(.S0 (n_3), .B (datain[5]), .A (ii_reg), .Y (n_10));
MX2XLTS g3321(.S0 (data_valid), .B (datain[2]), .A (ci_reg), .Y
    (n_9));
MX2XLTS g3322(.S0 (n_3), .B (datain[3]), .A (di_reg), .Y (n_8));
MX2XLTS g3323(.S0 (n_4), .B (datain[0]), .A (ai_reg), .Y (n_7));
MX2XLTS g3324(.S0 (n_1), .B (datain[1]), .A (bi_reg), .Y (n_6));
INVX1TS drc_bufs(.A (n_2), .Y (n_4));
INVX1TS drc_bufs3326(.A (n_2), .Y (n_3));
INVX1TS drc_bufs3327(.A (data_valid), .Y (n_2));
INVX1TS drc_bufs3331(.A (n_2), .Y (n_1));
OAI222XLTS g2(.A0 (n_18), .A1 (n_84), .B0 (feqg_reg), .B1 (heqj_reg),
    .C0 (n_72), .C1 (n_82), .Y (n_220));
AO22XLTS g5158(.A0 (n_56), .A1 (n_63), .B0 (n_61), .B1 (n_55), .Y
    (n_221));
AO22XLTS g5159(.A0 (n_60), .A1 (n_63), .B0 (n_61), .B1 (n_62), .Y
    (n_222));
ACCSIHCONX2TS g5160(.A (ji_4reg), .B (hi_4reg), .CO0N (n_223), .CO1N
    (n_224));
ACCSIHCONX2TS g5161(.A (n_28), .B (n_32), .CO0N (n_225), .CO1N
    (n_226));
DFFRX2TS p13_reg_reg(.RN (n_5), .CK (clk), .D (n_221), .Q (p13_reg),
    .QN (n_122));
DFFRX2TS hi_4reg_reg(.RN (n_5), .CK (clk), .D (n_44), .Q (hi_4reg),
    .QN (n_65));
DFFRX2TS ei_3reg_reg(.RN (n_5), .CK (clk), .D (ei_2reg), .Q (n_50),
    .QN (n_48));
DFFRX2TS gi_3reg_reg(.RN (n_5), .CK (clk), .D (gi_2reg), .Q (n_47),
    .QN (n_45));
DFFRX2TS hi_3reg_reg(.RN (n_5), .CK (clk), .D (hi_2reg), .Q (n_44),
    .QN (n_42));
DFFRX2TS p31_reg_reg(.RN (n_5), .CK (clk), .D (n_222), .Q (n_41), .QN
    (n_39));
DFFRX2TS ii_3reg_reg(.RN (n_5), .CK (clk), .D (n_36), .Q (ii_3reg),
    .QN (n_53));
DFFRX2TS ii_2reg_reg(.RN (n_5), .CK (clk), .D (ii_reg), .Q (n_37),
    .QN (n_35));
DFFRX2TS bi_2reg_reg(.RN (n_5), .CK (clk), .D (bi_reg), .Q (bi_2reg),
    .QN (n_51));
DFFRX2TS ci_2reg_reg(.RN (n_5), .CK (clk), .D (ci_reg), .Q (ci_2reg),
    .QN (n_52));
DFFRX2TS ci_4reg_reg(.RN (n_5), .CK (clk), .D (n_32), .Q (n_26), .QN
    (n_24));
DFFRX2TS ai_2reg_reg(.RN (n_5), .CK (clk), .D (ai_reg), .Q (n_21),
    .QN (n_19));
DFFRXLTS abei_reg_reg(.RN (n_5), .CK (clk), .D (n_76), .Q (abei_reg),
    .QN (UNCONNECTED5));
DFFRXLTS ei_2reg_reg(.RN (n_5), .CK (clk), .D (ei_reg), .Q (ei_2reg),
    .QN (UNCONNECTED6));
DFFRXLTS ai_3reg_reg(.RN (n_5), .CK (clk), .D (n_21), .Q (ai_3reg),
    .QN (UNCONNECTED7));
DFFRXLTS ai_4reg_reg(.RN (n_5), .CK (clk), .D (ai_3reg), .Q
```

```
    (ai_4reg), .QN (UNCONNECTED8));
  DFFRXLTS anbnenin_reg_reg(.RN (n_5), .CK (clk), .D (n_78), .Q
    (anbnenin_reg), .QN (UNCONNECTED9));
  DFFRXLTS compa_reg_reg(.RN (n_5), .CK (clk), .D (n_194), .Q
    (compa_reg), .QN (UNCONNECTED10));
  DFFRXLTS bi_3reg_reg(.RN (n_5), .CK (clk), .D (n_34), .Q (bi_3reg),
    .QN (UNCONNECTED11));
  DFFRXLTS bi_4reg_reg(.RN (n_5), .CK (clk), .D (bi_3reg), .Q
    (bi_4reg), .QN (UNCONNECTED12));
  DFFRXLTS compb_reg_reg(.RN (n_5), .CK (clk), .D (n_193), .Q
    (compb_reg), .QN (UNCONNECTED13));
  DFFRXLTS ci_3reg_reg(.RN (n_5), .CK (clk), .D (n_30), .Q (n_32), .QN
    (UNCONNECTED14));
  DFFRXLTS cndnenin_reg_reg(.RN (n_5), .CK (clk), .D (n_77), .Q
    (cndnenin_reg), .QN (UNCONNECTED15));
  DFFRXLTS di_2reg_reg(.RN (n_5), .CK (clk), .D (di_reg), .Q (di_2reg),
    .QN (UNCONNECTED16));
  DFFRXLTS compc_reg_reg(.RN (n_5), .CK (clk), .D (n_189), .Q
    (compc_reg), .QN (UNCONNECTED17));
  DFFRXLTS code_aux_a_reg_reg(.RN (n_5), .CK (clk), .D (n_188), .Q
    (code_aux_a_reg), .QN (UNCONNECTED18));
  DFFRXLTS fi_3reg_reg(.RN (n_5), .CK (clk), .D (fi_2reg), .Q
    (fi_3reg), .QN (UNCONNECTED19));
  DFFRXLTS ji_3reg_reg(.RN (n_5), .CK (clk), .D (ji_2reg), .Q
    (ji_3reg), .QN (UNCONNECTED20));
  DFFRXLTS p31_2reg_reg(.RN (n_5), .CK (clk), .D (n_40), .Q (p31_2reg),
    .QN (UNCONNECTED21));
  DFFRXLTS p13_2reg_reg(.RN (n_5), .CK (clk), .D (p13_reg), .Q
    (p13_2reg), .QN (UNCONNECTED22));
  DFFRXLTS p40_2reg_reg(.RN (n_5), .CK (clk), .D (p40_reg), .Q
    (p40_2reg), .QN (UNCONNECTED23));
  DFFRXLTS p04_2reg_reg(.RN (n_5), .CK (clk), .D (p04_reg), .Q
    (p04_2reg), .QN (UNCONNECTED24));
  DFFRXLTS code_aux_b_reg_reg(.RN (n_5), .CK (clk), .D (n_186), .Q
    (code_aux_b_reg), .QN (UNCONNECTED25));
  DFFRXLTS code_aux_c_reg_reg(.RN (n_5), .CK (clk), .D (n_126), .Q
    (code_aux_c_reg), .QN (UNCONNECTED26));
  DFFRXLTS di_3reg_reg(.RN (n_5), .CK (clk), .D (di_2reg), .Q (n_28),
    .QN (UNCONNECTED27));
  DFFRXLTS code_aux_d_reg_reg(.RN (n_5), .CK (clk), .D (n_125), .Q
    (code_aux_d_reg), .QN (UNCONNECTED28));
  DFFRXLTS p22_reg_reg(.RN (n_5), .CK (clk), .D (n_128), .Q (p22_reg),
    .QN (UNCONNECTED29));
  DFFRXLTS fghjp13_reg_reg(.RN (n_5), .CK (clk), .D (n_120), .Q
    (fghjp13_reg), .QN (UNCONNECTED30));
  DFFRXLTS fghjp31_reg_reg(.RN (n_5), .CK (clk), .D (n_141), .Q
    (fghjp31_reg), .QN (UNCONNECTED31));
  DFFRXLTS gi_4reg_reg(.RN (n_5), .CK (clk), .D (n_46), .Q (gi_4reg),
    .QN (UNCONNECTED32));
  DFFRXLTS fi_4reg_reg(.RN (n_5), .CK (clk), .D (fi_3reg), .Q
    (fi_4reg), .QN (UNCONNECTED33));
```

