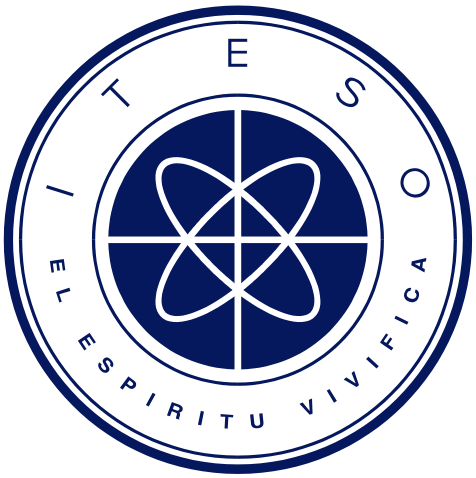
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

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



Diseño de módulo Serializador de un sistema SerDes para protocolo de comunicación PCI Express

Tesina para obtener el grado de:

Especialista en diseño de sistemas en chip

Presenta

Graciela Citlali González Morales

Directores de proyecto: Dr. Víctor Avendaño Fernández, Mtro. Alexandro Girón Allende, Mtro. Cuauhtémoc Rafael Aguilera Galicia

Guadalajara, Jalisco, Diciembre 2015

Agradecimientos

El autor desea agradecer a los tutores el tiempo y dedicación en la realización de este proyecto. Alex, Víctor y Cuauhtémoc gracias por su constancia y compromiso pero sobre todo por compartir su experiencia profesional y académica.

Un especial reconocimiento a Esteban por tu gran labor como coordinador de este gran proyecto, gracias por tu dedicación y por tu entrega.

Un agradecimiento a mis padres por brindarme una educación de bases sólidas y por las oportunidades en la vida. Gracias porque en la distancia siempre están conmigo.

Gracias a mi compañero en la escuela y en la vida, José Manuel por siempre impulsarme a mejorar. Gracias por su paciencia y por su amor.

Un especial agradecimiento a PNPC-CONACYT por el apoyo financiero a los proyectos académicos. Gracias por creer en los investigadores mexicanos y en la calidad de los programas.

Resumen

This document presents the process followed for designing of a VLSI circuit, specifically the Serializer module of a SerDes used in the PCI Express protocol communication. It is shown a proposed architecture, the implementation of a RTL model in Verilog, the logic synthesis, the physical synthesis and the generation of a GDSII file for manufacturing. The Serializer of a SerDes for PCIe obtains a parallel 8 bits datum and returns a serialized 10 bit encoded datum using 8b/10b encoding. The technology used for this project is the cmrf_7sf from IBM.

Tabla de contenidos

Lista de Figuras

Lista de tablas

Introducción

Este documento presenta procedimientos y resultados obtenidos en el desarrollo del Vehículo de Pruebas 1. El Vehículo de Pruebas 1 es un proyecto cuyo objetivo final es generar en el ITESO el conocimiento y la experiencia en el diseño, la implementación y la fabricación de sistemas en chip usando técnicas de diseño de circuitos de integración de muy alta escala (*VLSI* por sus siglas en inglés).

El Vehículo de Pruebas 1, es un sistema SerDes para protocolo de comunicación PCI Express generación 1. En este documento se mostrarán los pasos explorados del flujo de diseño de circuitos integrados *VLSI*. Se partirá desde su descripción en hardware hasta la generación de archivos GDSII para fabricación.

El sistema SerDes está conformado por cuatro subsistemas, el Serializador y Deserializador, el receptor y el transmisor. Para aplicaciones de alta velocidad como el protocolo PCI Express es necesario además el uso de un codificador y decodificador 8b/10b que permita una componente constante de DC.

El Capítulo 1 de este documento presenta un marco teórico para introducir al lector al sistema SerDes, su funcionamiento y principales características. También se explicará de forma concisa el protocolo PCI Express y la codificación 8b/10b.

El Capítulo 2 se enfoca en mostrar al lector el flujo de diseño del módulo serializador. Se muestra la arquitectura propuesta del módulo, se presenta también el código *RTL* implementado en Verilog, la síntesis lógica y la integración del módulo al sistema SerDes.

El Capítulo 3 muestra los bancos de prueba diseñados para la verificación del modelo propuesto en el Capítulo 2. Se mostrarán los resultados de la verificación funcional y de la verificación de equivalencia lógica *(Logic Equivalence Check)*. Se mostrará también los resultados de las simulaciones del módulo a nivel compuerta.

Finalmente el Capítulo 4 muestra el procedimiento para la realizar la síntesis física del sistema SerDes así como los resultados obtenidos.

CAPITULO 1: MARCO TEORICO DEL SISTEMA SERDES

1.1. *Descripción del sistema SerDes*

Un sistema SerDes (Serializer/Deserializer) es un bloque básico en la comunicación de los microprocesadores, se encarga de la conversión de datos serie a paralelos y paralelos a serie.

Su funcionamiento permite compartir y convertir datos entre interfaces seriales y paralelas. Está compuesto por cuatro bloques: un serializador y un Deserializador, un transmisor y un receptor analógico. La figura 1, es una representación de los bloques funcionales que integran el sistema SerDes “Vehículo de Pruebas 1”.

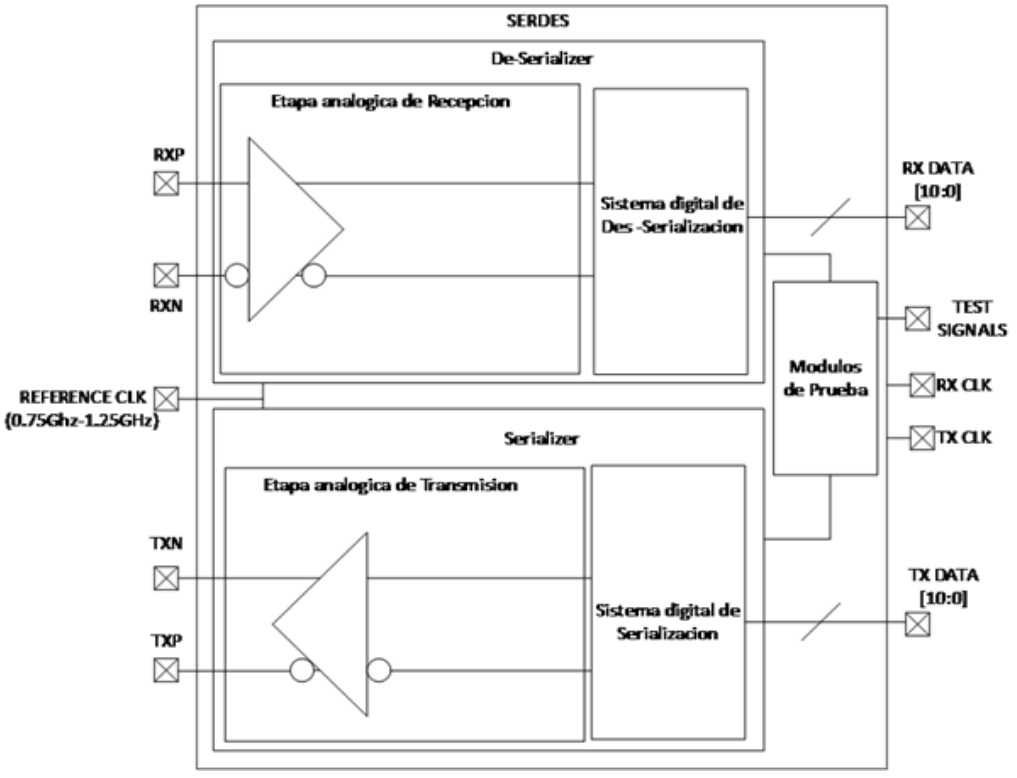


Figura 1. Bloques funcionales del Sistema SerDes

1.2. *Módulos del Sistema SerDes*

1.2.1. *Deserializador digital*

La etapa de **deserialización** tiene como objetivo convertir datos seriales a paralelos y alinear los datos recibidos con un reloj del sistema. Está compuesto de los siguientes bloques funcionales:

- ✓ CDR: *Clock and Data Recovery*. Dado que los datos recibidos no vienen acompañados de un reloj, es necesario recuperar o generar un reloj para poder muestrear los datos recibidos. El reloj de muestreo se genera a partir de un reloj de referencia cuya fase se alinea a las transiciones de los datos recibidos, a esto se le llama Clock Recovery. Al muestreo de los datos usando el reloj recuperado se le denomina Clock Data Recovery. Este módulo previene errores en la recepción de los datos debido a sobremuestro o submuestreo.

- ✓ SIPO (*Serial Input Parallel Output)*. Entrada serial, salida paralela. Módulo encargado de convertir los datos seriales recuperados por el CDR a un dato paralelo.

- ✓ Decodificador: Encargado de la decodificación de los símbolos de 10 bits a un dato de 8 bits. En la figura 1, el decodificador es un módulo interno del sistema digital del deserializador.

1.2.2. *Serializador digital*

La etapa de **serialización** tiene como objetivo convertir un bus de datos paralelos en datos seriales.

- ✓ PISO (*Parallel Input Serial Output). Entrada paralela, salida serial.* Módulo encargado de la conversión de datos paralelos a datos seriales.

- ✓ Codificador: Realiza la codificación del dato de 8 bits a símbolos de 10 bits. En la figura 1, el codificador es un módulo interno en del sistema digital del serializador.

1.2.3. *Deserializador y serializador analógicos*

La etapa analógica del Serializador tiene como objetivo compensar la atenuación en amplitud experimentada por los datos seriales al ser transmitidos a través del canal de comunicación.

La etapa analógica del Deserializador tiene como objetivo adecuar la señal digital diferencial recibida para eliminar o reducir el ruido de la señal, de tal forma que la señal generada pueda ser procesada adecuadamente por el circuito digital.

1.2.4. *Estructura y operación de la codificación 8b/10b*

La codificación 8b/10b es un código que se desarrolló en 1983 por Al Widmer y Peter Franaszek en el IBM Journal of Research and Development [1]. Actualmente es usada en Infiniband, Gigabit Ethernet, FiberChannel y la interfaz XAUI a 10 Gigabits Ethernet entre otros protocolos de comunicación.

De manera general el codificador convierte palabras de 8 bits a símbolos de 10 bits. Los 5 bits menos significativos los codifica en un grupo de 6 bits y los 3 bits más significativos los codifica en un grupo de 4 bits, los cuales se concatenan en un símbolo de datos de 10 bits. El protocolo 8b/10b incluye 12 caracteres especiales, los cuales son usados como delimitadores de los paquetes enviados. Algunos de ellos reciben el nombre de comas y son usados para marcar el inicio y/o fin de un paquete de datos enviado de manera serial. La figura 2 muestra gráficamente el funcionamiento de la codificación.

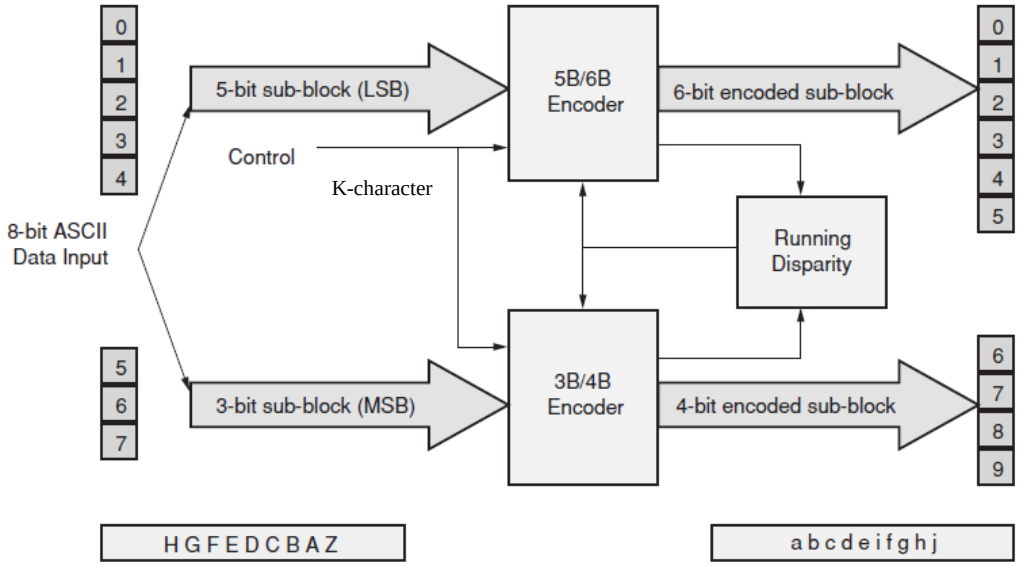


Figura 2. Codificación 8b/10b [2]

“La disparidad en cualquier bloque se define como la diferencia de unos y ceros. La disparidad positiva y negativa se refiere al exceso de 1s, o de 0s, respectivamente. El esquema de codificación garantiza que la disparidad de un símbolo codificado sea siempre 0 (5 unos y 5 ceros), +2 (6 unos y 4 ceros) ó -2 (4 unos y 6 ceros). La disparidad actual (running disparity) es un registro de la disparidad de los símbolos previamente codificados.” [3]

La tabla 1 muestra algunos valores de la codificación 8b/10b, la primera columna es el nombre del dato recibido. Los nombres de los símbolos de datos comúnmente son representados como D.x.y donde “x” está en el rango de 0–31 y “y” en el rango de 0–7. La segunda columna de la tabla 1 es la representación en binario a 8 bits de los símbolos de datos, donde el bit menos significativo es A y el más significativo es H. En la tercera y cuarta columna se muestra el dato codificado a 10 bits que depende de la disparidad actual (RD- : disparidad negativa, RD+: disparidad positiva). El dato codificado es enviado del bit menos significativo al más significativo (a b c d e i f g h j).

Los caracteres especiales son representados con el nombre *K.x.y* . Donde “x” puede tomar los valores 23, 27, 28, 29 y 30 y “y” está en el rango de 0–7. Estos símbolos son denominados especiales debido a que la secuencia de sus dígitos no se puede confundir con otras tramas de datos.

