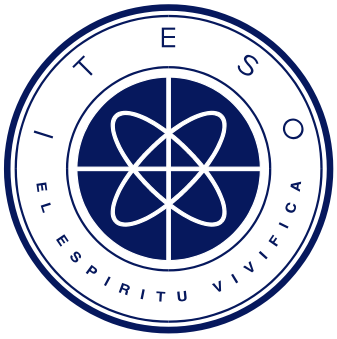
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN DISEÑO DE SISTEMAS EN CHIP



DISEÑO DE RECUPERADOR DE DATOS Y RELOJ ADAPTIVO A JITTER

Tesina para obtener el grado de:

ESPECIALISTA EN DISEÑO DE SISTEMAS EN CHIP

Presentan: Néstor Damián García Hernández

Director: Victor Avendaño Fernández
Manuel Salim Maza

San Pedro Tlaquepaque, Jalisco. 1 de Agosto de 2018.

Agradecimientos

Quisiera agradecer a Conacyt por el apoyo económico que me permitió realizar esta investigación.

Además, agradezco el tiempo invertido de mis asesores Manuel Salim y Victor Avendaño, pues sin sus correcciones y recomendaciones no habría obtenido el diseño y los resultados presentados.

Asímismo, agradezco a mi familia y a mi novia pues su apoyo incondicional y comprensión fue vital en este periodo.

Por último quiero reconocer al ITESO y a sus profesores por los conocimientos brindados, los cuales fueron esenciales para completar el flujo de diseño de este sistema en chip.

Abstract

This document approaches the development of a clock and data recovery (CDR) adaptive to jitter module working at 800MHz, which is part of a 130nm BiCMOS technology mixed signal System on Chip. The whole system on chip design flow is described and includes the logic design, logic synthesis, physical synthesis, verification and generation of files to manufacture it by MOSIS service. The CDR module uses 8 clock signals generated by an integrated PLL module on the system on chip, and also it recovers data generated by an LFSR module or by an external source. The development was done by using Verilog language and the Cadence toolset, which involved RTL Compiler, EDI Encounter and Virtuoso. On the other hand, the verification of the design was done on Simvision tool.

The successful recovery of clock and data results are presented through simulation signals and the achievement of the whole design flow is reported along with the required files to replicate them. Also conclusions and the suggested future work is included in the final chapter.

Resumen

Este documento se enfoca en el desarrollo de un módulo recuperador de reloj y datos (CDR) adaptivo al jitter operando a 800MHz, que será parte de un Sistema en Chip de señal mixta con tecnología BiCMOS de 130nm. El flujo de diseño completo de un sistema en chip es descrito e incluye el diseño lógico, la síntesis lógica, síntesis física, verificación y generación de archivos para manufacturarlo por MOSIS. El módulo CDR usa 8 señales de reloj producidas por un módulo PLL integrado al sistema en chip y recupera con éstas los datos generados por un módulo LFSR también integrado, o bien por una señal externa.

El desarrollo del módulo se realizó en lenguaje Verilog junto con las herramientas CAD de CADENCE que involucran RTL Compiler, EDI Encounter y Virtuoso. Por otro lado, la verificación fue realizada mediante la herramienta Simvision.

Los resultados de la recuperación de datos y reloj son presentados a través de señales de simulación y, tanto la evidencia del flujo completo de diseño como de los archivos necesarios para replicarlo, son incluidos. Además, las conclusiones y el trabajo a futuro sugerido están incluidos en el capítulo final.

Contenido

Lista de imágenes

Introducción

En los últimos años se ha visto una mejora en el nivel de densidad de transistores que puede contener un circuito integrado, tal como lo predijo la ley de Moore. Sin embargo, ante la madurez de la industria de computadoras personales y el creciente mercado de dispositivos móviles, la dinámica o el paradigma del desarrollo de dispositivos semiconductores ha tenido que cambiar. La ley de Moore en esta era está más definida por la habilidad de poder integrar no solo más transistores con mejor rendimiento, sino más funcionalidades, usando el mínimo consumo de potencia, área y costo posible, en un solo chip. Esto ha presentado nuevas oportunidades para nuevos actores dentro de este mercado, pero también han surgido nuevos retos.

Uno de los enfoques que se ha desarrollado para lidiar con la creciente complejidad de integración de los circuitos integrados es el desarrollo de la metodología de diseño de los denominados Sistemas en Chip.

Un Sistema en Chip consiste en la integración de varios bloques de hardware analógicos, digitales o de señal mixta (analógicos y digitales), que realizan diferentes funciones (por ejemplo administración de energía, procesadores de audio y video, GPS, de radio comunicación, etc.) y se encuentran en un solo sustrato de silicio, o chip, y en conjunto cumplen con una sola función específica. Al estar destinados a un solo propósito, estos están más personalizados y pueden ser más eficientes en términos de área ocupada, energía de alimentación requerida y velocidad alcanzada.

La importancia en el mercado de la metodología de los Sistemas en Chip ha ido aumentando y consolidando, por lo que se ha observado que grandes empresas tecnológicas han adquirido a otras empresas especializadas en este campo para mejorar su portafolio. Tal ha sido el caso de Apple, que desde el 2008 adquirió a PA-semiconductor, que le ha permitido diseñar sus propios chips especiales para dispositivos como el iPod, el iPhone y el iPad (Merritt, 2008) y con ellos poder incluir procesadores de imágenes y controladores de almacenamiento, que han servido para ofrecer características como el modo ráfaga al tomar fotografías, o bien funciones como Apple Pay o el escáner de huellas dactilares (Businessweek, 2016). Por otro lado, Qualcomm adquirió a Atheros en el año 2011 para mejorar su portafolio de soluciones inalámbricas; a Attansic

Technology por sus chips de tecnología Fast Ethernet, o bien a Summit Technology que poseía una gran experiencia en el desarrollo de chips dedicados a la administración de energía. (Qualcomm, 2012)

Los sistemas en chip, además de encontrarse en procesadores digitales de señales, en sistemas embebidos, en decodificadores de video, o en teléfonos inteligentes, son usados en los sistemas modernos de comunicación a alta frecuencia. Dada la relevancia de los últimos, en esta generación se ha elegido implementar la metodología de desarrollo de un sistema en chip de señal mixta que puede tener aplicación en dicho campo y éste integrará los bloques de hardware funcionales de la Figura 1, enlistados a continuación:

- **Módulo Phase Locked Loop (PLL)**, que a la vez posee los siguientes bloques funcionales:
  - **Detector de fase (PFD):** módulo de señal digital.
  - **Filtro de lazo cerrado invariante el tiempo (Charge pump):** módulo de señal analógica.
  - **Oscilador controlado por voltaje (VCO):** módulo de señal analógica.
- **Módulo recuperador de datos y reloj adaptivo al *jitter* (CDR):** módulo de señal digital.
- **Módulo generador de datos pseudo aleatorios (LFSR):** módulo de señal digital.

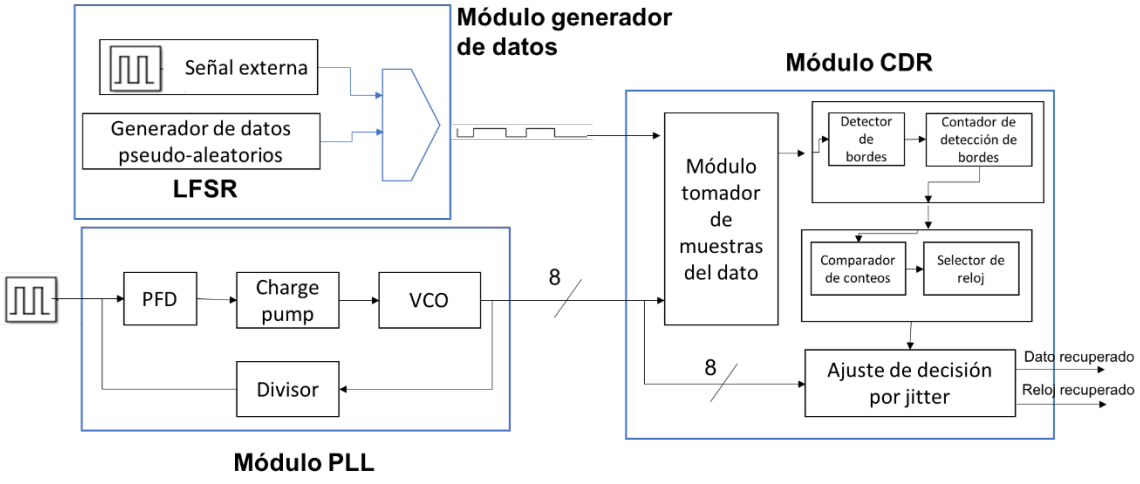


Figura 1: Diagrama bloques de Sistema en Chip a desarrollar

El emplear la metodología de diseño de un sistema en chip permite partir la complejidad de un circuito muy grande, entre diferentes diseñadores. Así pues, el desarrollo del sistema en chip

planteado anteriormente se llevó a cabo entre 6 diseñadores y, al finalizar el flujo de diseño de cada módulo, estos se integraron.

En este documento se detallará a profundidad el desarrollo de la metodología de diseño del módulo funcional "Recuperador de Reloj y Datos Adaptivo al *Jitter*" (referido como CDR), que corresponde a un módulo de señal digital. Se abordará la arquitectura del diseño, su descripción RTL, su síntesis lógica y física, así como la verificación lógica mediante simulación.

Por otro lado, dado que las señales de datos a alta velocidad son susceptibles a efectos como el *jitter* y el ruido, los cuales no pueden ser ignorados, debe existir una compensación ante estas variaciones en las señales, de tal forma que las tasas de errores al recuperarlas sean mínimas. Es por esto, que el circuito CDR diseñado incluye la característica de adaptación al *jitter*, que le permite ser resistente a dicho efecto.

Este circuito CDR empleará 8 señales de reloj a 800MHz desfadas una de otra 45º, que son generadas por el módulo PLL embebido en el sistema en chip. Dichas señales de reloj serán empleadas para muestrear al dato de entrada, el cual podrá provenir ya sea de un módulo generador de datos pseudo aleatorios, que también es parte del sistema en chip, o bien de una señal externa.

La descripción del módulo CDR se realizó en lenguaje Verilog 2011 y fue sintetizado y verificado usando la tecnología de celdas estándar BiCMOS de 130nm con las herramientas de diseño asistido por computadora (CAD) de Cadence. Por último, la fabricación del sistema en chip se realizará mediante MOSIS situado en California, EUA.

1. Antecedentes

En la generación 2016 de la especialidad de diseño de sistemas en chip del ITESO, se implementó un módulo deserializador que empleaba un recuperador de reloj y datos (CDR), el cual generaba internamente las 8 señales de reloj que usaba para muestrear la señal del dato de entrada. Dicho circuito se implementó en lenguaje verilog con tecnología de 130nm, empleando herramientas CAD de Cadence (Rivas-Villegas, 2016).

Por otro lado, dada la relevancia de los circuitos PLL en la industria, hay una gran cantidad de empresas que se dedican a la fabricación de estos dispositivos, entre las cuales las principales son:

- Texas Instruments.
- Analog Devices Inc.
- NXP.
- IDT (Integrated Device Technology).
- Microchip.
- ON Semiconductor.
- Maxim Integrated.
- STMicroelectronics.
- Semtech.
- NJR.
- Microsemi.
- Skyworks.
- Qorvo.
- Exar.
- Silicon Laboratories
- Parallax.
- Atmel.
- Cyoress Semiconductor.
- Pericom.

- Everspin Technologies.
- Intersil.
- Cirrus Logic.

En cuanto a los dispositivos CDR, existen algunos dispositivos de empresas como Analog Devices, Keysight, MACOM o Semtech que permiten la recuperación del reloj de los datos.

Keysight Technologies posee varios CDRs que funcionan para señales de entrada entre 50Mbaudios a 32Gbaudios, en sus modelos N1076A/N1077A o de 125Mbaudios a 64Gbaudios en los modelos N1076B/N1078A (Figura 2) y recuperan el reloj hasta de 16GHz (Keysight, 2018).

Figura 2: Dispositvo CDR que funciona para señales de entrada con velocidades hasta de 64Gbaudios

La empresa Analog Devices posee los circuitos integrados CDR ADN29xx que pueden recibir datos de entrada de 8.5Gbps a 11.3Gbps y brindan tarjetas de evaluación (Figura 3) para probar dicha funcionalidad (Analog Devices, 2017).

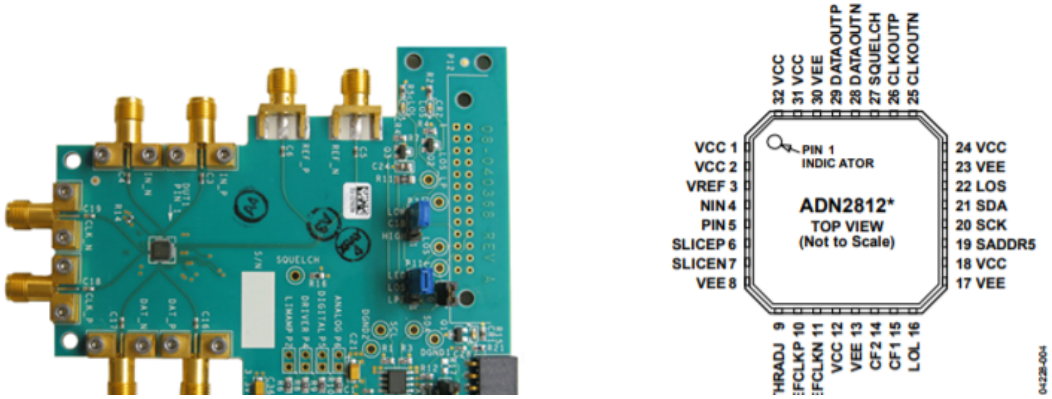


Figura 3: Tarjeta de evaluación de los circuitos ADN29xx (izquierda) y ADN2812 (derecha) son algunos ejemplos de sistemas de recuperación de datos y reloj de Analog Devices

Hay otras empresas como Gennum que poseen dispositivos CDRs y enuncian que operan con señales entre 9.95Gb/s a 11.3Gb/s con el modelo GN2003S, los cuales están enfocados a comunicaciones mediante fibra óptica (Gennum, 2007).

Otros ejemplos de CDR comerciales son los circuitos M21011-12/M21012-12 (Figura 4) que operan entre 42Mbps a 3.2Gbps y 1Gbps a 3.2Gbps, respectivamente (Macom, 2015).

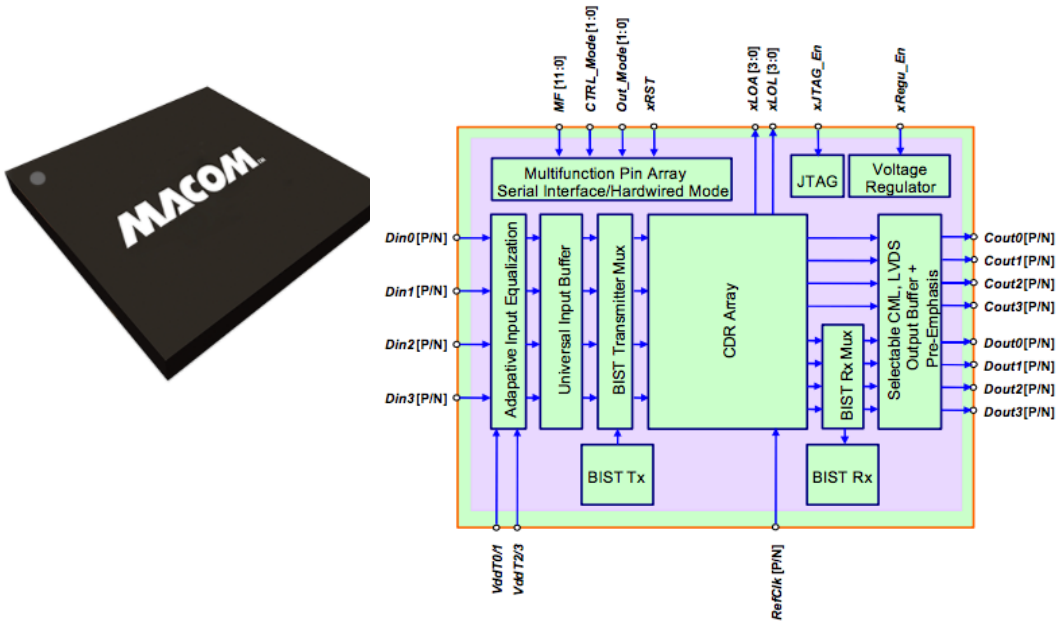


Figura 4: Dispositivo CDR M21012 de MACOM que puede operar con frecuencias hasta de 3.2Gbps

2. Marco teórico

2.1. Módulo recuperador de datos y reloj (CDR)

En la actualidad existen dispositivos electrónicos de comunicación y consumibles que requieren interfaces seriales de alta velocidad para transferir información. En estos enlaces la información enviada por la parte transmisora puede no contener una señal de reloj, y en estos casos a pesar de no poseer un reloj, el receptor debe hallar una forma de poder recibir la información e interpretarla correctamente. Así pues, a partir de la información asíncrona recibida, se debe generar un reloj con el cual se haga correctamente el muestreo e interpretación del dato recibido.

El circuito en el receptor que se dedica a generar este reloj se denomina recuperador de reloj. Para este propósito, el recuperador necesita crear una señal de referencia que tenga aproximadamente la misma frecuencia que el reloj original, y que esté alineada con las transiciones de la señal de dato, es decir, con los cambios de 0 a 1 o de 1 a 0.

Una vez que se generó la señal de reloj, se podrá muestrear el dato de entrada con la seguridad de que se podrá interpretar de forma correcta la información pues ya se conocerá su periodicidad, y a este proceso se le denomina recuperación de reloj y datos. (Nagaria, 2017)

Una de las arquitecturas empleadas para los sistemas CDR incluye un circuito PLL y un interpolador de fases. Un interpolador de fase es un circuito que, a partir de una señal de referencia, puede crear nuevas señales espaciadas uniformemente de 0 a 360˚. En el sistema en chip propuesto, el PLL es quien genera la señal de referencia, que será usada por el interpolador de fase para entregar 8 fases de reloj espaciadas 45˚ una de otra. Estas señales son usadas por el CDR para realizar la recuperación del reloj y datos. La explicación detallada de la generación de las señales del interpolador está fuera del alcance de este documento.

Las 8 señales generadas por el interpolador de fase son usadas para detectar las transiciones en la señal asíncrona de entrada, de tal forma que una de estas señales se alinee con la transición del dato de entrada como se observa en la Figura 5. En ella se aprecia que la séptima señal (clk_p270) está alineada con la transición de la señal de entrada (data).

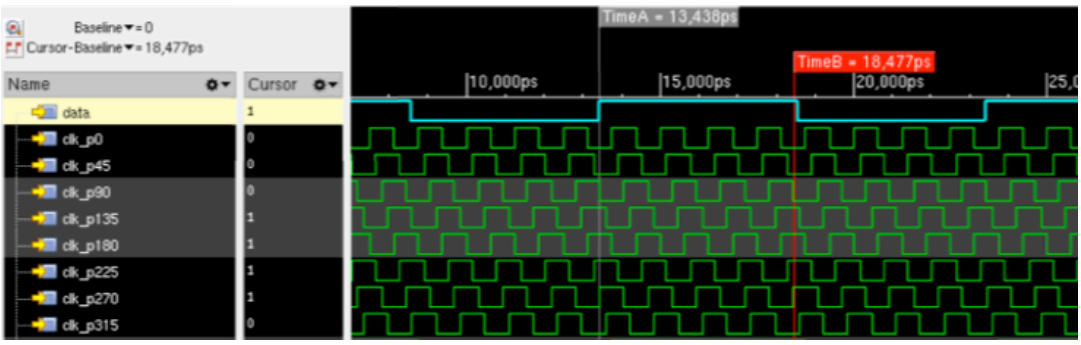


Figura 5: Señales generadas por el circuito PLL-interpolador de fase

El propósito de un CDR como ya se mencionó, es determinar la frecuencia aproximada de la señal de entrada, por lo que es útil detectar qué fase es la que está mejor alineada con las transiciones del dato de entrada, pues esta dará información sobre su periodicidad.

Es importante remarcar que se debe conocer la frecuencia de operación máxima del dato de entrada para poder determinar la frecuencia que deberán tener las señales generadas por el circuito PLL-interpolador, pues dado que se realizará un muestreo con ella, se debe cumplir con el teorema de Nyquist. Éste establece que una señal se puede reconstruir correctamente si se obtienen capturas o muestras de ella a una frecuencia 2 veces más grande que el ancho de banda de la señal a capturar. (Proakis & Manolakis, 1996)

Para la implementación considerada en este sistema en chip, la señal asíncrona de dato tendrá una frecuencia máxima de 200MHz, y el PLL-interpolador generará señales a 800MHz, con lo cual se realizará un sobre muestreo pues por cada ciclo de la señal de entrada, habrá 4 ciclos de las señales de reloj de referencia de 800Mhz, lo cual respeta el teorema de Nyquist.

Sin embargo, a pesar de poder determinar una señal de reloj alineada con la transición del dato, no es recomendable usarla para la recuperación del dato en un caso que no sea ideal, pues debido a efectos indeseados como el *jitter*, las transiciones de datos de alta velocidad son propensos a sufrir distorsión en el tiempo.

2.2. Jitter

El *jitter* es definido como una variación en fase de la posición ideal en el tiempo que debería tener una señal como se muestra en la Figura 6. Se observa que respecto a la forma de onda ideal con un periodo T, una señal con *jitter* realiza transiciones ya sea antes o después del periodo ideal.

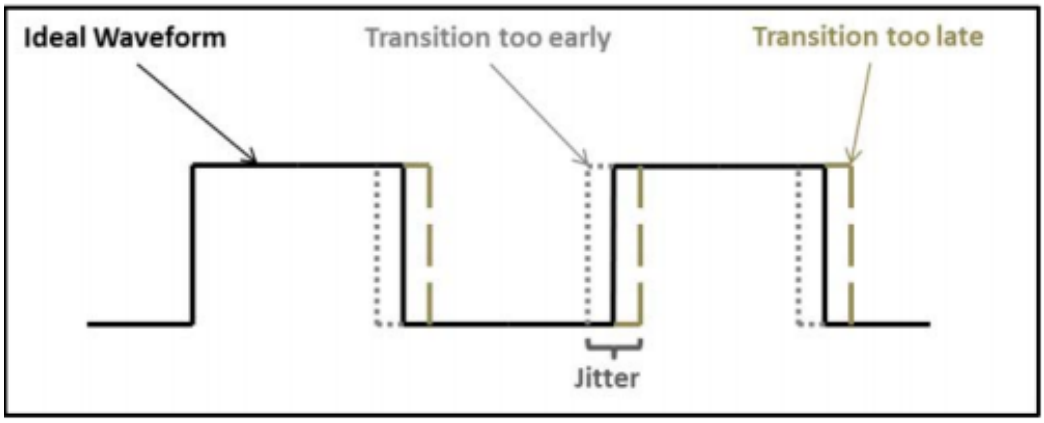


Figura 6: Definición de jitter (Bidaj, Begueret, & J., 2017)

En transmisiones de alta velocidad de las interfaces seriales, el *jitter* es un factor que no se puede descartar ya que afecta la tasa de error de bit (BER), siendo ésta una medida confiabilidad del dato recibido y corresponde a la probabilidad de tener un error al determinar el nivel de un bit recuperado como se muestra en la Figura 7. (Bidaj, Begueret, & J., 2017) Además, reduce el tiempo que las señales lógicas tienen para propagarse en los circuitos.

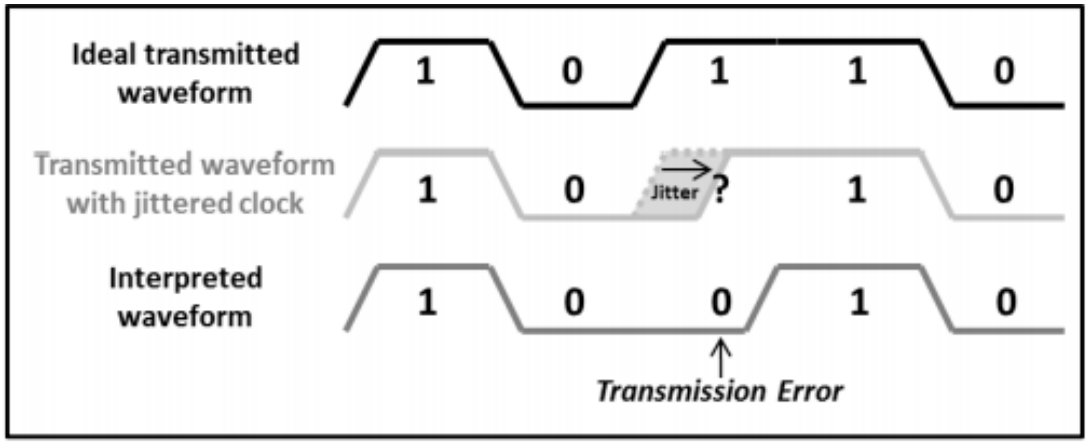


Figura 7: Señal en donde se recupera erróneamente la información debido al jitter (Bidaj, Begueret, & J., 2017)

Dado que el jitter es un efecto indeseable en las señales digitales, se debe considerar alguna técnica que permita contrarrestarlo o adaptarse a él.

2.3. Flujo de diseño de un Sistema en Chip

El flujo de diseño de un sistema en chip se ha segmentado en diversos pasos que ha permitido crear equipos de trabajo especializados en cada uno de ellos. De esta forma se ha particionado la complejidad del diseño de un circuito con alta densidad de integración de componentes. Principalmente este flujo de diseño se puede dividir en 2 áreas denominadas comúnmente, *front-end* y *back-end*.

2.3.1 Front-end

Las etapas que se consideran *front-end* están ilustradas en la Figura 8, y comienzan con el diseño de las "especificaciones del sistema", es decir, se detallan todas las características que deben cumplir los módulos que incluirá el sistema en chip. Posteriormente, con base en los requerimientos del sistema, se realiza el diseño de la "micro arquitectura", en la cual se definen las unidades funcionales que contendrá el sistema. Para esto es posible utilizar bloques abstractos que describan a grandes rasgos como estarán conectados unos con otros.

La siguiente etapa es el "diseño lógico" que consiste en describir como estarán construidas las unidades funcionales, es decir, definir si emplearán lógica combinacional, lógica secuencial, y la jerarquía de estos bloques. Cuando ya se tiene el diseño lógico, el siguiente paso es emplear un lenguaje de descripción de hardware (HDL por sus siglas en inglés) como VHDL ó Verilog con el que se abstraen estos bloques funcionales de hardware a código, que facilita los siguientes pasos del flujo de diseño. A esta descripción abstracta con lenguaje HDL se le llama "descripción Register transfer Level (RTL)".

Cuando ya se tiene lista la descripción RTL, es necesario verificar la funcionalidad lógica del diseño, lo cual se consigue con simulaciones en herramientas como Simvision de Cadence, Modelsim de Menthor Graphics o ISE de Xilinx. Si los resultados de la simulación son satisfactorios, se puede seguir con los siguientes pasos de la metodología de diseño, de lo contrario se debe revisar y posiblemente rediseñar el diseño lógico.

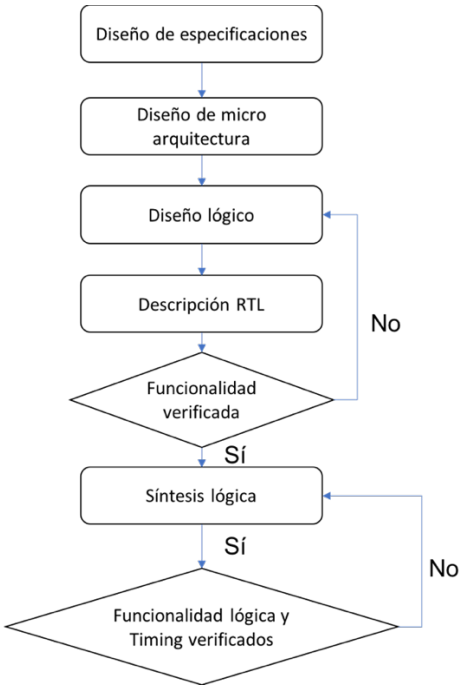


Figura 8: Front-end de la metodología de diseño de un sistema en chip.

Cuando la descripción RTL está verificada funcionalmente, se realiza la etapa de "síntesis lógica" mostrada en la Figura 9, que consiste en transformar la descripción del módulo hecha con lenguaje HDL a una representación del mismo circuito con las compuertas lógicas fundamentales (AND, OR, NOT, flip flops, buffer, sumadores, etc.), así como las interconexiones entre ellas. Las compuertas lógicas fundamentales que se emplean para esta etapa son denominadas *celdas estándar,* cuyas propiedades de fabricación, tales como timing, área, valores de capacitancias, o consumo de energía, son bien conocidas y permiten que los diseñadores del *front-end* se enfoquen solo en la funcionalidad lógica y no en la implementación física del diseño (*back-end*). Además, poseen dimensiones fijas (alto y ancho), que facilitan colocarlas en filas y aceleran el proceso de diseño de layout en herramientas automatizadas de diseño.

Para dicha etapa se definen una serie de restricciones con los valores de frecuencia a usar, valores de skew, retardos en señales de entrada y salida, etc., los cuales ayudan a refinar la elección de las compuertas para realizar la transformación del RTL a compuertas lógicas. En la misma etapa, las herramientas de síntesis lógica pueden realizar una optimización en las funciones lógicas implementadas en nuestro diseño.