```
DFFRXLTS ii_4reg_reg(.RN (n_5), .CK (clk), .D (n_38), .Q (ii_4reg),
    .QN (UNCONNECTED34));
DFFRXLTS ei_4reg_reg(.RN (n_5), .CK (clk), .D (n_49), .Q (ei_4reg),
    .QN (UNCONNECTED35));
DFFRXLTS ji_4reg_reg(.RN (n_5), .CK (clk), .D (ji_3reg), .Q
    (ji_4reg), .QN (UNCONNECTED36));
DFFRXLTS disp6n_2reg_reg(.RN (n_5), .CK (clk), .D (disp6n_reg), .Q
    (disp6n_2reg), .QN (UNCONNECTED37));
DFFRXLTS disp6p_2reg_reg(.RN (n_5), .CK (clk), .D (disp6p_reg), .Q
    (disp6p_2reg), .QN (UNCONNECTED38));
DFFRXLTS p13dei_reg_reg(.RN (n_5), .CK (clk), .D (n_144), .Q
    (p13dei_reg), .QN (UNCONNECTED39));
DFFRXLTS p13en_reg_reg(.RN (n_5), .CK (clk), .D (n_132), .Q
    (p13en_reg), .QN (UNCONNECTED40));
DFFRXLTS p31i_reg_reg(.RN (n_5), .CK (clk), .D (n_134), .Q
    (p31i_reg), .QN (UNCONNECTED41));
DFFRXLTS p22ancneeqi_reg_reg(.RN (n_5), .CK (clk), .D (n_162), .Q
    (p22ancneeqi_reg), .QN (UNCONNECTED42));
DFFRX1TS p22bncneeqi_reg_reg(.RN (n_5), .CK (clk), .D (n_160), .Q
    (p22bncneeqi_reg), .QN (UNCONNECTED43));
DFFRXLTS p22aceeqi_reg_reg(.RN (n_5), .CK (clk), .D (n_159), .Q
    (p22aceeqi_reg), .QN (UNCONNECTED44));
DFFRX1TS p22bceeqi_reg_reg(.RN (n_5), .CK (clk), .D (n_161), .Q
    (p22bceeqi_reg), .QN (UNCONNECTED45));
DFFRXLTS compd_reg_reg(.RN (n_5), .CK (clk), .D (n_192), .Q
    (compd_reg), .QN (UNCONNECTED46));
DFFRXLTS compe_reg_reg(.RN (n_5), .CK (clk), .D (n_190), .Q
    (compe_reg), .QN (UNCONNECTED47));
DFFRXLTS p13in_reg_reg(.RN (n_5), .CK (clk), .D (n_131), .Q
    (p13in_reg), .QN (UNCONNECTED48));
DFFRXLTS di_reg_reg(.RN (n_5), .CK (clk), .D (n_8), .Q (di_reg), .QN
    (UNCONNECTED49));
DFFRXLTS di_4reg_reg(.RN (n_5), .CK (clk), .D (n_28), .Q (di_4reg),
    .QN (UNCONNECTED50));
DFFRXLTS disp6a0_reg_reg(.RN (n_5), .CK (clk), .D (n_130), .Q
    (disp6a0_reg), .QN (UNCONNECTED51));
DFFRXLTS dispin_2reg_reg(.RN (n_5), .CK (clk), .D (dispin_reg), .Q
    (dispin_2reg), .QN (UNCONNECTED52));
DFFRXLTS disp6a2_reg_reg(.RN (n_5), .CK (clk), .D (n_133), .Q
    (disp6a2_reg), .QN (UNCONNECTED53));
DFFRXLTS disp6a_reg_reg(.RN (n_5), .CK (clk), .D (n_152), .Q
    (disp6a_reg), .QN (UNCONNECTED54));
DFFRXLTS disp6b_reg_reg(.RN (n_5), .CK (clk), .D (n_195), .Q
    (disp6b_reg), .QN (UNCONNECTED55));
DFFRXLTS disp6n_reg_reg(.RN (n_5), .CK (clk), .D (n_165), .Q
    (disp6n_reg), .QN (UNCONNECTED56));
DFFRXLTS disp6p_reg_reg(.RN (n_5), .CK (clk), .D (n_156), .Q
    (disp6p_reg), .QN (UNCONNECTED57));
DFFRXLTS dispaux_a_reg_reg(.RN (n_5), .CK (clk), .D (n_198), .Q
    (dispaux_a_reg), .QN (UNCONNECTED58));
DFFRXLTS dispin_4reg_reg(.RN (n_5), .CK (clk), .D (dispin_3reg), .Q
```