Data Byte Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
D0.0	000 00000	100111 0100	011000 1011
D1.0	000 00001	011101 0100	100010 1011
D2.0	000 00010	101101 0100	010010 1011
D3.0	000 00011	110001 1011	110001 0100
D4.0	000 00100	110101 0100	001010 1011
D5.0	000 00101	101001 1011	101001 0100
D6.0	000 00110	011001 1011	011001 0100
D7.0	000 00111	111000 1011	000111 0100
D8.0	000 01000	111001 0100	000110 1011
D9.0	000 01001	100101 1011	100101 0100

Tabla 1. Ejemplos de símbolos de datos válidos de la codificación 8b/10b [2]

En la Tabla 2, el símbolo *K28.7* recibe este nombre porque al dividirlo en 3 bits y 5 bits su conversión a decimal es 7 para los bits más significativos y 28 para los menos significativos. Los dos símbolos posibles para el dato codificado depende de la disparidad actual si es negativa (RD-) el dato codificado será *001111 1000*. Si la disparidad actual es positiva (RD+) el dato será *110000 0111.*

Special Code Name	Bits HGF EDCBA	Current RD – abcdei fghj	Current RD + abcdei fghj
K28.0	000 11100	001111 0100	110000 1011
K28.1	001 11100	001111 1001	110000 0110
K28.2	010 11100	001111 0101	110000 1010
K28.3	011 11100	001111 0011	110000 1100
K28.4	100 11100	001111 0010	110000 1101
K28.5	101 11100	001111 1010	110000 0101
K28.6	110 11100	001111 0110	110000 1001
K28.7[(1)]	111 11100	001111 1000	110000 0111
K23.7	111 10111	111010 1000	000101 0111
K27.7	111 11011	110110 1000	001001 0111
K29.7	111 11101	101110 1000	010001 0111
K30.7	111 11110	011110 1000	100001 0111

Tabla 2. Caracteres especiales de la codificación 8b/10b [2]

1.3. *Descripción del protocolo PCIe*

“PCI Express es la tercera generación de buses de I/O de alto rendimiento usado para interconectar dispositivos periféricos en aplicaciones computacionales y de comunicación. La primera generación de buses incluye ISA, EISA, VESA, and Miro

Channnel buses, mientras que la segunda generación incluye PCI, AGP, and PCI-X. PCI Express es un bus de interconexión que tiene muchas aplicaciones multiplataforma; móviles, escritorio, estaciones de trabajo, servidores, etc.” [4]

La velocidad de trasmisión/recepción para PCI Express Gen.1 por carril por dirección es 2.5 Gbits/s. La Tabla 3 muestra las distintas versiones del protocolo y las diferencias de ancho de banda.

PCI Express version	Line code	Transfer rate[a]	Bandwidth	
			Per lane[a]	In a ×16 (16-lane) slot[a]
1.0	8b/10b	2.5 GT/s	2 Gbit/s (250 MB/s)	32 Gbit/s (4 GB/s)
2.0	8b/10b	5 GT/s	4 Gbit/s (500 MB/s)	64 Gbit/s (8 GB/s)
3.0	128b/130b	8 GT/s	7.877 Gbit/s (984.6 MB/s)	126.032 Gbit/s (15.754 GB/s)
4.0	128b/130b	16 GT/s	15.754 Gbit/s (1969.2 MB/s)	252.064 Gbit/s (31.508 GB/s)

Tabla 3. Ancho de banda para versiones de PCI Express [5]

El protocolo PCIe implementa una comunicación dual-simplex lo que significa que los datos son transmitidos y recibidos en las líneas de transmisión y recepción al mismo tiempo. La tabla 4 muestra el ancho de banda de PCI Express para distintos números de carriles.

PCI Express Link Width	x1	x2	x4	x8	x12	x16	x32
Aggregate Bandwidth (GBytes/sec)	0.5	1	2	4	6	8	16

Tabla 4. Ancho de banda agregado para PCI Express [1]

1.4. *Especificaciones de diseño a nivel sistema del SerDes*

La tabla 5, muestra las especificaciones de diseño consideradas para este proyecto. Se definió que el vehículo de pruebas 1, sistema SerDes, tendría solo una línea de transmisión y recepción. La tecnología de fabricación sería de 180nm. El tamaño máximo del chip sería el aprobado por el convenio con MOSIS, 1.5mm x 1.5mm.

Especificaciones	
No. De carriles	Un carril de transmisión/recepción
Velocidad de transmisión	1.5 Gbits/s
Tecnología	180nm cmrf_7sf de IBM.
Dimensiones	1.5mm x1.5mm.
Voltaje	1.8V.
No. Pines del chip	40 pines

Tabla 5. Especificaciones de diseño

La tabla 6 muestra las principales señales de entrada y salida del vehículo de prueba 1, SerDes.

# Pin	Señal	Descripción
1	RXP	Terminal Positiva de Receptor
2	RXN	Terminal Negativa de Receptor
3	TXN	Terminal Positiva de Transmisor
4	TXP	Terminal Negativa de Transmisor
5	REF CLK	Señal de reloj de Referencia
6	RX DATA	Reloj de la señal de datos en el receptor
7	TEST SIGNAL	Señales para pruebas de funcionamiento
8	RX CLK	Reloj del Receptor
9	TX CLK	Reloj del Transmisor
10	TX DATA	Reloj de la señal de datos en el transmisor

Tabla 6. Entradas y salidas para SerDes

CAPITULO 2: DISEÑO DIGITAL DEL MÓDULO SERIALIZADOR

2.1. Objetivos del proyecto

El objetivo general de este proyecto es diseñar el módulo Serializador del sistema SerDes.

Los objetivos específicos del proyecto se enumeran a continuación:

1. Proponer una arquitectura para el módulo serializador.
2. Generar código RTL sintetizable usando Verilog.
3. Elaborar los bancos de prueba necesarios para la validación funcional.
4. Realizar la síntesis lógica del módulo.
5. Realizar la verificación de equivalencia lógica del módulo.
6. Realizar simulación a nivel compuerta.
7. Integrar el módulo Serializador a sistema SerDes.
8. Realizar la síntesis física.
9. Realizar verificación física.
10. Generar tapeout del sistema.
11. Fabricar el circuito integrado.

2.2. Alcance del proyecto.

A continuación se mencionan los puntos que delimitan el alcance de este proyecto:

1. Se recorrerá el flujo de diseño digital para el módulo Serializador.
2. El área del sistema SerDes está restringida al tamaño autorizado por MOSIS para proyectos de universidades. El diseño no debe exceder 1.5 mm x 1.5 mm.
3. El ITESO en conjunto con MOSIS serán los responsables de gestionar la fabricación final del sistema SerDes.

2.3. Descripción del funcionamiento del módulo Serializador

La etapa de serialización tiene como objetivo convertir un bus de datos paralelos en datos seriales. A continuación se describe de manera general el funcionamiento del Serializador diseñado en este proyecto.

1. Un dato paralelo de 8 bits es almacenado y sincronizado con el reloj de transmisión.

2. El dato es codificado a un símbolo válido de 10 bits usando la codificación 8b/10b.

3. El dato es serializado, es decir, cada bit del símbolo es enviado uno a uno en cada ciclo del reloj de transmisión.

4. El dato serializado será proveído a la etapa de transmisión analógica para ser enviado fuera del chip.

2.4. Arquitectura de diseño del módulo Serializador

La figura 3 muestra la arquitectura a bloques del sistema SerDes, incluye el Serializador y el Deserializador con los bloques para codificación y decodificación 8b/10b.

En la figura 3 la señal *"Tx_Data"* es un dato paralelo de 9 bits, de los cuales 8 bits son el dato que se quiere codificar y el noveno bit es un el bit de control que indica si el dato se debe codificar como un dato o como carácter especial. El bloque *"Encoder"* recibe como entrada la señal *"Tx_Data"* y codifica el dato en símbolos de datos y símbolos especiales. La señal *"Disparity _d"* es la disparidad actual del dato codificado y la señal *"Disparity _q"* es la disparidad del dato codificado anterior. La señal *"encoded_tx_data"* es el símbolo codificado a 10 bits que será serializado por el bloque *"Serializer"* usando reloj de transmisión *"ser_clock". La* señal "Tx_PCIe" es una señal serial la cual transmite el dato codificado.

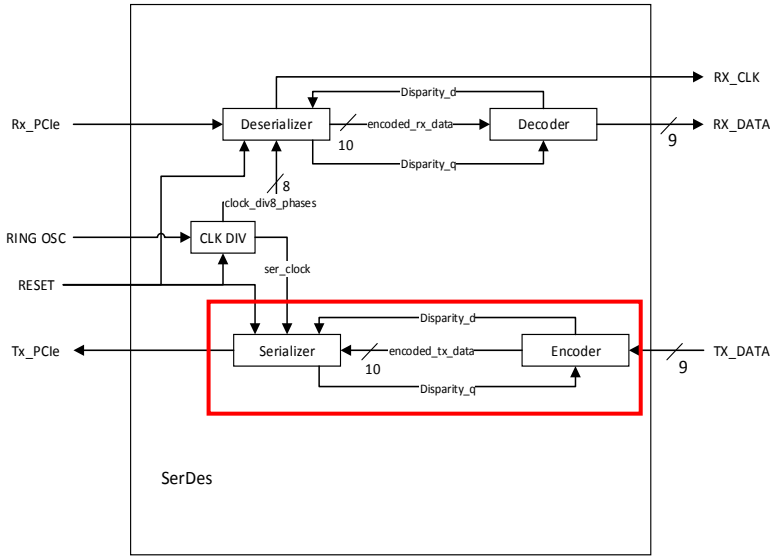


Figura 3. Arquitectura a bloques para el sistema SerDes

La figura 4 muestra la arquitectura digital propuesta para el Serializador. Esta arquitectura propone un multiplexor donde si el selector (*tx_frame_started*) es cero pasará el dato paralelo del bus, de lo contrario pasará el valor actual del registro de corrimiento. El registro de corrimiento es habilitado por la señal *tx_frame_started* la cual es generada a partir de un contador, que cuenta 10 ciclos de reloj ya que el dato a serializar contiene 10 bits. Se incluye también un registro para almacenar la disparidad del símbolo previo.

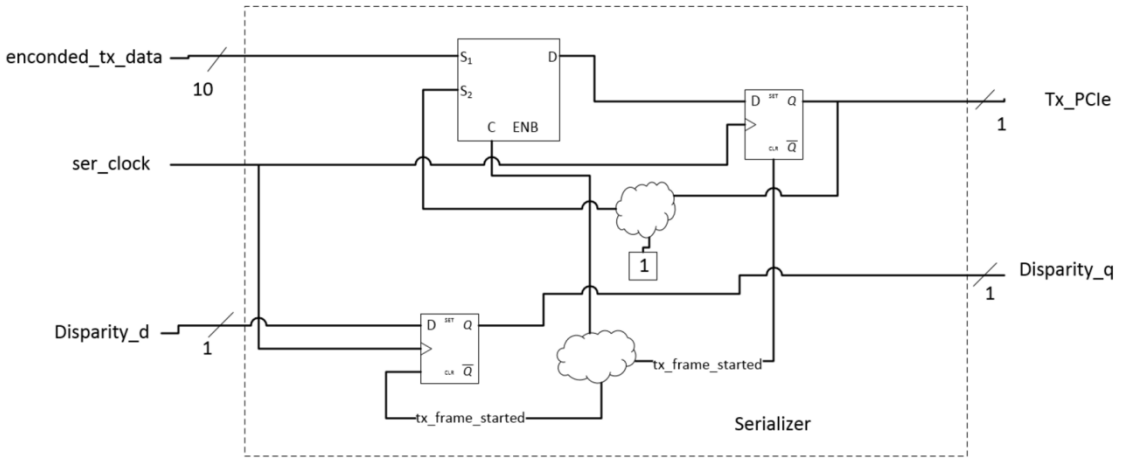


Figura 4. Arquitectura digital del Serializador

Se decidió reutilizar la implementación en Verilog del codificador que propone Hollis Chuck Benz. [5], ya que la implementación de la codificación 8b/10b no es uno de los objetivos de este proyecto.

2.5. Modelo en código RTL del módulo Serializador usando Verilog

A continuación se muestra el código Verilog que describe el funcionamiento del serializador.

```
module serializer(par_in, clk, rst, disparity_d, ser_out, disparity_q, tx_frame_started);
input[9:0]  par_in;   // data from the encoder
input       clk;     // shift clock
input       rst;     // reset signal
input       disparity_d ;
output reg         tx_frame_started;

// ------------ Outputs -------------------------//

output      ser_out;  // serial output data
output reg  disparity_q;

// -------------Internal Registers-----------------------//
reg [9:0]   par_reg;  // register to store data
reg [3:0]   counter;
// -------------Control Path-----------------------------//

always @(posedge clk, posedge rst ) begin
        if (rst) begin
              counter <= 4'b0000;
              tx_frame_started <= 1'b0;
        end

        //tx_frame_started should be 1 after counter is zero
        else if (counter == 4'b0)begin
                 tx_frame_started <= 1'b1;
                 counter <= counter + 1'b1;
        end

        //Count from 0 to 9
        else if (counter < 4'b1001) begin
                 tx_frame_started <= 1'b0;
                 counter <= counter + 1'b1;
        end

        // Restart counter
        else
        begin
                 counter <= 4'b0000;
                   tx_frame_started <= 1'b0;
          end
end

// -------------Data Path-----------------------------//
always @(posedge clk, posedge rst) begin
    if (rst)
          par_reg <= 10'h00;
    else
       if (tx_frame_started)
           par_reg <= par_in;
       else
           par_reg <= {1'b0, par_reg[9:1]};
end

//Serialized data
assign ser_out = par_reg[0];

//Assigns disparity
always @(posedge clk, posedge rst) begin
                        if (rst)
                                disparity_q <= 1'b0;

                        else if (tx_frame_started)
                                disparity_q <= disparity_d;
        end

endmodule
```

2.6. Síntesis lógica del módulo Serializador.

La síntesis lógica consiste en traducir un modelo *RTL* a celdas estándar, este proceso se realiza con la ayuda de una herramienta de síntesis. En este proyecto se utilizó ***Encounter RTL Compiler***. El flujo que se recorre para la síntesis lógica se muestra en el Apéndice B. Es conveniente automatizar esta operación puesto que el proceso de diseño es iterativo, debido a que los requerimientos pueden cambiar o bien los resultados no son los esperados. Por esta razón se decidió automatizar el flujo de la síntesis lógica con un script en TCL (Tool command language).