Las compuertas lógicas resultantes son genéricas o ideales, es decir no tienen ninguna información de parámetros físicos como capacitancias, retardos dependientes de la temperatura, tamaños, etc., los cuales son necesarios para poder predecir de mejor forma el comportamiento de nuestro diseño cuando ya esté fabricado. Por esto, se realiza un mapeo o conversión a compuertas con estos parámetros, cuya información viene en librería proveídas por los fabricantes, son dependientes de la tecnología a usar y serán elegidas dependiendo las restricciones que se hayan definido. Para este proyecto se usarán las librerías de la tecnología BiCMOS 8hp de 130nm de GLOBALFOUNDRIES.

Lo siguiente es insertar un "circuito scan" que sirve para poder hacerle pruebas al circuito fabricado y verificar su funcionamiento. Sin embargo, para este proyecto, está fuera del alcance.

Finalmente se obtiene un "gate level netlist" que es verificado en cuanto a funcionamiento lógico, como en análisis estático de tiempo, y en caso de no pasar satisfactoriamente dichas pruebas, se debe volver a las etapas anteriores para corregirlo.

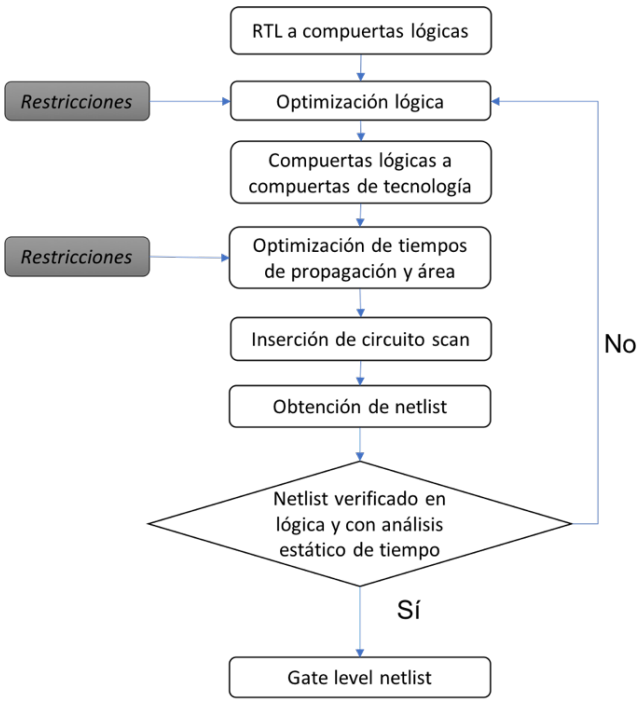


Figura 9: Metodología de síntesis lógica

2.3.2 Back-end

Después de realizar la síntesis física y obtener un *netlist* a nivel compuerta (*gate level netlist*), se realiza la síntesis física que consiste en los pasos apreciados en rojo en la Figura 10.

Se comienza realizando una planeación de donde se colocará cada uno de los módulos del sistema en chip. Además, se agregan anillos de alimentación y líneas de conexiones a alimentación, verticales y horizontales. Después se colocan las celdas estándar que fueron elegidas de acuerdo al netlist proveniente de la síntesis lógica, lo cual genera de forma automatiza un *layout*. Se realiza un enrutamiento de las celdas estándar y se verifican resultados de análisis de tiempos de propagación. Si estos resultados presentan *slack* negativo para los tiempos de *setup* y *hold*, se puede hacer una optimización para mejorarlos. En caso de seguir teniendo errores, se puede optar por rediseñar el circuito y volver a generar un *netlist* de él; de lo contrario, se realiza una serie de validaciones al *layout* generado y, finalmente, se exporta el diseño a formato GDS para mandarlo a fabricar.

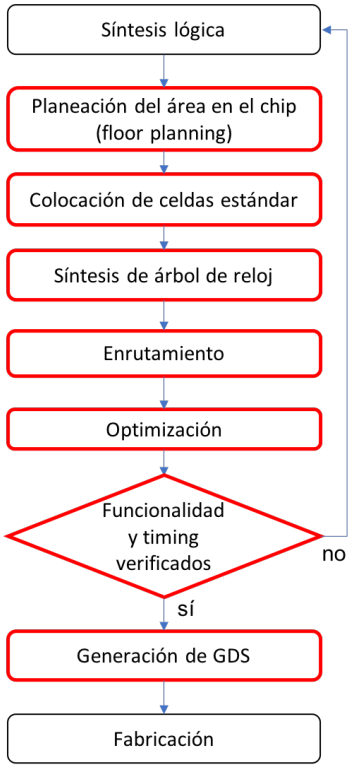


Figura 10: Pasos correspondientes al Back-end. Esta etapa también se conoce como diseño físico e incluye síntesis

3. Implementación de metodología de diseño de un sistema en chip

3.1. Microarquitectura del módulo recuperador de datos y reloj

La microarquitectura del sistema en chip que se realizará se presenta en la Figura 11. Se aprecia que contiene 3 bloques principales que corresponden al módulo PLL, al módulo generador de datos pseudo aleatorio (LFSR) y el módulo recuperador de datos y reloj (CDR). El que se detallará con profundidad en este documento será el módulo CDR.

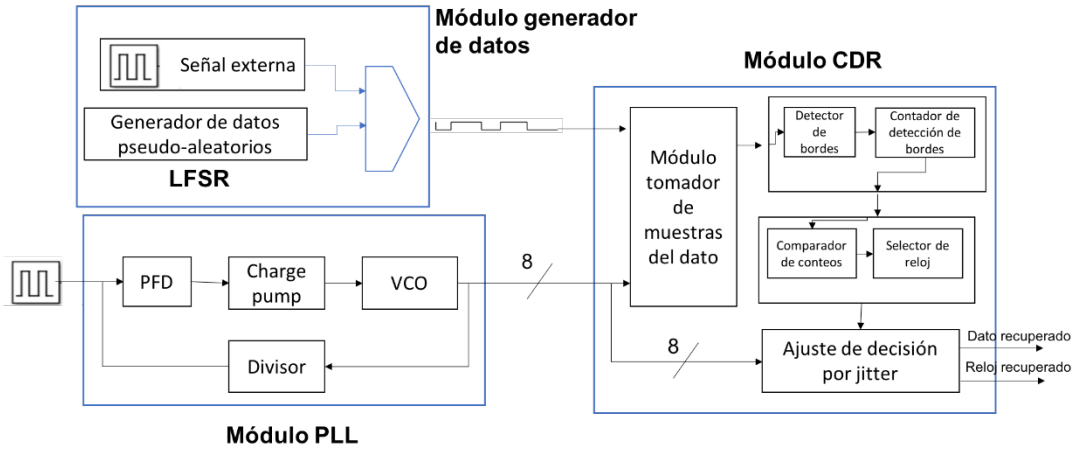


Figura 11: Micro arquitectura del sistema PLL-CDR-LFSR.

3.1.1 Etapa de muestreo

El módulo CDR recibirá una señal de datos seriales, con frecuencia de operación de 200MHz, que serán generados ya sea por el módulo LFSR embebido en el sistema en chip, o bien por una señal externa. Dicha señal es recibida por un bloque que realiza un muestreo de ella mediante el uso de 8 señales de reloj a 800MHz generadas por el módulo PLL. Dado que se están usando 8 señales de reloj, se generarán 8 muestras desfasadas una de otra 45º. Estos desfasamientos ayudarán después a detectar las transiciones en la señal. Por ejemplo, en la Figura 12 se despliegan las 8 señales de reloj desfasadas (señales "clk_px"), así como la señal del dato (color azul), y se puede percibir que para este caso que el reloj desfasado 270º detecta antes que las demás una transición de 0 a 1, y también la transición de 1 a 0 del dato.

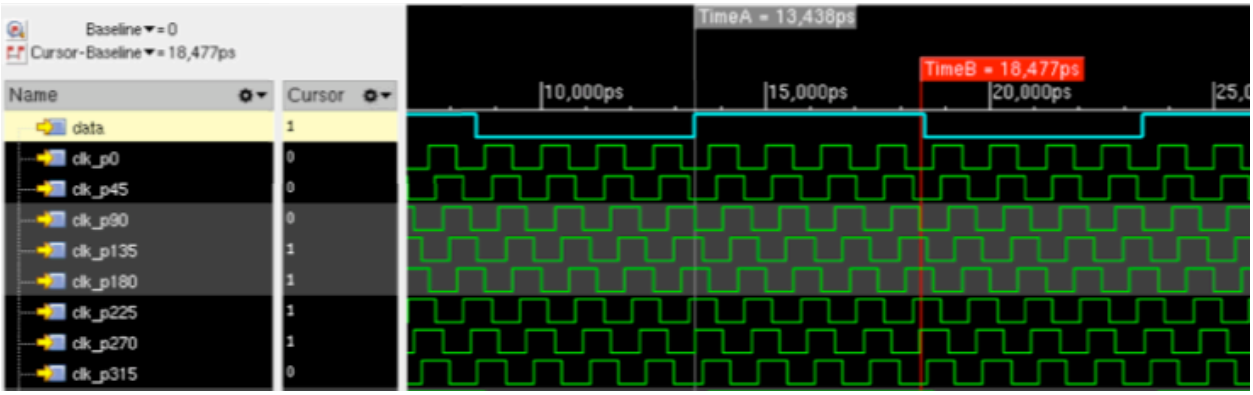


Figura 12: Formas de onda de detección de transiciones.

3.1.2 Detector de bordes

Se necesitará un módulo que compare las 8 muestras y genere un pulso para la muestra que capturó antes que las demás la transición o borde. Este módulo será el bloque "detector de bordes".

3.1.3 Etapa de conteo de detecciones de bordes

Por otro lado, cuando se haya ubicado un borde, una etapa de contadores acumulará la detección que haga cada muestra en una ventana análisis de 16 datos.

3.1.4 Comparador de conteos y selector de mejor reloj para muestreo

Cuando ya transcurrieron los 16 datos (bits), cada uno de los 8 contadores tendrá diferentes cuentas dependiendo de las detecciones de bordes que hayan hecho. Estos conteos son comparados para detectar cuál tuvo el mayor valor, es decir la mayor cantidad de detecciones. Al determinar cuál fue el mayor valor de conteo alcanzado, también se obtendrá la señal de reloj que produjo dichas detecciones, lo cual podrá sugerir que dicha señal de reloj es la más adecuada a usar para detectar los bordes en la información de entrada.

3.1.5 Módulo adaptivo al jitter

Dado que no se puede ignorar el efecto que tiene el *jitter* en la señal de dato de entrada, debemos contar con un módulo que compense las variaciones que habrá a lo largo del tiempo en ésta, pues conforme cambie el valor de *jitter* en dicha señal, la decisión que haya tomado el selector de reloj para muestreo, se deberá actualizar.

3.2. Diseño lógico

3.2.1 Etapa de muestreo

La implementación de la microarquitectura se describirá en esta sección. El primer bloque detallado en la microaquitectura es el "módulo muestreador". Este módulo se compone de dos etapas de 8 *flip flops* cada una como se aprecia en la Figura 13.

En la primera etapa de *flip flops*, todos reciben la señal de dato proveniente del módulo LSFR, pero cada uno de ellos recibe distinta señal de reloj. Cada señal de reloj proviene del módulo PLL y estarán desfasadas 45º una de otra.

La segunda etapa de *flip flops* sirve para sincronizar las muestras obtenidas en la primera etapa de *flip flops*, ya que se obtuvieron con 8 diferentes señales de reloj. En esta segunda etapa, los *flip flops* del 1 al 7, usarán como señal de reloj al reloj desfasado 315º; mientras que el 8vo *flip flop* usará una señal generada a partir de negar el reloj desfasado 180º. Lo anterior se realizó para permitirle a este último *flip flop* capturar el dato proveniente del *flip flop* 8 de la etapa 1 sin tener violaciones en el tiempo de *setup*.

Además de esos *flip flops*, se agregó otro más con el propósito de almacenar una muestra anterior del 8vo *flip flop* de la segunda etapa.

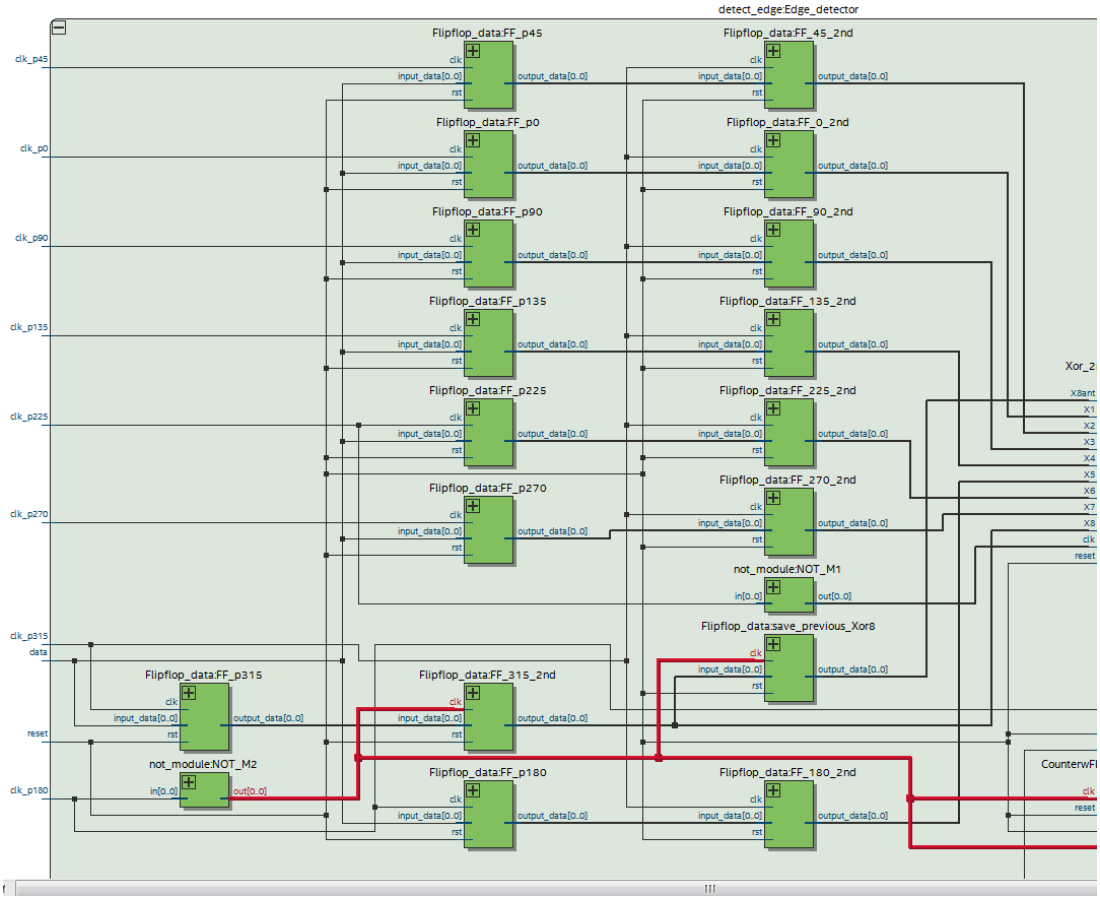


Figura 13: Etapa de muestreo.

3.2.2 Detector de transiciones

Después de las etapas de *flip flops* se tiene un módulo que compara las señales provenientes (Figura 14) de la segunda etapa de *flip flops*, mediante compuertas Xor, de la siguiente manera:

1. La muestra anterior del 8vo flip flop con la muestra del 1er flip flop.
2. La muestra del 1er flip flop con la del 2do.
3. La muestra del 2dor flip flop con la del 3er.
4. La muestra del 3er flip flop con la del 4to.
5. La muestra del 4to flip flop con la del 5to.
6. La muestra del 5to flip flop con la del 6to.
7. La muestra del 6to flip flop con la del 7mo.
8. La muestra del 7mo flip flop con la del 8vo.

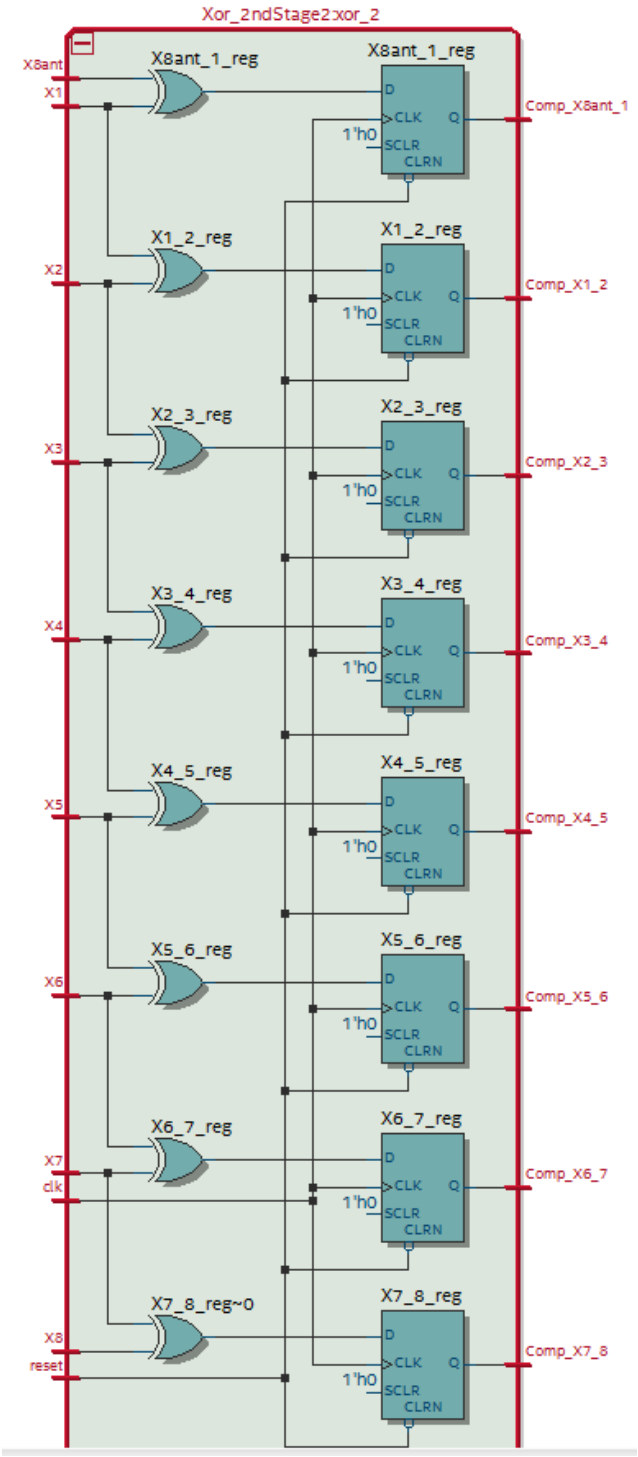


Figura 14: Módulo detector de transiciones.

La comparación de las señales se realiza con el propósito de identificar con qué señal de reloj se detectó mucho antes que las demás la transición de estado lógico del dato de entrada. Estas comparaciones se realizan de forma síncrona con cada flanco positivo de una señal de reloj

generada al negar el reloj desfasado 225º. Se emplea el de 225º negado para que no se presenten violaciones de *setup*, pues la información de entrada a este bloque se tiene actualizada con cada pulso del reloj de 180º negado que ocurre antes que la de 225º. Además se usan *flip flops* a las salidas como estrategia para permitir que la información tenga tiempo suficiente para propagarse

En el instante de comparación (flanco de subida de reloj de este módulo), las muestras de los *flip flops* podrían tener ceros o unos, dependiendo del instante en el que capturaron el dato de entrada. Así pues, al realizarse una comparación vertical entre ellas con compuertas Xor, las señales que tengan la misma información se descartarán y, a la vez, los pulsos generados indicarán cuál fue la primera señal en detectar una transición y en qué instante. Para ilustrar mejor el concepto se agrega la Figura 15, en donde se puede notar que las señales verdes contienen las muestras del dato de entrada, y se observa que a partir de la muestra de 90º (señal Prev_sample_Ph90) se identifica un 1 lógico. Las demás señales también lo identifican, pero esa información será redundante por lo que se puede descartar. Así pues, al comparar la muestra de 90º contra la de 45º se generará un pulso, ya que en ese instante estas señales son diferentes. Dicho pulso se muestra en la señal en color azul "Xor_2_3" en el instante donde está el marcador rojo.

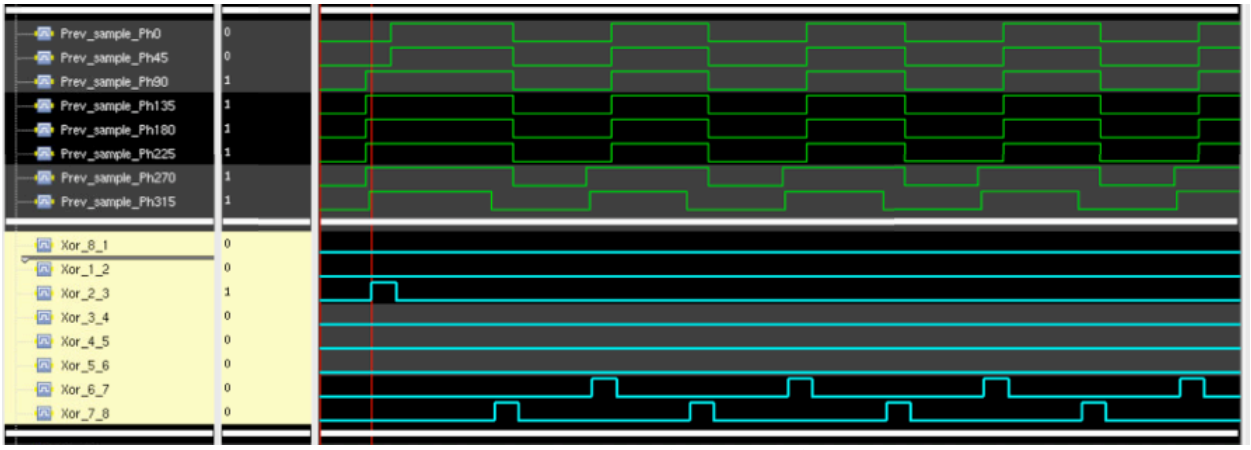


Figura 15: Generación de pulsos en los instantes de transiciones.

3.2.3 Etapa de conteo de detecciones de bordes

Los pulsos generados por la etapa de detección de transiciones, son procesados por una etapa de contadores (Figura 16). Estos detectan cuando se genera un pulso por el módulo de Xors y ante estos pulsos, un conteo se aumenta en una unidad. Existen 8 contadores y cada uno es asignado a una salida del bloque detector de transiciones.

Además, se agregó también un contador de 16 datos seriales de entrada, el cual dictará la ventana de conteo o captura de detección de transiciones. Así pues, los 8 contadores de transiciones acumularán conteos mientras transcurren 16 datos solamente, después de esto serán reiniciados por otro bloque generador de un pulso de reinicio.

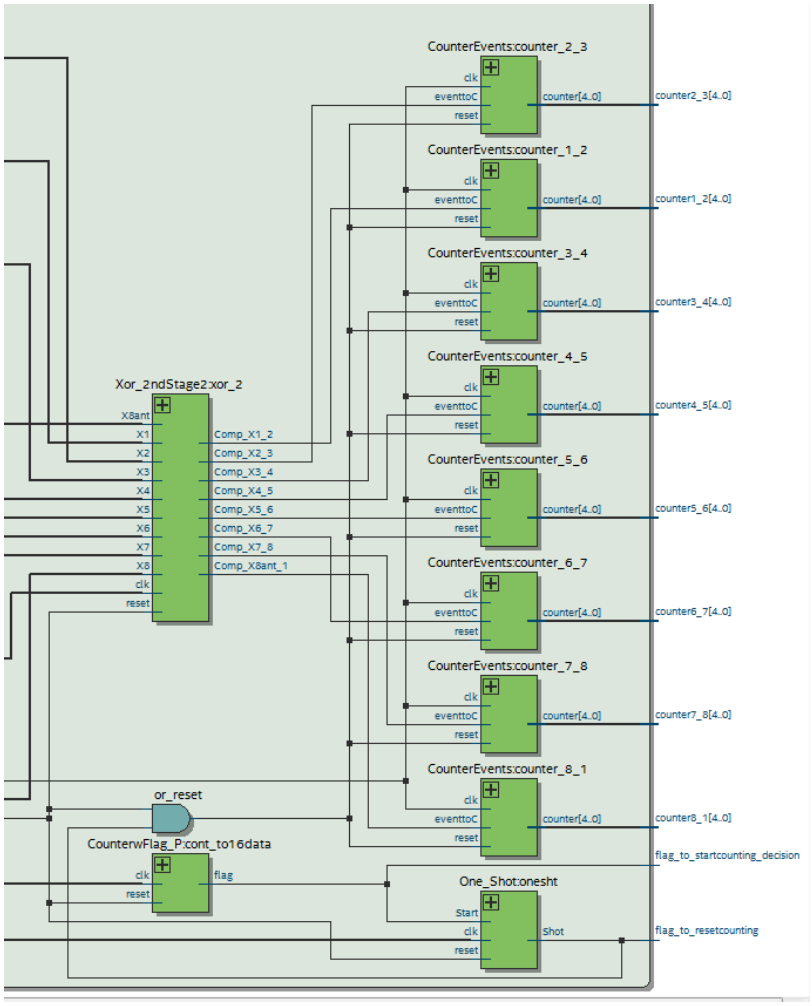


Figura 16: Etapa de contadores de transiciones

3.2.4 Módulo selector de reloj

Posterior a la etapa de conteos, se encuentra una etapa de *flip flops* (Figura 17 en color azul) que sirven para sincronizar la información del módulo de conteos y transferirla al módulo "selector de fase" (Figura 17 en color rojo). Ambos bloques funcionan de manera síncrona usando de reloj a la señal desfasada 180º negada.

El módulo selector de reloj basa su principio de operación en realizar comparaciones de los conteos que se realizaron en la etapa de contadores. Éste determinará cuál fue el mayor conteo y qué señal lo produjo, dando como salida una señal de 3 bits que indican cuál de los 8 relojes es el que muestrearon mayores transiciones en la señal de dato de entrada. Los bloques involucrados para realizar estas comparaciones y determinar a la señal de reloj más adecuada se pueden apreciar en la Figura 18.

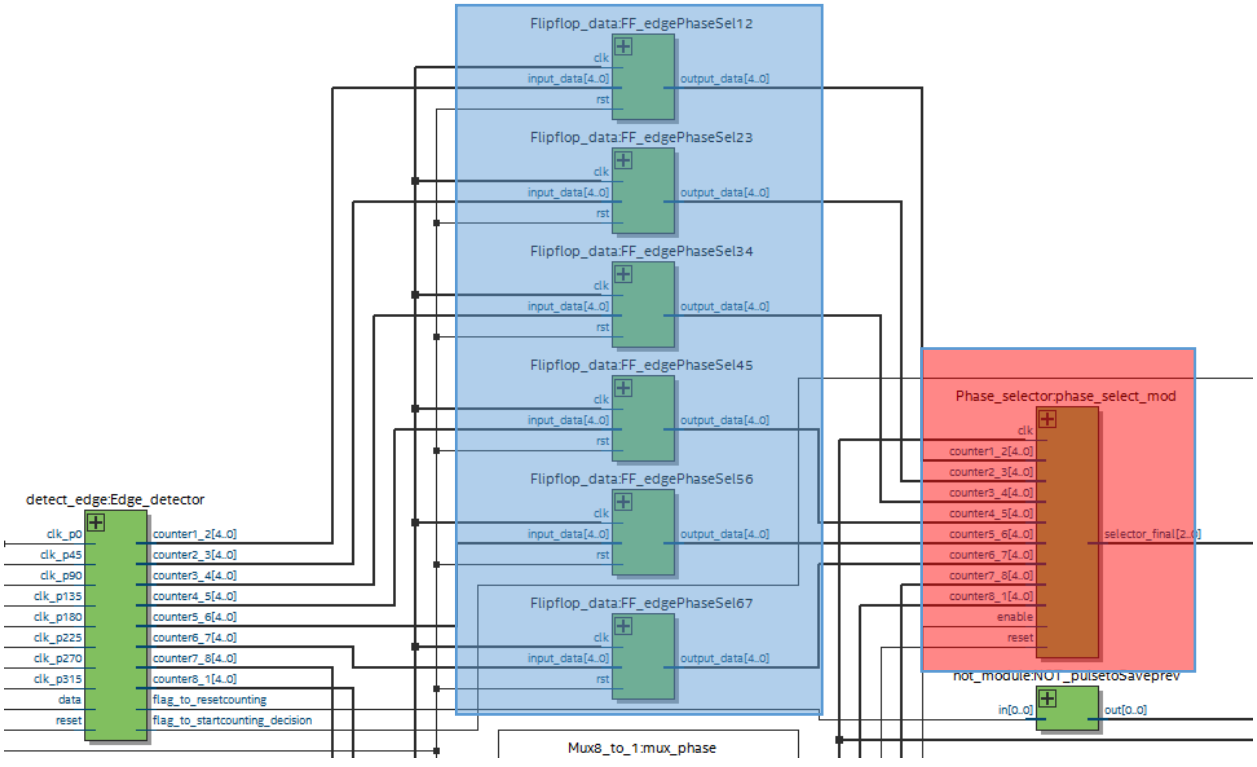


Figura 17: Etapas de flip flops sincronizadores y módulo selector de reloj

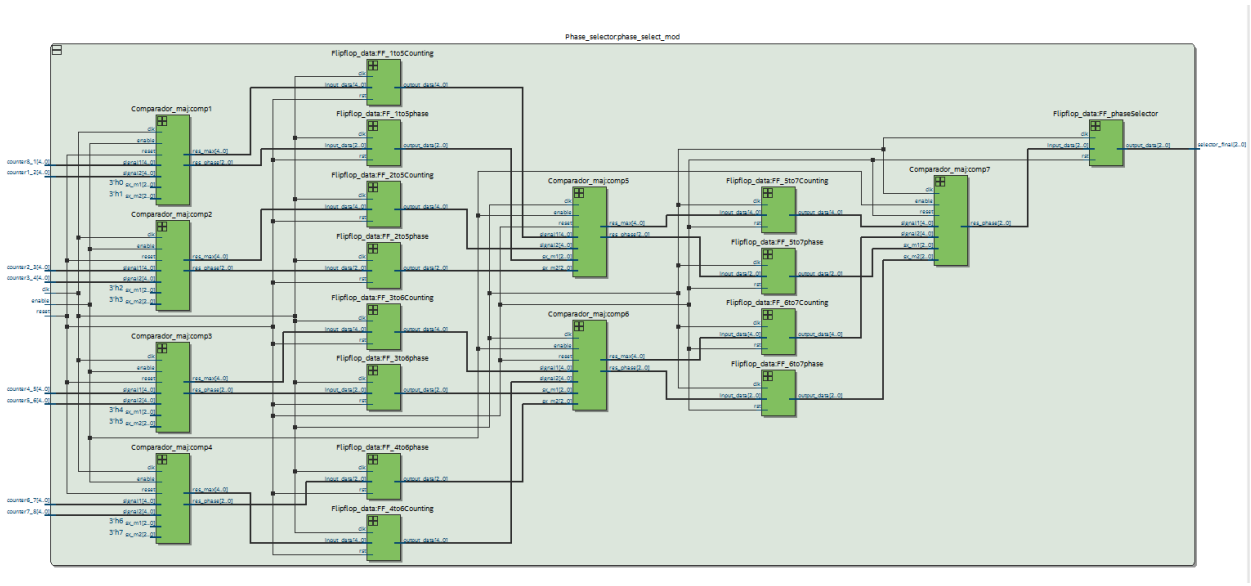


Figura 18: Módulo selector de reloj. Este bloque realiza comparaciones de los conteos de transiciones

3.2.5 Módulo adaptivo a jitter

Ya se mencionó que los efectos de *jitter* en las señales digitales no deben ser ignorados, por lo que para compensar dicho efecto, se propusieron para este CDR los bloques señalados en la Figura 19.