```
      (dispin_4reg), .QN (UNCONNECTED59));
  DFFRXLTS dispin_3reg_reg(.RN (n_5), .CK (clk), .D (dispin_2reg), .Q
      (dispin_3reg), .QN (UNCONNECTED60));
  DFFRXLTS dispin_reg_reg(.RN (n_5), .CK (clk), .D (n_15), .Q
      (dispin_reg), .QN (UNCONNECTED61));
  DFFRXLTS fghj22_reg_reg(.RN (n_5), .CK (clk), .D (n_220), .Q
      (fghj22_reg), .QN (UNCONNECTED62));
  DFFRXLTS ei_reg_reg(.RN (n_5), .CK (clk), .D (n_14), .Q (ei_reg), .QN
      (UNCONNECTED63));
  DFFRXLTS feqg_reg_reg(.RN (n_5), .CK (clk), .D (n_74), .Q (feqg_reg),
      .QN (UNCONNECTED64));
  DFFRXLTS fi_2reg_reg(.RN (n_5), .CK (clk), .D (fi_reg), .Q (fi_2reg),
      .QN (UNCONNECTED65));
  DFFRXLTS gi_2reg_reg(.RN (n_5), .CK (clk), .D (gi_reg), .Q (gi_2reg),
      .QN (UNCONNECTED66));
  DFFRXLTS heqj_reg_reg(.RN (n_5), .CK (clk), .D (n_75), .Q (heqj_reg),
      .QN (UNCONNECTED67));
  DFFRXLTS fi_reg_reg(.RN (n_5), .CK (clk), .D (n_13), .Q (fi_reg), .QN
      (UNCONNECTED68));
  DFFRXLTS k28p_reg_reg(.RN (n_5), .CK (clk), .D (n_103), .Q
      (k28p_reg), .QN (UNCONNECTED69));
  DFFRXLTS gi_reg_reg(.RN (n_5), .CK (clk), .D (n_12), .Q (gi_reg), .QN
      (UNCONNECTED70));
  DFFRXLTS hi_2reg_reg(.RN (n_5), .CK (clk), .D (hi_reg), .Q (hi_2reg),
      .QN (UNCONNECTED71));
  DFFRXLTS ji_2reg_reg(.RN (n_5), .CK (clk), .D (ji_reg), .Q (ji_2reg),
      .QN (UNCONNECTED72));
  DFFRXLTS hi_reg_reg(.RN (n_5), .CK (clk), .D (n_11), .Q (hi_reg), .QN
      (UNCONNECTED73));
  DFFRXLTS ji_reg_reg(.RN (n_5), .CK (clk), .D (n_16), .Q (ji_reg), .QN
      (UNCONNECTED74));
  DFFRXLTS p13_3reg_reg(.RN (n_5), .CK (clk), .D (p13_2reg), .Q
      (p13_3reg), .QN (UNCONNECTED75));
  DFFRXLTS p31_3reg_reg(.RN (n_5), .CK (clk), .D (p31_2reg), .Q
      (p31_3reg), .QN (UNCONNECTED76));
  DFFRX1TS p04_reg_reg(.RN (n_5), .CK (clk), .D (n_96), .Q (p04_reg),
      .QN (UNCONNECTED77));
  DFFRXLTS ai_reg_reg(.RN (n_5), .CK (clk), .D (n_7), .Q (ai_reg), .QN
      (UNCONNECTED78));
  DFFRXLTS ci_reg_reg(.RN (n_5), .CK (clk), .D (n_9), .Q (ci_reg), .QN
      (UNCONNECTED79));
  DFFRXLTS bi_reg_reg(.RN (n_5), .CK (clk), .D (n_6), .Q (bi_reg), .QN
      (UNCONNECTED80));
  DFFRXLTS p40_reg_reg(.RN (n_5), .CK (clk), .D (n_95), .Q (p40_reg),
      .QN (UNCONNECTED81));
  DFFRXLTS ii_reg_reg(.RN (n_5), .CK (clk), .D (n_10), .Q (ii_reg), .QN
      (UNCONNECTED82));
endmodule

module CDR(rst, clks_in, a_rx, samp_test);
  input rst, a_rx;
```

```
  input [3:0] clks_in;
  output samp_test;
  wire rst, a_rx;
  wire [3:0] clks_in;
  wire samp_test;
  wire [3:0] a_xor_reg;
  wire [1:0] best_samp;
  wire [3:0] c_rx_upsampled;
  wire [3:0] c_rx_upsampled_4reg;
  wire [3:0] c_rx_upsampled_3reg;
  wire [3:0] c_rx_upsampled_2reg;
  wire UNCONNECTED83, UNCONNECTED84, UNCONNECTED85, UNCONNECTED86,
       UNCONNECTED87, UNCONNECTED88, UNCONNECTED89, UNCONNECTED90;
  wire UNCONNECTED91, UNCONNECTED92, UNCONNECTED93, UNCONNECTED94,
       UNCONNECTED95, UNCONNECTED96, UNCONNECTED97, UNCONNECTED98;
  wire UNCONNECTED99, UNCONNECTED100, UNCONNECTED101,
UNCONNECTED102,
       UNCONNECTED103, UNCONNECTED104, UNCONNECTED105,
UNCONNECTED106;
  wire UNCONNECTED107, UNCONNECTED108, UNCONNECTED109,
UNCONNECTED110,
       \best_samp_reg[0]_296 , \best_samp_reg[1]_297 ,
       \c_rx_upsampled_reg[0]_244 , \c_rx_upsampled_reg[1]_247 ;
  wire \c_rx_upsampled_reg[2]_250 , \c_rx_upsampled_reg[3]_253 , n_0,
       n_1, n_2, n_3, n_4, n_5;
  wire n_6, n_7, n_19;
  INVX2TS g162(.A (rst), .Y (n_0));
  OR2XLTS g144(.A (a_xor_reg[3]), .B (n_5), .Y (n_7));
  OR3XLTS g146(.A (n_4), .B (a_xor_reg[1]), .C (a_xor_reg[3]), .Y
       (n_6));
  NOR3BXLTS g147(.AN (best_samp[0]), .B (a_xor_reg[1]), .C
       (a_xor_reg[0]), .Y (n_5));
  NOR2BXLTS g148(.AN (best_samp[1]), .B (a_xor_reg[0]), .Y (n_4));
  XOR2XLTS g152(.A (c_rx_upsampled[1]), .B (c_rx_upsampled[2]), .Y
       (n_3));
  XOR2XLTS g153(.A (c_rx_upsampled[2]), .B (c_rx_upsampled[3]), .Y
       (n_2));
  XOR2XLTS g154(.A (c_rx_upsampled[1]), .B (c_rx_upsampled[0]), .Y
       (n_1));
  MX4XLTS g2(.S1 (\best_samp_reg[1]_297 ), .S0 (\best_samp_reg[0]_296
       ), .D (c_rx_upsampled_4reg[3]), .C (c_rx_upsampled_4reg[2]), .B
       (c_rx_upsampled_4reg[1]), .A (c_rx_upsampled_4reg[0]), .Y
       (n_19));
  DFFRXLTS samp_test_reg(.RN (n_0), .CK (clks_in[0]), .D (n_19), .Q
       (samp_test), .QN (UNCONNECTED83));
  DFFRXLTS \best_samp_reg_reg[1] (.RN (n_0), .CK (clks_in[0]), .D
       (best_samp[1]), .Q (\best_samp_reg[1]_297 ), .QN
       (UNCONNECTED84));
  DFFRXLTS \best_samp_reg_reg[0] (.RN (n_0), .CK (clks_in[0]), .D
       (best_samp[0]), .Q (\best_samp_reg[0]_296 ), .QN
       (UNCONNECTED85));
```