La figura 5 muestra las entradas necesarias para la herramienta y los archivos generados por ella.

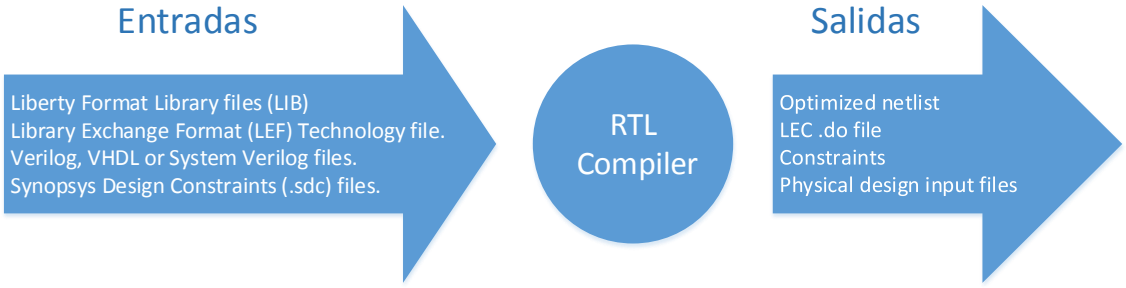


Figura 5. Entradas y salidas para RTL Compiler

El script apunta a las rutas donde se encuentran los archivos .lib, el *technology lef* y los *timing constraints* dentro de los equipos del ITESO. El script se puede encontrar en el Apéndice B y en el Apéndice D se puede encontrar el archivo de *constraints* utilizados en este proyecto.

La figura 6 muestra los archivos utilizados en la síntesis lógica así como los productos de ella.

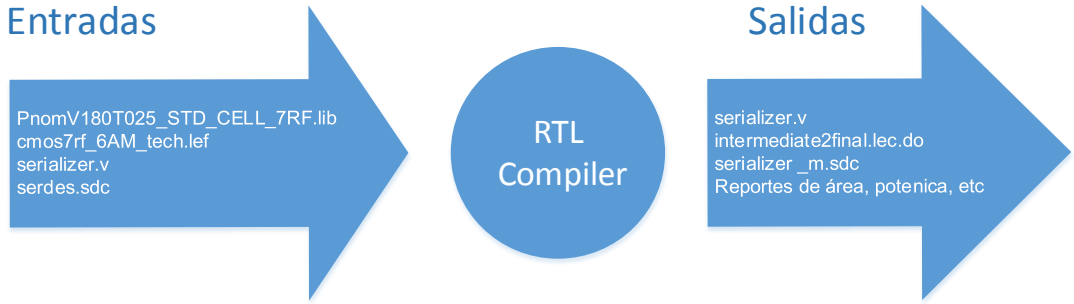


Figura 6. Entradas y salidas de RTL Compiler para Serializador

Al finalizar la síntesis lógica, es posible obtener un diagrama esquemático del netlist generado por la herramienta. La figura 7 muestra los resultados de la síntesis lógica del Serializador. Se observa que la herramienta interpretó correctamente los 10 registros necesarios para el registro de corrimiento que se modeló en Verilog. La herramienta también interpretó las compuertas combinacionales necesarias para la ruta de control.

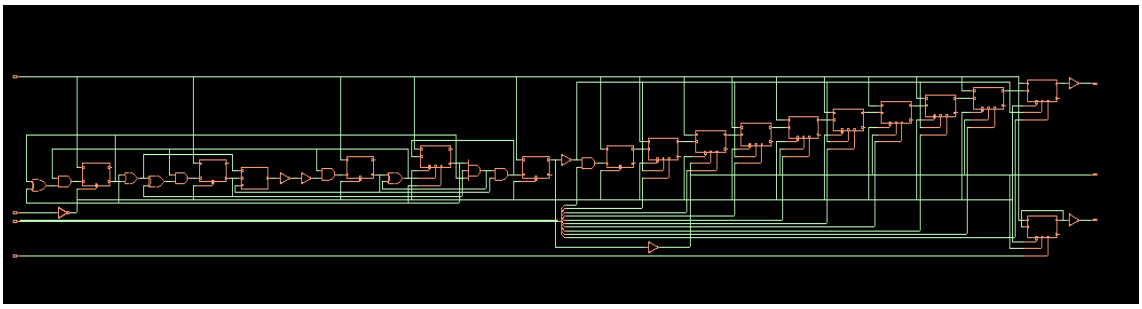


Figura 7. Síntesis lógica de Serializador usando la tecnología cmrf_7sf de IBM

En la figura 8, se muestra el reporte de área, obtenido de *RTL Compiler*. El área ocupada sólo por celdas se obtiene de multiplicar el área que ocupa cada celda por el número de instancias usadas en el diseño. El área requerida para el Serializador es 3128.56 μ^2, lo que representa el 0.13% del área del chip (2.25mm^2).

Report Area

Generated by: Encounter(R) RTL Compiler v12.10-s012_1 (Jan 26 2013)
Generated on: Oct 24 2015 19:14:18
Module: serializer
Technology library: PnomV180T025_STD_CELL_7RF
Operating conditions: PnomV180T025_STD_CELL_7RF
Interconnect mode: ple

Instance	Cells	Cell Area	Net Area	Total Area	Wireload	WL Flag
serializer	34	1998.26	1130.30	3128.56	<none>	(S)

Figura 8. Área utilizada por celdas estándar para Serializador

La figura 9 muestra el reporte de celdas utilizadas para implementar el diseño del Serializador. La herramienta de síntesis reportó que para el modelo propuesto es necesario: 2 celdas AND de 2 y 3 entradas, 6 buffers de distintos tipos, 5 registros tipo D *master-slave* con reset (DFFR), un inversor, una celda OR-AND-INVERSOR (OAI), una OR de 2 entradas y 11 registros de tipo D con reset asíncrono.

El área ocupada por las celdas requeridas para el módulo digital serializador es de 1998.27μ^2. Considerando que el chip completo tiene un área máxima de 2.25 mm^2, el área usada por las celdas estándar para el serializador es muy baja puesto que representa el 0.08% del área total.

Gate	Instances	Area	Library
	4	90.32	
AND2_H	1	22.58	PnomV180T025_STD_CELL_7RF
AND3_I	1	26.34	PnomV180T025_STD_CELL_7RF
BUFFER_D	1	18.82	PnomV180T025_STD_CELL_7RF
BUFFER_F	1	18.82	PnomV180T025_STD_CELL_7RF
BUFFER_K	2	52.69	PnomV180T025_STD_CELL_7RF
BUFFER_L	1	26.34	PnomV180T025_STD_CELL_7RF
BUFFER_M	1	41.40	PnomV180T025_STD_CELL_7RF
DFFR_H	1	71.50	PnomV180T025_STD_CELL_7RF
DFFR_K	4	376.32	PnomV180T025_STD_CELL_7RF
INVERT_A	1	11.29	PnomV180T025_STD_CELL_7RF
OAI21_E	1	26.34	PnomV180T025_STD_CELL_7RF
OR2_H	1	22.58	PnomV180T025_STD_CELL_7RF
SDFFR_H	9	846.72	PnomV180T025_STD_CELL_7RF
SDFFR_K	2	233.32	PnomV180T025_STD_CELL_7RF
XOR2_E	2	75.26	PnomV180T025_STD_CELL_7RF
XOR2_I	1	37.63	PnomV180T025_STD_CELL_7RF
TOTAL	34	1998.27	

Figura 9.Celdas estándar utilizadas por el Serializador

La figura 10 muestra el reporte las celdas requeridas para el Serializador, agrupadas por tipo. Las celdas *Sequential* se refieren a celdas del tipo registros o *flip-flop*. Las celdas tipo *inverter*, *buffer y logic* se refieren a celdas combinacionales. Los resultados corresponden a los esperados del modelo RT. El 76.5% de las celdas utilizadas son del tipo secuenciales esto es debido a que el Serializador se modeló como un registro de corrimiento, celdas combinacionales reportadas se utilizaron en la ruta de control.

Generated by: Encounter(R) RTL Compiler v12.10-s012_1 (Jan 26 2013)
Generated on: Oct 24 2015 19:14:51
Module: serializer
Technology library: PnomV180T025_STD_CELL_7RF
Operating conditions: PnomV180T025_STD_CELL_7RF
Interconnect mode: ple

Type	Instances	Area	Area %
sequential	16	1527.86	76.50
inverter	1	11.29	0.60
buffer	6	158.05	7.90
logic	11	301.06	15.10
TOTAL	34	1998.26	100.10

Figura 10. Reporte de área utilizada por las celdas estándar en la síntesis lógica

2.7. Integración de módulo Serializador al SerDes

La integración del módulo Serializador al sistema SerDes se realizó instanciando el módulo en un proyecto *toplevel* y haciendo las conexiones necesarias entre los módulos de acuerdo a la arquitectura presentada en la sección 2.4.

La figura 11 es un diagrama esquemático del *netlist* de la síntesis lógica del sistema SerDes. Se obtiene de ejecutar el comando *gui_show* en la terminal de Linux una vez que la síntesis lógica se realizó exitosamente.

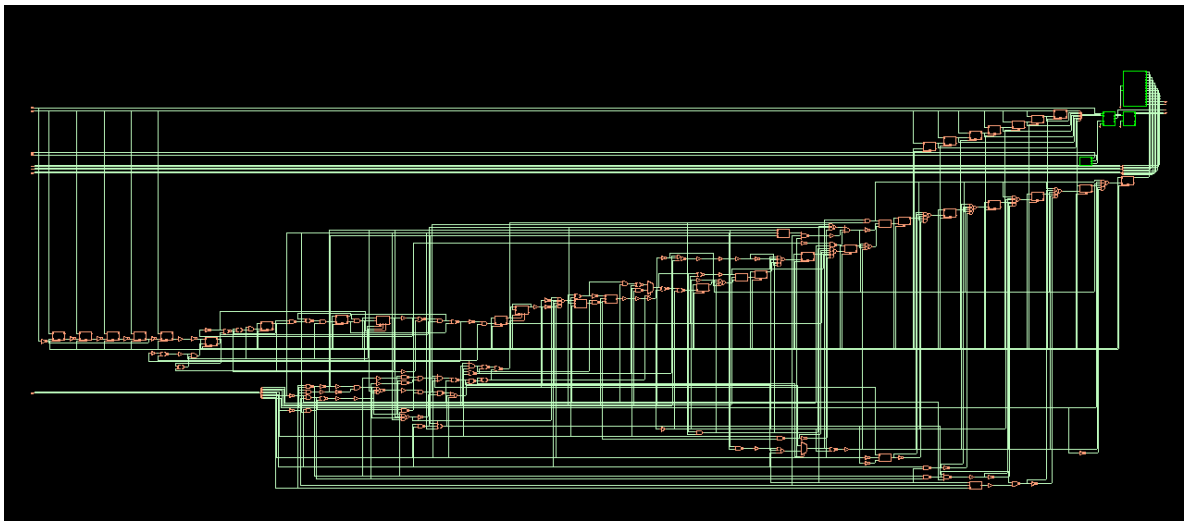
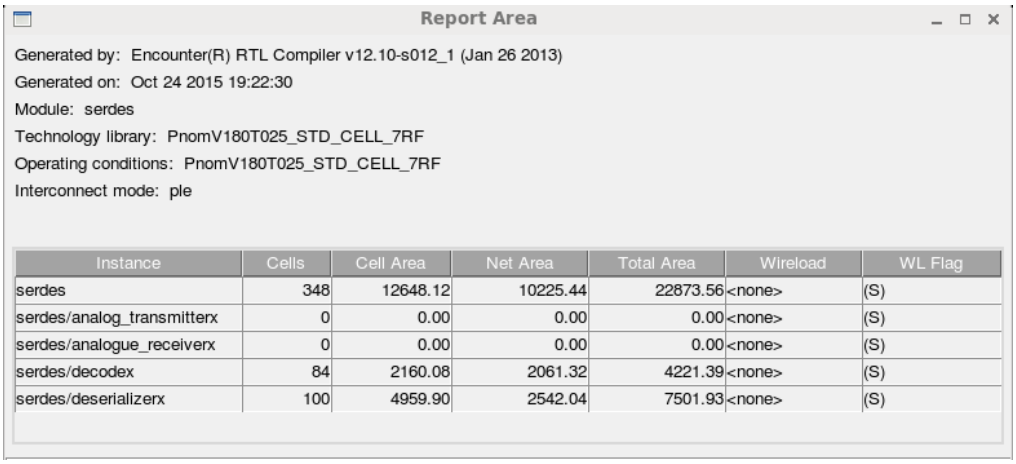


Figura 11. Síntesis lógica del SerDes

La figura 12, presenta el reporte de área de la síntesis lógica del SerDes. Este reporte se obtiene de ejecutar el comando *report area* en la línea de comandos dentro de RTL Compiler. En estos resultados se observa que los módulos *analog_transmittex* y el módulo *analog_receiverx* reportan celdas igual cero y área igual a cero. Este resultado es esperado puesto que fueron instanciados como cajas negras.