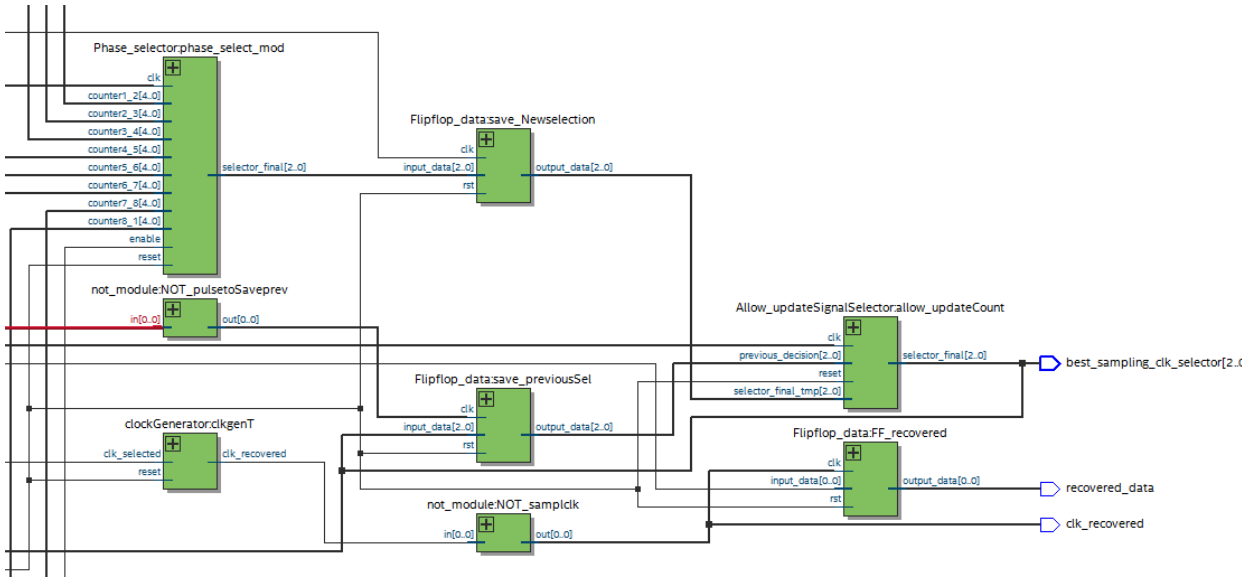


Figura 19: Bloques involucrados en la parte adaptiva del CDR

Una vez que el módulo “selector de reloj” determina cuál es el reloj que detectó más transiciones, tendremos un valor de 3 bits que corresponderá al selector de dicho reloj, y este valor será guardado en un flip flop denominado “save_Newselection”. Este flip flop se actualiza cada que transcurren 16 datos seriales de entrada, justo antes de que los contadores se reinicien.

El valor que se almacenó se considera solo una propuesta de posible nuevo selector de reloj a usar, así pues, no será usado inmediatamente, sino que primero será evaluado por el módulo llamado “Allow_updateSignalSelector”.

El módulo “Allow_updateSignalSelector” es el que dictará cuál será el valor de selector de reloj a usar, el cual considera los efectos de jitter en el dato de entrada. Este módulo procura no realizar actualizaciones en el valor del selector de reloj si no es tan necesario, de acuerdo al siguiente criterio: habrá un “margen de tolerancia” de 2 fases de reloj, (1 hacia adelante y 1 hacia atrás) en el cual no se realizará una actualización.

Este bloque opera de la siguiente forma: recibe una propuesta de una nueva fase con la que se detectaron más transiciones, y también recibe el valor de la última fase seleccionada (guardada en el flip flop llamado “save_previousSel”). Posteriormente, compara ambos valores y decide si tomará la nueva propuesta de fase, o si mantiene la selección previa.

Esta selección la hace comparando qué tan desfasados están ambos valores, y si hay una distancia mayor a 2 fases (1 hacia adelante ó 1 hacia atrás), cambiará; de lo contrario, se mantendrá la decisión tomada anteriormente. Este margen se propuso para evitar realizar demasiadas actualizaciones y evitar así que oscile tanto esa señal selectora. Por ejemplo, si la decisión anterior había sido usar la fase de 0º y después de analizar los 16 datos de entrada se determina que la fase 7 (de 315º) es ahora la mejor fase a elegir, la decisión será no hacer ningún cambio pues la fase 0º está separada 1 fase de la fase 7 como se puede ver en la siguiente secuencia.

Fase 315º	Fase 0º	Fase 45º	Fase 90º	Fase 135º	Fase 180º	Fase 225º	Fase 270º	Fase 315º	Fase 0º

Así pues, cada fase tiene un rango de inmunidad de 2 (1 hacia adelante y 1 hacia atrás), de tal forma que la fase 0º no cambia ante la fase de 315º y 45º. La fase de 45º no cambia ante la fase de 0º y 90º; la fase de 90º no cambia ante la fase de 45º y de 135º, etc. Se propuso este rango para cada una para evitar muchos cambios de decisión que hicieran oscilar a la señal selectora.

3.2.6 Multiplexor de relojes y divisor de frecuencia

Una vez que el módulo “Allow_updateSignalSelector” determina qué selector de reloj es el más adecuado, ese valor se usa como selector en un multiplexor de 8 a 1 (en color rojo en la Figura 20), el cual multiplexa las 8 señales de reloj desfasadas. La salida de este multiplexor indicará cuál es la señal de reloj que detecta antes que las otras 8, a las transiciones del dato de entrada. Sin embargo, este reloj no es el adecuado para realizar un muestreo o recuperación de la señal, pues su flanco de subida estaría muy cercano al flanco de subida del dato de entrada. Así pues, esta señal se acondicionará mediante un divisor de frecuencia de tal forma que obtengamos una señal de reloj cuyo flanco de subida esté al centro del bit de dato de entrada a recuperar.

En la Figura 20 se nota el módulo “clockGenerator” (en color morado en la Figura 20) que realiza la función de divisor de frecuencia. Y una vez que se obtiene esta señal de reloj, se observó que era más adecuado negar esta señal y usar el flanco de subida resultante, por lo tanto se agregó un inversor (en color azul en Figura 20).

Ya que se tiene la señal de reloj más adecuada, un flip flop (en color amarillo en la Figura 20) se encargará de recuperar el dato de entrada con la señal de reloj generada.

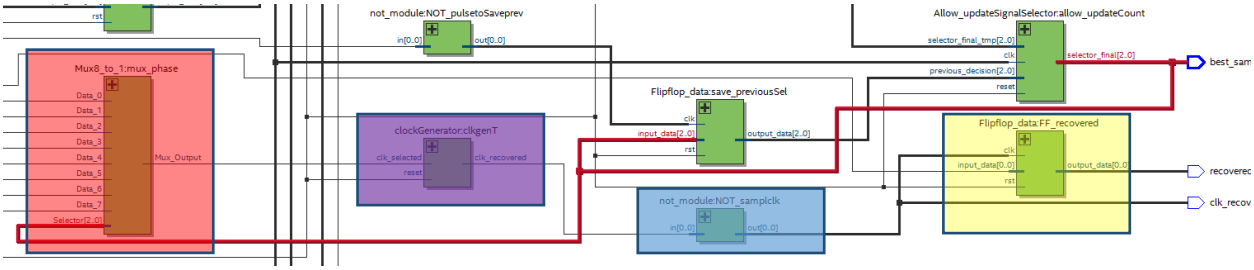


Figura 20: Bloque multiplexor de relojes y recuperación de dato

3.3. Síntesis lógica

Para realizar la síntesis lógica del diseño descrito anteriormente se empleó la herramienta de diseño asistido por computadora RTL Compiler de Cadence. Además de los archivos en verilog del diseño lógico (incluidos en los Apéndices), se empleó un script en lenguaje tcl en donde se especifican qué archivos en verilog se usarán para realizar la síntesis lógica; además se agregan comandos para optimizar el diseño, restricciones para poder elegir las compuertas que se usarán, y librerías de las celdas estándar con información de *timing*, archivos lef, y de puertos de entrada y salida.

El script usado está detallado en el "Apéndice A" y el código en verilog del módulo top se incluyó en el "Apéndice B", y para ejecutarlo se empleó el comando siguiente que dio como resultado la síntesis de la Figura 21:

- rc -gui -f ../CDR.tcl -log CDRlog.log

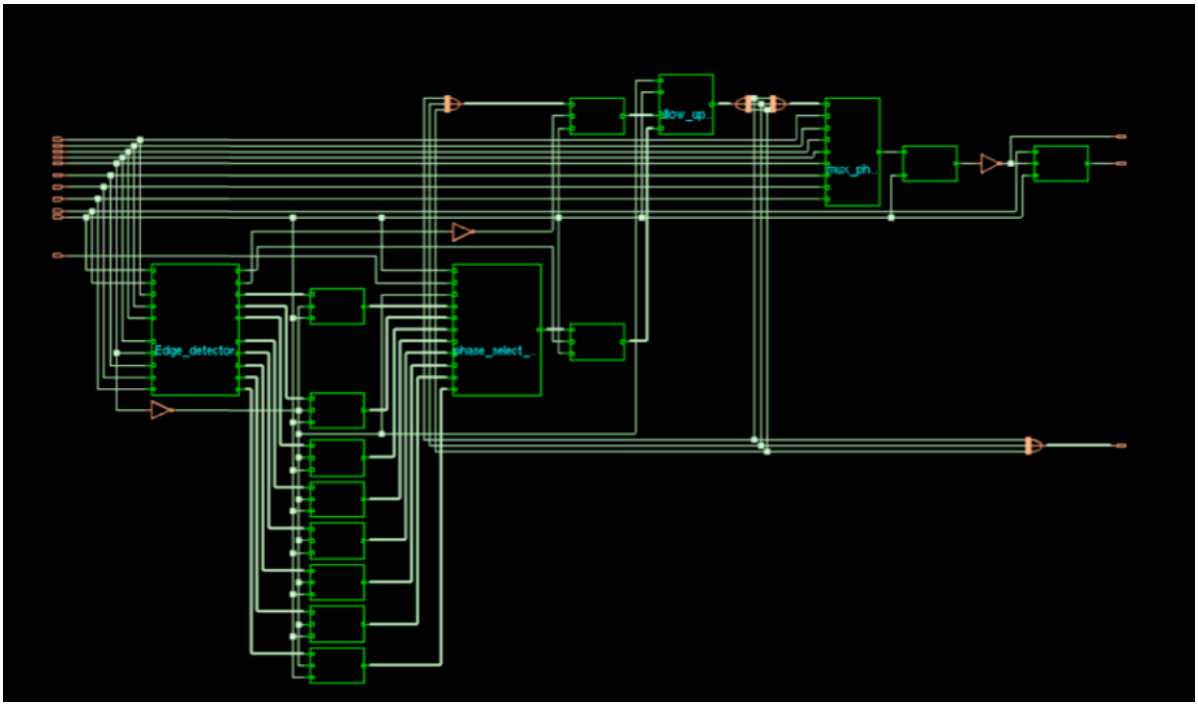


Figura 21: Resultado de la síntesis lógica con RTL Compiler del circuito CDR adaptivo

3.3.1 Etapa de muestreo

De esta síntesis se desprende el bloque detector de bordes cuya estructura se aprecia mejor en la Figura 22; además el código en verilog de los bloques que componen el detector de bordes está plasmado en el "Apéndice C". Éste consta de:

- La 1ra. etapa de muestreo donde el reloj de cada flip flop corresponde a una fase diferente. Vemos en color rojo el reloj del flip flop 1 al 7.
- La 2da. etapa de muestreo, donde el reloj del flip flop 1 al 7 es la fase 8 (de 315º), mientras que el reloj del flip flop 8 es la fase 4 (de 180º) negada. En magenta se ve el reloj proveniente de la señal de 180º negada.
- La etapa de Xor para detectar transiciones.
- La etapa de contadores.
- El módulo contador de 16 datos que reinicia el conteo en la etapa de contadores.

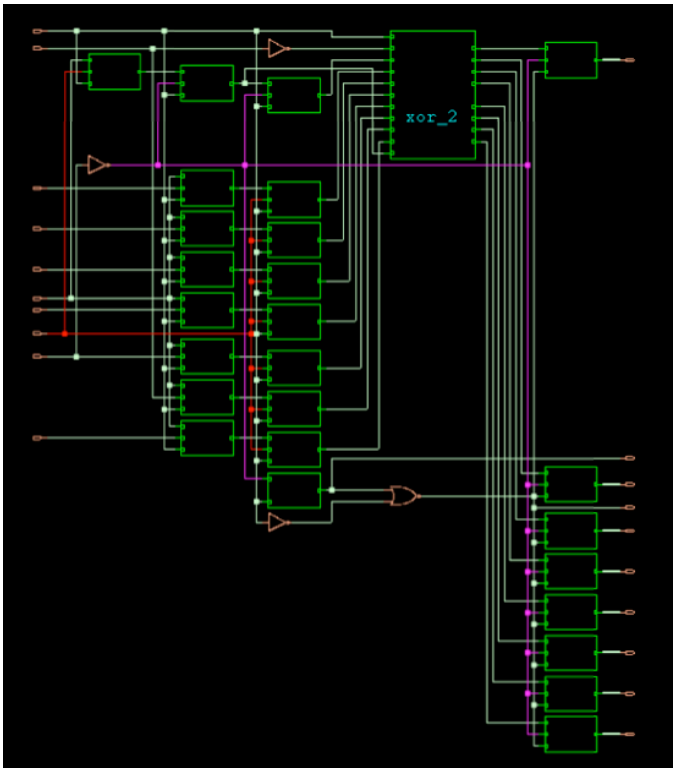


Figura 22: Bloques funcionales de módulo detector de bordes

La primera etapa de muestreo consta de 8 *flip flops*, que capturan a una frecuencia de 800MHz el dato de entrada. Dichas señales de reloj de los *flip flops* están desfasadas 45º una de

otra, teniendo así 0, 45, 90, 135, 180, 225, 270 y 315 grados de desfasamiento. En la Figura 23 se observa el resultado de la simulación en Simvision del *netlist* obtenido, en donde estas señales tienen el nombre "clk_px". Ahora bien, para esta implementación, el dato de entrada posee una frecuencia fundamental de 200MHz; es por eso que se observan 4 ciclos de las señales de muestreo dentro de cada bit del dato (señal "data" en color rojo en la Figura 23). Por otro lado, las señales "New_samplePhx" contienen el muestreo de la primera etapa de *flip flops*. La señal "New_samplePh225", como se observa, detectó antes que las demás, un borde.

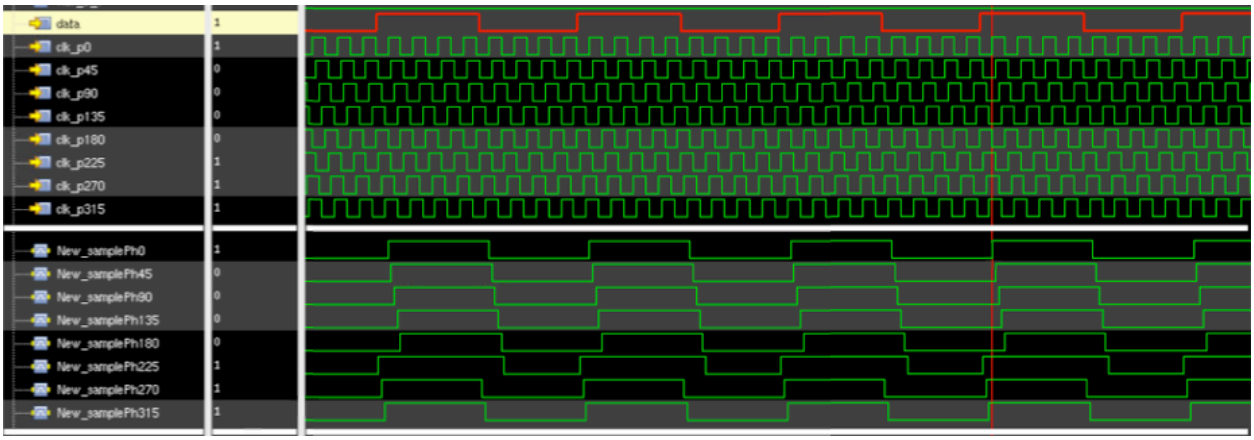


Figura 23: Señales de simulación de primer etapa de muestreo

Ahora bien, es necesario tener sincronizadas estas muestreas para poder compararlas, para lo cual se colocó la segunda etapa de *flip flops*. Parecería una buena opción emplear el reloj de 315º de desfasamiento para transferir los datos de la etapa 1 hacia la etapa 2 de *flip flops*; sin embargo, para el *flip flop* 8 de la etapa 2 no es una buena opción porque no se cumplirían los tiempos de *setup* y *hold*. Así pues, del *flip flop* 1 al 7 de la 2da etapa se emplea como reloj a la señal de 315º, mientras que para el *flip flop* 8 se emplea como reloj a la señal con fase de 180º negada, pues su flanco de subida se ubica después del flanco de subida de la señal de 315º.

En la Figura 24, se observa el resultado del muestreo de la 2da etapa de *flip flops*. En ella se puede notar que en el flanco de subida de la señal "clk_p315" se capturan las muestras del 1 al 7 de la primera etapa de *flip flops*, mientras que en el flanco de subida de la señal "neg180" se captura la muestra del *flip flop* 8 de la primera etapa; así pues, después del flanco de subida del reloj "neg180" ya se tendrían todos los datos que se usarán para las siguientes etapas de procesamiento.

Figura 24: señales de 1ra y 2da etapa de muestreo

3.3.2 Detector de transiciones

Posteriormente, las muestras que provienen de la salida de la segunda etapa de muestreo son procesadas por un módulo de operaciones Xor que se ve en la Figura 25 y cuyo código está en el "Apéndice H".

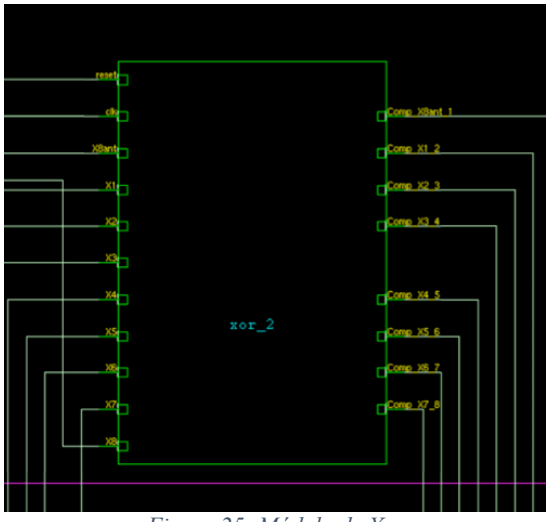


Figura 25: Módulo de Xor

Este módulo recibe las 8 muestras de la 2da etapa de *flip flops*, y también la muestra anterior de la fase de 315 de la 2da etapa. Las operaciones Xor se hacen de la siguiente forma:

- La señal anterior de 315 con la señal de 0º.
- La fase 0 con la de 45º.
- La de 45º con la de 90º.
- La de 90º con la de 135º.
- La de 135º con la de 180º.
- La de 180º con la de 225º.
- La de 225º con la de 270º.
- La de 270º con la de 315º.

El módulo Xor produce las formas de onda mostradas en la Figura 26, donde vemos que la señal "Xor_5_6" tiene un pulso, pues en ese instante hubo una transición de la señal "Prev_sample_Ph180" a la "Prev_sample_Ph225".

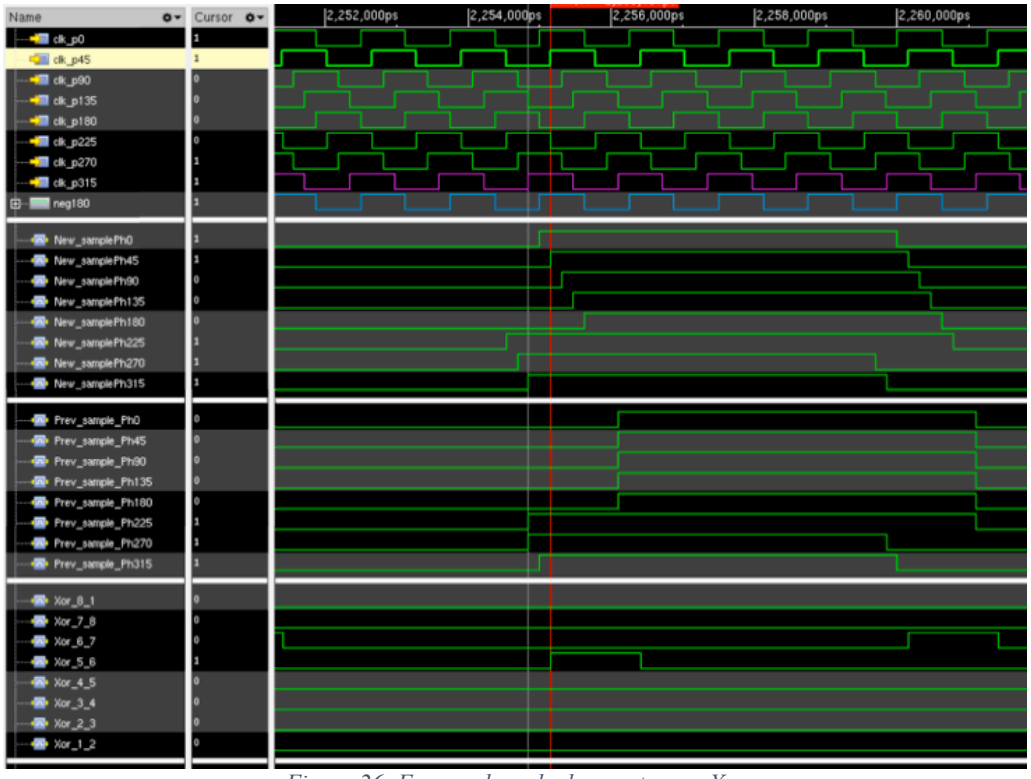


Figura 26: Formas de onda de muestreos y Xor

3.3.3 Etapa de conteo de detecciones de transiciones

Posteriormente, estos pulsos del módulo Xor serán las entradas a la etapa de los módulos contadores, cuyo código en verilog está en el "Apéndice J". En la Figura 27 se muestra cómo con cada pulso de la etapa de Xor se incrementa en 1 el conteo en las señales "counterx_x". Esta etapa lleva el registro de qué señales han producido detecciones de bordes o transiciones de 1 a 0, ó 0 a 1; y después de 16 datos de 200MHz, se genera el pulso "flag_to_startcounting_decision", que servirá para comenzar a analizar qué contador registró el mayor conteo. A su vez este pulso es el detonador del circuito "One shot" que genera un pulso de reset para el reinicio de los contadores en 0, es por esto que se pueden notar momentos en que el conteo de la etapa de contadores se reinicia a 0.

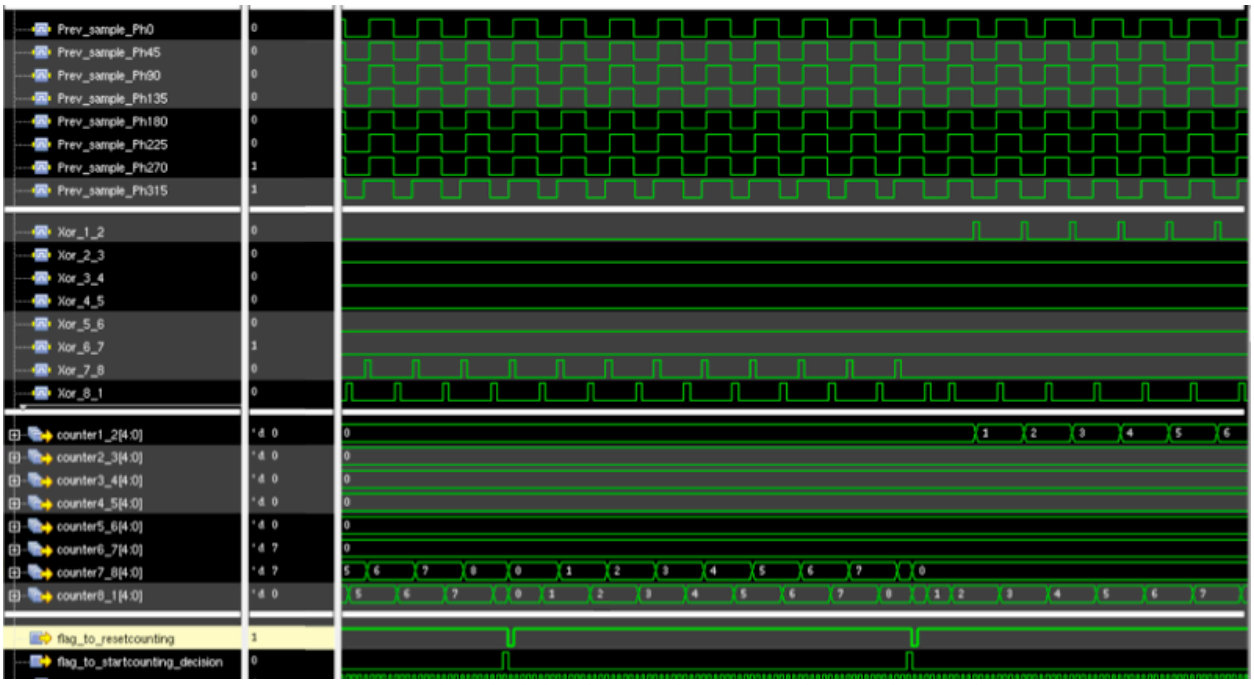


Figura 27: Formas de onda de la 2da etapa de muestreo, de Xor y de contadores

Las señales descritas anteriormente corresponden al circuito detector de bordes que, a su vez, son las entradas a los siguientes módulos del circuito CDR.