```
DFFRXLTS \c_rx_upsampled_4reg_reg[3] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled_3reg[3]), .Q (c_rx_upsampled_4reg[3]), .QN
    (UNCONNECTED86));
DFFRXLTS \c_rx_upsampled_4reg_reg[2] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled_3reg[2]), .Q (c_rx_upsampled_4reg[2]), .QN
    (UNCONNECTED87));
DFFRXLTS \c_rx_upsampled_4reg_reg[1] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled_3reg[1]), .Q (c_rx_upsampled_4reg[1]), .QN
    (UNCONNECTED88));
DFFRXLTS \c_rx_upsampled_4reg_reg[0] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled_3reg[0]), .Q (c_rx_upsampled_4reg[0]), .QN
    (UNCONNECTED89));
DFFRXLTS \c_rx_upsampled_3reg_reg[3] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled_2reg[3]), .Q (c_rx_upsampled_3reg[3]), .QN
    (UNCONNECTED90));
DFFRXLTS \c_rx_upsampled_3reg_reg[0] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled_2reg[0]), .Q (c_rx_upsampled_3reg[0]), .QN
    (UNCONNECTED91));
DFFRXLTS \c_rx_upsampled_3reg_reg[1] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled_2reg[1]), .Q (c_rx_upsampled_3reg[1]), .QN
    (UNCONNECTED92));
DFFRXLTS \c_rx_upsampled_3reg_reg[2] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled_2reg[2]), .Q (c_rx_upsampled_3reg[2]), .QN
    (UNCONNECTED93));
DFFRXLTS \c_rx_upsampled_2reg_reg[1] (.RN (n_0), .CK (clks_in[0]), .D
    (\c_rx_upsampled_reg[1]_247 ), .Q (c_rx_upsampled_2reg[1]), .QN
    (UNCONNECTED94));
DFFRXLTS \c_rx_upsampled_2reg_reg[2] (.RN (n_0), .CK (clks_in[0]), .D
    (\c_rx_upsampled_reg[2]_250 ), .Q (c_rx_upsampled_2reg[2]), .QN
    (UNCONNECTED95));
DFFRXLTS \c_rx_upsampled_2reg_reg[3] (.RN (n_0), .CK (clks_in[0]), .D
    (\c_rx_upsampled_reg[3]_253 ), .Q (c_rx_upsampled_2reg[3]), .QN
    (UNCONNECTED96));
DFFRXLTS \c_rx_upsampled_2reg_reg[0] (.RN (n_0), .CK (clks_in[0]), .D
    (\c_rx_upsampled_reg[0]_244 ), .Q (c_rx_upsampled_2reg[0]), .QN
    (UNCONNECTED97));
DFFRXLTS \c_rx_upsampled_reg_reg[0] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled[0]), .Q (\c_rx_upsampled_reg[0]_244 ), .QN
    (UNCONNECTED98));
DFFRXLTS \c_rx_upsampled_reg_reg[1] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled[1]), .Q (\c_rx_upsampled_reg[1]_247 ), .QN
    (UNCONNECTED99));
DFFRXLTS \c_rx_upsampled_reg_reg[2] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled[2]), .Q (\c_rx_upsampled_reg[2]_250 ), .QN
    (UNCONNECTED100));
DFFRXLTS \c_rx_upsampled_reg_reg[3] (.RN (n_0), .CK (clks_in[0]), .D
    (c_rx_upsampled[3]), .Q (\c_rx_upsampled_reg[3]_253 ), .QN
    (UNCONNECTED101));
DFFRXLTS \c_rx_upsampled_reg[0] (.RN (n_0), .CK (clks_in[0]), .D
    (a_rx), .Q (c_rx_upsampled[0]), .QN (UNCONNECTED102));
DFFRXLTS \c_rx_upsampled_reg[1] (.RN (n_0), .CK (clks_in[1]), .D
```

```
      (a_rx), .Q (c_rx_upsampled[1]), .QN (UNCONNECTED103));
  DFFRXLTS \c_rx_upsampled_reg[2] (.RN (n_0), .CK (clks_in[2]), .D
      (a_rx), .Q (c_rx_upsampled[2]), .QN (UNCONNECTED104));
  DFFRXLTS \c_rx_upsampled_reg[3] (.RN (n_0), .CK (clks_in[3]), .D
      (a_rx), .Q (c_rx_upsampled[3]), .QN (UNCONNECTED105));
  DFFRXLTS \best_samp_reg[1] (.RN (n_0), .CK (clks_in[0]), .D (n_6), .Q
      (best_samp[1]), .QN (UNCONNECTED106));
  DFFRXLTS \best_samp_reg[0] (.RN (n_0), .CK (clks_in[0]), .D (n_7), .Q
      (best_samp[0]), .QN (UNCONNECTED107));
  DFFRXLTS \a_xor_reg_reg[3] (.RN (n_0), .CK (clks_in[0]), .D (n_3), .Q
      (a_xor_reg[3]), .QN (UNCONNECTED108));
  DFFRXLTS \a_xor_reg_reg[1] (.RN (n_0), .CK (clks_in[0]), .D (n_1), .Q
      (a_xor_reg[1]), .QN (UNCONNECTED109));
  DFFRXLTS \a_xor_reg_reg[0] (.RN (n_0), .CK (clks_in[0]), .D (n_2), .Q
      (a_xor_reg[0]), .QN (UNCONNECTED110));
endmodule

module deserializer(rst, clks_in, a_rx, c_parallel_out, clk_out,
    disparity_d, disparity_q, c_data_valid);
  input rst, a_rx, disparity_d;
  input [3:0] clks_in;
  output [9:0] c_parallel_out;
  output clk_out, disparity_q, c_data_valid;
  wire rst, a_rx, disparity_d;
  wire [3:0] clks_in;
  wire [9:0] c_parallel_out;
  wire clk_out, disparity_q, c_data_valid;
  wire [9:0] shift_2reg;
  wire [3:0] cycle_count;
  wire [9:0] shift_reg;
  wire UNCONNECTED111, UNCONNECTED112, UNCONNECTED113,
UNCONNECTED114,
      UNCONNECTED115, UNCONNECTED116, UNCONNECTED117,
UNCONNECTED118;
  wire UNCONNECTED119, UNCONNECTED120, UNCONNECTED121,
UNCONNECTED122,
      UNCONNECTED123, UNCONNECTED124, UNCONNECTED125,
UNCONNECTED126;
  wire UNCONNECTED127, UNCONNECTED128, UNCONNECTED129,
UNCONNECTED130,
      UNCONNECTED131, UNCONNECTED132, UNCONNECTED133,
UNCONNECTED134;
  wire UNCONNECTED135, comma_detected, comma_detected_reg, n_0, n_1,
      n_2, n_3, n_4;
  wire n_5, n_6, n_7, n_8, n_9, n_10, n_11, n_12;
  wire n_13, n_14, n_15, n_16, n_17, n_18, n_19, n_20;
  wire n_21, n_22, n_23, n_24, n_25, n_26, n_28, n_29;
  wire n_30, n_31, n_32, n_33, n_34, n_36, n_38, n_40;
  wire samp_test;
  CDR CDR1(.rst (rst), .clks_in (clks_in), .a_rx (a_rx), .samp_test
      (samp_test));
```