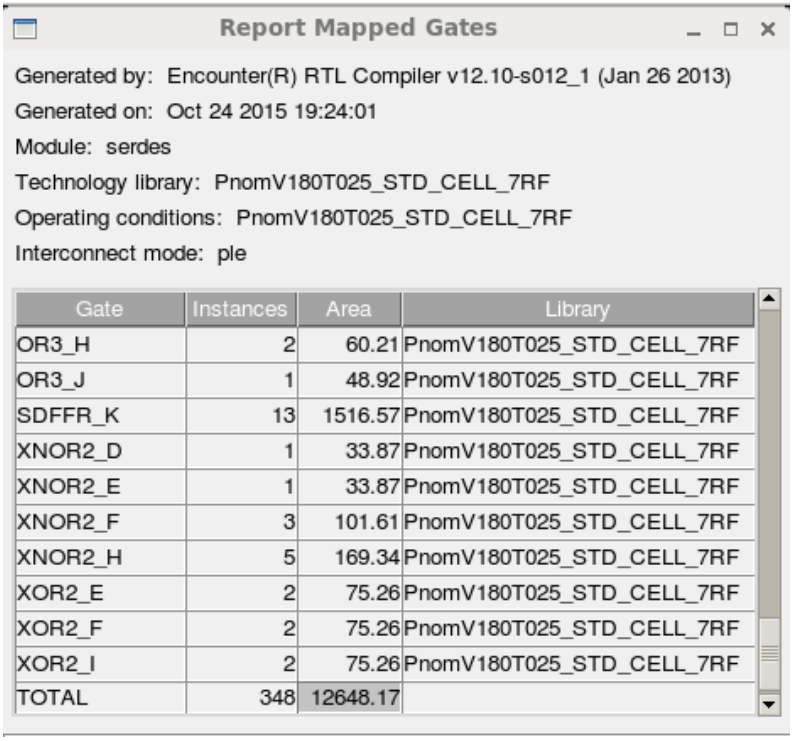
Report Area

Generated by: Encounter(R) RTL Compiler v12.10-s012_1 (Jan 26 2013)
Generated on: Oct 24 2015 19:22:30
Module: serdes
Technology library: PnomV180T025_STD_CELL_7RF
Operating conditions: PnomV180T025_STD_CELL_7RF
Interconnect mode: ple

Instance	Cells	Cell Area	Net Area	Total Area	Wireload	WL Flag
serdes	348	12648.12	10225.44	22873.56	<none>	(S)
serdes/analog_transmitterx	0	0.00	0.00	0.00	<none>	(S)
serdes/analogue_receiverx	0	0.00	0.00	0.00	<none>	(S)
serdes/decodex	84	2160.08	2061.32	4221.39	<none>	(S)
serdes/deserializerx	100	4959.90	2542.04	7501.93	<none>	(S)

Figura 12. Reporte de área utilizada por celdas estándar en sistema SerDes

La figura 13, presenta el reporte de las celdas estándar utilizadas para implementar el sistema SerDes. Este reporte se puede obtener usando el comando *report gates* dentro de ***RT Compiler.*** El área utilizada por las celdas estándar necesarias para implementar los módulos digitales del sistema SerDes es igual a 0.01264800mm^2 lo que representa el 0.56% del área del chip. La herramienta reportó que se requieren tres celdas tipo OR de tres entradas de dos tipos distintos, 13 registros tipo D con reset asíncrono (SDFFR), 10 celdas tipo XNOR de cuatro tipos distintos y celdas tipo XOR de tres tipos distintos.



Report Mapped Gates

Generated by: Encounter(R) RTL Compiler v12.10-s012_1 (Jan 26 2013)
Generated on: Oct 24 2015 19:24:01
Module: serdes
Technology library: PnomV180T025_STD_CELL_7RF
Operating conditions: PnomV180T025_STD_CELL_7RF
Interconnect mode: ple

Gate	Instances	Area	Library
OR3_H	2	60.21	PnomV180T025_STD_CELL_7RF
OR3_J	1	48.92	PnomV180T025_STD_CELL_7RF
SDFFR_K	13	1516.57	PnomV180T025_STD_CELL_7RF
XNOR2_D	1	33.87	PnomV180T025_STD_CELL_7RF
XNOR2_E	1	33.87	PnomV180T025_STD_CELL_7RF
XNOR2_F	3	101.61	PnomV180T025_STD_CELL_7RF
XNOR2_H	5	169.34	PnomV180T025_STD_CELL_7RF
XOR2_E	2	75.26	PnomV180T025_STD_CELL_7RF
XOR2_F	2	75.26	PnomV180T025_STD_CELL_7RF
XOR2_I	2	75.26	PnomV180T025_STD_CELL_7RF
TOTAL	348	12648.17	

Figura 13. Celdas estándar mapeadas para sistema SerDes

CAPITULO 3: VERIFICACIÓN Y VALIDACIÓN DEL MÓDULO SERIALIZADOR

3.1. Plan de verificación y validación del módulo Serializador

En el flujo de diseño es necesario realizar la verificación del modelo del serializador en sus diferentes etapas de desarrollo, con el fin de asegurar que la funcionalidad siga siendo la misma. La figura 14, muestra el orden en la que se realizarán las diferentes verificaciones.

Figura 14. Niveles de verificación del Serializador

3.1.1. *Verificación funcional.*

Una vez modelado el módulo Serializador a nivel RTL, se elaboró un banco de pruebas en Verilog para verificar que el funcionamiento correspondiera al esperado. En este banco de pruebas se estimuló el módulo con valores conocidos y se revisó que las salidas correspondieran al valor esperado. E el código Verilog se puede encontrar el Apéndice C.

3.1.2. *Verificación de equivalencia lógica. Logic Equivalence Check.*

Se verificará que cada elemento descrito en hardware haya sido correctamente interpretado por la herramienta y se haya mapeado a una celda estándar de la librería cmrf_7sf. En la sección 3.3 se pueden encontrar los resultados.

3.1.3. *Verificación a nivel compuerta.*

Usando el mismo banco de pruebas implementado para la verificación funcional se realizó una simulación usando el netlist sintetizado por RTL Compiler, se verificó que el módulo tuviera la misma funcionalidad.

3.1.4. *Verificación física.*

Después de integrar el módulo serializador al sistema SerDes, se realizará la verificación física en Virtuoso, donde se harán pruebas de DRC y LVS.

3.2. Banco de prueba para el módulo Serializador

El banco de pruebas fue implementado en Verilog, es un código algorítmico el cual inyecta valores conocidos al dispositivo bajo prueba (DUT) y verifica que a la salida del DUT los resultados correspondan al valor esperado. El código Verilog se puede encontrar en el Apéndice C.

3.3. Resultados de simulación funcional del módulo Serializador

Usando el banco de pruebas mostrado en el apéndice B. Se realizó una simulación en Modelsim. La figura 15 muestra los resultados para el Serializador de datos de 10 bits.

La señal *"data_in"* es un puerto de entrada que recibe el dato paralelo de 10 bits generado por el codificador. En el modelo del Serializador, el bus es leído exclusivamente cada 10 ciclos de reloj, es decir, cada vez que se termina de serializar un dato de 10 de bits, se volverá a leer un nuevo dato. La señal *tx_frame_started* es igual a '1' cuando el contador es igual a '0' por haber alcanzado la cuenta de 9 o porque se recibió un *reset*.

La señal *"ser_out"* es el dato serializado, corresponde al bit menos significativo del registro de corrimiento. El dato serializado se empieza a transmitir justo cuando la señal *tx_frame_started* cambia su estado de uno a cero.

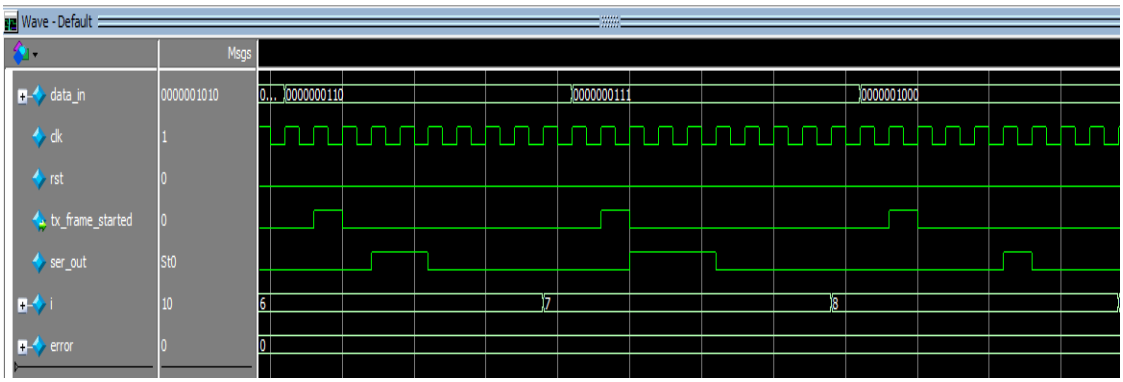


Figura 15. Resultados de verificación funcional del serializador de datos de 10 bits

Se comprobó el correcto funcionamiento del codificador y decodificador de Hollis Chuck Benz [5] conectando en herradura el módulo codificador y el módulo decodificador. La figura 16 muestra la implementación para la verificación de estos módulos.

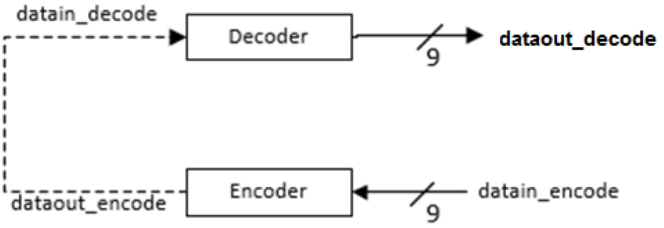


Figura 16. Conexión en herradura del codificador y del decodificador

En la figura 17, la salida *datain_encode* es la entrada al codificador, esta señal es el estímulo generado por el banco de pruebas. La señal *dataout_encode* es el resultado de la codificación, esta señal es conectada a la entrada del decodificador (*datain_decode)*. La señal *dataout_decode* es la salida del decodificador. Se puede observar que la señal *dataout_decode* es igual al valor de la señal *datain_encode*, lo cual es el comportamiento correcto.

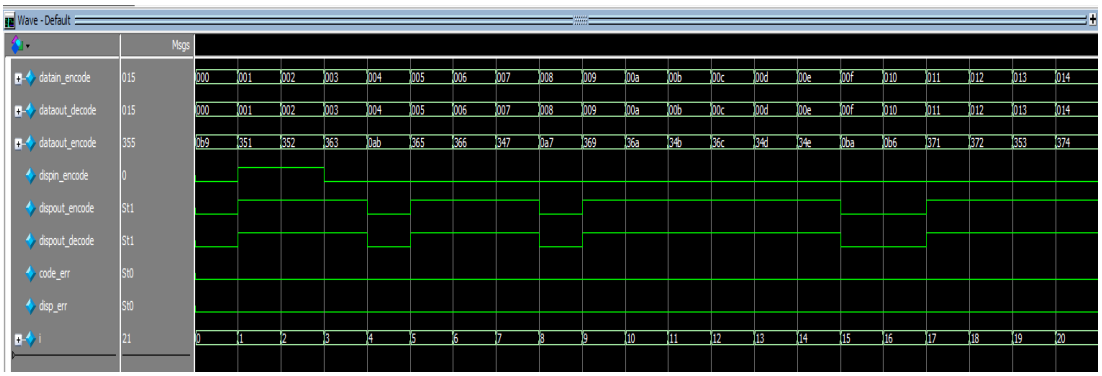


Figura 17. Loop del Serializador y Deserializador con codificador y decodificador

La figura 18 muestra las formas de onda para la serialización de los símbolos de datos de 10 bits codificados. Por ejemplo el dato "*8'b000000110"* de acuerdo a la Tabla 1 del Capítulo 1 se codifica en "*10'b1101100110".* Por lo que la señal *"Error"* es igual a cero.

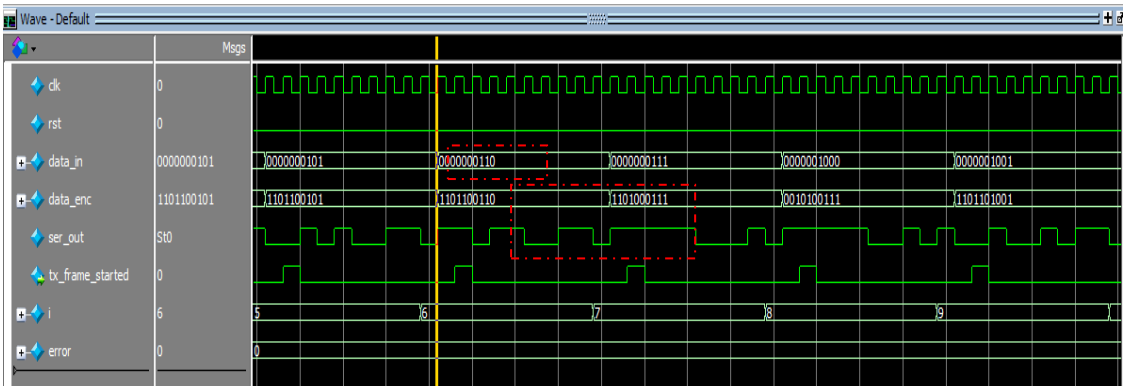


Figura 18. Formas de onda del Serializador para PCI Express

3.4. Verificación de equivalencia lógica

Comúnmente llamado LEC (por sus siglas en inglés *Logic Equivalence Check)*, es una verificación que se recomienda realizar inmediatamente después ejecutarse la síntesis lógica, puesto que determina si el modelo RTL y el netlist obtenido de la síntesis lógica son iguales o no. Esta verificación se realizó usando la herramienta **Cadence *Encounter Conformal* Equivalence Checker**. La figura 19 muestra los distintos niveles de abstracción donde se puede realizar la verificación de equivalencia lógica. En este proyecto solo se realizó una verificación lógica entre el diseño a nivel RTL y el netlist generado por RTL Compiler en la síntesis lógica.