3.3.4 Seleccionador de reloj de muestreo

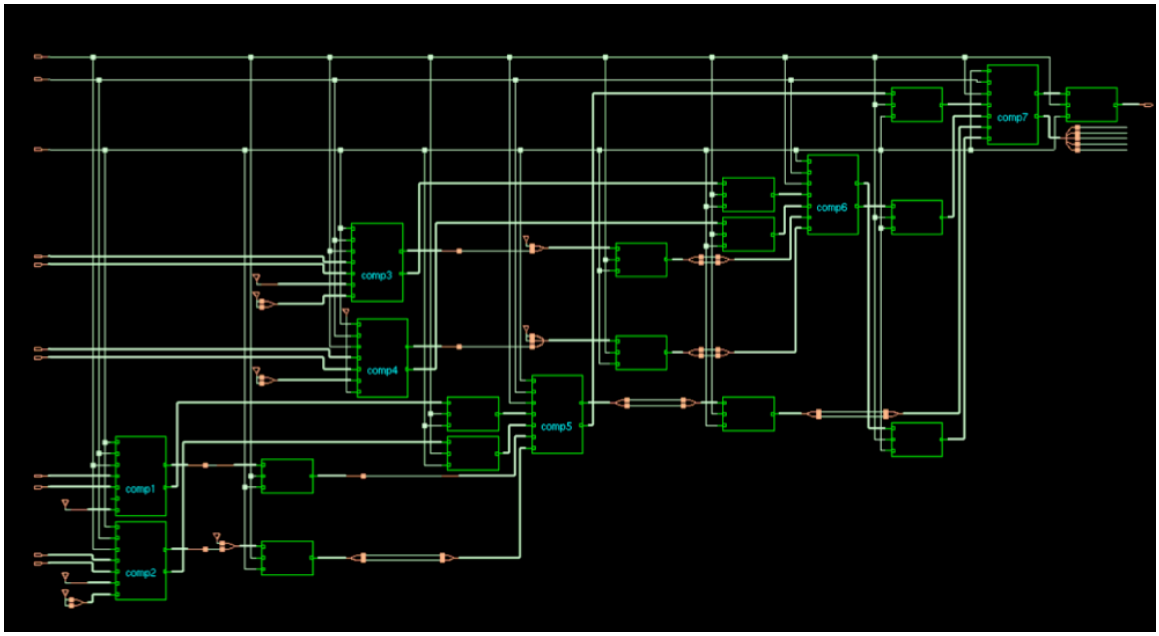


Figura 28: Módulo seleccionador de fase de muestreo

En la Figura 28, se muestra el circuito que determina cuál es la fase que presentó mayores detecciones de bordes, y cuyo código se encuentra en el "Apéndice D". Éste recibe los conteos de la etapa de contadores, y los compara por pares. En la Figura 29 se muestran las formas de onda de dicho módulo y se observa cómo va cambiando el seleccionador de la fase con mayores bordes detectados. El seleccionador (señal "selector_final") es de 3 bits, por lo que indica del 0 al 7 la fase que detectó más bordes.

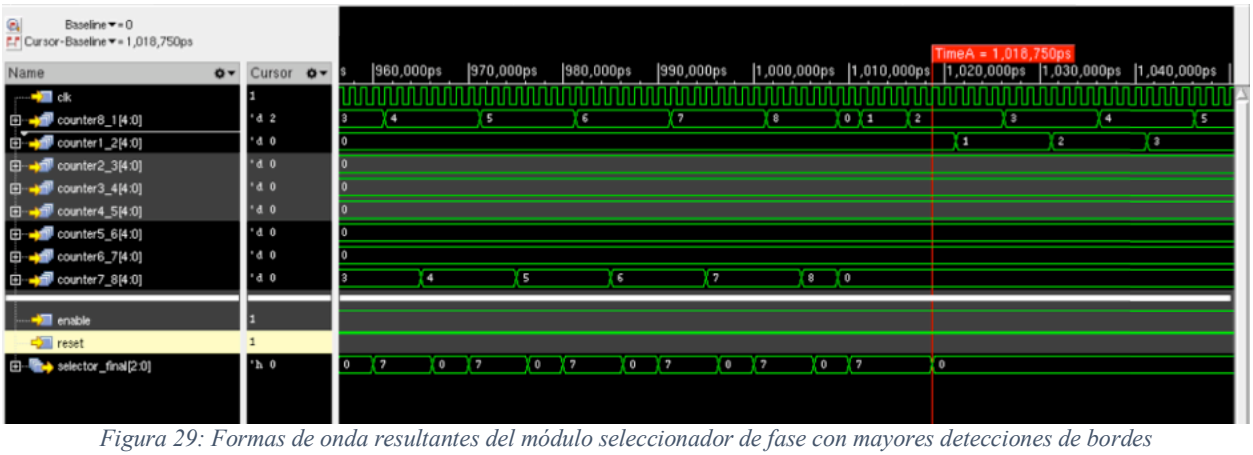


Figura 29: Formas de onda resultantes del módulo seleccionador de fase con mayores detecciones de bordes

El resultado anterior es capturado por un flip flop, como se puede ver en la Figura 30, cuyo reloj es el pulso que se genera después de 16 datos de 200MHz.

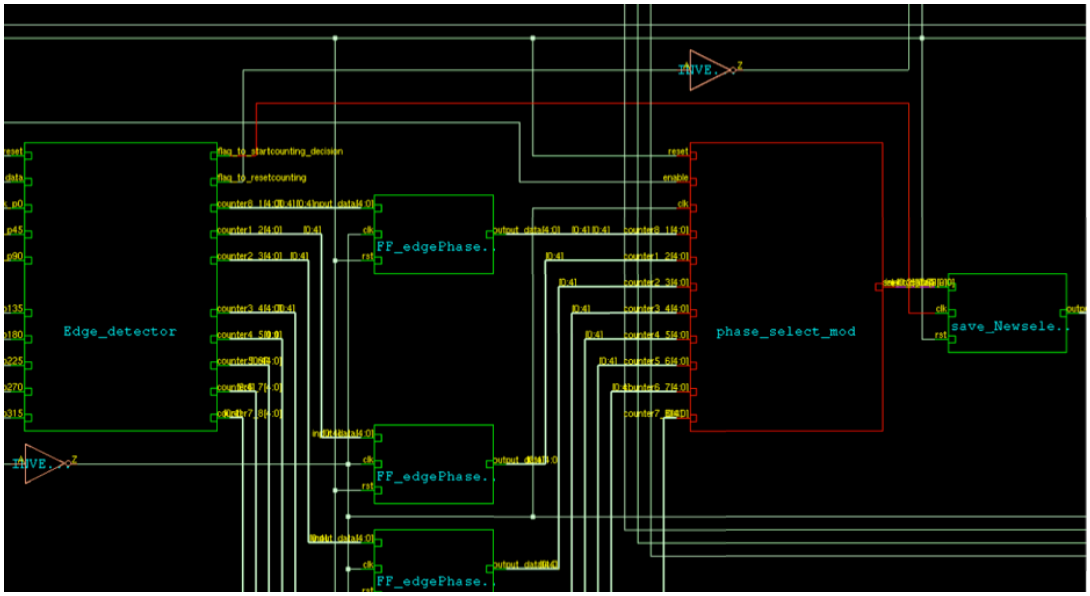


Figura 30: Se observa el flip flop que guarda el resultado del seleccionador de mayor conteo

Las señales generadas se pueden observar en la Figura 31. En ella se aprecia cómo con el pulso de la señal "clk" se guarda el valor que tiene en ese momento la señal "selector_final". Ahora bien, a partir de este punto, esa información es procesada en la parte adaptiva del sistema para contrarrestar los efectos del *jitter* que pueda haber.

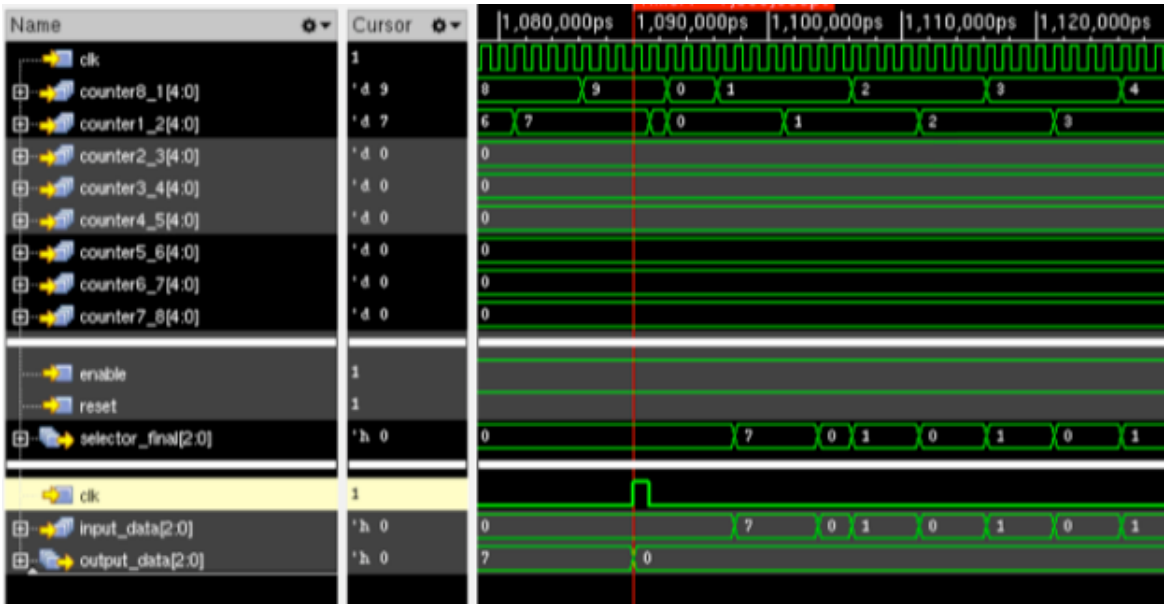


Figura 31: Se observa el momento en que se captura la decisión de la fase con el mayor conteo

3.3.5 Mejora a circuito selector de fase

Al observar los resultados de *timing* de la síntesis lógica se notó que el módulo selector de fase tenía problemas para poder realizar las comparaciones en un solo ciclo de reloj de 800Mhz. En este módulo selector de fase, había comparadores cuya lógica combinacional incluía muchas compuertas, como se nota en la Figura 32, que contribuían para obtener un *slack* de sólo 5ps como se aprecia en el reporte de la Figura 33.

Lo anterior se mejoró al segmentar el módulo comparador mediante la adición de *flip flops*, con lo cual las comparaciones se realizan con más ciclos de reloj. En la Figura 34 se observa la nueva estructura de una instancia de un módulo comparador de conteos, en el cual se muestra el bloque "C1" el cual contiene la lógica combinacional dividida por medio de *flip flops* como se presenta en la Figura 35. Del reporte de la Figura 36 se observa que con las modificaciones, se obtiene un *slack* de 213ps lo cual es una mejora sustancial.

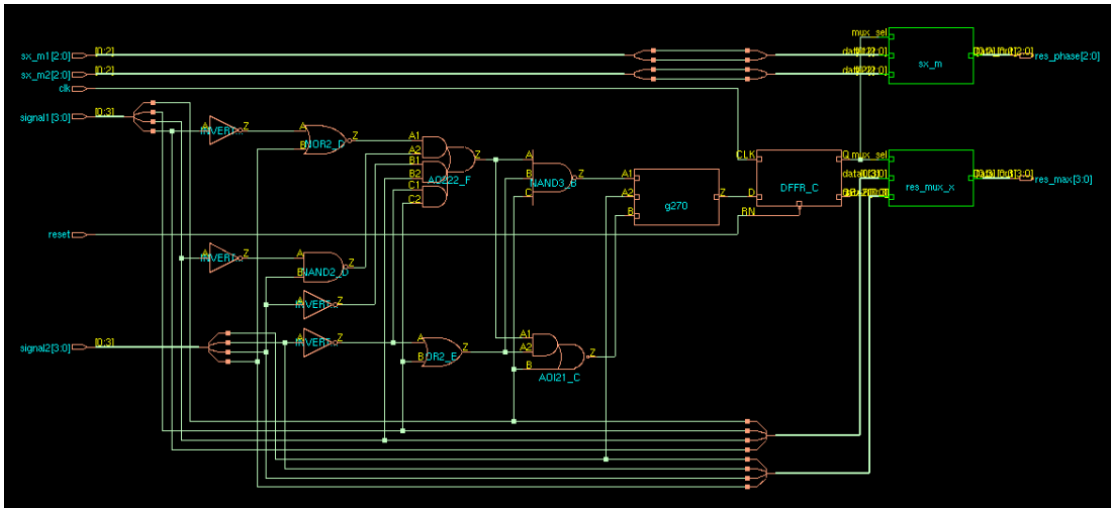


Figura 32: Módulo comparador sin optimización

```
Path   2:

            Pin                 Type     Fanout Load Slew Delay Arrival
                                            (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock FF_edgePhaseSelclk67)   launch                                0 R
FF_edgePhaseSel67
  output_data_reg[1]/CLK                          0                  0 R
  output_data_reg[1]/Q         DFFR_K         3 30.0   96  +346     346 R
FF_edgePhaseSel67/output_data[1]
phase_select_mod/counter6_7[1]
  comp4/signal1[1]
    g277/A                                                +0     346
    g277/Z                     INVERT_H       1 10.6   46   +42     388 F
    g274/A                                                +0     388
    g274/Z                     NAND2_D        1 10.4  150   +86     474 R
    g273/A1                                               +0     474
    g273/Z                     AO222_H        2 16.2  115  +182     656 R
    g272/A                                                +0     656
    g272/Z                     NAND3_B        1 11.7  221  +132     788 F
    g270/A1                                               +0     788
    g270/Z                     AO21_H         1  8.8   69  +205     993 F
    selector_reg_reg/D         DFFR_C                     +0     993
    selector_reg_reg/CLK       setup                   0  +251    1245 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock Phase_select_modClk)    capture                            1250 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :       5ps
Start-point  : FF_edgePhaseSel67/output_data_reg[1]/CLK
End-point    : phase_select_mod/comp4/selector_reg_reg/D
```

Figura 33: Reporte de timing del path de un módulo comparador sin optimización

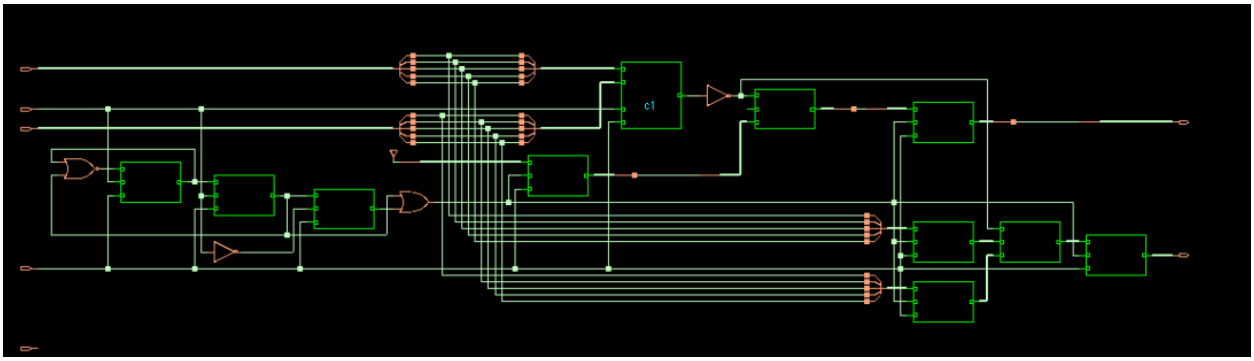


Figura 34: Interior de bloque comparador con nueva arquitectura

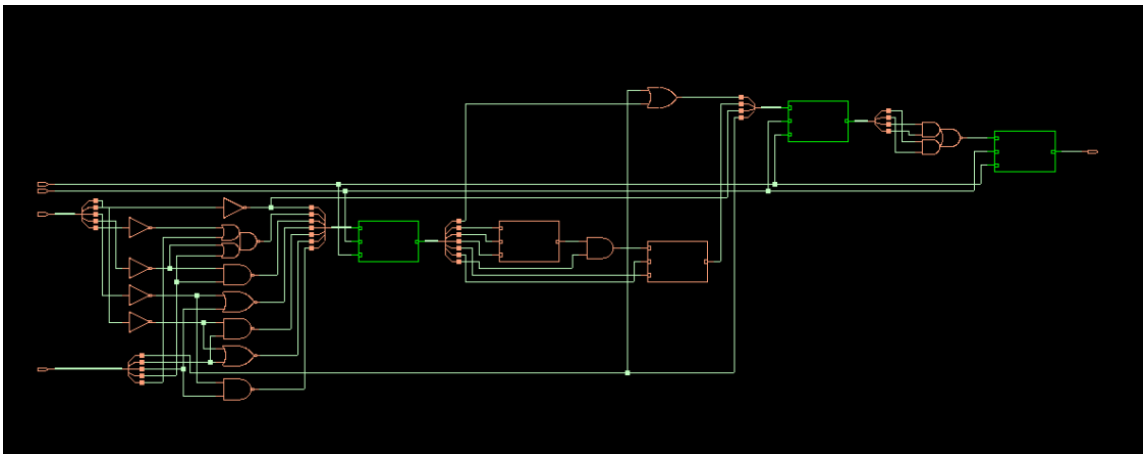


Figura 35: Lógica combinacional del comparador dividida con flip flops

```
path  35:

             Pin                 Type     Fanout Load Slew Delay Arrival
                                                 (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock Phase_select_modClk)      launch                              0 R
phase_select_mod
  comp7
    c1
      FF_tmp1
        output_data_reg[3]/CLK                              0          0 R
        output_data_reg[3]/Q     DFFR_D        1 12.5  127  +342     342 R
      FF_tmp1/output_data[3]
      g104/B                                                +0     342
      g104/Z                     AO21_F        1  9.0   90  +205     547 R
      g103/A                                                +0     547
      g103/Z                     AND2_E        1 10.0   86  +128     674 R
      g102/A1                                               +0     674
      g102/Z                     OAI21_B       1  8.8  141   +92     766 F
      FF_tmp2/input_data[2]
        output_data_reg[2]/D     DFFR_C                       +0     766
        output_data_reg[2]/CLK   setup                     0 +270    1037 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
(clock Phase_select_modClk)      capture                          1250 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :     213ps
Start-point  : phase_select_mod/comp7/c1/FF_tmp1/output_data_reg[3]/CLK
End-point    : phase_select_mod/comp7/c1/FF_tmp2/output_data_reg[2]/D
```

Figura 36: Reporte de timing del comparador partido por flip flops

3.3.6 Mejora a circuito contador

Se realizó una mejora a la etapa de conteos, pues en estos *paths* había problemas de *timing* de *setup* como se observa en la Figura 37. Anteriormente cada contador poseía 5 bits para ir almacenando detecciones de bordes, por lo que el número máximo de detecciones que se podían almacenar eran 32. En este escenario, el sintetizador lógico agregaba varias compuertas que a su vez aumentaban el retardo en la propagación de la señal. En la Figura 38 se nota en color azul el peor *path* de *timing* que genera un slack positivo de 1ps.

Sin embargo, para esta implementación de CDR, al usar una ventana de análisis de 16 bits del dato, no se espera que haya más de 16 bordes detectados. Por lo tanto, se optó por reducir el número de bits de los contadores de 5 bits, a 4 bits. Con esto se redujo el número de flip flops a usar y se prescindió de un par de compuertas que afectaban el timing como se nota en la Figura 39.

En la Figura 40 se aprecia como el módulo “counter_7_8” de poseer el peor path de timing, ahora es el *path* 10 y posee un *slack* de +29ps.

```
  Technology libraries:   PwcV1p08T125_STD_CELL_8HP_12T
                          IBM_CMOS8HP_BASE_WB_IO_SLOW_V108_V140_T125
                          physical_cells
  Operating conditions:   Pwc_V108_V140_T125
  Interconnect mode:      global
  Area mode:              physical library
=============================================================

path   1:

           Pin               Type     Fanout Load Slew Delay Arrival
                                             (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock CountingStageclk78)   launch                                 0 R
Edge_detector
  counter_7_8
    counter_reg_reg[0]/CLK                                 0          0 R
    counter_reg_reg[0]/Q     DFFR_H        3 20.5  122  +424      424 R
    g255/B                                             +0      424
    g255/Z                   AND3_E        2 18.1  141  +188      613 R
    g253/A                                             +0      613
    g253/Z                   AND2_E        2 21.8  153  +172      785 R
    g249/A                                             +0      785
    g249/Z                   NAND2_E       1 11.7  106   +64      849 F
    g245/B                                             +0      849
    g245/Z                   XNOR2_C       1  8.8  148  +129      977 F
    counter_reg_reg[4]/D     DFFR_C                    +0      977
    counter_reg_reg[4]/CLK   setup                   0  +272     1249 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock CountingStageclk78)   capture                           1250 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :       1ps
Start-point  : Edge_detector/counter_7_8/counter_reg_reg[0]/CLK
End-point    : Edge_detector/counter_7_8/counter_reg_reg[4]/D
```

Figura 37: Reporte de timing del peor path cuando los contadores tienen 5 bits

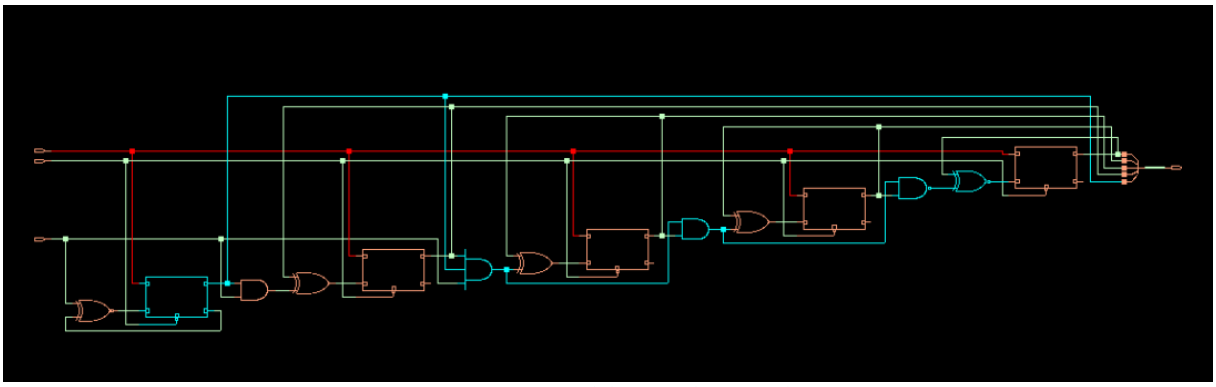


Figura 38: Módulo contador con 5 bits

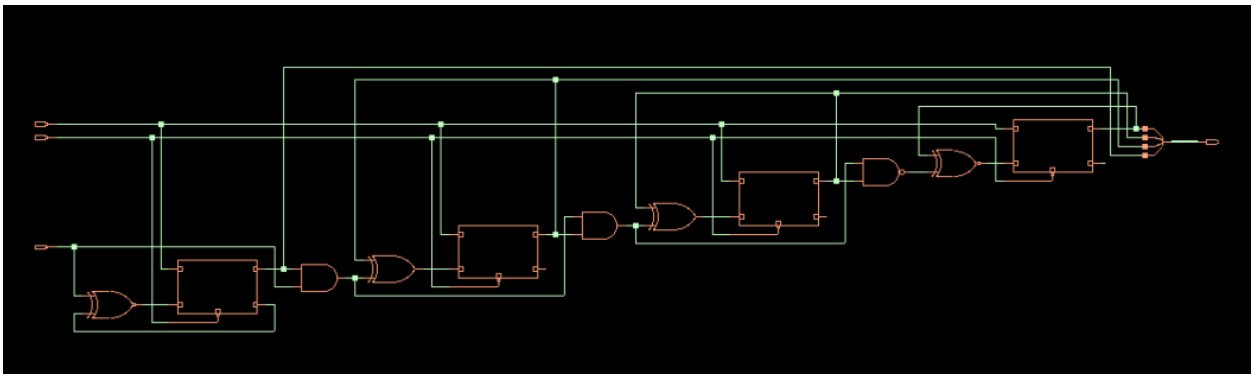


Figura 39: Síntesis resultante del módulo contador después de reducir un bit para el conteo

```
path  10:

           Pin               Type     Fanout Load Slew Delay Arrival
                                             (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------
(clock CountingStageclk78)   launch                              0 R
Edge_detector
  counter_7_8
    counter_reg_reg[0]/CLK                                       0 R
    counter_reg_reg[0]/Q     DFFR_H       2 14.8  108  +414     414 R
    g190/A                                             +0     414
    g190/Z                   AND2_E       2 18.1  129  +154     569 R
    g188/A                                             +0     569
    g188/Z                   AND2_E       2 17.7  132  +158     727 R
    g185/A                                             +0     727
    g185/Z                   NAND2_D      1 11.7  133   +86     813 F
    g181/B                                             +0     813
    g181/Z                   XNOR2_C      1  8.8  133  +138     951 F
    counter_reg_reg[3]/D     DFFR_E                    +0     951
    counter_reg_reg[3]/CLK   setup                   0  +270    1221 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock CountingStageclk78)   capture                          1250 R
-------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :      29ps
Start-point  : Edge_detector/counter_7_8/counter_reg_reg[0]/CLK
End-point    : Edge_detector/counter_7_8/counter_reg_reg[3]/D
```

Figura 40: Reporte de timing resultante después de reducir un bit al módulo contador

3.3.7 Etapa adaptiva a jitter

Los bloques mostrados en la Figura 41 son los que están involucrados en la parte adaptiva a *jitter* y cuyo código se encuentra en el "Apéndice E". En ella se nota el bloque en rojo, que se encarga de decidir si la fase seleccionada por el módulo "Selector de fase" (en color verde) cambiará o no.

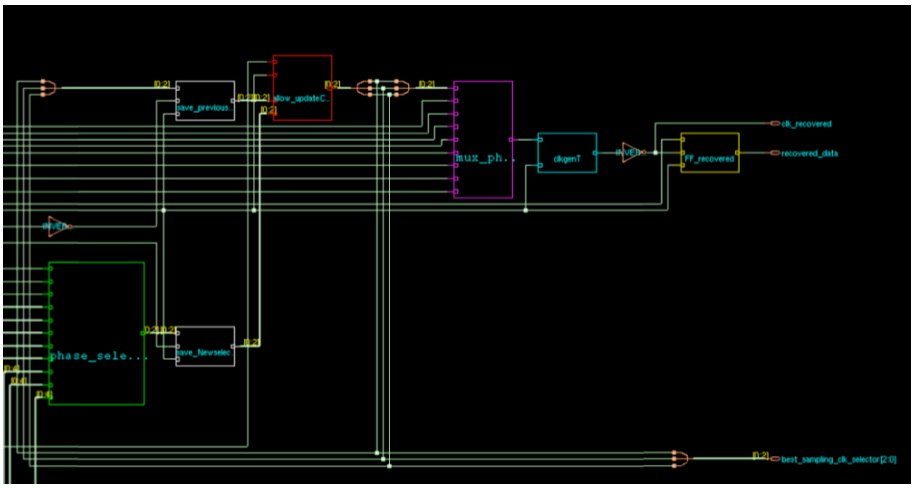


Figura 41: Bloques involucrados en la parte adaptiva del Sistema

En la Figura 42 se nota en las señales de abajo, la operación del circuito como se describió anteriormente. La señal morada es la nueva propuesta de fase a usar; la señal en naranja es la decisión anterior y la señal en rojo es la respuesta del módulo adaptivo. En este caso, dado que en el pulso de análisis (de la señal "clk") la fase anterior era 0º y la nueva propuesta es la fase 6 (de 270º), la señal "selector_final" (en rojo) no cambia pues está en su rango de inmunidad.

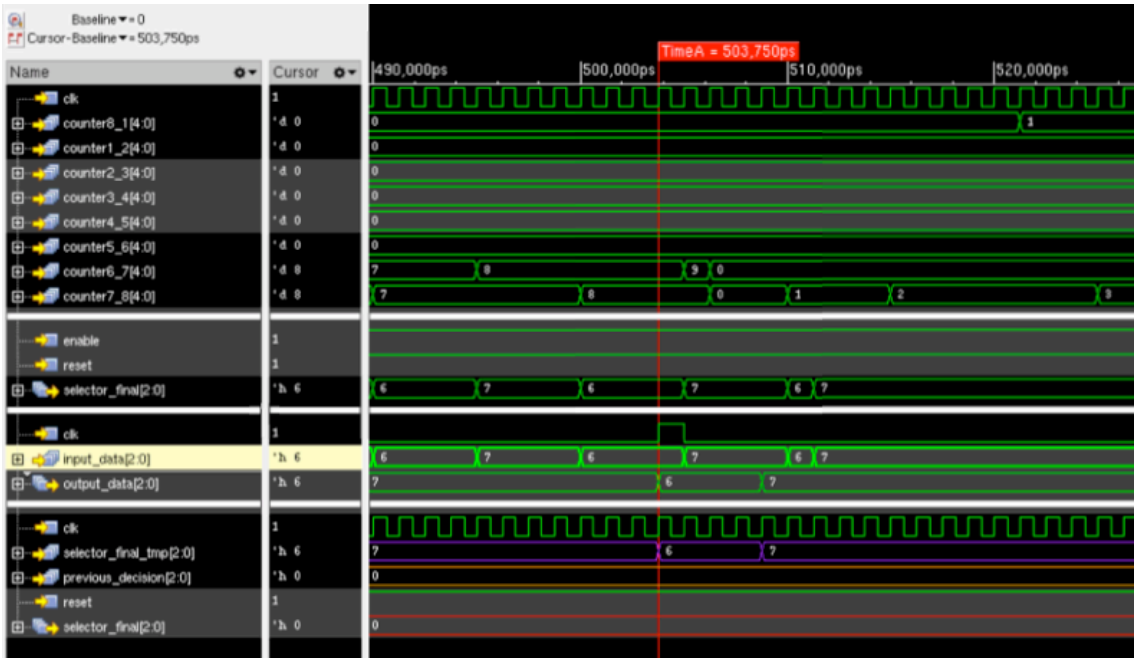


Figura 42: Formas de onda del módulo adaptivo a jitter

En contraste, en la Figura 43 se observa el momento en el que se hace un cambio de decisión. La decisión anterior fue la fase 3, y la nueva es la fase 6; dado que están separadas más de 2 fases, se realiza un cambio mostrado en la señal en rojo "selector_final".

Figura 43: Se observa el cambio de selección de fase por el módulo adaptivo (señal en rojo)

Posteriormente, ya que se tiene una decisión de la fase a usar, ésta es empleada como selector en un multiplexor de 8 a 1, el cual multiplexa las 8 fases de muestreo (de 0º a 315º), el cual es el bloque en morado de la Figura 41.