```
NOR2XLTS g295(.A (n_28), .B (comma_detected), .Y (n_36));
NAND2X1TS g297(.A (n_34), .B (n_33), .Y (comma_detected));
NAND4BBXLTS g298(.AN (n_31), .BN (shift_2reg[7]), .C (n_30), .D
    (shift_2reg[8]), .Y (n_34));
NAND4BBXLTS g299(.AN (n_29), .BN (shift_2reg[8]), .C (n_32), .D
    (shift_2reg[7]), .Y (n_33));
NOR4XLTS g300(.A (shift_2reg[0]), .B (shift_2reg[1]), .C
    (shift_2reg[9]), .D (shift_2reg[2]), .Y (n_32));
NAND4XLTS g301(.A (shift_2reg[1]), .B (shift_2reg[2]), .C
    (shift_2reg[0]), .D (shift_2reg[9]), .Y (n_31));
NOR4XLTS g303(.A (shift_2reg[5]), .B (shift_2reg[3]), .C
    (shift_2reg[6]), .D (shift_2reg[4]), .Y (n_30));
NAND4XLTS g304(.A (shift_2reg[3]), .B (shift_2reg[4]), .C
    (shift_2reg[5]), .D (shift_2reg[6]), .Y (n_29));
NAND3BXLTS g312(.AN (n_40), .B (cycle_count[0]), .C
    (comma_detected_reg), .Y (n_28));
OR3XLTS g314(.A (cycle_count[3]), .B (cycle_count[2]), .C
    (cycle_count[1]), .Y (n_40));
INVX2TS g319(.A (rst), .Y (n_4));
DFFSX1TS \cycle_count_reg[3] (.SN (n_4), .CK (clks_in[0]), .D (n_26),
    .Q (cycle_count[3]), .QN (UNCONNECTED111));
NAND2XLTS g224(.A (n_25), .B (n_19), .Y (n_26));
XNOR2XLTS g225(.A (n_22), .B (cycle_count[3]), .Y (n_25));
NOR2BXLTS g227(.AN (n_19), .B (n_23), .Y (n_24));
ADDHXLTS g230(.A (n_7), .B (n_5), .S (n_23), .CO (n_22));
NOR2BXLTS g231(.AN (n_19), .B (n_18), .Y (n_21));
NAND2BXLTS g232(.AN (n_38), .B (n_19), .Y (n_20));
AOI2BB1XLTS g243(.A0N (n_40), .A1N (cycle_count[0]), .B0
    (comma_detected), .Y (n_19));
XNOR2XLTS g244(.A (n_38), .B (cycle_count[1]), .Y (n_18));
MX2XLTS g245(.S0 (n_0), .B (shift_2reg[8]), .A (c_parallel_out[8]),
    .Y (n_17));
MX2XLTS g246(.S0 (n_3), .B (shift_2reg[1]), .A (c_parallel_out[1]),
    .Y (n_16));
MX2XLTS g247(.S0 (n_0), .B (shift_2reg[9]), .A (c_parallel_out[9]),
    .Y (n_15));
MX2XLTS g248(.S0 (n_2), .B (shift_2reg[2]), .A (c_parallel_out[2]),
    .Y (n_14));
MX2XLTS g249(.S0 (n_2), .B (shift_2reg[3]), .A (c_parallel_out[3]),
    .Y (n_13));
MX2XLTS g250(.S0 (n_0), .B (shift_2reg[0]), .A (c_parallel_out[0]),
    .Y (n_12));
MX2XLTS g251(.S0 (n_3), .B (shift_2reg[4]), .A (c_parallel_out[4]),
    .Y (n_11));
MX2XLTS g252(.S0 (n_3), .B (shift_2reg[5]), .A (c_parallel_out[5]),
    .Y (n_10));
MX2XLTS g253(.S0 (n_0), .B (shift_2reg[6]), .A (c_parallel_out[6]),
    .Y (n_9));
MX2XLTS g254(.S0 (n_2), .B (shift_2reg[7]), .A (c_parallel_out[7]),
    .Y (n_8));
NOR2XLTS g256(.A (cycle_count[0]), .B (cycle_count[1]), .Y (n_7));
```

```
OR2XLTS g257(.A (comma_detected), .B (comma_detected_reg), .Y (n_6));
INVX1TS drc_bufs(.A (n_1), .Y (n_3));
INVX1TS drc_bufs265(.A (n_1), .Y (n_2));
INVX1TS drc_bufs270(.A (n_1), .Y (n_0));
DFFSX1TS \cycle_count_reg[0] (.SN (n_4), .CK (clks_in[0]), .D (n_20),
    .Q (cycle_count[0]), .QN (n_38));
DFFRX2TS \cycle_count_reg[2] (.RN (n_4), .CK (clks_in[0]), .D (n_24),
    .Q (cycle_count[2]), .QN (n_5));
DFFRX2TS c_data_valid_reg(.RN (n_4), .CK (clks_in[0]), .D (n_36), .Q
    (c_data_valid), .QN (n_1));
DFFRXLTS \c_parallel_out_reg[6] (.RN (n_4), .CK (clks_in[0]), .D
    (n_9), .Q (c_parallel_out[6]), .QN (UNCONNECTED112));
DFFRXLTS \c_parallel_out_reg[0] (.RN (n_4), .CK (clks_in[0]), .D
    (n_12), .Q (c_parallel_out[0]), .QN (UNCONNECTED113));
DFFRXLTS \c_parallel_out_reg[1] (.RN (n_4), .CK (clks_in[0]), .D
    (n_16), .Q (c_parallel_out[1]), .QN (UNCONNECTED114));
DFFRXLTS \c_parallel_out_reg[2] (.RN (n_4), .CK (clks_in[0]), .D
    (n_14), .Q (c_parallel_out[2]), .QN (UNCONNECTED115));
DFFRXLTS \c_parallel_out_reg[5] (.RN (n_4), .CK (clks_in[0]), .D
    (n_10), .Q (c_parallel_out[5]), .QN (UNCONNECTED116));
DFFRXLTS \c_parallel_out_reg[9] (.RN (n_4), .CK (clks_in[0]), .D
    (n_15), .Q (c_parallel_out[9]), .QN (UNCONNECTED117));
DFFRXLTS \c_parallel_out_reg[3] (.RN (n_4), .CK (clks_in[0]), .D
    (n_13), .Q (c_parallel_out[3]), .QN (UNCONNECTED118));
DFFRXLTS disparity_q_reg(.RN (n_4), .CK (clks_in[0]), .D
    (disparity_d), .Q (disparity_q), .QN (UNCONNECTED119));
DFFRXLTS \c_parallel_out_reg[4] (.RN (n_4), .CK (clks_in[0]), .D
    (n_11), .Q (c_parallel_out[4]), .QN (UNCONNECTED120));
DFFRXLTS \c_parallel_out_reg[7] (.RN (n_4), .CK (clks_in[0]), .D
    (n_8), .Q (c_parallel_out[7]), .QN (UNCONNECTED121));
DFFRXLTS \c_parallel_out_reg[8] (.RN (n_4), .CK (clks_in[0]), .D
    (n_17), .Q (c_parallel_out[8]), .QN (UNCONNECTED122));
DFFRXLTS \shift_2reg_reg[9] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[8]), .Q (shift_2reg[9]), .QN (UNCONNECTED123));
DFFRXLTS \shift_reg_reg[9] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[7]), .Q (shift_2reg[8]), .QN (UNCONNECTED124));
DFFRXLTS \shift_reg_reg[8] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[6]), .Q (shift_2reg[7]), .QN (UNCONNECTED125));
DFFRXLTS \shift_reg_reg[7] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[5]), .Q (shift_2reg[6]), .QN (UNCONNECTED126));
DFFRXLTS \shift_reg_reg[6] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[4]), .Q (shift_2reg[5]), .QN (UNCONNECTED127));
DFFRXLTS \shift_reg_reg[5] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[3]), .Q (shift_2reg[4]), .QN (UNCONNECTED128));
DFFRXLTS \shift_reg_reg[4] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[2]), .Q (shift_2reg[3]), .QN (UNCONNECTED129));
DFFRXLTS \shift_reg_reg[3] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[1]), .Q (shift_2reg[2]), .QN (UNCONNECTED130));
DFFRXLTS \shift_reg_reg[2] (.RN (n_4), .CK (clks_in[0]), .D
    (shift_2reg[0]), .Q (shift_2reg[1]), .QN (UNCONNECTED131));
DFFRXLTS \shift_reg_reg[1] (.RN (n_4), .CK (clks_in[0]), .D
```