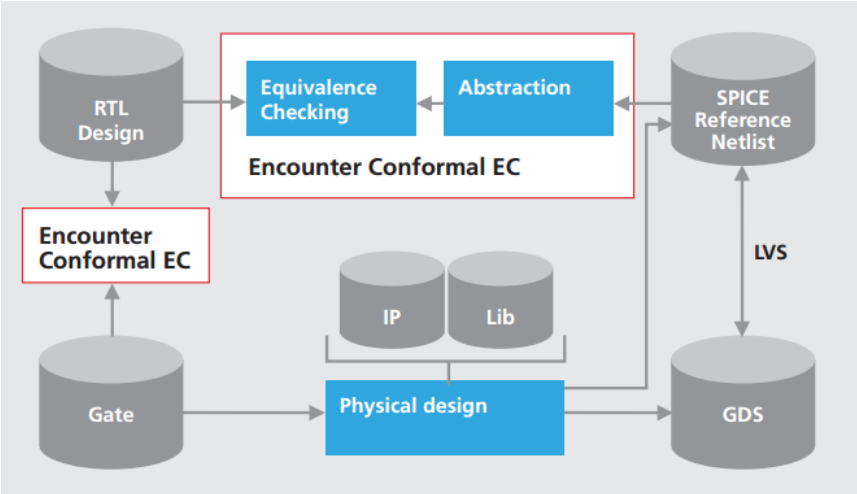


Figura 19. Utilización de LEC en distintos niveles de abstracción

Durante la síntesis lógica se generó un archivo .do, este archivo es usado por la herramienta para verificar que el modelo sintetizado es equivalente al modelo RTL. La figura 20 muestra los resultados para el módulo Serializador a nivel RTL y el módulo sintetizado a nivel intermedio, es decir, compara el modelo RTL con un modelo sintetizado que usa celdas genéricas. Se observa que el resultado es PASS lo que significa que los dos modelos son equivalentes.

```
--------------------------------------------------------------------------------
6. Compare Results:                                                       PASS
     Number of EQ compare points:                          19
     Number of NON-EQ compare points:                      0
     Number of Aborted compare points:                     0
     Number of Uncompared compare points :                 0
================================================================================
// Command: report statistics
Mapping and compare statistics
================================================================================
                     Compare Result       Golden            Revised
--------------------------------------------------------------------------------
Root module name                          serializer        serializer

Primary inputs                               13                13
   Mapped                                    13                13

Primary outputs                               3                 3
   Mapped                                     3                 3
      Equivalent                  3

State key points                             16                16
   Mapped                                    16                16
      Equivalent                 16
================================================================================

SETUP> dofile rtl2intermediate.lec.do
```

Figura 20. Resultados de LEC entre RTL y netlist intermedio

La figura 21 muestra los resultados para el módulo sintetizado a nivel intermedio y el módulo final, es decir, compara el modelo sintetizado que usa celdas genéricas y el modelo que usa las celdas estándar de la librería cmrf_7sf. Se observa que el resultado es PASS lo que significa que los dos modelos son equivalentes.

```
6. Compare Results:                                                    PASS
     Number of EQ compare points:                          19
     Number of NON-EQ compare points:                      0
     Number of Aborted compare points:                     0
     Number of Uncompared compare points :                 0
================================================================================
// Command: report statistics
Mapping and compare statistics
================================================================================
                     Compare Result       Golden            Revised
--------------------------------------------------------------------------------
Root module name                          serializer        serializer

Primary inputs                               13                13
   Mapped                                    13                13

Primary outputs                               3                 3
   Mapped                                     3                 3
      Equivalent                 3

State key points                             16                16
   Mapped                                    16                16
      Equivalent                16
================================================================================
```

Figura 21. Resultados LEC entre netlist intermedio y final

3.5. Resultados de simulación del módulo Serializador a nivel compuerta

Usando el banco de pruebas mostrado en el Apéndice. Se realizó una simulación en ***Modelsim*** donde se instanció el módulo Serializador sintetizado en RTL **Compiler**. El comportamiento esperado es que el funcionamiento no haya cambiado. En la figura 22 se puede observar que el dato 8'b00000110 es serializado un ciclo después de que el registro es habilitado por la señal *tx_frame_started.* El comportamiento es correcto, por lo que la señal "error" tiene valor de cero.

El reloj para esta simulación está corriendo a 125 MHz, esto es porque a nivel sistema se observó que un reloj más rápido provocaría violaciones de los tiempos de *setup* o *hold* de los registros de la librería digital de la tecnología cmrf_7sf.

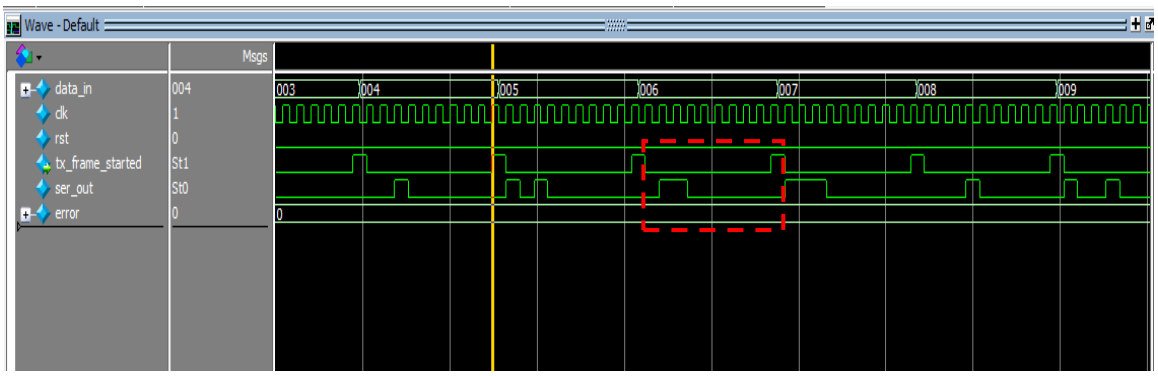


Figura 22. Resultados de la simulación a nivel compuerta del Serializador

CAPITULO 4: DISEÑO FÍSICO DEL SERDES

4.1. Síntesis física del sistema SerDes

La síntesis física se realizó usando ***Cadence Encounter Digital Implementation System (EDI).*** En las páginas siguientes se mostrará el procedimiento para generar el *floorplan* del sistema SerDes usando la interfaz gráfica.

Con el fin de no repetir manualmente los pasos necesarios cada vez que se requirió hacer ajustes al diseño se decidió automatizar la síntesis física con un script en TCL (*Tool command language*), el cual se puede encontrar en el Apéndice F.

4.1.1. Analysis View

Una de las entradas más importantes para la herramienta es la definición del *Analysis View,* este archivo se puede generar dentro de la misma interfaz gráfica o se puede escribir un script (recomendable) para después ser llamado desde otros scripts. El *Analysis View* especifica las esquinas de operación del diseño. La figura 23 muestra la estructura para crear un *Analysis View.*

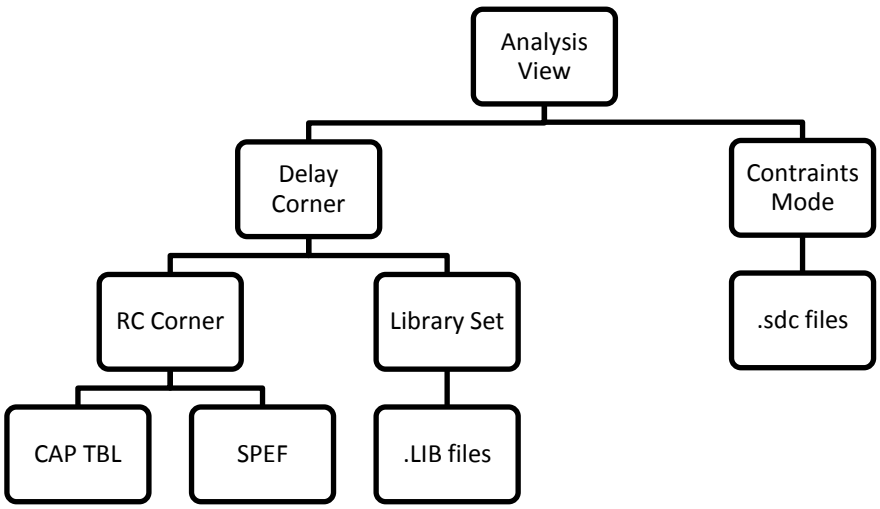


Figura 23. Estructura del Analysis View

A continuación se muestra el archivo .tcl utilizado para crear el *Analysis View de* este proyecto.

```
# Version:1.0 MMMC View Definition File

# Do Not Remove Above Line

create_rc_corner -name rctyp_t25 -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_op_cond -name op_cond_1v8_t25 -library_file
{../lib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_20111130/std_cell/v.20111130/synops
ys/nom/PnomV180T025_STD_CELL_7RF.lib} -P {180} -V {1.8} -T {25}

create_library_set -name nom_1v8_t25 -timing
{../lib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_cell_20111130/std_cell/v.20111130/synops
ys/nom/PnomV180T025_STD_CELL_7RF.lib}

create_constraint_mode -name serdes_func -sdc_files {../rc/outputs/serdes_m.sdc}

create_delay_corner -name dc_rctyp_1v8_t25 -library_set {nom_1v8_t25} -opcond
{op_cond_1v8_t25} -rc_corner {rctyp_t25}

create_analysis_view -name av_rctyp_1p8_t25 -constraint_mode {serdes_func} -
delay_corner {dc_rctyp_1v8_t25}

set_analysis_view -setup {av_rctyp_1p8_t25} -hold {av_rctyp_1p8_t25}
```

4.1.2. Archivo .globals

Es importante crear un archivo .globals (script en TCL) el cual establece las rutas de los archivos que se usarán en la síntesis física, por ejemplo, los archivos .LEF, los archivos .LIB, etc. Esto evita hacer una y otra vez estas asignaciones. A continuación se muestra el archivo .globals utilizado en este proyecto.

```
#################################################################
#  Generated by:      Cadence Encounter 13.17-s018_1
#  OS:                Linux x86_64(Host ID fv00)
#  Generated on:      Tue Oct  6 22:03:15 2015
#  Design:
#  Command:           save_global nominal.globals
#################################################################
#
# Version 1.1
#

set conf_qxconf_file {NULL}
set conf_qxlib_file {NULL}
set defHierChar {/}
set init_gnd_net {GND}
set init_lef_file {../lef/cmos7rf_6AM_tech.lef ../lef/ibm_cmos7rf_analog_cells_serdes.lef
../lef/ibm_cmos7rf_sc_12Track.lef}
set init_mmmc_file {nominal.view}
set init_pwr_net {VDD}
set init_verilog {../rc/outputs/serdes_m.v}
set lsgOCPGainMult 1.000000
```

Para realizar la síntesis física se inicializó la herramienta EDI escribiendo *Encounter -64* en una terminal de Linux (esto abre la interfaz gráfica de Encounter). Luego cargamos los archivos de configuración, el *Analysis View* y el archivo .globals.

Esto se hace a través del menú *File*, donde se seleccionó *Import Design (figura 24).*

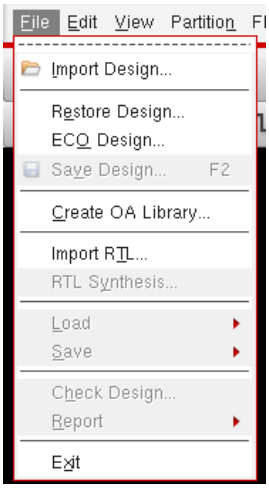


Figura 24. Menú File de EDI

Para cargar los archivos se abrió la ventana *Load desde Design Import* (figura 25).

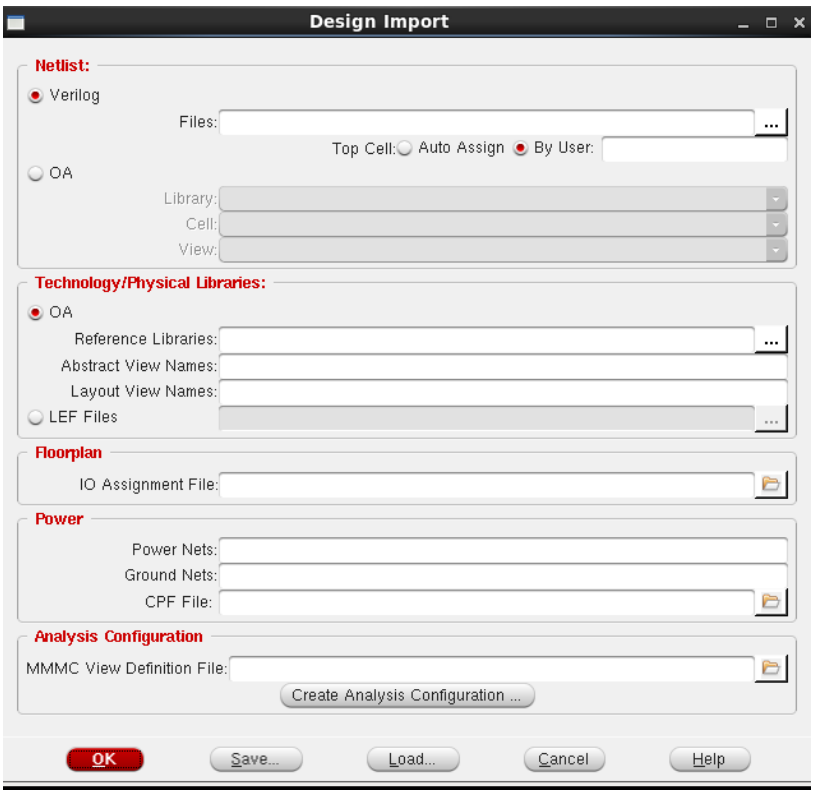


Figura 25. Ventana para importar diseño.

En la ventana *Load Global Variables*, se seleccionó el archivo .globals previamente elaborado, este archivo configuró todos los campos de la ventana *Import Design* mostrada en la figura 25.