La salida del multiplexor entra a un módulo divisor de frecuencia que ayuda a generar una señal de muestreo cuyo flanco de subida está más en el centro del dato a muestrear. Este bloque es el de color azul en la Figura 41. Es importante señalar que se empleó un inversor a la salida de dicho divisor de frecuencia, pues se observó que el flanco de bajada de este módulo era más adecuado para tomar las muestras.

En la Figura 44 es posible distinguir la señal “best_sampling_clk” que es la fase más adecuada para tomar muestras del dato de entrada, seleccionada por el módulo adaptivo. La señal “clk_recovered” es la salida del divisor de frecuencia y cuyo flanco de subida se ve centrado en en el dato a muestrear (señal “original data”); y, finalmente, la señal “recovered_data” es el dato resultante del muestreo con el reloj seleccionado. Se observa que se recupera correctamente la información en este instante.

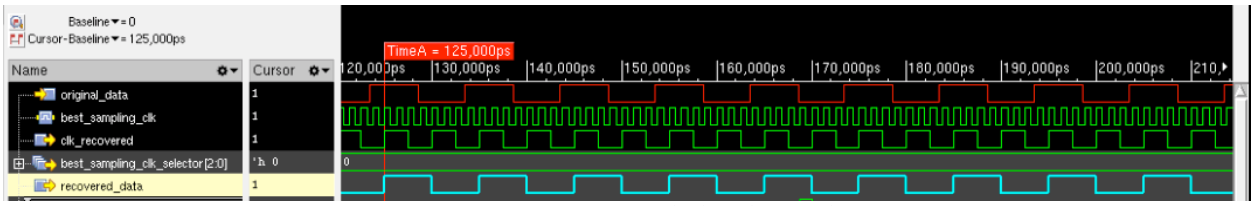


Figura 44: Recuperación de dato de entrada

En la Figura 45, se aprecia que al inicio las señales de los contadores "counter2_3" y "counter3_4" incrementan su cuenta al detectar transiciones con las muestras del reloj 2 al 3 y del 3 al 4, pero ante el *jitter* hay un momento en el que los contadores "counter5_6" y "counter6_7" comienzan a detectar transiciones en lugar de los contadores anteriores. Así pues, estos valores ayudan al módulo adaptivo a determinar que se debe realizar un cambio de fase para muestrear, causando un cambio de la fase 3 a la fase 6 en la señal "best_sampling_clk_selector" (en morado).

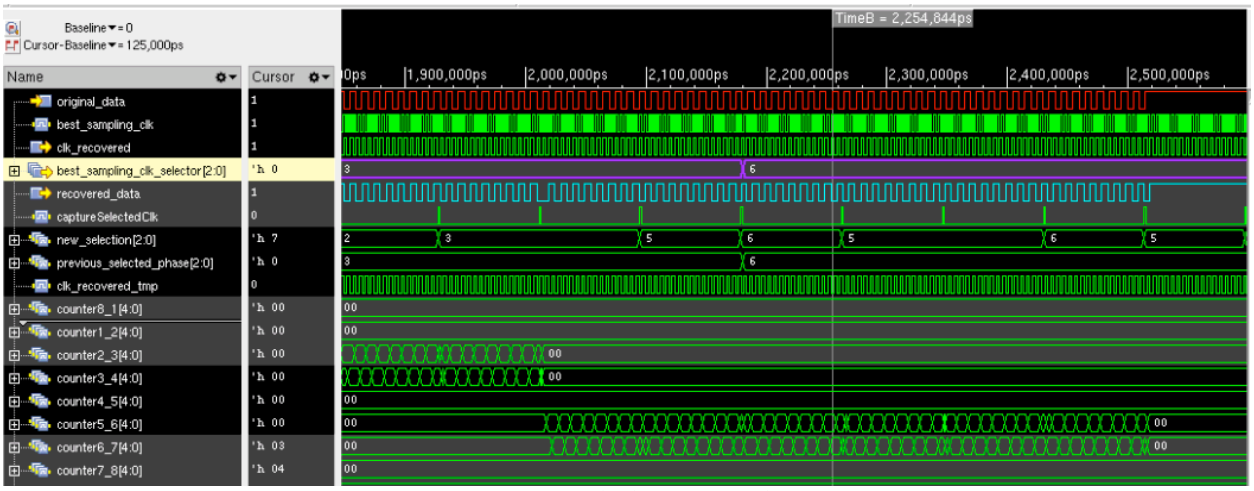


Figura 45: Resultados con jitter de 5%

En la Figura 46, se presenta con más detalle el instante en el que se realiza el cambio de fase. Se observa que cuando la fase 3 es empleada para el muestreo, el flanco de subida de la señal "clk_recovered" no está tan centrada en el dato de entrada; mientras que cuando se realiza el cambio de fase a la fase 6, el flanco de subida de la señal "clk_recovered" ya se encuentra más centrada y sigue recuperando el dato de entrada (señal azul "recovered_data").

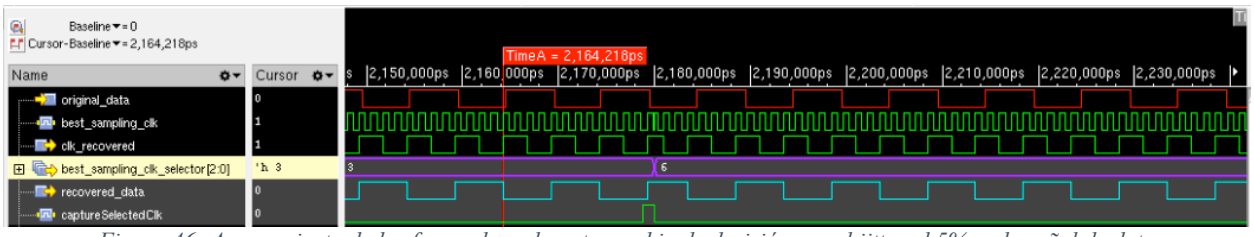


Figura 46: Acercamiento de las forma de onda ante cambio de decisión por el jitter al 5% en la señal de dato

3.3.8 Resumen de resultados de síntesis lógica

Para la síntesis lógica con RTL Compiler, empleando las librerías de celdas estándar con transistores lentos y un voltaje de 1.08v se obtuvo un *slack* positivo de 3ps como se muestra en la Figura 47. Se puede notar que este *path* está en el contador que determina la ventana de análisis, es decir, cuántos datos se analizarán. Este módulo posee un gran número de compuertas lógicas que contribuyen a que el retardo en la propagación de la señal sea mayor.

Para este mismo caso con transistores lentos, se aprecia en la Figura 48 que el área total que se requiere para esta síntesis lógica es de 33585.512um^2. Además, el módulo que requirió mayor área (19592 um^2) es el selector de fase, pues los bloques que lo conforman, principalmente los comparadores, poseen mucha lógica combinacional que abarcarán dicha área, como se nota en la Figura 49.

Por otro lado, el módulo detector de transiciones abarca un área de 7407um^2, mientras que el módulo que realiza la parte adaptiva al *jitter* requiere 1412 um^2. Además por la Figura 50 podemos ver que la potencia necesaria para el circuito es de 7.37mW, con las librerías de transistores lentos.

```
============================================================
  Generated by:           Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1
  Generated on:           Aug 02 2018  06:44:40 pm
  Module:                 CDR
  Technology libraries:   PwcV1p08T125_STD_CELL_8HP_12T
                          IBM_CMOS8HP_BASE_WB_IO_SLOW_V108_V140_T125
                          physical_cells
  Operating conditions:   Pwc_V108_V140_T125
  Interconnect mode:      global
  Area mode:              physical library
============================================================

path   1:

          Pin                Type      Fanout Load Slew Delay Arrival
                                             (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock not_180clk)           launch                                0 R
Edge_detector
  cont_to16data
    counter_reg_reg[1]/CLK                              0           0 R
    counter_reg_reg[1]/Q     DFFR_H           3 29.4  143  +439     439 R
    g287/B                                             +0     439
    g287/Z                   AND2_I           3 38.7  104  +140     580 R
    g276/A                                             +0     580
    g276/Z                   NAND4_E          2 29.4  148   +93     673 F
    g275/A                                             +0     673
    g275/Z                   INVERT_I         2 19.0   90   +74     746 R
    g272/A                                             +0     746
    g272/Z                   NAND2_D          1 15.4  121   +87     834 F
    g265/B                                             +0     834
    g265/Z                   NAND2BAL_E       1 13.8   95   +78     912 R
    g261/A                                             +0     912
    g261/Z                   AOI21_C          1  8.8  109   +71     983 F
    counter_reg_reg[6]/D     DFFR_H                    +0     983
    counter_reg_reg[6]/CLK   setup                   0  +264    1247 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock not_180clk)           capture                           1250 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :       3ps
Start-point  : Edge_detector/cont_to16data/counter_reg_reg[1]/CLK
End-point    : Edge_detector/cont_to16data/counter_reg_reg[6]/D
```

Figura 47: Resultado de timing para síntesis lógica con transistores lentos

```
Instance Count
--------------
Leaf Instance Count             637
Sequential Instance Count       323
Combinational Instance Count    314
Hierarchical Instance Count     153

Area
----
Cell Area                         16366.080
Physical Cell Area                0.000
Total Cell Area (Cell+Physical)   16366.080
Net Area                          17219.432
Total Area (Cell+Physical+Net)    33585.512

Max Fanout                        292 (reset)
Min Fanout                        0 (res_max_counting[0])
Average Fanout                    2.3
Terms to net ratio                3.6
Terms to instance ratio           4.1
Runtime                           34.113 seconds
Elapsed Runtime                   40 seconds
RC peak memory usage:             285.00
EDI peak memory usage:            no_value
Hostname                          FV00
```

Figura 48: Resultados de área al usar la librería de celdas estándar de transistores lentos

```
============================================================
  Generated by:           Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1
  Generated on:           Aug 02 2018  06:44:41 pm
  Module:                 CDR
  Technology libraries:   PwcV1p08T125_STD_CELL_8HP_12T
                          IBM_CMOS8HP_BASE_WB_IO_SLOW_V108_V140_T125
                          physical_cells
  Operating conditions:   Pwc_V108_V140_T125
  Interconnect mode:      global
  Area mode:              physical library
============================================================

       Instance         Cells  Cell Area  Net Area  Total Area
-------------------------------------------------------------
CDR                       637      16366     17219       33586
  phase_select_mod        390      10222      9370       19592
    comp6                  61       1638      1293        2930
      c1                   28        620       647        1267
```

Figura 49: Reporte de los módulos que ocupan más área al emplear transistores lentos

```
============================================================
  Generated by:           Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1
  Generated on:           Aug 02 2018  06:44:42 pm
  Module:                 CDR
  Technology libraries:   PwcV1p08T125_STD_CELL_8HP_12T
                          IBM_CMOS8HP_BASE_WB_IO_SLOW_V108_V140_T125
                          physical_cells
  Operating conditions:   Pwc_V108_V140_T125
  Interconnect mode:      global
  Area mode:              physical library
============================================================

                            Leakage    Dynamic      Total
      Instance        Cells Power(nW)  Power(nW)   Power(nW)
-------------------------------------------------------------
CDR                     637 18010.686 7360923.718 7378934.404
  phase_select_mod      390 10904.674 3910949.512 3921854.186
    comp6                61  1753.893  649038.902  650792.795
```

Figura 50: Reporte de potencia requerida para el circuito CDR

Por otro lado, al emplear las librerías de transistores con *timing* típico se obtiene que el peor *path* tiene un *slack* positivo de 382ps. En este escenario, el peor *path* de *timing* se encontró en el módulo seleccionador de fase como se nota en la Figura 51. Además, se pudo observar que el *path* que causaba problemas de *timing* con librerías de transistores lentos (Figura 47), ahora no tiene problemas pues da un slack positivo de 428ps (Figura 52).

```
============================================================
  Generated by:           Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1
  Generated on:           Jul 22 2018  06:51:56 pm
  Module:                 CDR
  Technology libraries:   PnomV1p50T025_STD_CELL_8HP_12T
                          IBM_CMOS8HP_BASE_WB_IO_TYP_V150_V150_T25
                          physical_cells
  Operating conditions:   Pnc_V150_V150_T25
  Interconnect mode:      global
  Area mode:              physical library
============================================================

path   1:

           Pin                Type     Fanout Load Slew Delay Arrival
                                             (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock Phase_select_modClk)   launch                              625 R
phase_select_mod
  comp7
    FF_delay2
      q_reg[0]/CLK                                  0             625 R
      q_reg[0]/Q              DFFR_C        3 20.1  139  +189     814 R
    FF_delay2/q[0]
    FF_delay3/input_data[0]
      q_reg[0]/D              DFFR_C                 +0     814
      q_reg[0]/CLK            setup                  0   +54     868 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock Phase_select_modClk)   capture                            1250 F
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :     382ps
Start-point  : phase_select_mod/comp7/FF_delay2/q_reg[0]/CLK
End-point    : phase_select_mod/comp7/FF_delay3/q_reg[0]/D
```

Figura 51: Resultado de timing para síntesis lógica con transistores típicos

```
path  10:

           Pin               Type     Fanout Load Slew Delay Arrival
                                             (fF) (ps)  (ps)   (ps)
-------------------------------------------------------------------------
(clock not_180clk)           launch                                 0 R
Edge_detector
  cont_to16data
    counter_reg_reg[0]/CLK                                 0          0 R
    counter_reg_reg[0]/Q     DFFR_H        3 22.3   62  +190      190 R
    g225/A                                                +0      190
    g225/Z                   NAND2_D       2 18.2   65   +44      234 F
    g222/A                                                +0      234
    g222/Z                   OR2_E         2 18.2   55   +89      323 F
    g219/A                                                +0      323
    g219/Z                   OR2_E         2 18.2   55   +86      410 F
    g214/A                                                +0      410
    g214/Z                   OR2_E         2 18.2   55   +86      496 F
    g211/A                                                +0      496
    g211/Z                   OR2_E         1 12.5   45   +79      575 F
    g204/B                                                +0      575
    g204/Z                   XOR2_C        1  9.5  113   +68      644 R
    g201/A                                                +0      644
    g201/Z                   NOR2_B        1  8.9   96   +63      706 F
    counter_reg_reg[6]/D     DFFR_C                      +0      706
    counter_reg_reg[6]/CLK   setup                    0  +116      822 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock not_180clk)           capture                             1250 R
-------------------------------------------------------------------------
Cost Group   : 'C2C' (path_group 'C2C')
Timing slack :     428ps
Start-point  : Edge_detector/cont_to16data/counter_reg_reg[0]/CLK
End-point    : Edge_detector/cont_to16data/counter_reg_reg[6]/D
```

Figura 52: Path que presentaba problemas de timing con librerías de transistores lentos, ahora tienen un mejor slack

El área total requerida es de 33486.664 um2 como se muestra en la Figura 53.

```
Instance Count
--------------
Leaf Instance Count             638
Sequential Instance Count       323
Combinational Instance Count    315
Hierarchical Instance Count     153

Area
----
Cell Area                           16277.760
Physical Cell Area                  0.000
Total Cell Area (Cell+Physical)     16277.760
Net Area                            17208.904
Total Area (Cell+Physical+Net)      33486.664

Max Fanout                          292 (reset)
Min Fanout                          0 (res_max_counting[0])
Average Fanout                      2.4
Terms to net ratio                  3.7
Terms to instance ratio             4.1
Runtime                             33.323 seconds
Elapsed Runtime                     39 seconds
RC peak memory usage:               286.00
EDI peak memory usage:              no_value
Hostname                            FV00
```

Figura 53: Resultados de área para síntesis lógica con transistores con timing típico

El resultado de potencia para esta síntesis lógica fue de 14.1mW como se ve en la Figura 54.

```
=============================================================
  Generated by:           Encounter(R) RTL Compiler RC14.26 - v14.20-s058_1
  Generated on:           Aug 02 2018  07:04:51 pm
  Module:                 CDR
  Technology libraries:   PnomV1p50T025_STD_CELL_8HP_12T
                          IBM_CMOS8HP_BASE_WB_IO_TYP_V150_V150_T25
                          physical_cells
  Operating conditions:   Pnc_V150_V150_T25
  Interconnect mode:      global
  Area mode:              physical library
=============================================================

                                Leakage     Dynamic       Total
      Instance        Cells Power(nW)   Power(nW)    Power(nW)
-------------------------------------------------------------------
CDR                     638  2508.501 14133260.545 14135769.046
  phase_select_mod      390  1561.078  7467837.579  7469398.657
    comp6                61   254.989  1250812.540  1251067.528
```

Figura 54: Resultados de potencia

Al simular en Simvision, el testbench descrito en el "Apéndice T" el netlist generado con librerías de transistores lentos y una señal de dato sin jitter se obtiene un resultado ideal, en donde la recuperación del dato no presenta problemas. Se puede apreciar en la Figura 55 que la señal "decisión_final" tiene el selector de la fase a emplear para realizar la generación del reloj recuperador del dato. En este caso como no hay jitter en la señal de entrada (señal en color rojo), se determina que no es necesario realizar un cambio de fase y con la fase '0' se podrá generar un reloj recuperador de dato.

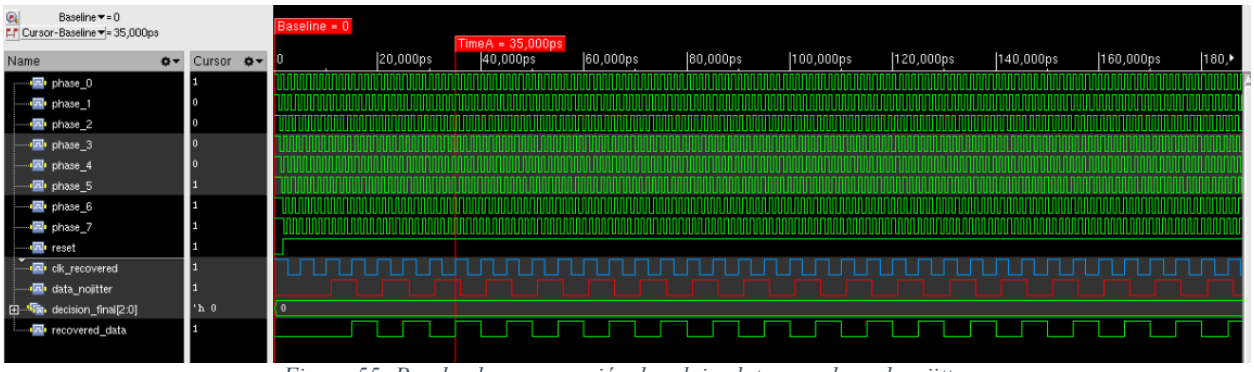


Figura 55: Prueba de recuperación de reloj y dato cuando no hay jitter

3.3.9 Jitter al 5% del periodo de la señal de dato

Ahora bien, introduciendo un jitter de 5% del periodo de la señal de entrada se ejecutó una simulación mediante Simvision para un tiempo de 3000us, cuyas señales se pueden ver en la Figura 56 y en la Figura 57.

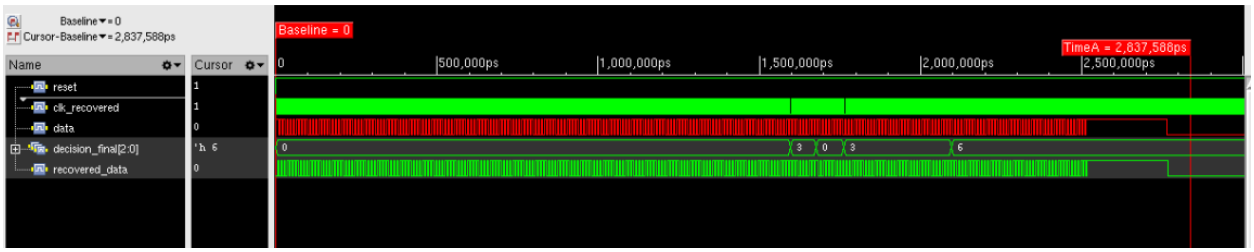


Figura 56: Simulación de operación de CDR con una señal de dato de entrada con jitter del 5% del periodo de la señal

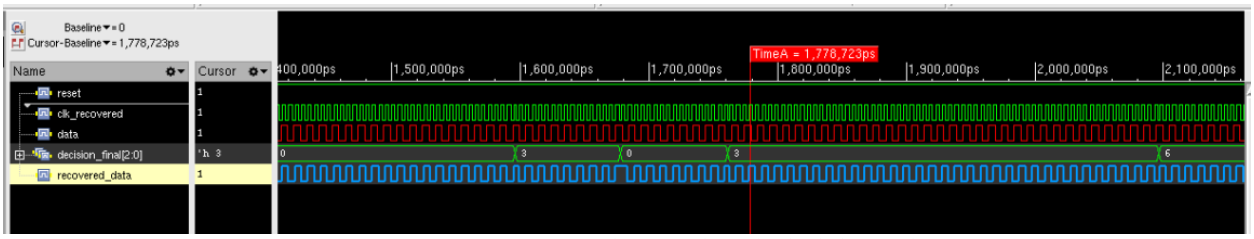


Figura 57: Acercamiento, para observar el resultado de la simulación

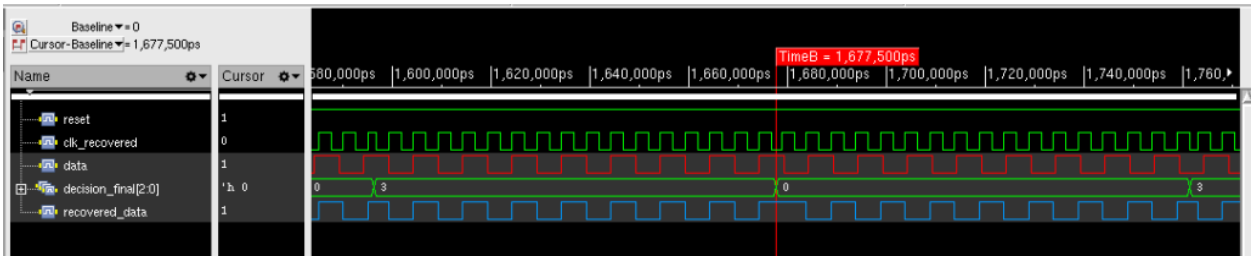


Figura 58: Acercamiento del cambio de decisión de fase 3 a fase 0

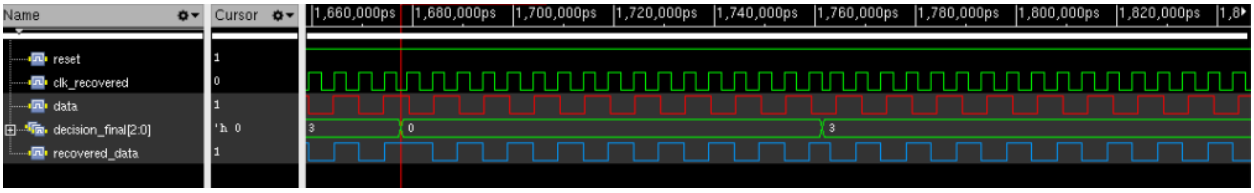


Figura 59: Acercamiento de la transición de fase 0 a fase 3

De las figuras anteriores se aprecia que debido al jitter introducido, la decisión del módulo adaptivo dicta que debe haber algunos cambios en la fase elegida. Para este caso se cambió de la fase 0 a la fase 3, posteriormente regresó a la fase 0 y después vuelve a cambiar a la fase 3 y a la fase 6 después de 336ns aproximadamente. En el cambio de la fase 3 a la fase 0 se puede ver que los flancos de subida de la señal de reloj recuperado ("clk_recovered") no están centrados y debido al jitter, llega un punto en el que la captura del dato se realiza en el borde de la señal; es por esto que se cambia a la fase 0 para tratar de contrarrestar el efecto del jitter. Sin embargo, en el instante en el que se cambia de fase se pierde un bit. Además, se observa en la Figura 58 que la fase seleccionada no fue adecuada pues el reloj recuperado sigue sin capturar el dato en el centro del bit. En la Figura 59 se puede apreciar mejor ese efecto, por lo cual el circuito decide realizar

nuevamente un cambio a la fase 3, con la cual se empieza a notar que el flanco de subida de la señal “clk_recovered” se sitúa más al centro del bit de dato.

3.3.10 Jitter al 10% del periodo de la señal de dato

Al aumentar el valor del jitter al 10% del periodo de la señal de datos se observa que hay un instante en el que no se puede recuperar correctamente la información, lo cual es mostrado en la Figura 60 y Figura 61. Además, se observa que se realizan más cambios de fase comparando con el jitter al 5%.

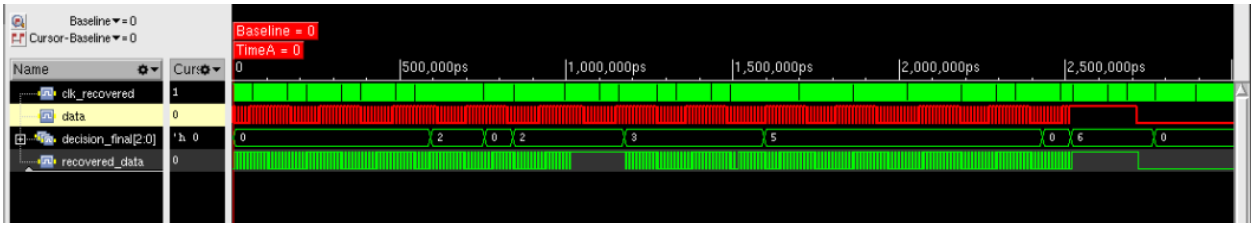


Figura 60: Resultado de simulación con jitter al 10%

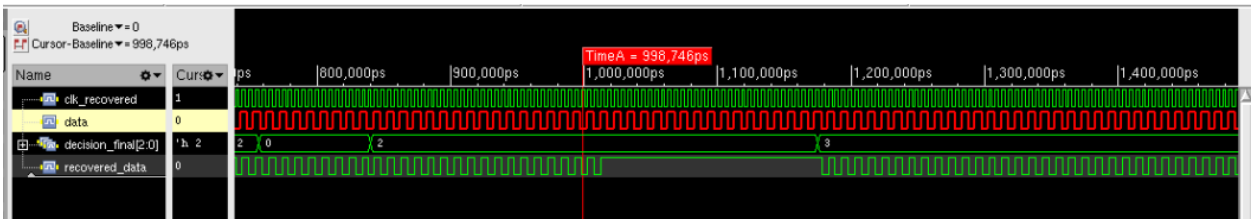


Figura 61: Acercamiento de la transición en donde se pierde la recuperación del dato

3.3.1 Jitter al 15% del periodo de la señal de dato

Cuando se agrega un jitter al 15% del periodo de la señal de datos se aprecia que se pierde por unos instantes el dato de entrada, por lo que en la Figura 62 hay un pulso más ancho en el dato recuperado. En la Figura 63 se muestra un acercamiento a este efecto. Sin embargo, también se aprecia que se realizan más transiciones de fase, comparando con el jitter al 5% y al 10%, para poder recuperar el dato de entrada.

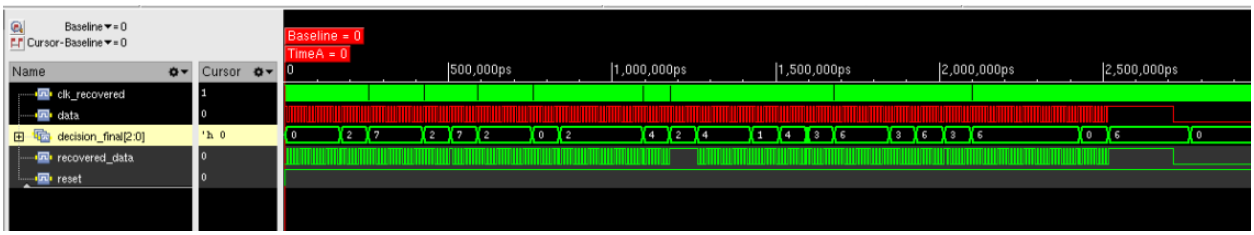


Figura 62: Resultado de simulación con jitter al 15%

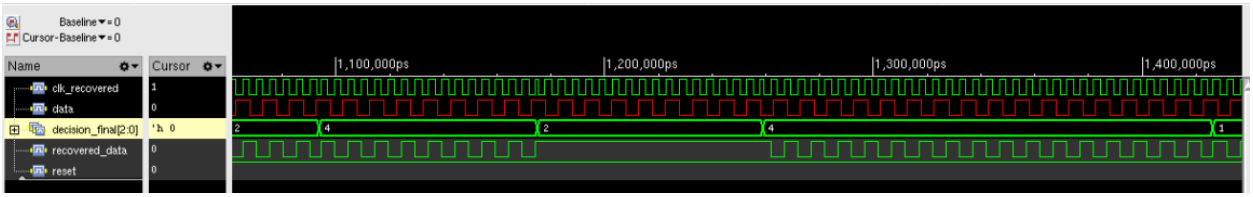


Figura 63: Acercamiento de la transición en donde se pierde la recuperación del dato

3.4. Síntesis física

3.4.1 Síntesis física de módulo CDR (routing and placement)

Para realizar la síntesis física se usó el software de Cadence EDI Encounter, así como los *netlist* generados por la síntesis lógica mediante RTL Compiler, archivos de constraint generados por la misma herramienta y el script del "Apéndice P".

Dentro de la carpeta que contiene los archivos de *netlist* y *constraint*, se creó una carpeta de apoyo titulada "EDI_implementation", que albergará los archivos generados por Encounter al realizar el siguiente paso en el flujo de diseño: la síntesis física.