```
      (shift_reg[0]), .Q (shift_2reg[0]), .QN (UNCONNECTED132));
  DFFRXLTS \shift_reg_reg[0] (.RN (n_4), .CK (clks_in[0]), .D
      (samp_test), .Q (shift_reg[0]), .QN (UNCONNECTED133));
  DFFRXLTS \cycle_count_reg[1] (.RN (n_4), .CK (clks_in[0]), .D (n_21),
      .Q (cycle_count[1]), .QN (UNCONNECTED134));
  DFFRXLTS comma_detected_reg_reg(.RN (n_4), .CK (clks_in[0]), .D
      (n_6), .Q (comma_detected_reg), .QN (UNCONNECTED135));
endmodule

module digitalRx_wrapper(VDD, VSS, DVDD, DVSS, rst, clk, a_rx,
    a_rx_diff, alfa, dataout, dispout, code_err, disp_err,
    data_valid_pipe, clk_4f, ko);
  input VDD, VSS, DVDD, DVSS, rst, clk, a_rx, a_rx_diff, alfa;
  output [7:0] dataout;
  output dispout, code_err, disp_err, data_valid_pipe, clk_4f, ko;
  wire VDD, VSS, DVDD, DVSS, rst, clk, a_rx, a_rx_diff, alfa;
  wire [7:0] dataout;
  wire dispout, code_err, disp_err, data_valid_pipe, clk_4f, ko;
  wire [3:0] clks_in;
  wire [9:0] c_parallel_out;
  wire [7:0] dataout_w;
  wire a_rx_n, a_rx_n_w, a_rx_w, beta, c_data_valid, clk_out, clk_w,
      code_err_w;
  wire data_valid_pipe_w, disp_err_w, disparity_d, disparity_d_w,
      disparity_q, dispout_w, ko_w, n_9;
  wire n_10, rst_w;
  clock_divider clock_divider1(rst_w, clk_w, clks_in);
  decodePipe decode1(rst_w, clks_in[0], c_data_valid, c_parallel_out,
      disparity_q, dataout_w, data_valid_pipe_w, dispout_w,
      code_err_w, disp_err_w, ko_w);
  deserializer deserializer1(rst_w, clks_in, a_rx_w, c_parallel_out,
      clk_out, disparity_d_w, disparity_q, c_data_valid);
  CLKBUFX2TS cdn_loop_breaker(.A (VSS), .Y (n_10));
  CLKBUFX2TS cdn_loop_breaker4(.A (VSS), .Y (n_9));
  PDVDD pad_DVDD(.DVDD (DVDD));
  PDVSS pad_DVSS(.DVSS (DVSS));
  PVDD pad_VDD(.VDD (VDD));
  PVSS pad_VSS(.VSS (VSS));
  PIC pad_a_rx(.P (a_rx), .IE (VDD), .Y (a_rx_w));
  PIC pad_a_rx_n(.P (a_rx_n), .IE (VDD), .Y (a_rx_n_w));
  PIC pad_alfa(.P (alfa), .IE (n_10), .Y (VSS));
  PIC pad_beta(.P (beta), .IE (n_9), .Y (VSS));
  PIC pad_clk(.P (clk), .IE (VDD), .Y (clk_w));
  POC4C pad_clk_4f(.A (clks_in[0]), .P (clk_4f));
  POC4C pad_code_err(.A (code_err_w), .P (code_err));
  POC4C pad_dataout0(.A (dataout_w[0]), .P (dataout[0]));
  POC4C pad_dataout1(.A (dataout_w[1]), .P (dataout[1]));
  POC4C pad_dataout2(.A (dataout_w[2]), .P (dataout[2]));
  POC4C pad_dataout3(.A (dataout_w[3]), .P (dataout[3]));
  POC4C pad_dataout4(.A (dataout_w[4]), .P (dataout[4]));
  POC4C pad_dataout5(.A (dataout_w[5]), .P (dataout[5]));
```