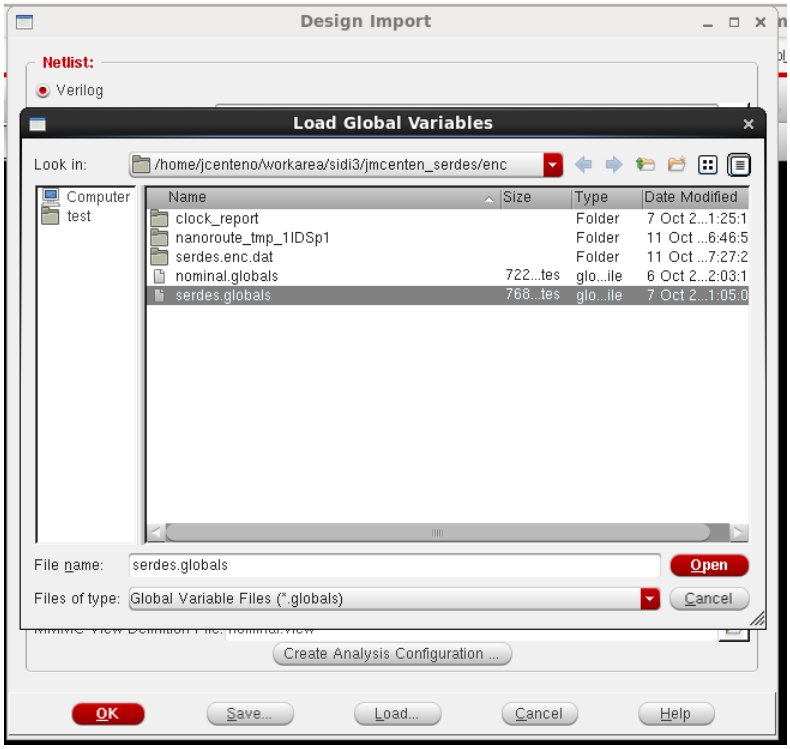


Figura 26. Selección de archivo .globals para EDI

Los comando en TCL que se muestran a continuación carga el archivo .globals del proyecto .

```
source serdes.globals
init_design
```

Hasta este punto EDI sabe de dónde se tomarán todos los archivos necesarios para la síntesis física. El siguiente paso fue crear un *Floorplan,* es decir, definir el tamaño del chip, asignar un área para las celdas, definir la distancia entre el core y el anillo de pines de entrada y salidas, etc.

4.1.3. Definición del Floorplan

Desde el menú *Floorplan se seleccionó Specify Floorplan*. En este proyecto se eligió definir el diseño especificando las dimensiones del chip que en este caso es 1.5mm x 1.5mm. También se configuró un espacio de 20um de cada lado entre el *core* y el *pad* de salidas y entradas (*Core to IO Boundary*). La figura 27 muestra las opciones de configuración para *floorplan* en EDI.

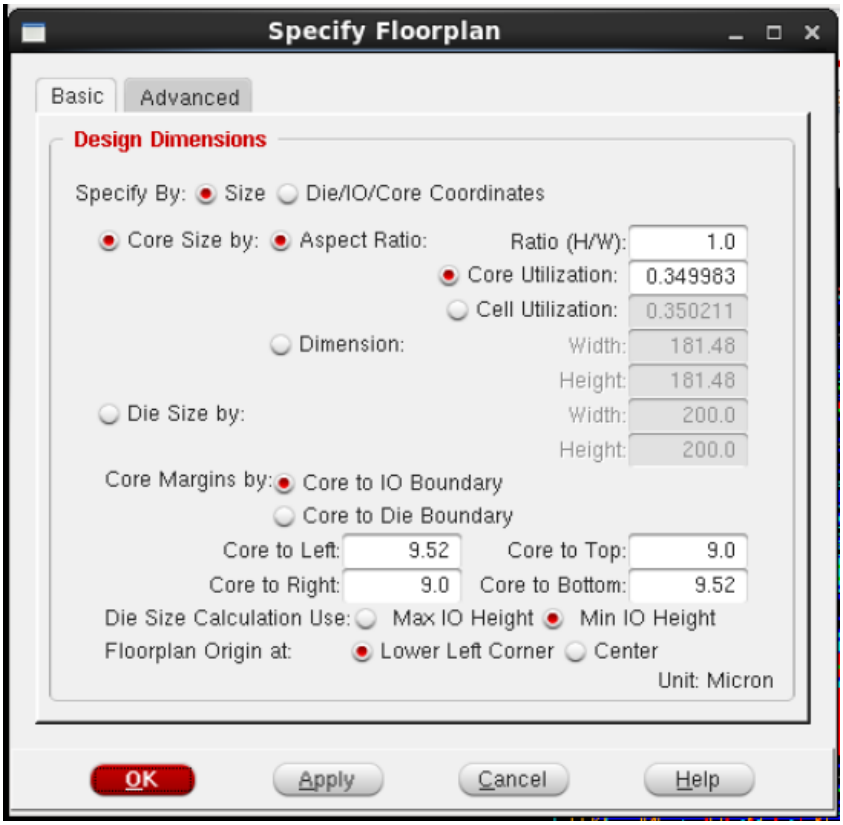


Figura 27. Configuración de Floorplan en EDI

Con las especificaciones del tiny chip se elaboró el comando en TCL equivalente a la figura 27.

```
# Floorplan specification using 20 microns for IO pad ring.
set tiny_chip_size 1500.0
set io_pad_ring 20.0
floorPlan -site CORE -d $tiny_chip_size $tiny_chip_size \
$io_pad_ring $io_pad_ring $io_pad_ring $io_pad_ring
```

4.1.4. Anillo de voltaje y tierra.

Una vez que se estableció el *floorplan,* se definió el anillo de voltaje y tierra el cual se encuentra alrededor del *core*. Desde el menú *Power → Power Planning* se seleccionó *Add Ring*. En este proyecto tierra recibe el nombre de GND y la fuente de voltaje VDD. Se indicó que los metales usados para VDD y GND serían MT(M5) y AM (M6). Para las rutas de alimentación se recomienda usar los metales más altos, ya que son los más anchos y resistentes. La figura 28 muestra las opciones de configuración para el anillo de voltaje y tierra en EDI.

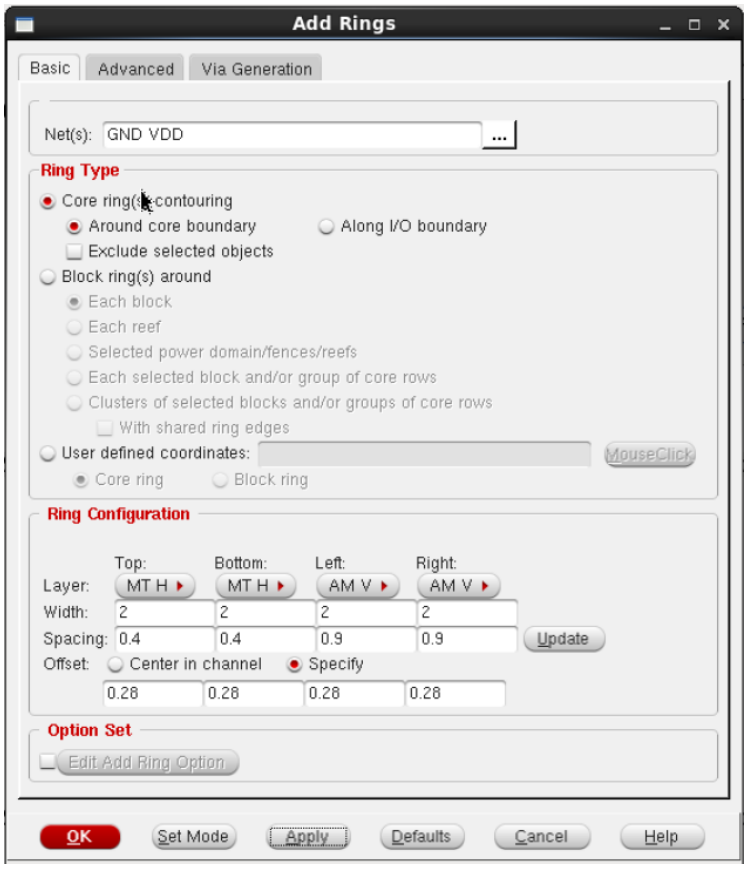


Figura 28. Anillo de voltaje y tierra

Se elaboró un script en TCL para configurar la herramienta con los valores recomendados de diseño para la tecnología cmrf_7sf.

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

4.1.5. Agregar tiras de voltaje y tierra.

Ya que se creó un anillo de voltaje y tierra alrededor del *core,* se crearon las tiras o rutas de voltaje y tierra para alimentar a las celdas estándar del diseño. Desde el menú *Power → Power Planning* se seleccionó *Add Stripe*. La figura 29 muestra la configuración por default de esta ventana. En este proyecto se estableció una distancia de 20um entre tiras, por lo que la herramienta creará todas las tiras que quepan en el espacio que fue configurado en el *floorplan* con una distancia entre sí de 20um.

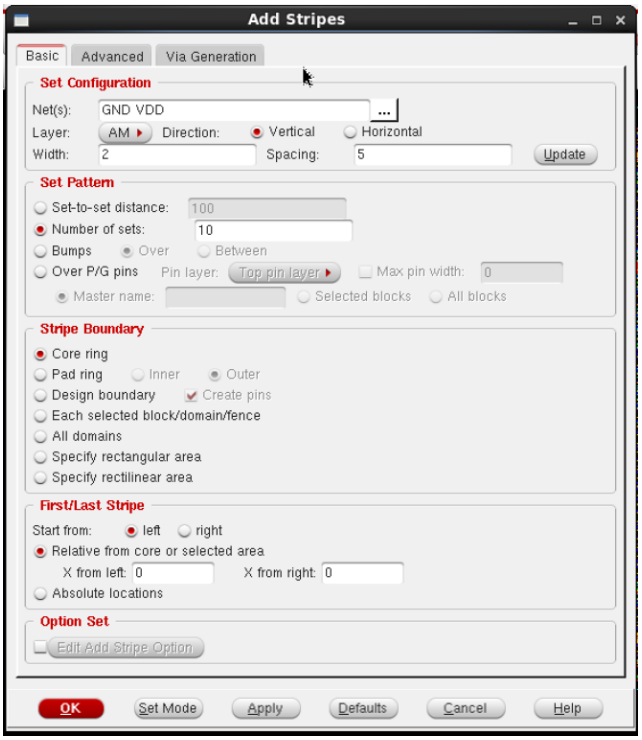


Figura 29. Configuración para tiras de VDD y GND

El comando en TCL que se presenta a continuación configura tiras o rutas en este proyecto.

```
addStripe\
    -block_ring_top_layer_limit AM\
    -max_same_layer_jog_length 4\
    -padcore_ring_bottom_layer_limit MT\
    -set_to_set_distance 20\
    -stacked_via_top_layer AM\
    -padcore_ring_top_layer_limit AM\
    -spacing 5\
    -merge_stripes_value 0.28\
    -layer AM\
    -block_ring_bottom_layer_limit MT\
    -width 2\
    -nets {GND VDD}\
    -stacked_via_bottom_layer M1\
    -break_stripes_at_block_rings 1
```

4.1.6. Ruteo de voltaje y tierra a metales bajos.

Para conectar voltaje y tierra a los metales más bajos (M1), en el menú Route → Special Route se indicó el nombre la fuente y tierra (VDD y GND), además se deseleccionó Block, Pins y Pad Pins. La figura 30 muestra la configuración de la interfaz gráfica de EDI.

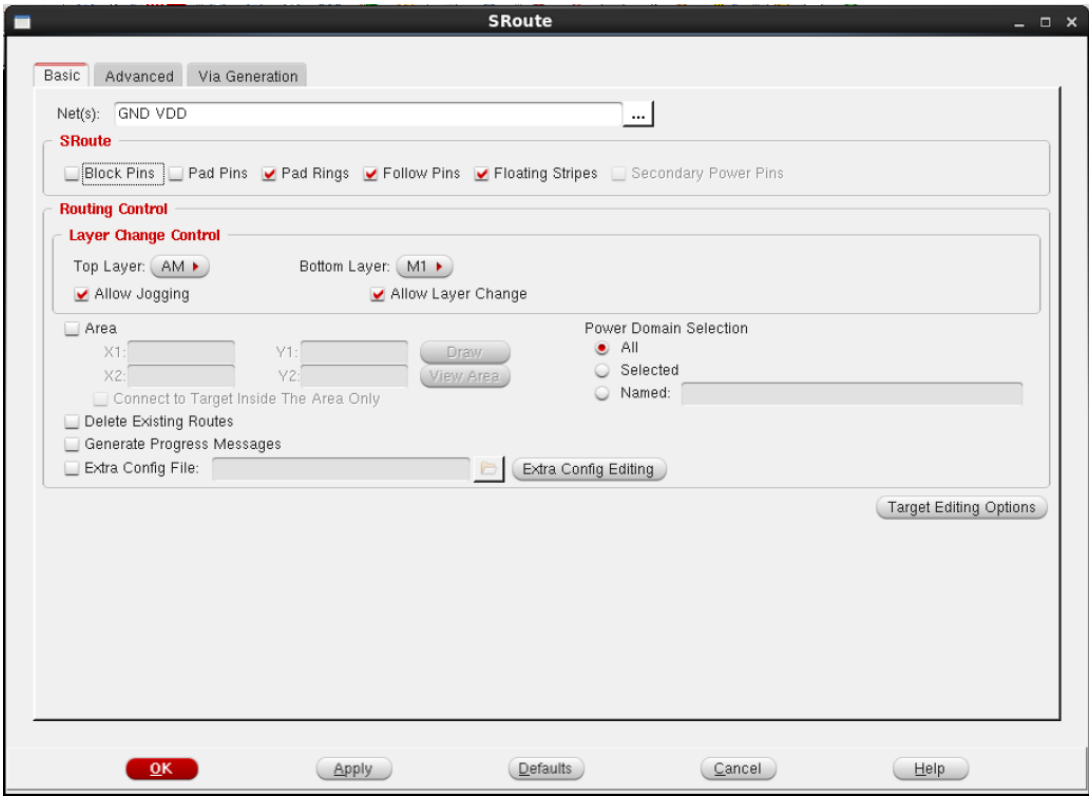


Figura 30. Configuración para red de energía a metales bajos

El comando en TCL que se muestra a continuación configura la herramienta de la misma forma en que se muestra en la figura 30.

```
sroute\
    -connect { padRing corePin floatingStripe }\
    -layerChangeRange { M1 AM }\
    -blockPinTarget { nearestTarget }\
    -stripeSCpinTarget { blockring padring ring stripe ringpin blockpin }\
    -checkAlignedSecondaryPin 1\
    -allowJogging 1\
    -crossoverViaBottomLayer M1\
    -allowLayerChange 1\
    -targetViaTopLayer AM\
    -crossoverViaTopLayer AM\
    -targetViaBottomLayer M1\
    -nets { GND VDD }
```

4.1.7. Colocación de celdas estándar.

Una vez que todo el chip está conectado a voltaje y tierra. Se procedió a hacer la colocación (*placement*) de las celdas estándar. Desde el menú Place → Standard Cell se configuró la ventana con los valores por default excepto la opción *Reorder scan Connection,* está opción se deshabilitó. La figura 31 muestra la configuración para la colocación de las celdas estándar.