Para abrir el software Encounter, se emplea el comando "encounter" en una terminal, dentro del directorio de trabajo "EDI_implementation", como se observa en la Figura 64. Y una vez ejecutado dicho comando se abrirá la interfaz gráfica de éste, como se muestra en la Figura 65.

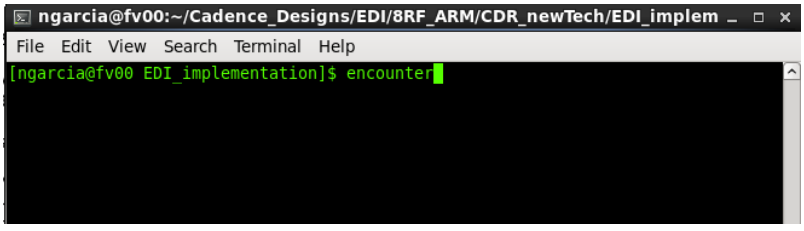


Figura 64: Comando para ejecutar el software CAD Encounter

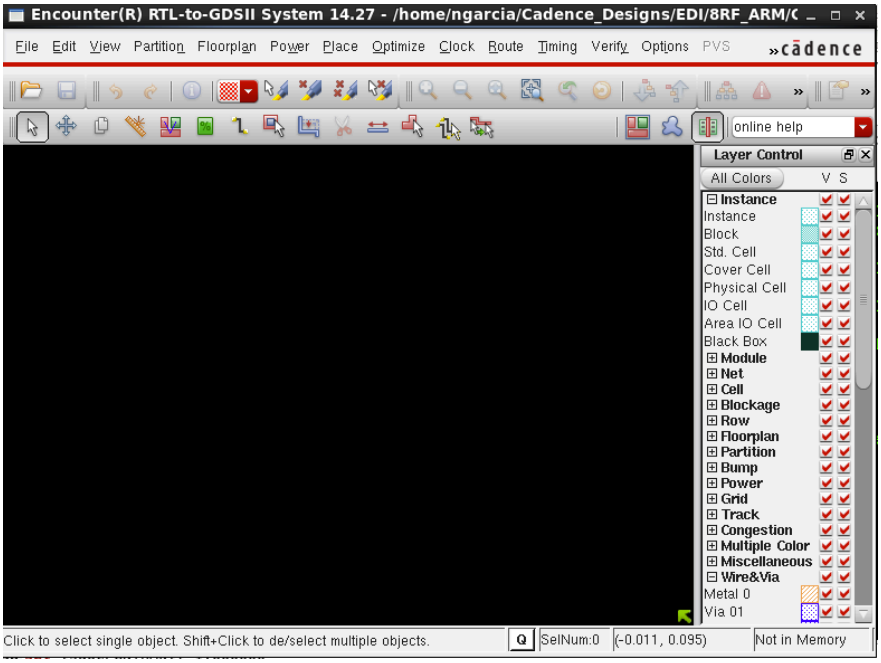


Figura 65: Interfaz gráfica de Encounter

Posteriormente, cuando ya tenemos abierto Encounter, se comienza creando la configuración del ambiente en Encounter, para poder realizar la síntesis física y el análisis de esquinas como se puede apreciar en la Figura 66.

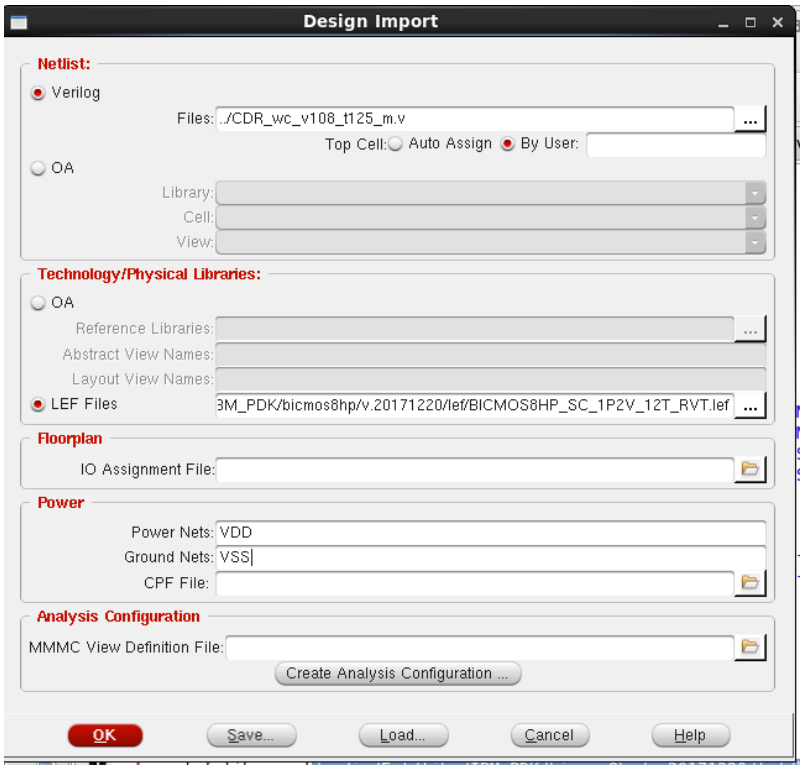


Figura 66: Configuración de Encounter para realizar síntesis física

En el campo Verilog es importante especificar el netlist generado para el peor caso (con transistores lentos). Por otro lado, en el campo LEF files es importante agregar el archivo LEF de tecnología y el de macros que para la tecnología BiCMOS se encuentran en las siguientes rutas:

- /media/Ext/Libs/IBM_PDK/bicmos8hp/v.20171220/lef/bicmos8hp_5AM_21_tech.lef
- /media/Ext/Libs/IBM_PDK/bicmos8hp/v.20171220/lef/BICMOS8HP_SC_1P2V_12T_RVT.lef

También se pueden agregar los lef de entradas y salidas que están en:

- /opt/libs/IBM_PDK/bicmos8hp/v.20160727/lef/CMOS8HP_BASE_WB_IO_5AM.lef

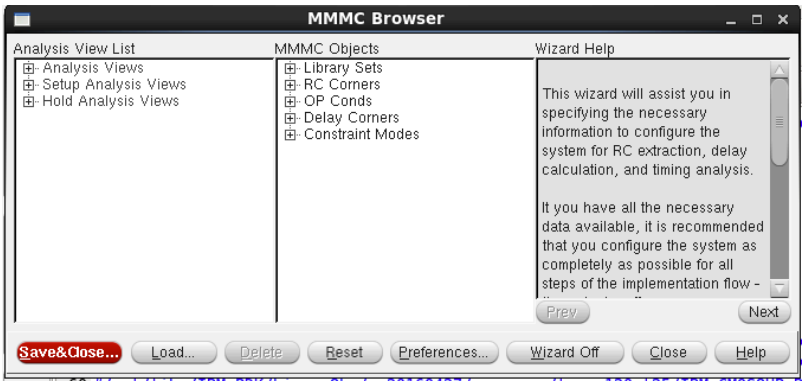


Figura 67: Asistente para creación de análisis Multimode-Multi Corner

Posteriormente se creó una configuración de análisis multi modo, multi esquina (Multi mode-multi corner) con el asistente que ofrece Encounter, como se ve en la Figura 67.

Al crear una nueva configuración se empieza con el panel derecho "MMMC Objects". Se da doble click en "Library sets", y se crean 2 elementos:

1. Typ_timing_lib: donde se pueden agregar librerías de timing y de Signal integrity.

Para este paso, se agregó solo la librería de timing de las celdas estándar (Figura 68), que se encuentra en el siguiente directorio:

- /media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/synopsys/typ_v150_t025/PnomV1p50T025_STD_CELL_8HP_12T.lib

Ahora bien, la librería de entradas/salidas (I/O) que se podría usar es:

- /media/Ext/libs/IBM_PDK/bicmos8hp/v.20160727/synopsys/typ_v150_v150_t25/IBM_CMOS8HP_BASE_WB_IO_TYP_V150_V150_T25.lib

2. WC_timing_lib: para la cual también se agregó solo la librería de timing:
   - /media/Ext/libs/IBM_PDK/bicmos8hp/v.20160727/synopsys/slow_v108_v140_t125/ IBM_CMOS8HP_BASE_WB_IO_SLOW_V108_V140_T125.lib

Ahora bien, la librería de entradas/salidas (I/O) que se podría usar para este caso es:

- /media/Ext/libs/IBM_PDK/bicmos8hp/v.20160727/synopsys/slow_v108_v140_t125/IBM_CMOS8HP_BASE_WB_IO_SLOW_V108_V140_T125.lib

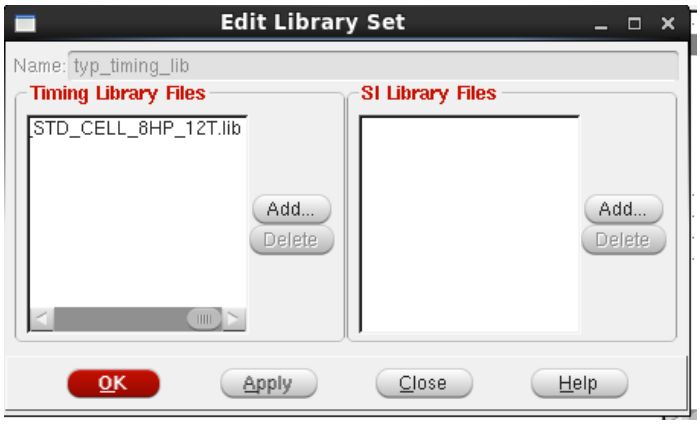


Figura 68: Inserción de librería de timing

Posteriormente se deben agregar RC Corners como se aprecia en la Figura 69 y la Figura 70.

1. Se agregan para escenario típico

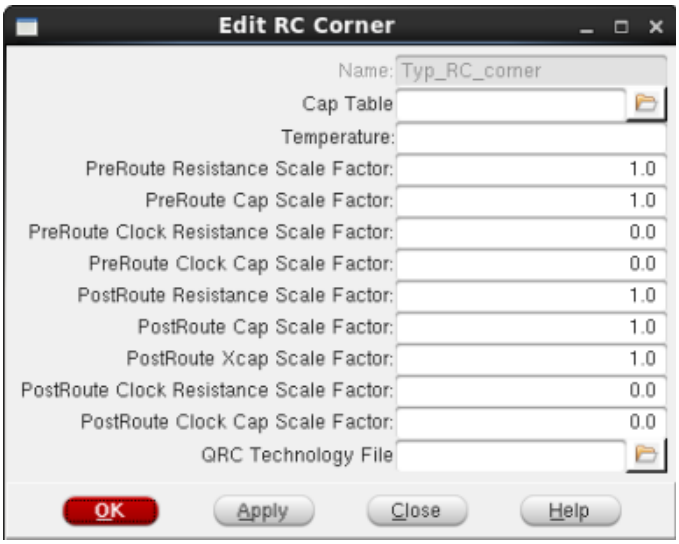


Figura 69: RC corner para escenario típico

2. Se agrega otro para el worst case en donde se modifica la temperatura a 125°C:

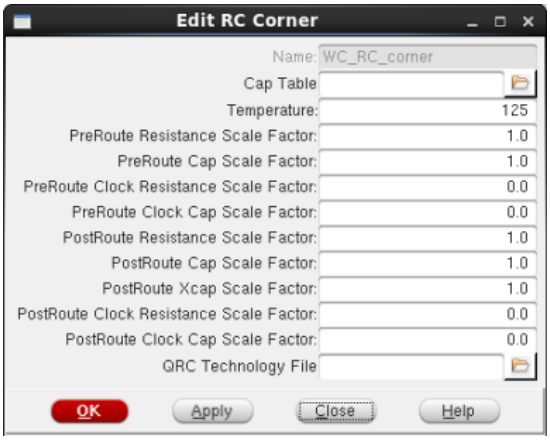


Figura 70: RC corner para caso de transistores lentos

Posteriormente se agregan Delay Corners**.** Se agregan 2, donde una es para parámetros típicos y otra para parámetros de peor escenario con transistores lentos.

Para el caso típico, en el campo "RC Corner" se emplea el RC corner típico previamente definido. Mientras que en el campo "Library set" se usa del Library Set, el típico previamente definido "typ_timing_lib."

Se realiza lo equivalente para el peor escenario, con transistores lentos como se muestra en la Figura 71.

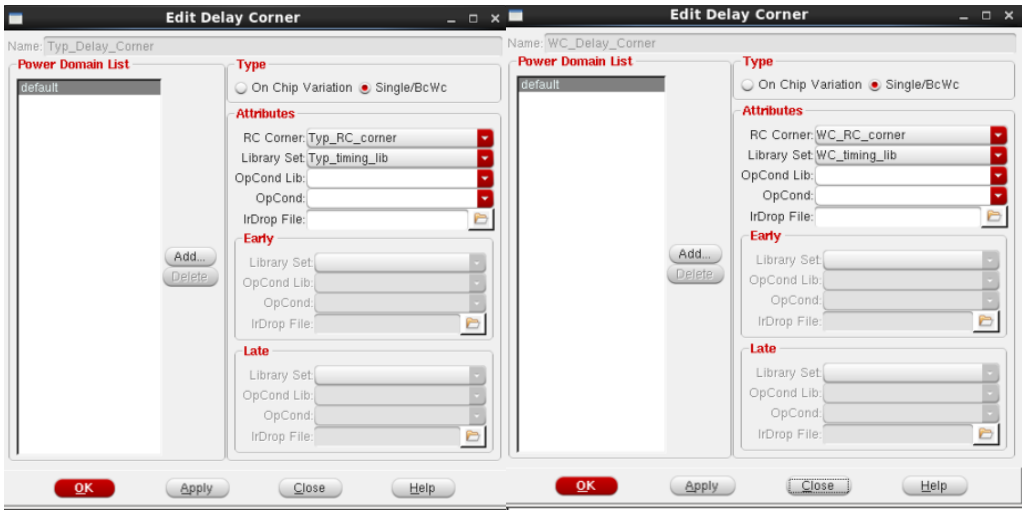


Figura 71: Delay corners para caso típico (izquierda) y peor escenario (derecha)

Posteriormente se definen “Constraint modes”. Se agregarán 2 que corresponden al caso típico y al peor escenario respectivamente.

Para el caso típico se da doble clic, se da un nombre como “typ_constraint” (Figura 72) y “wc_constraint” (Figura 73); y se especifican los archivos “.sdc” resultantes de los *netlist* generados con RTL Compiler.

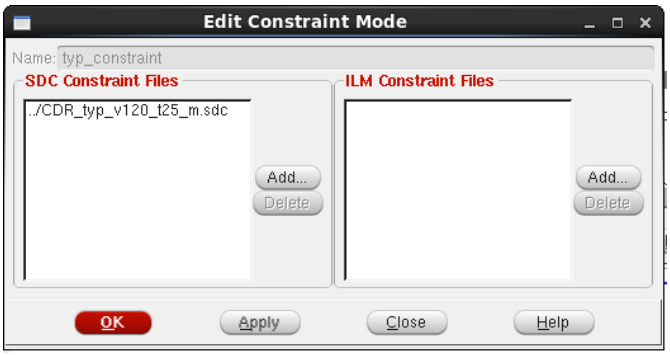


Figura 72: Constraint mode para escenario típico

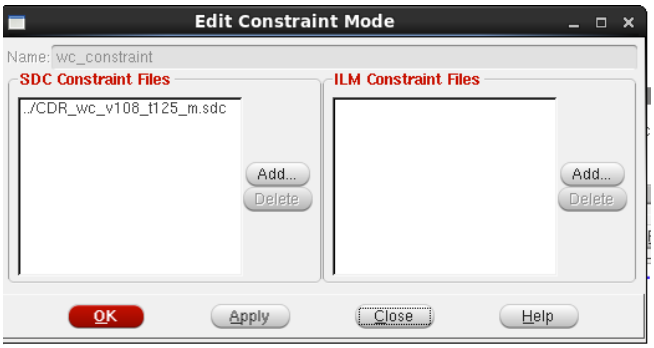


Figura 73: Constraint mode para peor escenario

Después de los pasos anteriores, se definen los Views que consisten en “Constraint + DelayCorners”, de la siguiente forma.

Se define un View para caso típico y otro para peor caso, y se elige el constraint mode y el delay corner correspondiente como se ve en la Figura 74.

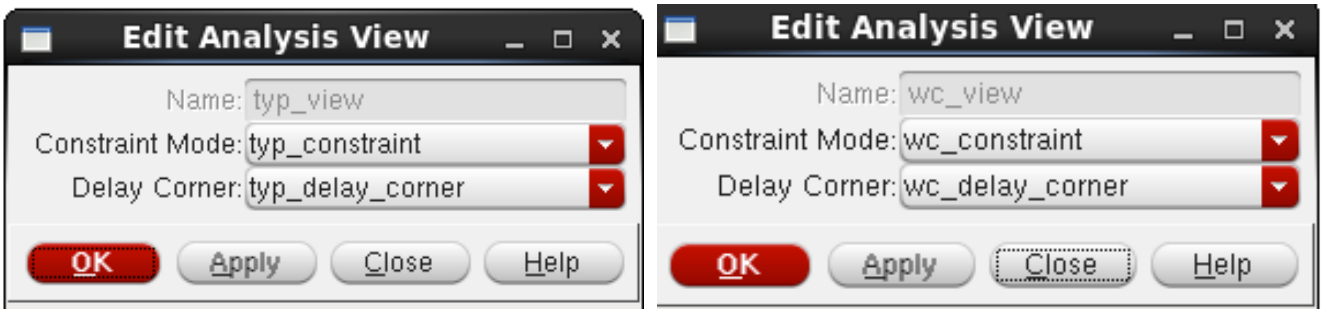


Figura 74: Analysis view para escenario típico (izquierda) y peor escenario (derecha)

Posteriormente se indica que:

- El "Setup Analysis View" se hará con el Worst case.
- El "Hold Analysis View" se hará con el Typical case.

Al finalizar lo anterior, se pueden guardar los parámetros anteriores como en la Figura 75.

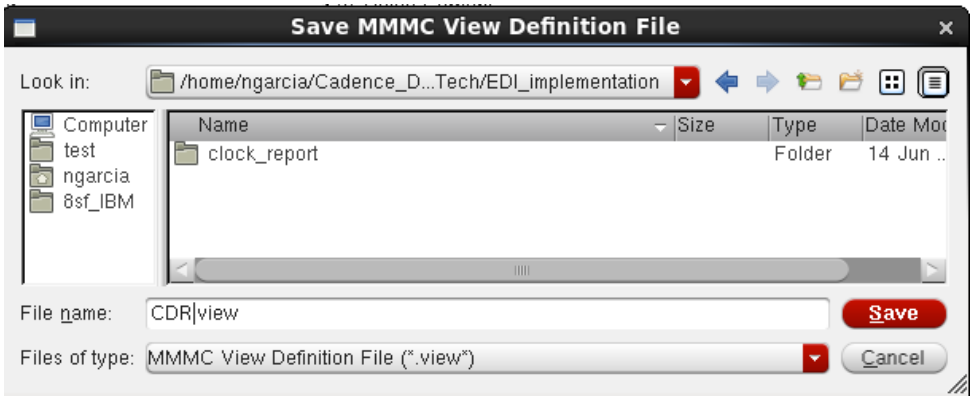


Figura 75: La configuración MMMC se puede guardar para ahorrar tiempo al realizar iteraciones de ajuste

Posteriormente se define el proceso del diseño, mediante el comando en consola:

- setDesign -process 130

Y se obtiene el siguiente resultado (Figura 76) que muestra el área sugerida por Encounter para el diseño. Los archivos resultantes "globals" y "views" están en el Apéndice Q y Apéndice R, respectivamente.

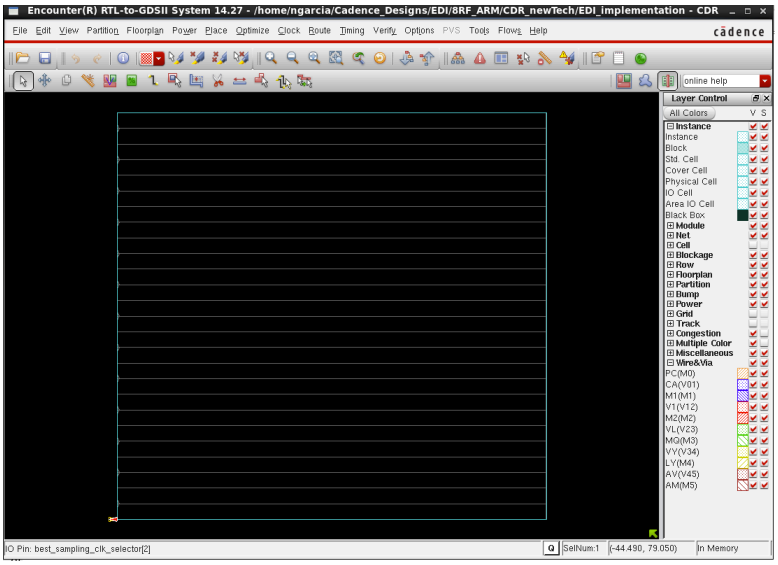


Figura 76: Encounter después de agregar la configuración de análisis multi modo, multi esquina

Lo siguiente es definir el Floorplan. Se usa el menú "Floorplan" en la GUI de encounter (Figura 77), y se elige la opción Specify Floorplan. El resultado de estos parámetros se aprecia en la Figura 78. Se puede observar que la distancia del core al margen donde estaría los puertos de entrada y salida, hay 11um, mientras que el espacio vertical entre stripes es de 4.8um pues es el ancho de las celdas estándar BiCMOS. Aproximadamente se necesitaría un área de 301.80um2 para el módulo CDR a sintetizar.

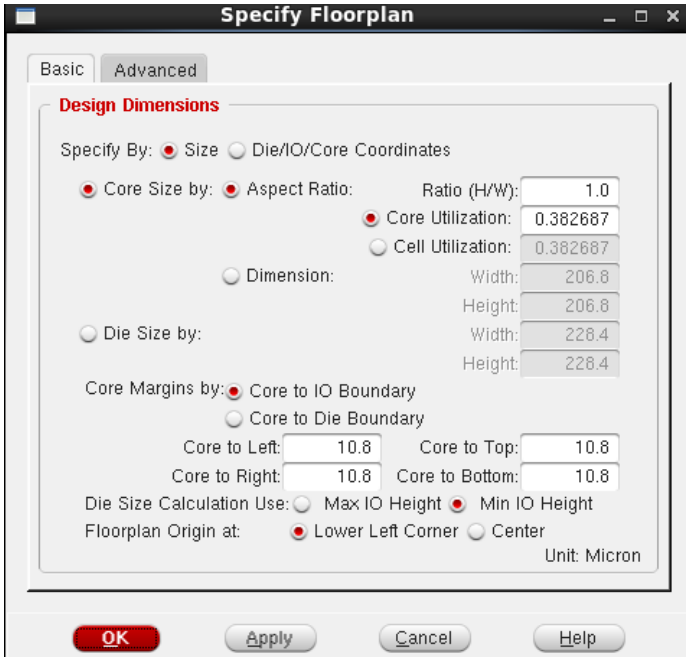


Figura 77: Especificación de Floorplan

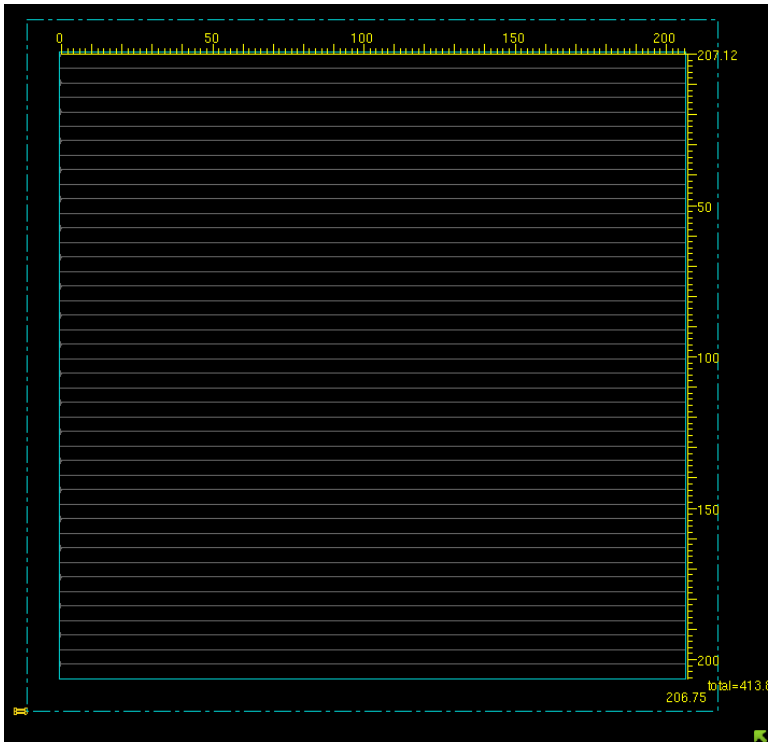


Figura 78: Floorplan obtenido

Al tener el floorplan preliminar, se continúa con la definición de las conexiones a las terminales de alimentación, para lo cual se puede emplear la interface gráfica de "Global Net Connections".

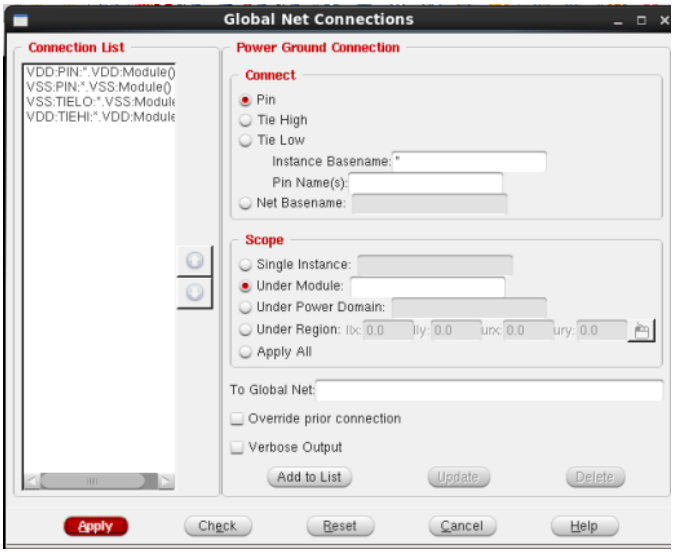


Figura 79: Asistente gráfico para especificar los global net connections

Una vez que se tiene lo anterior, se puede comenzar a construir el power grid para el diseño. Para esto, se agregan anillos alrededor del core del chip. Lo anterior se realizó mediante la GUI del menú Power→ Power Planning→ Add ring (Figura 80).

Se seleccionaron los nodos de alimentación como se muestra en la siguiente ventana de la Figura 82. Es importante señalar que el orden en el que se agreguen en esta interface, determina como aparecerán en el floorplan. Por ejemplo, al agregar VDD primero, quiere decir que éste aparecerá en la parte interna del anillo, mientras que VSS aparecerá al exterior del anillo.

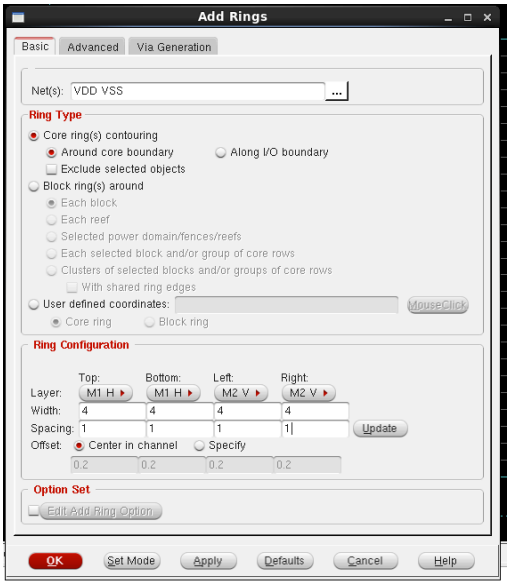


Figura 80: Especificación de anillo de alimentación

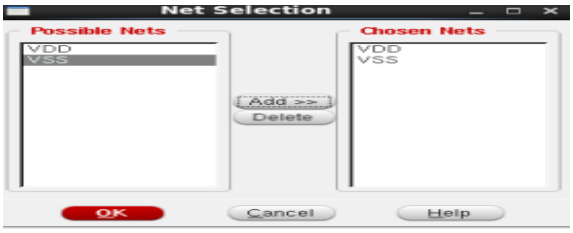


Figura 81: Selección de terminales a agregar en el anillo de alimentación.

Dado que tenemos 11um de espacio entre el core del chip y los bordes, se eligió para el anillo un ancho de 4um y una separación de 1um. Además, se definió que el anillo estuviera alrededor del borde del core. Posteriormente, en la configuración del anillo, se definen qué tipo de metales se emplearán dependiendo de la orientación del segmento del anillo.

Como criterio se tomó que las partes del anillo que están en posición horizontal (top y bottom) emplearán metal 1 (número impar, color azul); mientras que aquellos en posición vertical (left y right) emplearán metal 2 (número par, color rojo).

En la Figura 82 se puede observar que el anillo interno es de VDD, pues fue el primero en agregarse.