```
  POC4C pad_dataout6(.A (dataout_w[6]), .P (dataout[6]));
  POC4C pad_dataout7(.A (dataout_w[7]), .P (dataout[7]));
  POC4C pad_disp_err(.A (disp_err_w), .P (disp_err));
  PIC pad_disparity_d(.P (disparity_d), .IE (VDD), .Y (disparity_d_w));
  POC4C pad_dispout(.A (dispout_w), .P (dispout));
  POC4C pad_ko(.A (ko_w), .P (ko));
  PIC pad_rst(.P (rst), .IE (VDD), .Y (rst_w));
endmodule
```

c. Logic synthesis script

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

if {[file exists /proc/cpuinfo]} {
  sh grep "model name" /proc/cpuinfo
  sh grep "cpu MHz"    /proc/cpuinfo
}
puts "Hostname : [info hostname]"

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


set DESIGN digitalRx_wrapper
set SYN_EFF medium
#set MAP_EFF medium
set MAP_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH outputs_${DATE}
set _REPORTS_PATH reports_${DATE}
set _LOG_PATH logs_${DATE}

# Variable to specify the technology .lib file name
set timing_library {scx3_cmos8rf_lpvt_tt_1p2v_25c.lib
iogpil_cmrf8sf_rvt_tt_1p2v_2p5v_25c.lib}

set my_lef_library {/opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-
x/lef/ibm13_8lm_2thick_3rf_tech.lef /opt/libs/ARM/IB03IG502-FB-00000-r0p0-
00rel0/aci/io/lef/iogpil_cmrf8sf_rvt_M2_3_3.lef /opt/libs/ARM/IB03LB501-FB-00000-r0p0-
00rel0/aci/sc-x/lef/ibm13rflpvt_macros.lef}

set_attribute lib_search_path {/opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-
x/synopsys /opt/libs/ARM/IB03IG502-FB-00000-r0p0-00rel0/aci/io/synopsys/} /
set_attribute script_search_path {..} /
set_attribute hdl_search_path {..} /
```

```
##Uncomment and specify machine names to enable super-threading.
##set_attribute super_thread_servers {<machine names>} /

##Default undriven/unconnected setting is 'none'.
##set_attribute hdl_unconnected_input_port_value 0 | 1 | x | none /
##set_attribute hdl_undriven_output_port_value   0 | 1 | x | none /
##set_attribute hdl_undriven_signal_value        0 | 1 | x | none /


##set_attribute wireload_mode <value> /
set_attribute information_level 9 /

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

set_attribute library $timing_library
set_attribute lef_library $my_lef_library /
#set_attribute cap_table_file <file> /
set_attribute auto_ungroup none /


##set_attribute congestion_effort <low|medium|high> /
##generates <signal>_reg[<bit_width>] format
#set_attribute hdl_array_naming_style %s\[%d\] /
#################


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


read_hdl -v2001 { ../verilogfiles/adderA.v \
../verilogfiles/adderB.v \
../verilogfiles/CDR.v \
../verilogfiles/decodePipe.v \
../verilogfiles/deserializer.v \
../verilogfiles/digitalRx_wrapper.v \
../verilogfiles/inputBuffer.v \
../verilogfiles/clock_divider.v \
}
elaborate $DESIGN
puts "Runtime & Memory after 'read_hdl'"
timestat Elaboration


check_design -unresolved

#####################################################################
```

```
## Constraints Setup
################################################################################

read_sdc { ../digitalRx.sdc}
puts "The number of exceptions is [llength [find /designs/$DESIGN -exception *]]"


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

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

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

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


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

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

if {[llength [all::all_seqs]] > 0} {
  define_cost_group -name I2C -design $DESIGN
  define_cost_group -name C2O -design $DESIGN
  define_cost_group -name C2C -design $DESIGN
  path_group -from [all::all_seqs] -to [all::all_seqs] -group C2C -name C2C
  path_group -from [all::all_seqs] -to [all::all_outs] -group C2O -name C2O
  path_group -from [all::all_inps]  -to [all::all_seqs] -group I2C -name I2C
}

define_cost_group -name I2O -design $DESIGN
path_group -from [all::all_inps]  -to [all::all_outs] -group I2O -name I2O
foreach cg [find / -cost_group *] {
  report timing -cost_group [list $cg] >> $_REPORTS_PATH/${DESIGN}_pretim.rpt
}
################################################################################
##############
```

```
## Leakage/Dynamic power/Clock Gating setup.
################################################################################
#############


#set_attribute lp_clock_gating_cell [find /lib* -libcell <cg_libcell_name>] "/designs/$DESIGN"
#set_attribute lp_power_unit {uW}
#set_attribute lp_pso_aware_estimation true
#set_attribute leakage_power_effort high
#set_attribute max_leakage_power 100 "/designs/$DESIGN"
#set_attribute lp_power_optimization_weight 0.5 "/designs/$DESIGN"
#set_attribute max_dynamic_power 100 "/designs/$DESIGN"
## read_tcf <TCF file name>
## read_saif <SAIF file name>
## read_vcd <VCD file name>



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



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


synthesize -to_generic -eff $SYN_EFF
puts "Runtime & Memory after 'synthesize -to_generic'"
timestat GENERIC

report power > $_REPORTS_PATH/${DESIGN}_generic_power.rpt
#write_design -encounter -gzip -basename ${_OUTPUTS_PATH}/generic/${DESIGN}
generate_reports -outdir $_REPORTS_PATH -tag generic
## syntax : generate_reports -outdir <out dir> -tag <tag> [-encounter]
## syntax : summary_table -outdir <out dir>
#generate_reports -outdir $_REPORTS_PATH -tag generic
summary_table -outdir $_REPORTS_PATH



################################################################################
############################
## Synthesizing to gates
```

```
##############################################################################
############################

synthesize -to_mapped -eff $MAP_EFF -no_incr
puts "Runtime & Memory after 'synthesize -to_map -no_incr'"
timestat MAPPED

report power > $_REPORTS_PATH/${DESIGN}_map_power.rpt
foreach cg [find / -cost_group *] {
  report timing -cost_group [list $cg] > $_REPORTS_PATH/${DESIGN}_[basename
$cg]_post_map.rpt
}
generate_reports -outdir $_REPORTS_PATH -tag map
summary_table -outdir $_REPORTS_PATH
write_design -encounter -gzip -basename ${_OUTPUTS_PATH}/map/${DESIGN}


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

## ungroup -threshold <value>

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

## Uncomment to remove assigns & insert tiehilo cells during Incremental synthesis
##set_attribute remove_assigns true /
##set_remove_assign_options -buffer_or_inverter <libcell> -design <design|subdesign>
##set_attribute use_tiehilo_for_const <none|duplicate|unique> /
synthesize -to_mapped -eff $MAP_EFF -incr
report power > $_REPORTS_PATH/${DESIGN}_incremental_power.rpt
generate_reports -outdir $_REPORTS_PATH -tag incremental
summary_table -outdir $_REPORTS_PATH

puts "Runtime & Memory after incremental synthesis"
timestat INCREMENTAL

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


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

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


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


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

report clock_gating > $_REPORTS_PATH/${DESIGN}_clockgating.rpt
report power -depth 0 > $_REPORTS_PATH/${DESIGN}_power.rpt
report gates -power > $_REPORTS_PATH/${DESIGN}_gates_power.rpt

##report qor > $_REPORTS_PATH/${DESIGN}_qor.rpt
report area > $_REPORTS_PATH/${DESIGN}_area.rpt
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_incr.rpt
report messages > $_REPORTS_PATH/${DESIGN}_messages.rpt
write_design -basename ${_OUTPUTS_PATH}/${DESIGN}_m
write_hdl  > ${_OUTPUTS_PATH}/${DESIGN}_m.v
## write_script > ${_OUTPUTS_PATH}/${DESIGN}_m.script
write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_m.sdc


###################################
### write_do_lec
###################################


write_do_lec -golden_design ${_OUTPUTS_PATH}/${DESIGN}_intermediate.v -
revised_design ${_OUTPUTS_PATH}/${DESIGN}_m.v -logfile
${_LOG_PATH}/intermediate2final.lec.log > ${_OUTPUTS_PATH}/intermediate2final.lec.do
##Uncomment if the RTL is to be compared with the final netlist..
write_do_lec -revised_design ${_OUTPUTS_PATH}/${DESIGN}_m.v -logfile
${_LOG_PATH}/rtl2final.lec.log > ${_OUTPUTS_PATH}/rtl2final.lec.do

puts "Final Runtime & Memory."
timestat FINAL
puts "============================"
puts "Synthesis Finished ....."
puts "============================"

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

##quit
```