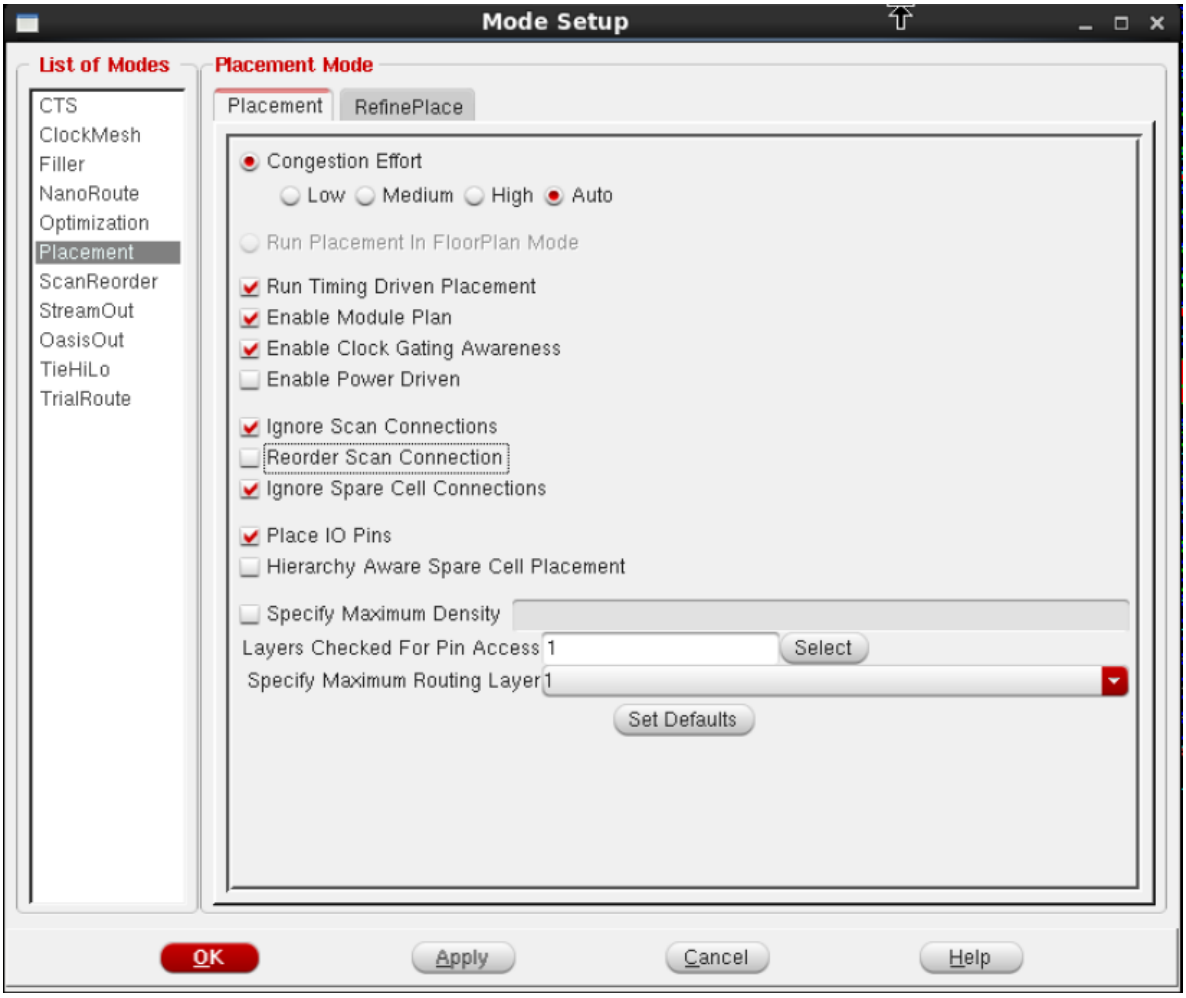


Figura 31. Configuración de EDI para colocación de celdas estándar

Los comandos en TCL que se muestran a continuación configuran la herramienta exactamente de la misma forma que en la figura 31.

```
setPlaceMode -reset
setPlaceMode -congEffort auto -timingDriven 1 -modulePlan 1 -clkGateAware 1 -
powerDriven 0 -ignoreScan 1 -reorderScan 0 -ignoreSpare 1 -placeIOPins 1 -
moduleAwareSpare 0 -checkPinLayerForAccess {  1 } -preserveRouting 0 -
rmAffectedRouting 0 -checkRoute 0 -swapEEQ 0
setPlaceMode -fp false
placeDesign -prePlaceOpt
```

4.1.8. Clock Tree.

Para mantener la integridad de la señal en un nivel aceptable, se introducen inversores y/o buffers en el diseño, estás estructuras reciben el nombre de *Clock Trees.* Desde el menú Clock → Synthesize Clock Tree se abrió la ventana *Gen Spec.* Donde se seleccionaron los inversores como elementos del *clock tree,* está información es guardada en un archivo Clock.ctstch que se puede utilizar en otros proyectos.

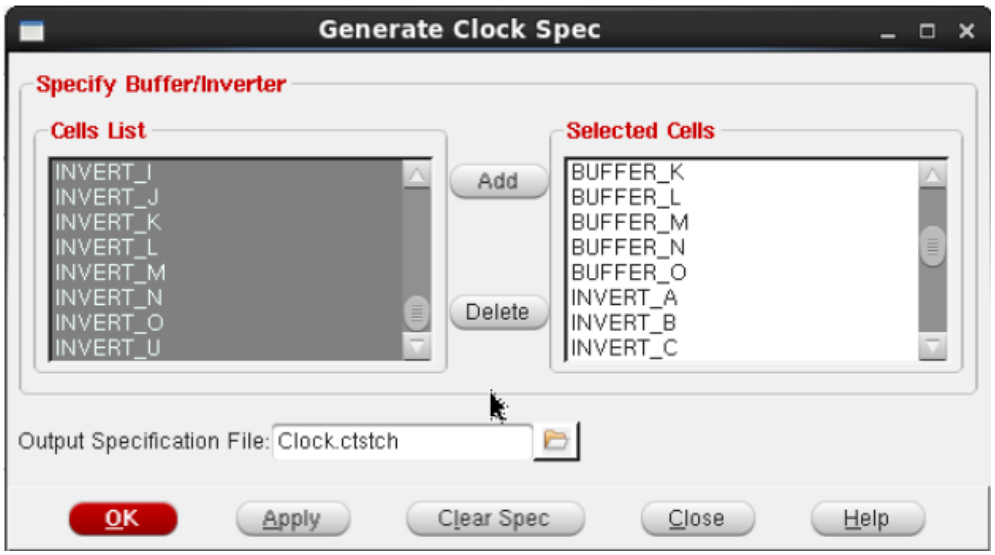


Figura 32. Selección de celdas para Clock Tree

En comando en TCL que se muestra a continuación configura la herramienta exactamente igual que en la figura 32.

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

4.1.9. Autoruteo

Ya que celdas estándar se colocaron y que se insertaron los elementos del *Clock Tree* se procedió a generar el autoruteo, este proceso se realiza a través de la herramienta NanoRoute dentro de EDI. Desde el menú Route → NanoRoute → Route. Se seleccionaron las siguiente opc iones.

- ✓ Fix Antenna
- ✓ Timing Driven
- ✓ SI Driven (Signal Integrity Driven): Ayuda a disminuir el ruido.

Seleccionar estas opciones ayudan a prevenir el ruido y las perturbaciones electromagnéticas (crosstalk).

Figura 33. Configuración de autoruteo

Los comandos en TCL que se muestran a continuación configuran la herramienta exactamente igual que la figura 33.

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

```
setNanoRouteMode -quiet -routeWithSiDriven true
routeDesign –globalDetail
```

Al terminar el autoruteo se obtuvo el *floorplan* mostrado en la figura 34.

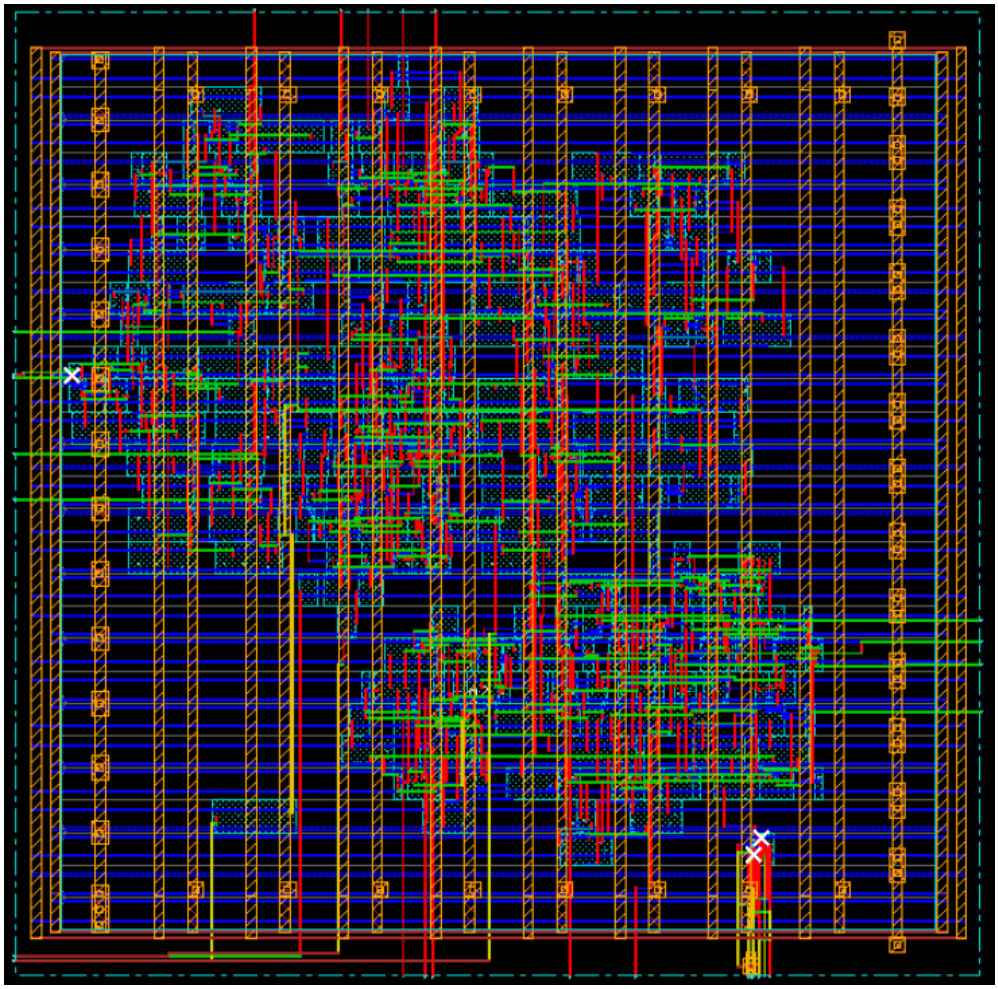


Figura 34. Síntesis física del sistema SerDes

4.1.10. Verificación de la síntesis física

Existen 2 tipos de verificación física que se pueden hacer en Encounter, conectividad y geometría. Los comandos en TCL que se muestran a continuación configuran la herramienta para realizar estas dos verificaciones.

```
# connectivity and geometry verifications
verifyConnectivity -type all -error 1000 -warning 50
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 -warning 50
```

```
verifyGeometry
```

4.1.11. Exportación de GSDII

Finalmente una vez que el diseño esta completo y no contiene errores se exportó un archivo GDSII. Desde el menú File → Save → GDS/OASIS se abrió la ventana que se muestra en la figura 35, donde se asigno un nombre para el archivo de salida y se asoció un archivo *.map* el cual contiene la información de los metales usados en el diseño (este archivo debe ser compatible con los metales en virtuoso, si se desea importar el diseño).