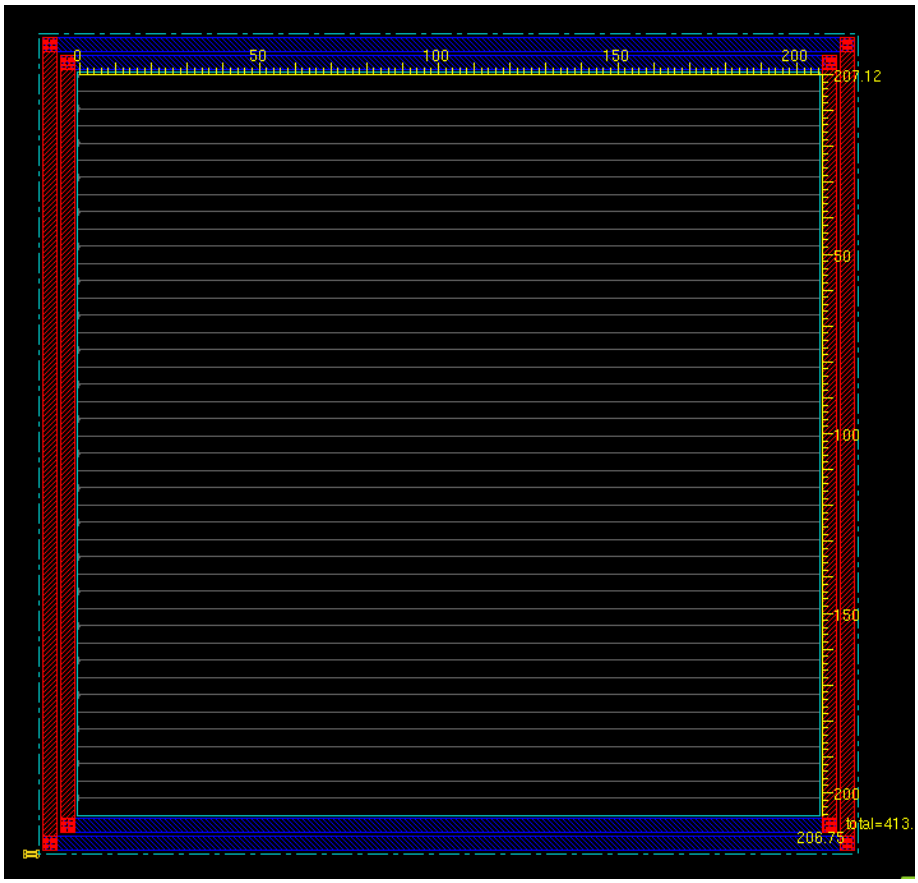


Figura 82: Floorplan y anillos de alimentación

Posteriormente se agregaron stripes o barras horizontales así como verticales, de la siguiente forma: menú Route→ Special route. Se agregaron los nodos de alimentación VDD y VSS, además se seleccionaron las opciones “Block pins y Pad pins” como se observa en la Figura

83. Las barras horizontales tendrán un color azul que corresponde al metal 1 como se muestra en la Figura 84.

Figura 83: Adición de stripes horizontales

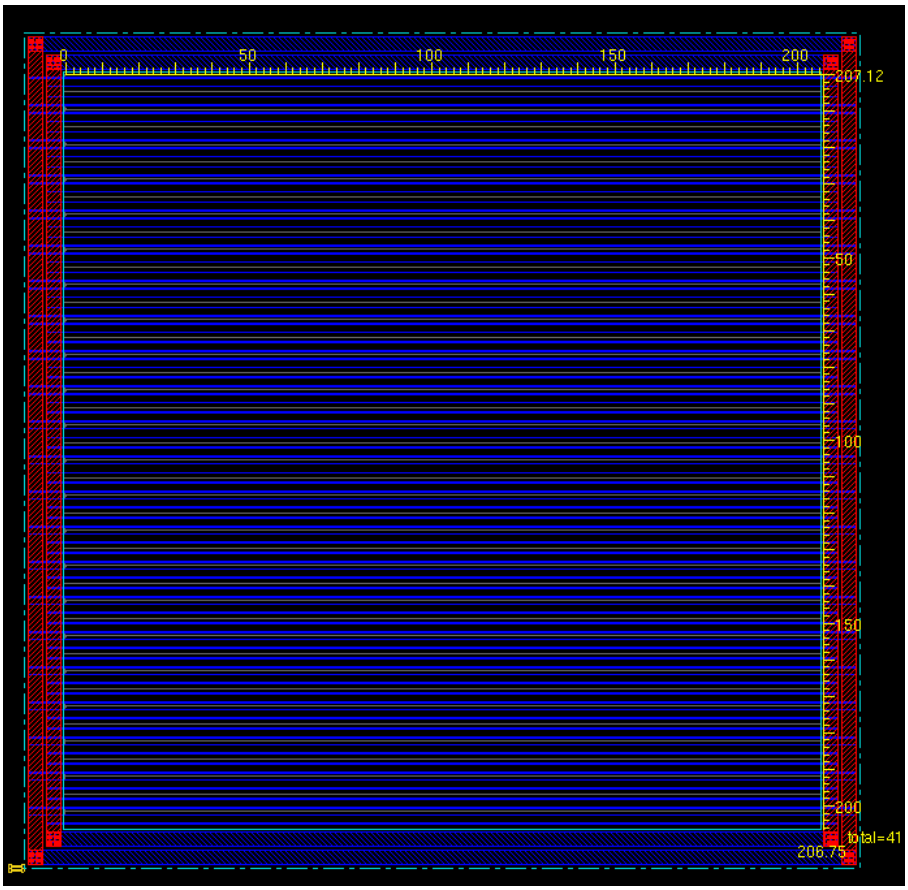


Figura 84: Powergrid con stripes horizontales

Es posible realizar verificaciones de DRC en este punto del diseño para asegurar que el floorplan y powergrid no tengan problemas de este tipo. Los resultados de estas verificaciones se aprecian en la Figura 85 y Figura 86.

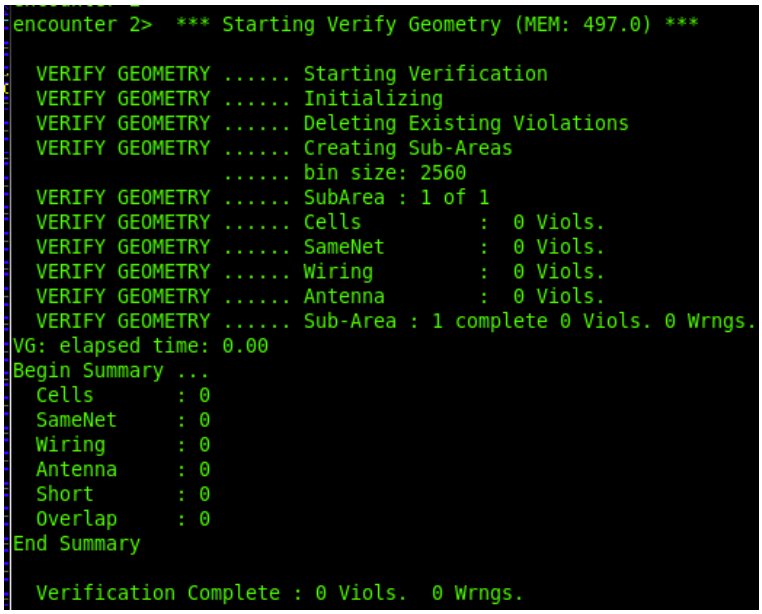


Figura 85: Reporte de violaciones de geometría

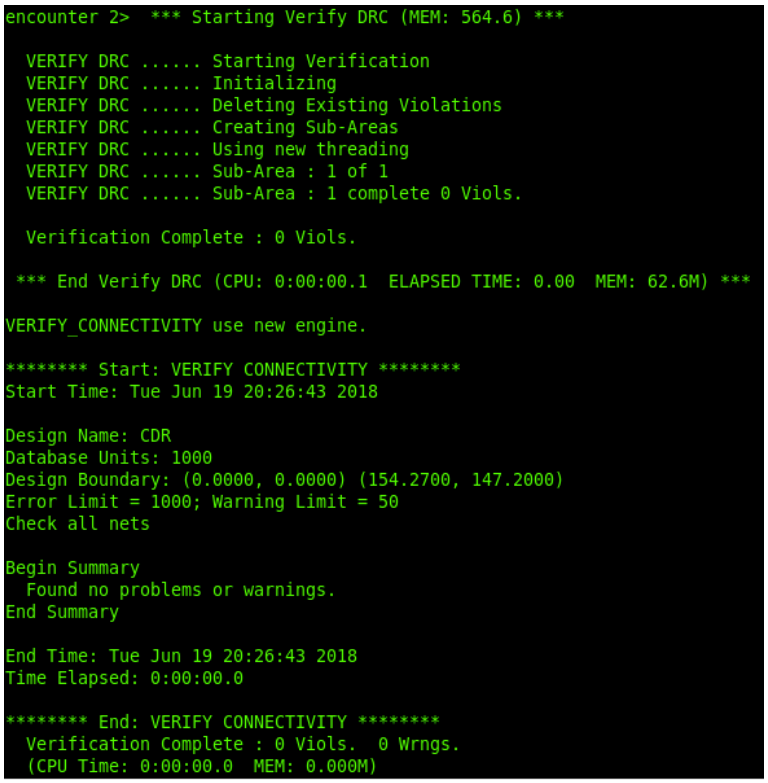


Figura 86: Reporte de violaciones de conectividad y DRC

Posteriormente, se agregaron stripes verticales mediante el menú: Power → Power Planning → Add Stripes y se configuraron como se muestra en la Figura 87. Por otro lado, el espacio entre el borde del chip y los stripes verticales se definió de 32um aproximadamente.

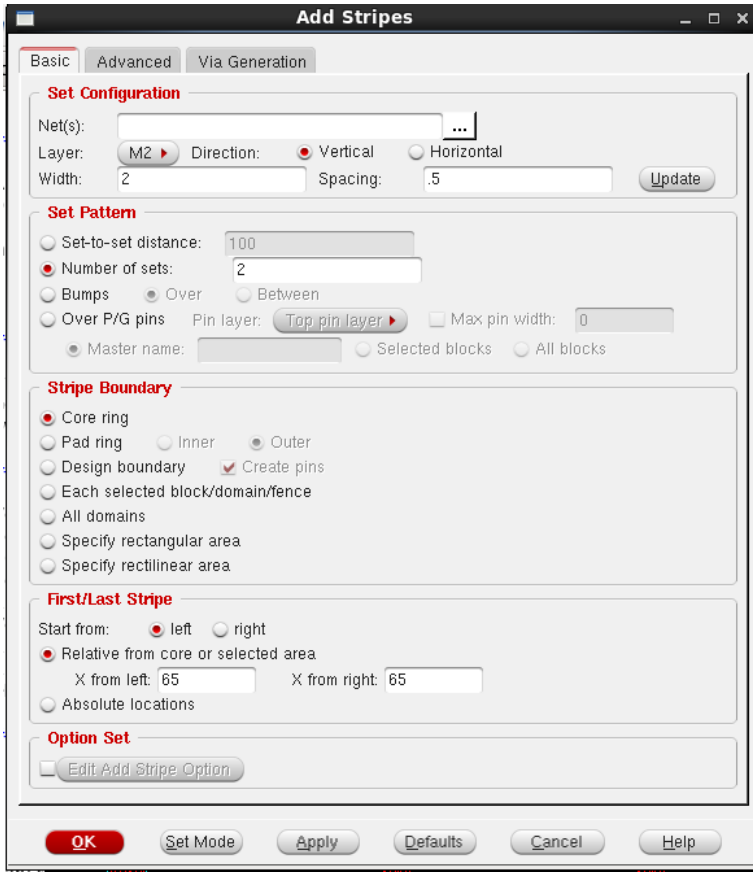


Figura 87: Se agregaron 3 stripes verticales con un ancho de 2um, separados 0.5um. .

El resultado de los parámetros anteriores se puede ver en la Figura 88.

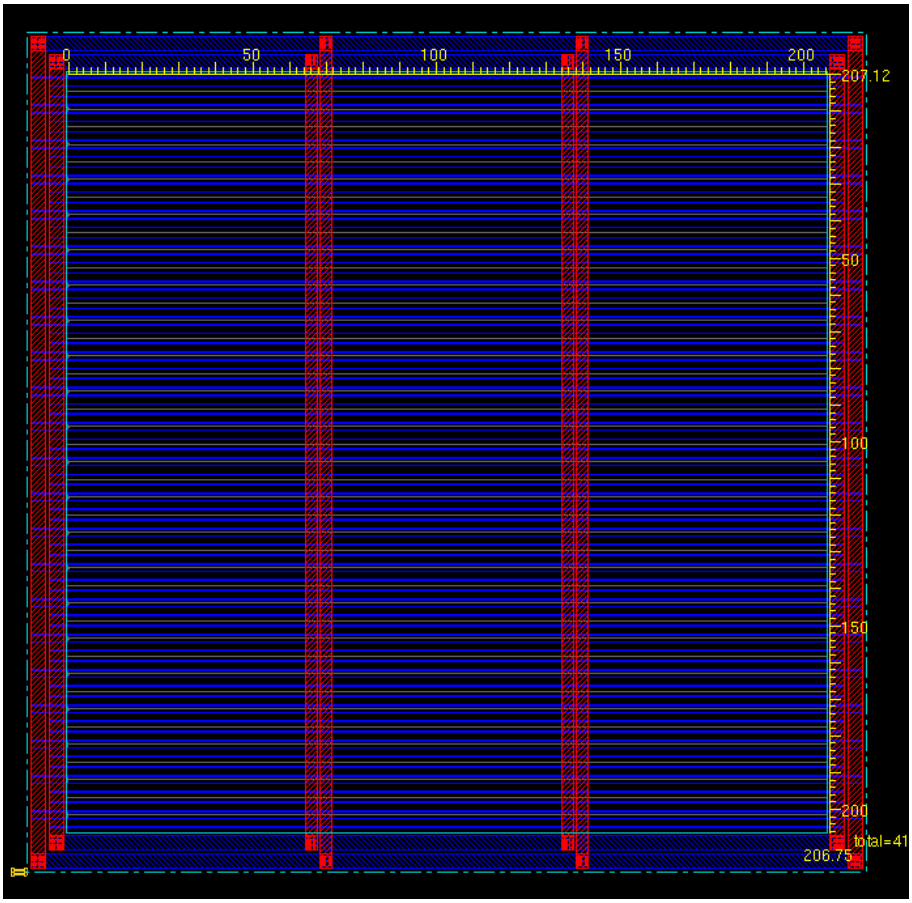


Figura 88: Floorplan con powergrid y stripes horizontals y vertical

Una vez que están definidos los stripes, tanto horizontales como verticales, se colocaron las celdas estándar del diseño de la siguiente forma. Se eligió el menú Place-→ Place Standard Cell. Posteriormente se abrió la ventana "Mode" y se desactivaron las opciones "Ignore Scan Connection" y "Reorder Scan connection" como en la Figura 89.

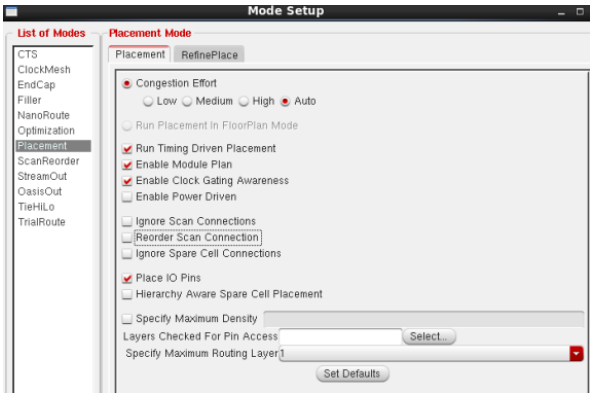


Figura 89: Opciones para colocación de las celdas estándar

El resultado de estos pasos es el que se observa en la Figura 90.

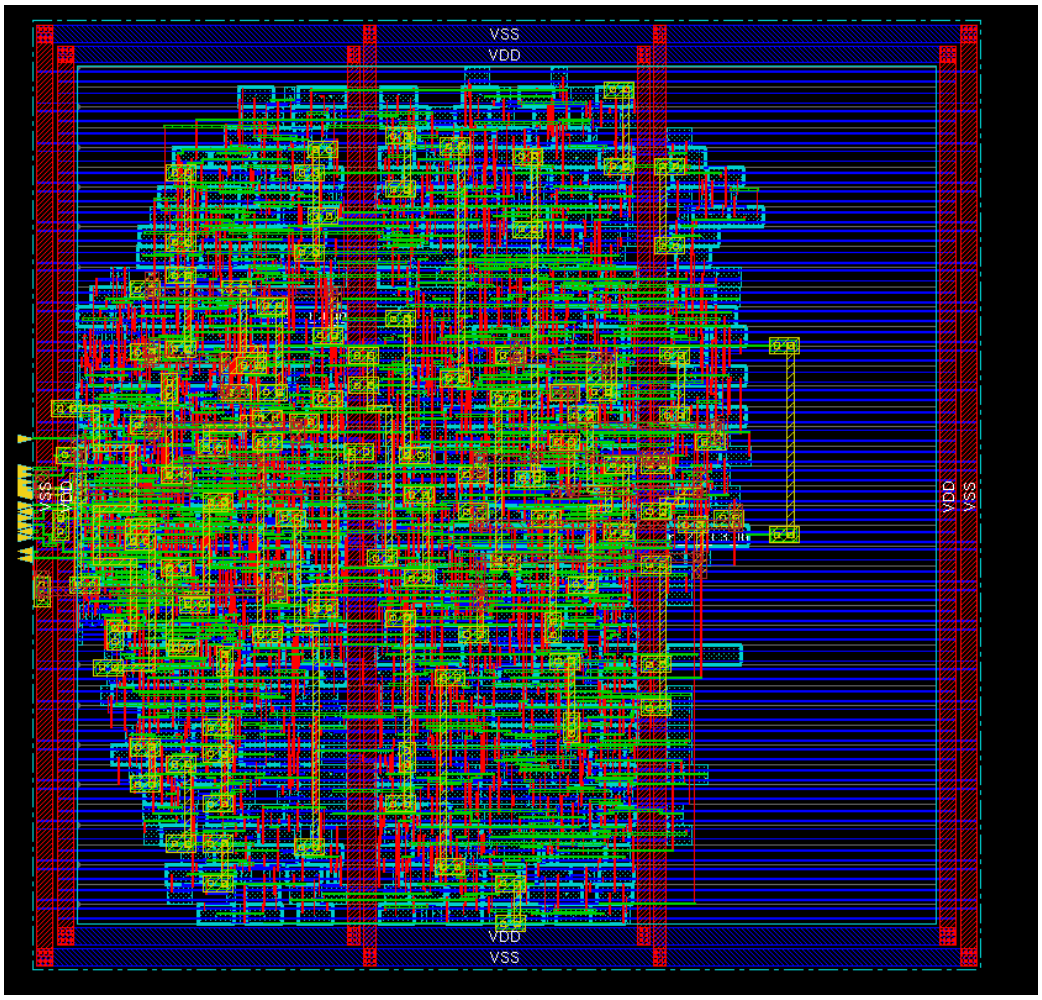


Figura 90: Síntesis física sin optimización

Al observar la vista Amoeba, se nota cómo se distribuyeron los bloques funcionales del CDR, como en la Figura 91.

Figura 91: Vista amoeba

Al cambiar la vista, se pueden observar como se colocaron las celdas estándar en el floorplan tal como se muestra en la Figura 92.

Posterior a esto, se debe construir el árbol de reloj para el circuito, lo cual se realiza de la siguiente manera. Primero se ejecuta el comando:

- setCTSMode -engine ck

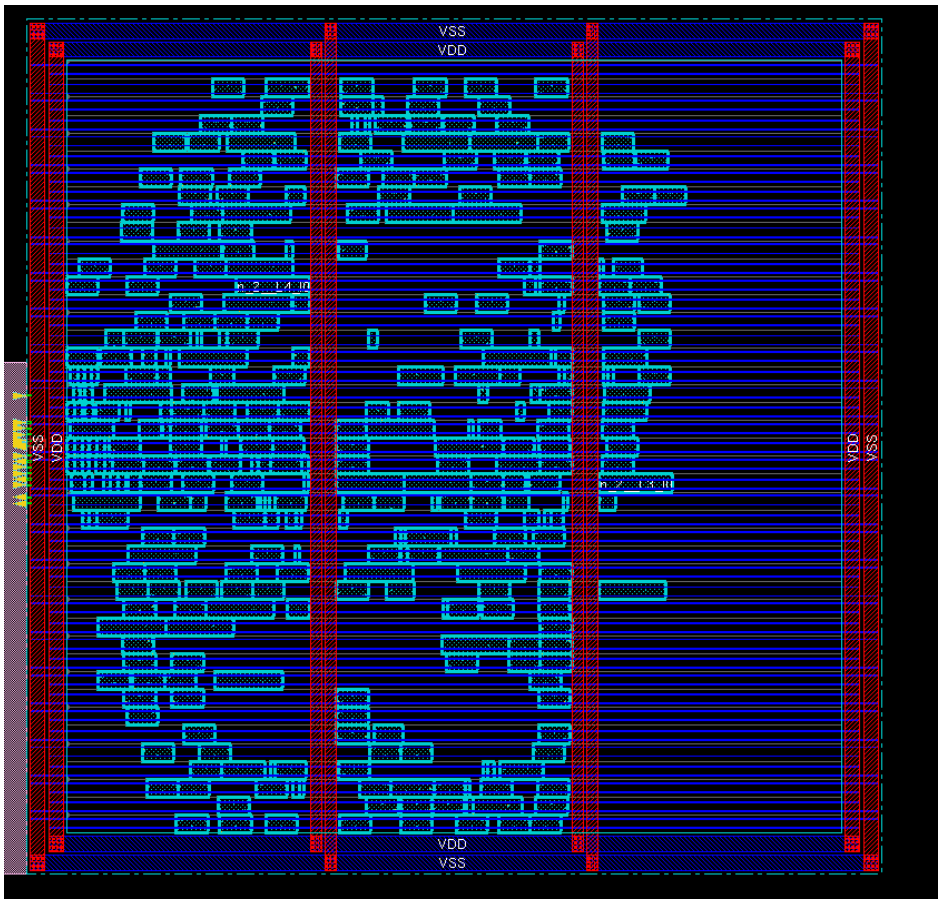


Figura 92: Las celdas estándar se colocan en el área definida por el floorplan

Después de obtener el ruteo de las celdas estándar se realizó la síntesis del árbol de reloj (clock tree synthesis). Para lo anterior, se empleó el menú Clock-→ synthesis Clock tree; después se selecciona la opción "Gen Spec" que abre una nueva ventana en donde se seleccionaron todas las celdas disponibles y se especificó el nombre del archivo ".ctstch" que se generará como se ve en la Figura 93. Se puede desplegar el árbol de reloj como se ve en la Figura 94.

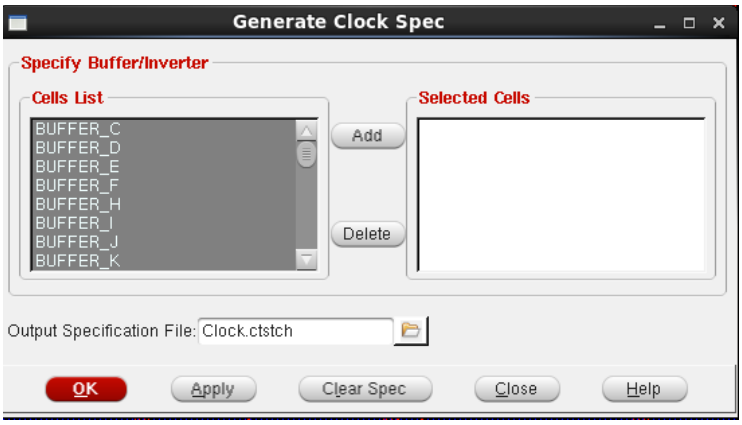


Figura 93: Celdas disponibles para la síntesis del árbol de reloj

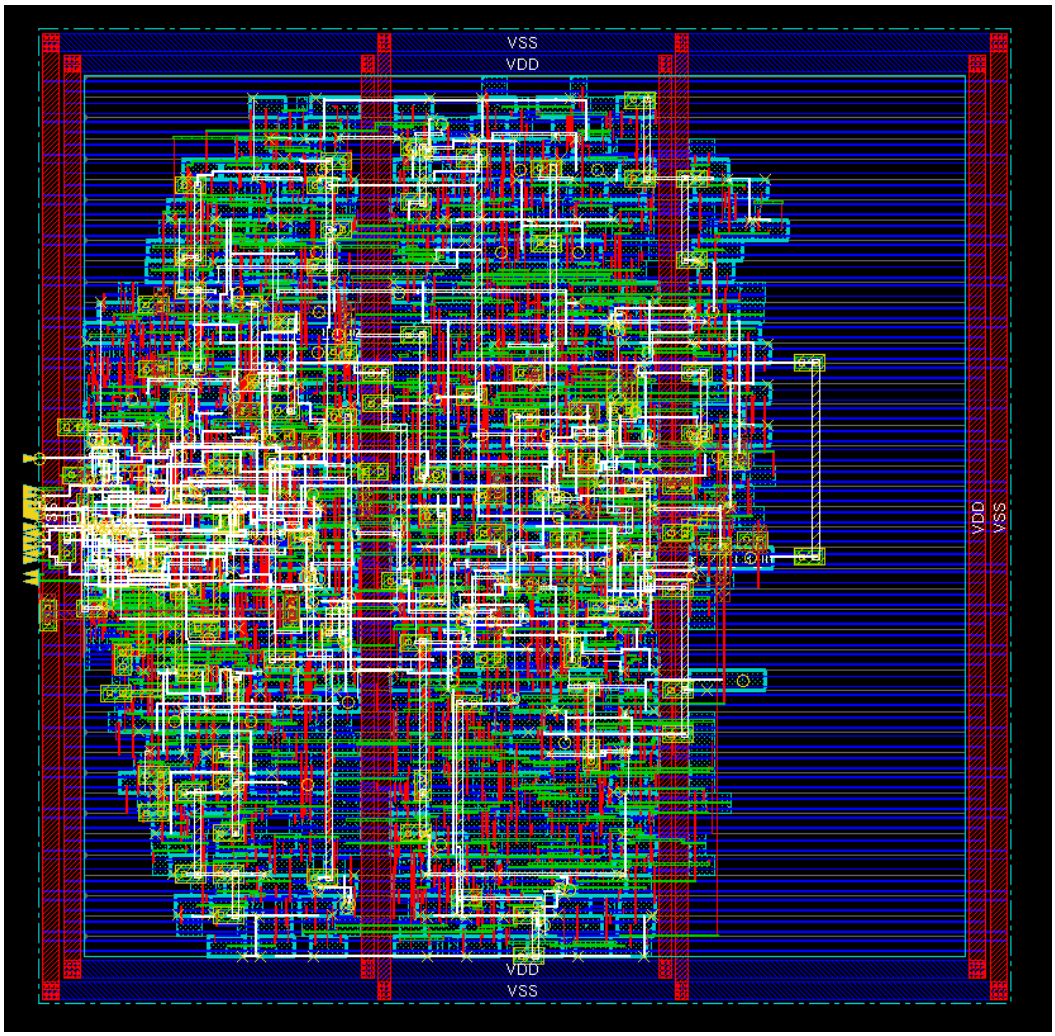


Figura 94: Síntesis del árbol de reloj

Después de esto, el ruteo comenzó mediante la herramienta “Nanoroute”.

Se realizaron pruebas de DRC, conectividad y geometría con los resultados de la Figura 95 y Figura 96.

```
encounter 3>  *** Starting Verify Geometry (MEM: 934.3) ***

  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: 0.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.
```

Figura 95: Reporte de violaciones de geometría

```
 *** verify geometry (CPU: 0:00:00.3  MEM: 63.5M)

encounter 3>  *** Starting Verify DRC (MEM: 997.7) ***

  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: 0:00:00.2  ELAPSED TIME: 0.00  MEM: 0.0M) ***

VERIFY_CONNECTIVITY use new engine.

******** Start: VERIFY CONNECTIVITY ********
Start Time: Wed Jun 20 19:31:47 2018

Design Name: CDR
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (159.7800, 142.4000)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Wed Jun 20 19:31:47 2018
Time Elapsed: 0:00:00.0

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

encounter 3>
```

Figura 96: Reporte de violaciones de conectividad y DRC

Al ejecutar el análisis de timing post CTS, se observa que el tiempo de setup tiene problemas (Figura 97), mientras que el de hold (Figura 98) está en el límite de lo aceptable.

```
#############################################################
#  Generated by:      Cadence Encounter 14.27-s035_1
#  OS:                Linux x86_64(Host ID fv00)
#  Generated on:      Fri Aug  3 21:18:21 2018
#  Design:            CDR
#  Command:           timeDesign -postCTS -prefix postCTS_setup
#############################################################

------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):| -7.532  | -5.662  | -7.532  |
|           TNS (ns):|-208.990 |-201.458 | -7.532  |
|    Violating Paths:|   76    |   75    |    1    |
|          All Paths:|   303   |   293   |   10    |
+--------------------+---------+---------+---------+
```

Figura 97: Resultado de timing post CTS para el setup

```
#############################################################
#  Generated by:      Cadence Encounter 14.27-s035_1
#  OS:                Linux x86_64(Host ID fv00)
#  Generated on:      Fri Aug  3 21:18:22 2018
#  Design:            CDR
#  Command:           timeDesign -postCTS -prefix postCTS_hold -hold
#############################################################

------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):| -0.255  | -0.255  |  0.170  |
|           TNS (ns):| -8.930  | -8.930  |  0.000  |
|    Violating Paths:|   45    |   45    |    0    |
|          All Paths:|   303   |   293   |   10    |
+--------------------+---------+---------+---------+
```

Figura 98: Resultado de análisis de timing post CTS para hold

Dado que hubo problemas de timing se ejecutó un script para optimizarlo, con los siguientes resultados.