d. Constraint file

```
set_time_unit -picoseconds
set_load_unit -femtofarads

current_design digitalRx

set ref_clk_period 2000

# Clock definition
#define_clock -name 1000MHz_CLK -period 1000 [get_ports clk]
#define_clock -name 1000MHz_CLK -period 4000 [clock_ports]
#define_clock -name 1000MHz_CLK -period 1000 [get_ports clock clocks_in[0] clocks_in[1]
clocks_in[2] clocks_in[3]]
#define_clock -name 250MHz_CLK -period 4000 [get_ports clocks_in[0]]
create_clock -name 1000MHz_CLK -period $ref_clk_period [get_ports clk]

create_generated_clock \
  -name clock_ph0 \
  -source clock_divider1/clk \
  -divide_by 4 [get_pins clock_divider1/clks_out[0]]

create_generated_clock \
  -name clock_ph1 \
  -source clock_divider1/clks_out[0] \
  [get_pins clock_divider1/clks_out[1]] \
  -edges {1 2 3} \
  -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock \
  -name clock_ph2 \
  -source clock_divider1/clks_out[0] \
  [get_pins clock_divider1/clks_out[2]] \
  -edges {1 2 3} \
  -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

create_generated_clock \
  -name clock_ph3 \
  -source clock_divider1/clks_out[0] \
  [get_pins clock_divider1/clks_out[3]] \
  -edges {1 2 3} \
  -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

# 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 } 1000MHz_CLK
#set_attribute slew { 28 28 28 28 } 250MHz_CLK

# network clock latency
set_attribute clock_network_late_latency 100 1000MHz_CLK
```

```
set_attribute clock_network_early_latency 90 1000MHz_CLK
#set_attribute clock_network_late_latency 100 250MHz_CLK
#set_attribute clock_network_early_latency 90 250MHz_CLK
# source clock latency
set_attribute clock_source_late_latency 50 1000MHz_CLK
set_attribute clock_source_early_latency 40 1000MHz_CLK
#set_attribute clock_source_late_latency 50 250MHz_CLK
#set_attribute clock_source_early_latency 40 250MHz_CLK

# clock skew
set_attribute clock_setup_uncertainty {17 10} 1000MHz_CLK
set_attribute clock_hold_uncertainty {13 5} 1000MHz_CLK
#set_attribute clock_setup_uncertainty {17 10} 250MHz_CLK
#set_attribute clock_hold_uncertainty {13 5} 250MHz_CLK


# 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 1000MHz_CLK] -input 200 -name IDelay [find /des* -port
ports_in/*]
#external_delay -clock [find / -clock 250MHz_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 1000MHz_CLK] -output 200 -name ODelay [find /des* -port
ports_out/*]
#external_delay -clock [find / -clock 250MHz_CLK] -output 200 -name ODelay [find /des* -port
ports_out/*]

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


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

# 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/digitalRx/ports_out/*

# Setting maximum value of fanout
set_attribute max_fanout 4 /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.
```

C. Physical Synthesis

a. Automation script

```
## 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 ../digitalRx_frame_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_wrapper.view
```

```
set init_pwr_net {VDD DVDD}
set init_verilog ../digitalRx_wrapper_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 623.4 550.4 16.8 16.8 16.8 16.8
floorPlan -dieSizeByIoHeight max -site IBM13SITE -s 150 150 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
#addRing -skip_via_on_wire_shape Noshape -skip_via_on_pin Standardcell -center 1 -
stacked_via_top_layer MA -type core_rings -jog_distance 0.2 -threshold 0.2 -nets {VDD VSS}
-follow core -stacked_via_bottom_layer M1 -layer {bottom M1 top M1 right M2 left M2} -width
1 -spacing 0.5 -offset 0.2

## 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 2 -
skip_via_on_pin Standardcell -stacked_via_top_layer MA -padcore_ring_top_layer_limit M3 -
spacing 5 -xleft_offset 130 -xright_offset 130 -merge_stripes_value 0.2 -layer M2 -
block_ring_bottom_layer_limit M1 -width 4 -nets {VDD VSS} -stacked_via_bottom_layer M1
#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 2 -
skip_via_on_pin Standardcell -stacked_via_top_layer MA -padcore_ring_top_layer_limit M3 -
spacing .28 -xleft_offset 150 -xright_offset 150 -merge_stripes_value 0.2 -layer M2 -
block_ring_bottom_layer_limit M1 -width .56 -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 3
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_digitalRx.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 3
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 digitalRx_verif.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/digitalRx_wrapper.rpt


# save design

saveDesign digitalRx_wrapper_script.enc
```

b. Analysis 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
{../digitalRx_wrapper_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}
```

LIST OF FILES

Stage	Description	Path
RTL design	Verilog files required for the RTL synthesis	/home/rvillegas/Cadence_Designs/RC/8RF_ARM/digitalRx_beta2_wrapper/verilogfiles/*
Logic synthesis	Constraint file	/home/rvillegas/Cadence_Designs/RC/8RF_ARM/digitalRx_beta2_wrapper/digitalRx.sdc
	Automation script	/home/rvillegas/Cadence_Designs/RC/8RF_ARM/digitalRx_beta2_wrapper/gnr_elab.tcl
	Report files	/home/rvillegas/Cadence_Designs/RC/8RF_ARM/digitalRx_beta2_wrapper/TIEMPO_FIXED/reports_Jul05-19:44:03
	Output netlist	/home/rvillegas/Cadence_Designs/EDI/8RF_ARM/digitalRx_beta2_wrapper/digitalRx_wrapper_m.v
	Output constraint file	/home/rvillegas/Cadence_Designs/EDI/8RF_ARM/digitalRx_beta2_wrapper/digitalRx_wrapper_m.sdc
Gate level simulation	Standard cells models	/opt/libs/ARM/IB03LB501-FB-00000-r0p0-00rel0/aci/sc-x/verilog/
Physical Synthesis	Automation script	/home/rvillegas/Cadence_Designs/EDI/8RF_ARM/digitalRx_beta2_wrapper/Full_Synthesis_EDI_digitalRxWrapper.tcl
	IO configuration file	/home/rvillegas/Cadence_Designs/EDI/8RF_ARM/digitalRx_beta2_wrapper/digitalRx_frame_arm.ioc
	Analysis View	/home/rvillegas/Cadence_Designs/EDI/8RF_ARM/digitalRx_beta2_wrapper/Typ_WC_wrapper.view
	Encounter DB	/home/rvillegas/Cadence_Designs/EDI/8RF_ARM/digitalRx_beta2_wrapper/LATEST/digitalRx_wrapper_gds.gds
Virtuoso	GDS output file	Library: digLatest Cell: digitalRx_wrapper View: Layout