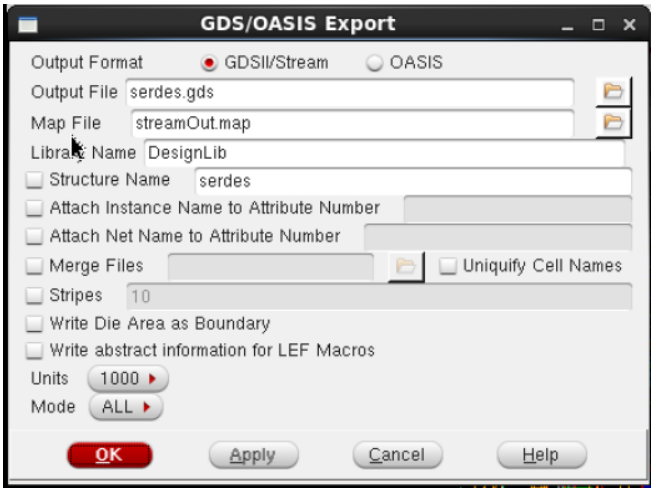


Figura 35. Exportación de GSDII

Es importante mencionar que los archivos .LEF reales de los módulos analógicos del transmisor y el receptor no se obtuvieron a tiempo por lo que no se pudieron integrar al sistema SerDes. Sin embargo se creó un archivo .lef ficticio tomando en cuenta sus entradas y salidas así como una geometría aproximada a los modelos reales, esto con el fin de simular su presencia en la síntesis física. Los archivos .lef de estas celdas se pueden encontrar en el Apéndice G.

CONCLUSIONES

En este documento se presentó el diseño de un sistema integrado SerDes para el protocolo PCI Express en tecnología de 180nm de IBM. El sistema SerDes está compuesto por un transmisor analógico, un receptor analógico, un Serializador digital, un Deserializador digital.

En este documento se muestran los pasos seguidos para recorrer el flujo de diseño de circuitos integrados VLSI. Primero se presentó la descripción a nivel RTL para el modelo Serializador en Verilog. Posteriormente se realizó la síntesis lógica usando Encounter RTL Compiler y el kit digital de la tecnología cmrf_7sf el cual incluye una librería de celdas estándar. Se presentaron los resultados de las pruebas funcionales del módulo Serializador a nivel RTL y a nivel compuerta. Se realizó también la verificación del proceso de síntesis mediante la herramienta Encounter Conformal Checker.

Se presentó el procedimiento para integrar el módulo Serializador al sistema SerDes usando la herramienta Encounter Digital Implementation. Se logró generar un layout a partir del netlist de sistema generado usando las mismas técnicas que en el módulo Serializador. La colocación y el ruteo del sistema se hicieron de forma automática incluyendo celdas analógicas como cajas negras.

Es importante mencionar que aún quedan áreas de oportunidad para la continuación de este trabajo. Las celdas analógicas del sistema SerDes se deben aislar durante la creación de la malla de potencia para evitar conexiones incorrectas. Es necesario agregar al diseño el *PAD frame* que conecta el core de diseño con las entradas y salidas del chip. Finalmente es necesario generar un archivo .map para traducir el layout de geometrías de EDI a un layout compatible con Virtuoso para no perder información de los metales.

REFERENCIAS

[1] A. W. P.A.Franaszek, «"A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code",» *Journal of research and development,* vol. 27, nº 18-8646, p. 5, 1983.

[2] A. Athavale y C. Christensen, “High-Speed Serial I/O Made Simple A Designers’ Guide, with FPGA Applications", 1.0 ed., San Jose, CA: Xcell Publications, 2005, pp. 26-28.

[3] I. Xilinx, *8b/10b Encoder v5.0 Product Specification,* Xilinx, Inc, 2004.

[4] R. Budruk, D. Anderson y S. Tom, PCI Express System Architecture, J. Winkles, Ed., Colorado Springs: Addiso-Wesley Developer's Press, 2003, pp. 9-65.

[5] C. Benz, «Chuck Benz ASIC and FPGA Design,» [En línea]. Available: http://asics.chuckbenz.com/. [Último acceso: 2015 06 16].

[6] E. Castorena, *Logic Synthesis, comunicaión privada,* Tlaquepaque, 2015.

[7] E. Castorena, *Physical Design, comunicación privada,* Tlaquepaque, 2015.

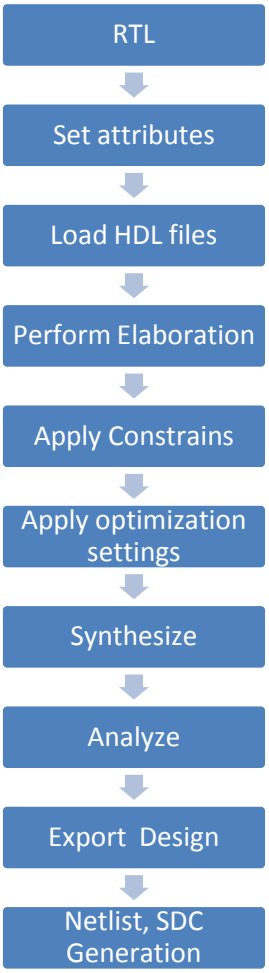
[8] B. Zimmer, «Introduction to the Custom Design Flow: Building a standard cell,» 04 2013. [En línea]. Available: http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee241_s13/Assignments/lab3-stdcell.pdf. [Último acceso: 10 08 2015].

[9] IBM Systems and Technology Group, *CMOS 7RF (CMRF7SF) 1.8V (12-Track) Standard Cell Databook Foundry IP,* Hopewell Junction, NY: IBM, 2010.

[10] «Wikipedia,» 2 10 2015. [En línea]. Available: https://es.wikipedia.org/wiki/PCI-Express. [Último acceso: 25 11 2015].

APENDICES

Apéndice A. Flujo de la síntesis lógica [6]



Apéndice B. Script para flujo de la síntesis lógica.

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

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

set DESIGN serializer
set SYN_EFF medium
set MAP_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH outputs
set _REPORTS_PATH reports
set _LOG_PATH logs/tcl_logs_${DATE}
##set ET_WORKDIR <ET work directory>
set_attribute lib_search_path {.
/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_ce
ll_20111130/std_cell/v.20111130/synopsys/nom} /
set_attribute hdl_search_path {. ../rtl} /

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

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


#set_attribute library <libname>
set_attr library {
    PnomV180T025_STD_CELL_7RF.lib
}
## PLE
set_attribute lef_library
/opt/libs/IBM_PDK/cmrf7sf/relAM/cdslib/CMRF7SF_Digital_Kit/ibm_cmos7rf_std_ce
ll_20111130/std_cell/v.20111130/lef/cmos7rf_6AM_tech.lef
## Provide either cap_table_file or the qrc_tech_file
##set_attribute cap_table_file <file> /
#set_attribute qrc_tech_file <file> /
##generates <signal>_reg[<bit_width>] format
#set_attribute hdl_array_naming_style %s\[%d\] /
######

#####################################################################
## Load Design
#####################################################################
#read_hdl <hdl file name(s)>
read_hdl -v2001 {

    serializer.v
}

set_attribute hdl_track_filename_row_col true
```

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

check_design -unresolved

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

read_sdc ../constraints/serdes.sdc

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


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

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

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

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


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

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

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

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

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



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


synthesize -to_generic -eff $SYN_EFF
write_hdl > ${DESIGN}_generic.v
puts "Runtime & Memory after 'synthesize -to_generic'"
timestat GENERIC
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_generic.rpt
generate_reports -outdir $_REPORTS_PATH -tag generic
summary_table -outdir $_REPORTS_PATH




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

synthesize -to_mapped -eff $MAP_EFF -no_incr
puts "Runtime & Memory after 'synthesize -to_map -no_incr'"
timestat MAPPED
report datapath > $_REPORTS_PATH/${DESIGN}_datapath_map.rpt

foreach cg [find / -cost_group *] {
  report timing -num_paths 10 -cost_group [list $cg] >
$_REPORTS_PATH/${DESIGN}_[basename $cg]_post_map.rpt
}
generate_reports -outdir $_REPORTS_PATH -tag map
summary_table -outdir $_REPORTS_PATH



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

## ungroup -threshold <value>

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

```
## Incremental Synthesis
################################################################################
##########################

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

puts "Runtime & Memory after incremental synthesis"
timestat INCREMENTAL

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


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

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


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


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

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


#################################
### write_do_lecl
#################################
```

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

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

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

##quit
```

Apéndice C. Banco de prueba para el módulo Serializador

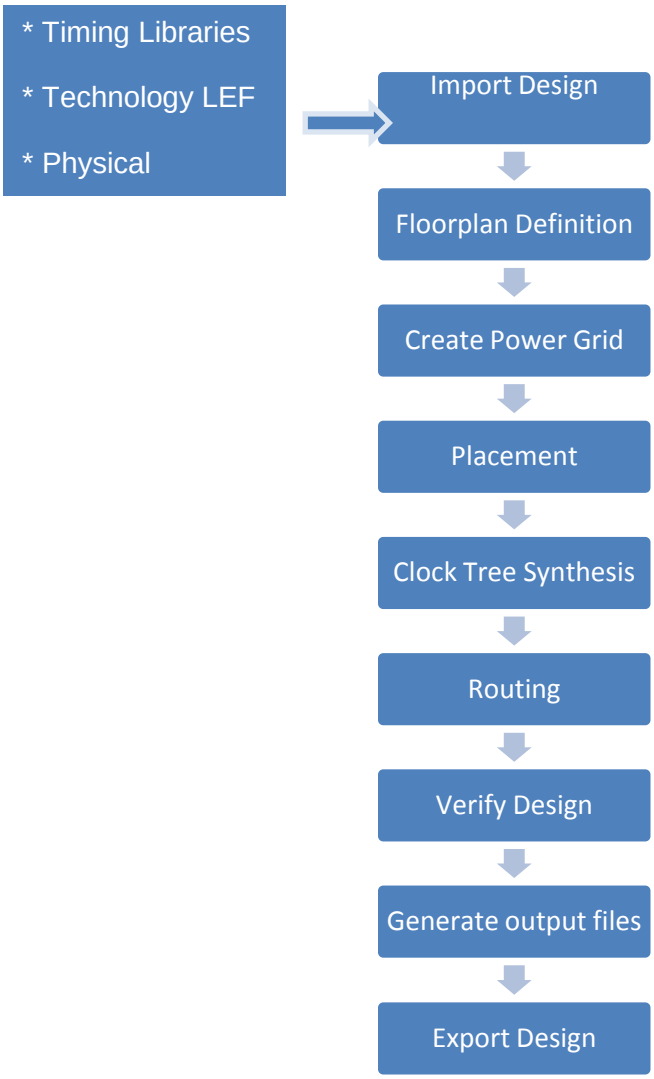
```
`timescale 1ns / 100ps
module tb_serializer();

reg[9:0]   data_in;
reg        clk;
reg        rst;
reg        dispout;

wire       ser_out;
wire       dispin;

integer    i;

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

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

// parallel input generation
Initial begin
    data_in <= 10'b0;


    for (i = 0; i < 1024; i = i + 1)
               parallel(i[9:0]);


end

task parallel(input[9:0] data);
begin
  @(posedge clk);
       data_in <= data;
   repeat (10)
       @(posedge clk);

end
endtask


//
initial begin
  dispout = 1'b0;

  @(negedge rst)
  @(posedge clk);

  #10 dispout = 1'b1;
  #20 dispout = 1'b0;

end


/* //Process for checking
```

```
always (*) begin
      if (enable) begin
            datain_DUT = serializer_inst.par_in;
            dataout_DUT = serializer_inst.ser_out(ser_out);

       end


       else begin

         end

end */

integer error;

always @(posedge rst) begin
    @(negedge rst);

    error = 0;
    if (serializer_inst. tx_frame_started) begin
       $display("Error Enable shouldn't be 1 when reset deasserts");
          error = error + 1;
       end

       @(posedge clk);
       @(posedge clk);
       if(!serializer_inst. tx_frame_started) begin
          $display("Error : Enable shouldn't be 0 on the first clock cycle after
reset");
          error = error + 1;
       end
       repeat (9)
           @(posedge clk);
                if(serializer_inst. tx_frame_started) begin
                       $display("Error : Enable shouldn't be 1 during 9 clock
cycles");
          error = error + 1;
       end

    @(posedge clk);
        if(!serializer_inst. tx_frame_started) begin
                  $display("Error : Enable shouldn't be 0 after 10 clock cycles");
          error = error + 1;
       end
end

//Serializer DUT instance
serializer serializer_inst(
.clk(clk),
.rst(rst),
.par_in(data_in),
.ser_out(ser_out),
.disparity_q(dispin),
.disparity_d(dispout))  ;

endmodule
```

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

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

current_design serdes

set_clock_gating_check -setup 0.0

set_wire_load_mode "enclosed"

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

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

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

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

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

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

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

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

```
    -edge_shift [list $ref_clk_period $ref_clk_period $ref_clk_period]

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

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

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


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

clock_uncertainty -setup 3.0 [get_clocks ref_clk]

clock_uncertainty -hold 1.0 [get_clocks ref_clk]

set_clock_latency -source 10 [get_clocks ref_clk]


set_input_delay -name IDelay -clock clocks_out_reg_0 100 [all_inputs]

set_output_delay -name ODelay -clock clocks_out_reg_0 100 [all_outputs]

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

set_max_transition 200 [current_design]

set_max_fanout 10 [current_design]

set_load 40 [all_outputs]
```

Apéndice E. Flujo de síntesis Física [7]



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

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

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

```
setPlaceMode -congEffort auto -timingDriven 1 -modulePlan 1 -clkGateAware 1 -
powerDriven 0 -ignoreScan 1 -reorderScan 0 -ignoreSpare 1 -placeIOPins 1 -
moduleAwareSpare 0 -checkPinLayerForAccess {  1 } -preserveRouting 0 -
rmAffectedRouting 0 -checkRoute 0 -swapEEQ 0
setPlaceMode -fp false
placeDesign -prePlaceOpt

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

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

# connectivity and geometry verifications
verifyConnectivity -type all -error 1000 -warning 50
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 -warning 50
verifyGeometry
```

Apéndice G. LEF para celdas análogicas

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

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

END analogue_receiver

MACRO analog_transmitter
    CLASS CORE ;
    FOREIGN analog_transmitter -0.28 -0.28 ;
    ORIGIN 0 0 ;
    SIZE 4.48 BY 6.72 ;
    SYMMETRY X Y ;
    SITE CORE ;
    PIN Data
        DIRECTION INPUT ;
        ANTENNAMODEL OXIDE1 ;
```

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

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


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

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



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

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