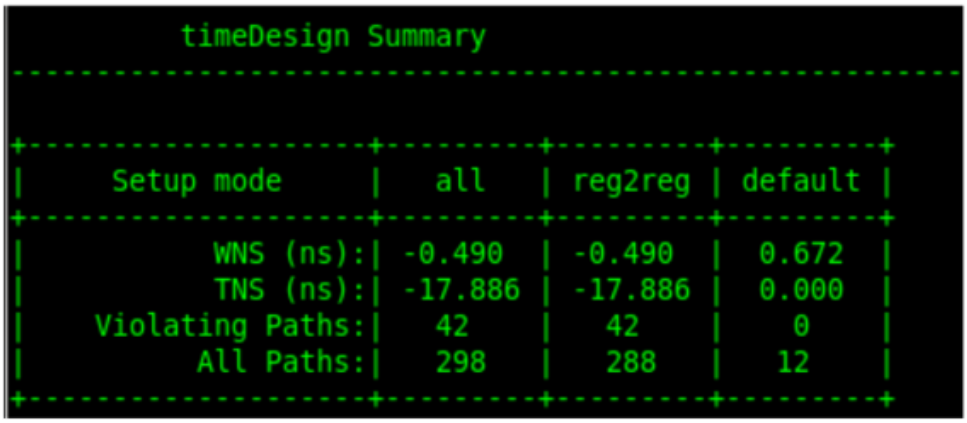


Figura 99: Resultado de timing después de la optimización para el setup

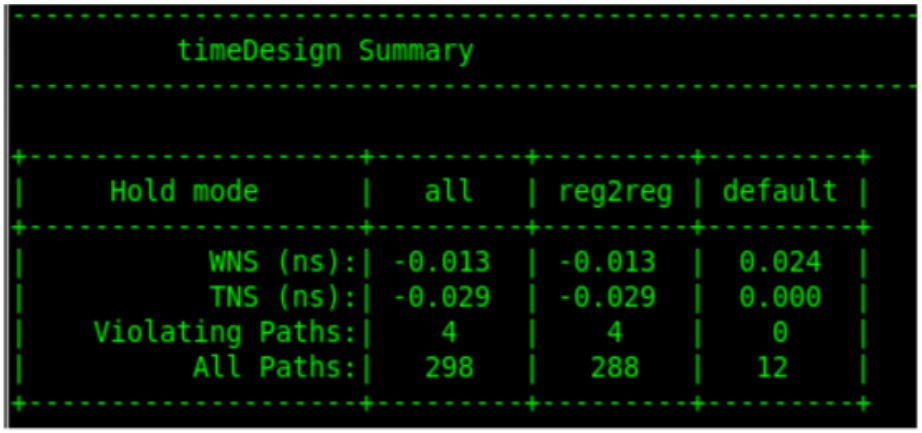


Figura 100: Resultado de timing después de la optimización para el hold

Con 2 stripes verticales separados 65um de distancia de los bordes del chip, considerando un floorplan de 207um por lado, se obtiene: hold -0.013ns y setup: -0.49ns, para señales de reloj de 800MHz. Sin embargo, se observa en las figuras anteriores que aún hay problemas de timing, los cuales se deberían ajustar.

Después de esto, se le agregan celdas “Fillers” al diseño, de lo contrario, la empresa que manufactura el chip le agregaría sus propios fillers con el riesgo de que haya errores en producción. En la Figura 101 se aprecia el resultado de agregar Fillers.

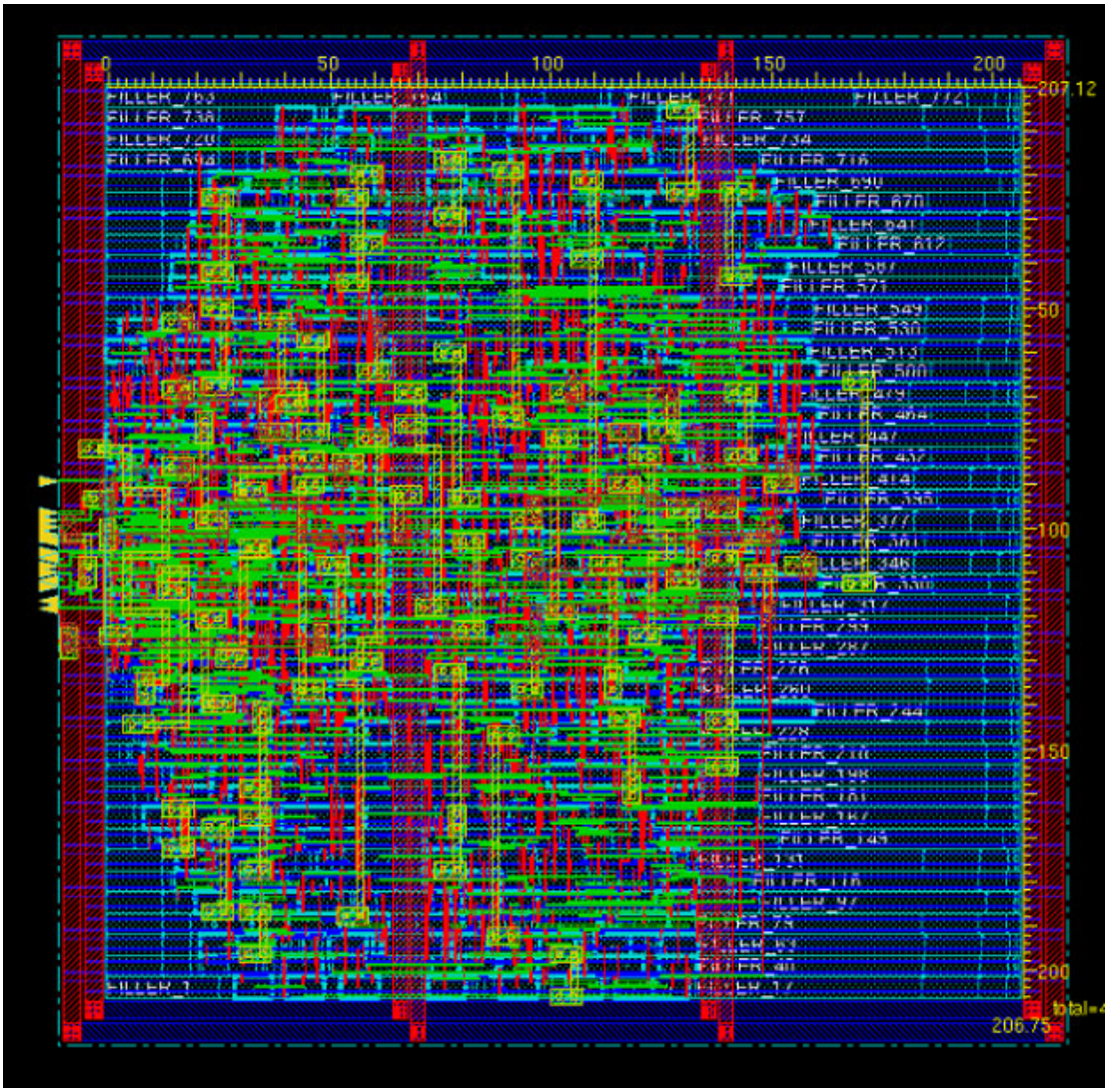


Figura 101: Diseño de CDR con Fillers

Posteriormente se exportó el diseño a GDS para poder continuar con el flujo de diseño en el software CAD Virtuoso. Así pues, se eligió el menú File->Save->GDS/OASIS, para abrir el cuadro de mensaje GDS/OASIS Export, que exportará el diseño. Se eligió una ruta en el campo "Output File" y en la opción "Map File" se usó el archivo "/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/bicmos8hp_soce2gds.map" (Figura 102 y Figura 103), pues con esto se está permitieno que Encounter use el archivo proveído por Global Foundries "bicmos8hp_soce2gds.map" para realizar el mapeo de las capas de metales.

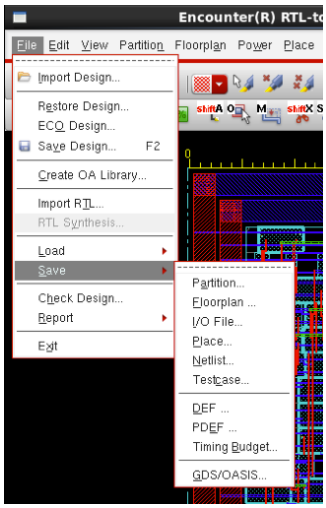


Figura 102: Se exportó el diseño a formato GDS

Además, se debe vincular el archivo gds de las celdas estándar, cuya información es necesaria para poder leer nuestro diseño en el software Virtuoso. Así pues. en el campo "Merge files" se elige el siguiente archivo:

- /media/Ext/libs/8HP_IP_CELL_AND_IO_Libs/BiCMOS8HP_Digital_Kit/ibm_cmos8hp/sc_1p2v_12t_rvt/v.20171220/gds2/BICMOS8HP_SC_1P2V_12T_RVT.gds

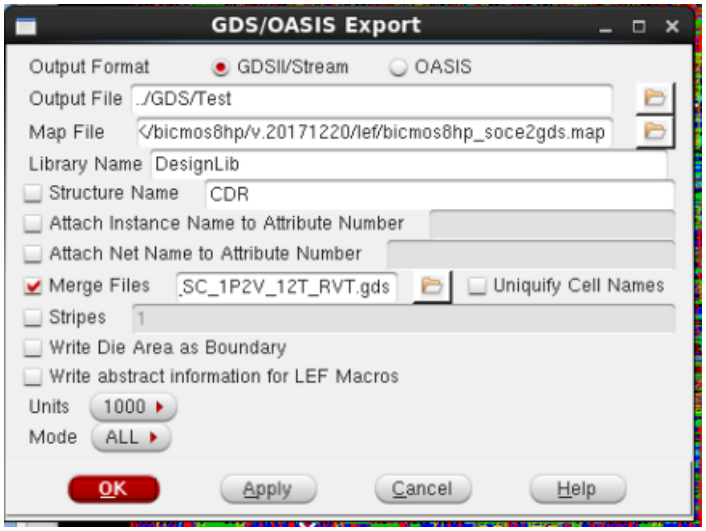


Figura 103: Configuración para exportar el diseño a formato GDS

Posteriormente, se abrió el PDK de Virtuoso para la tecnología "GF_8hp", se eligió el menú File->New->Library como se muestra en la Figura 104. Se dio un nombre a la librería y con la opción "Reference existing technology libraries", se escogió "bicmos8hp".

Figura 104: Se agregó una nueva librería con esta opción del menú File

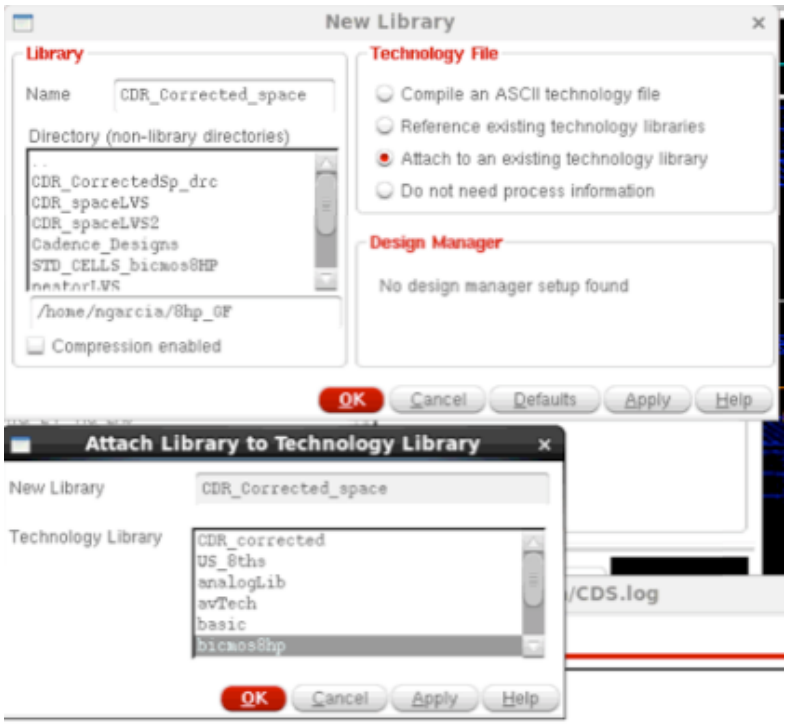


Figura 105: Adición de nueva librería bicmos8hp

Posteriormente se estableció que el editor fuera “gedit” mediante el comando editor=”gedit” como se aprecia en la Figura 106.

Figura 106: Definición de editor de texto para abrir futuras advertencias

Después se elige el Stream in con el GDS generado mediante Encounter y se especificó cuál es el "top level cell" de nuestro diseño.

Figura 107: Los campos relevantes contuvieron la información mostrada en la imagen

Posteriormente en la tab "Layers" se debe elegir en la sección "Map file" el botón Load-> Elegir archivo de la siguiente ruta:

- /media/Ext/libs/IBM_PDK/bicmos8hp/V1.7_0.6HP/cdslib/bicmos8hp/bicmos8hp.layermap

Con lo anterior se desplegarán las capas de metales para la librería que se empleó que fue “5AM_21” como se ve en la Figura 108.

Posteriormente se agregaron librerías mostradas en la Figura 109, en donde se agregó el archivo de la siguiente ruta (previamente agregado en el path editor como se ve en la Figura 110):

- /media/Ext/libs/IBM_PDK/bicmos8hp/v.20170531/cdslib61/BICMOS8HP_SC_1P2V_12T_RVT_CDSLIB

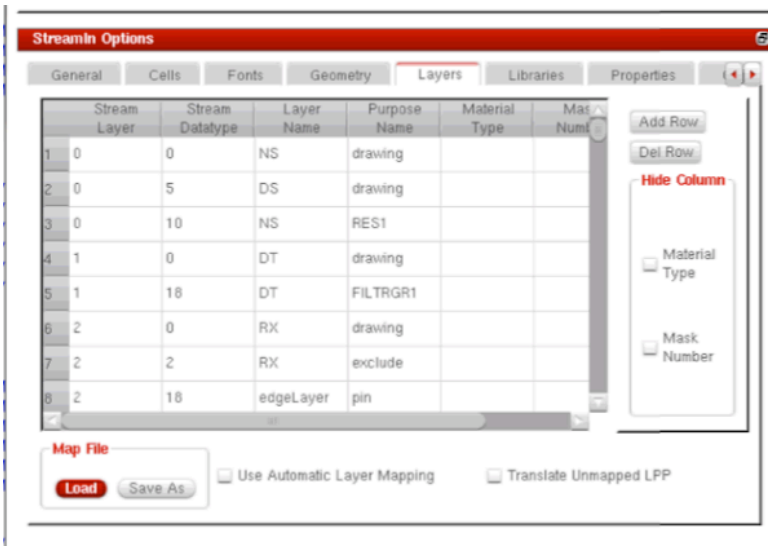


Figura 108: Configuración de capas de metales

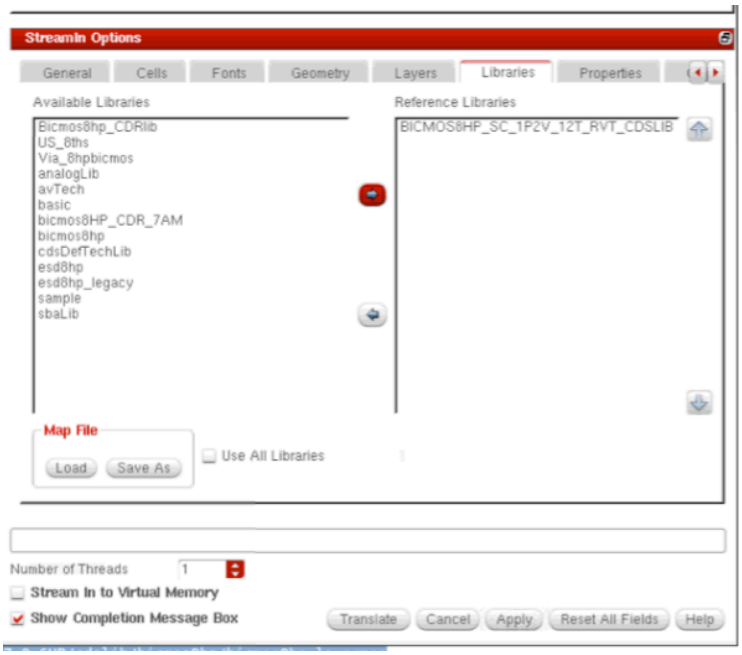


Figura 109: Configuración de librerías

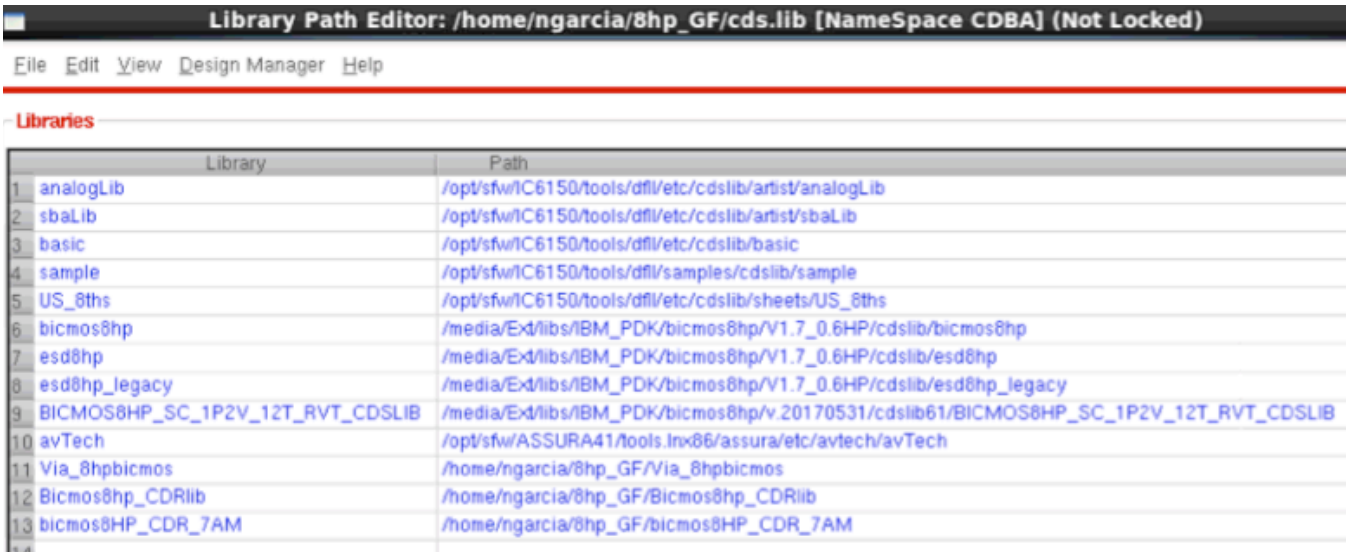


Figura 110: Librerías en Path editor

Una vez que ya se agregó la librería de las celdas estándar, se presiona el botón Translate, y se observa si hay errores y advertencias en el archivo de logs.

Una vez hecho lo anterior, se podrá ver en el Library Manager, que en la librería que acabamos de crear, se agregan las celdas estándar involucradas en el diseño importado del CDR, con sus vistas de layout.

Figura 111: Librería creada para el diseño CDR

Al abrir la top level cell “CDR” se puede ver que el layout del diseño importado se despliega con todas las capas de metales involucradas, tal como lo muestra la Figura 112, Figura 113 y Figura 114. A este diseño se le aplicará un análisis de DRC y LVS.

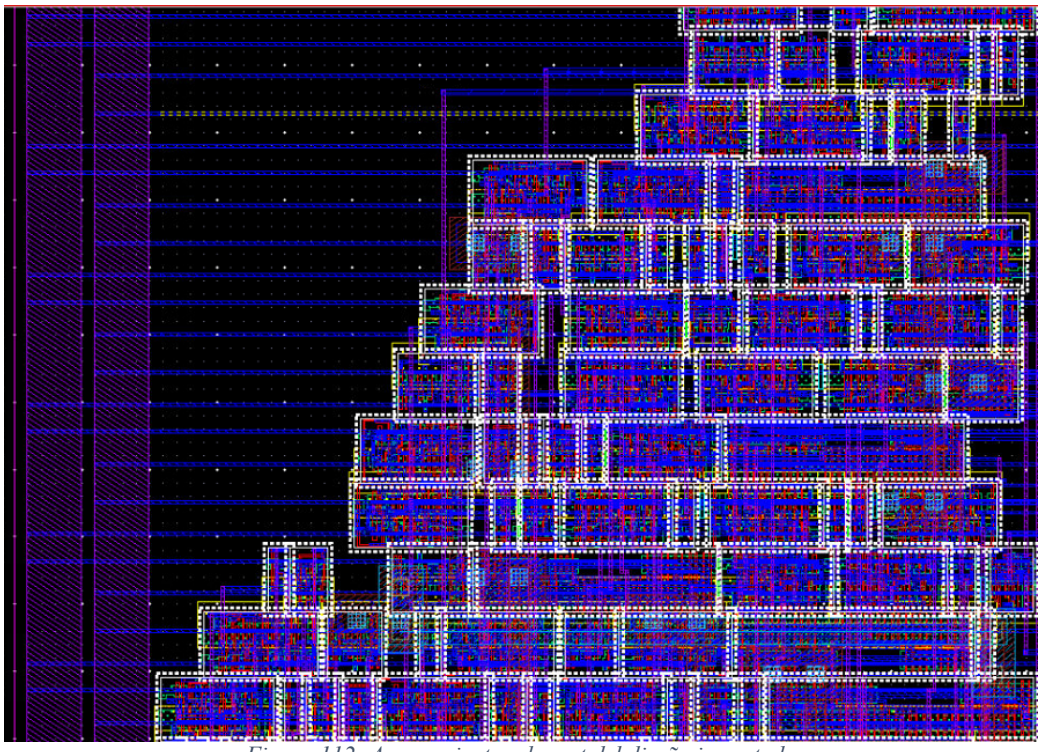


Figura 112: Acercamiento a layout del diseño importado

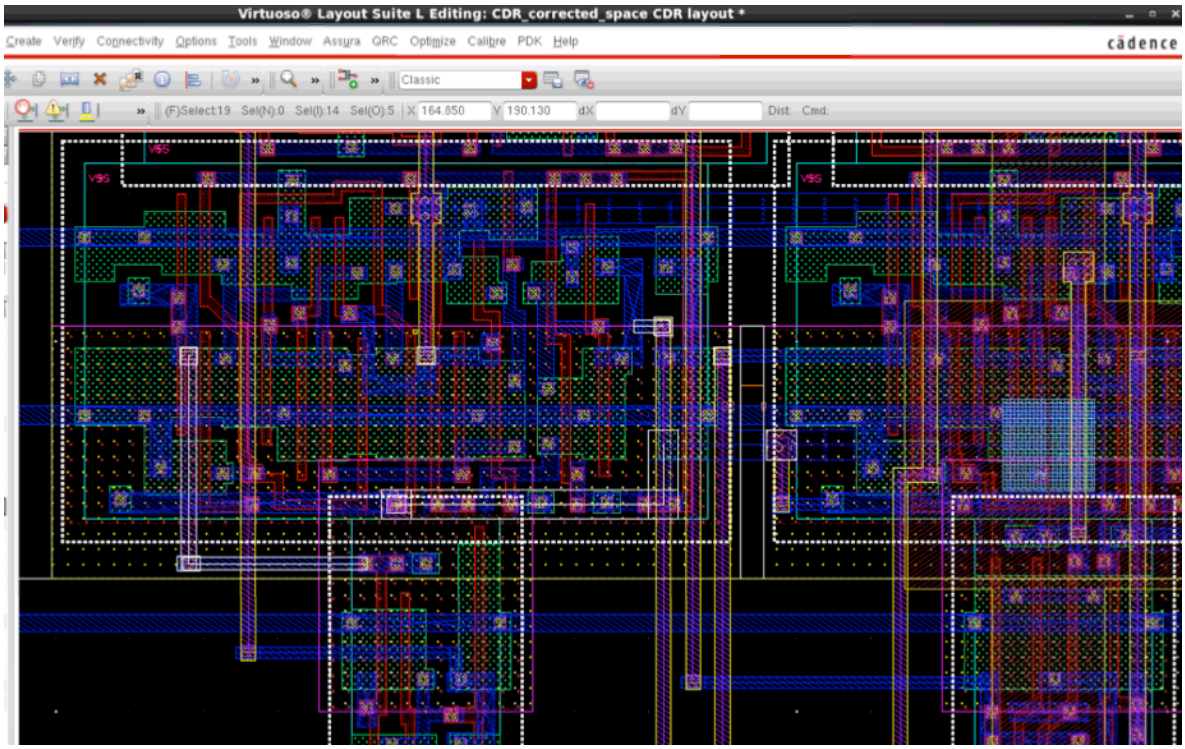


Figura 113: Acercamiento al layout del diseño importado

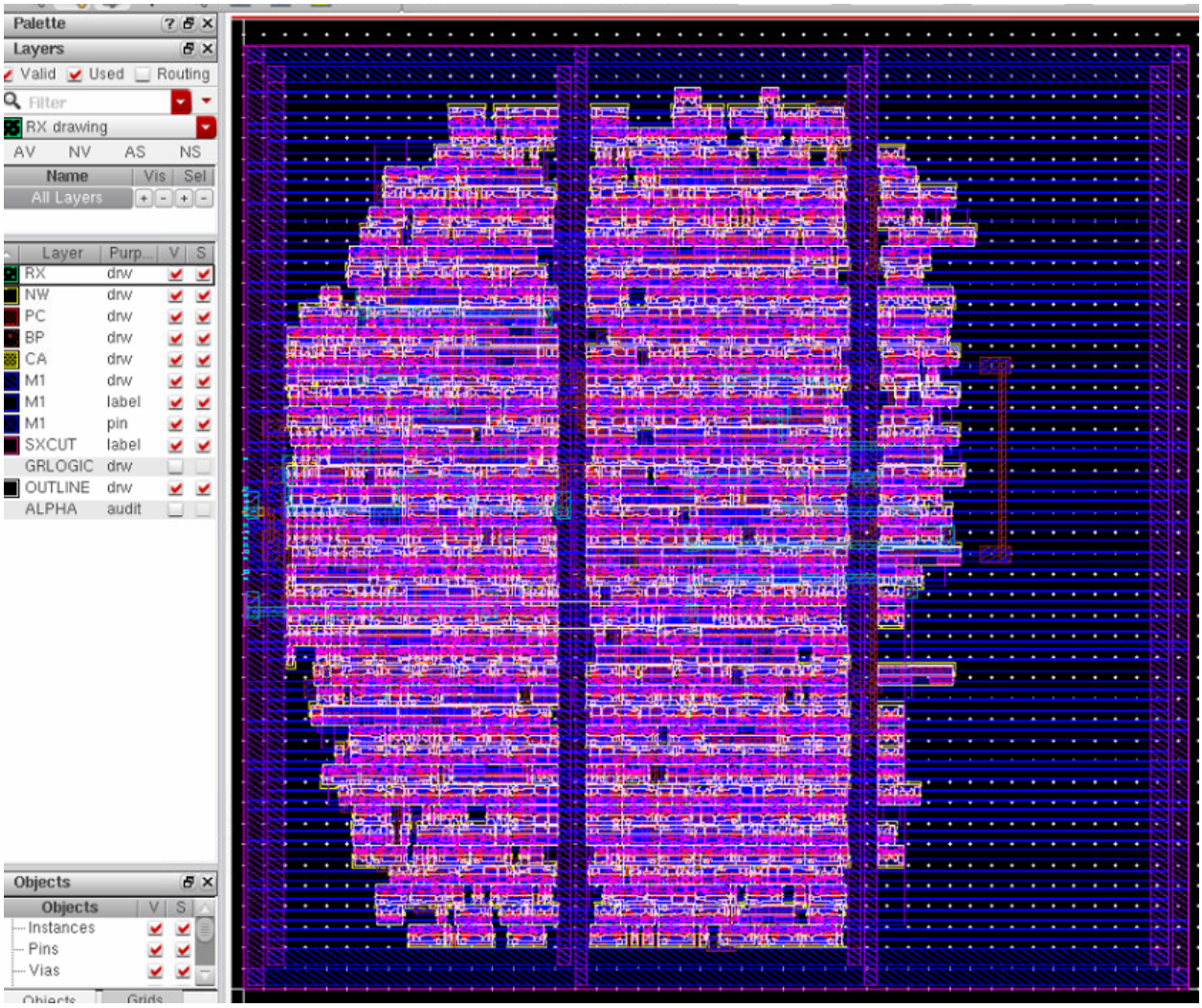


Figura 114: Layout del diseño importado desde Encounter a Virtuoso

3.5. Pruebas de DRC y LVS en Virtuoso

3.5.1 Ejecución de DRC

Para correr DRC se usa Assura. Se elige el menú Assura -> Run DRC. Como se nota en la Figura 115, en Run Name: se pone un nombre, y en "Run directory": se pone el directorio en donde se va a guardar.

En Rules File se especifica:

/opt/libs/IBM_PDK/bicmos8hp/V1.7_0.6HP/Assura/DRC/drc.rul

En Switch Names: “BEOL_STACK_211” y “Cell”.

Se presiona Apply, y se espera la generación del reporte.

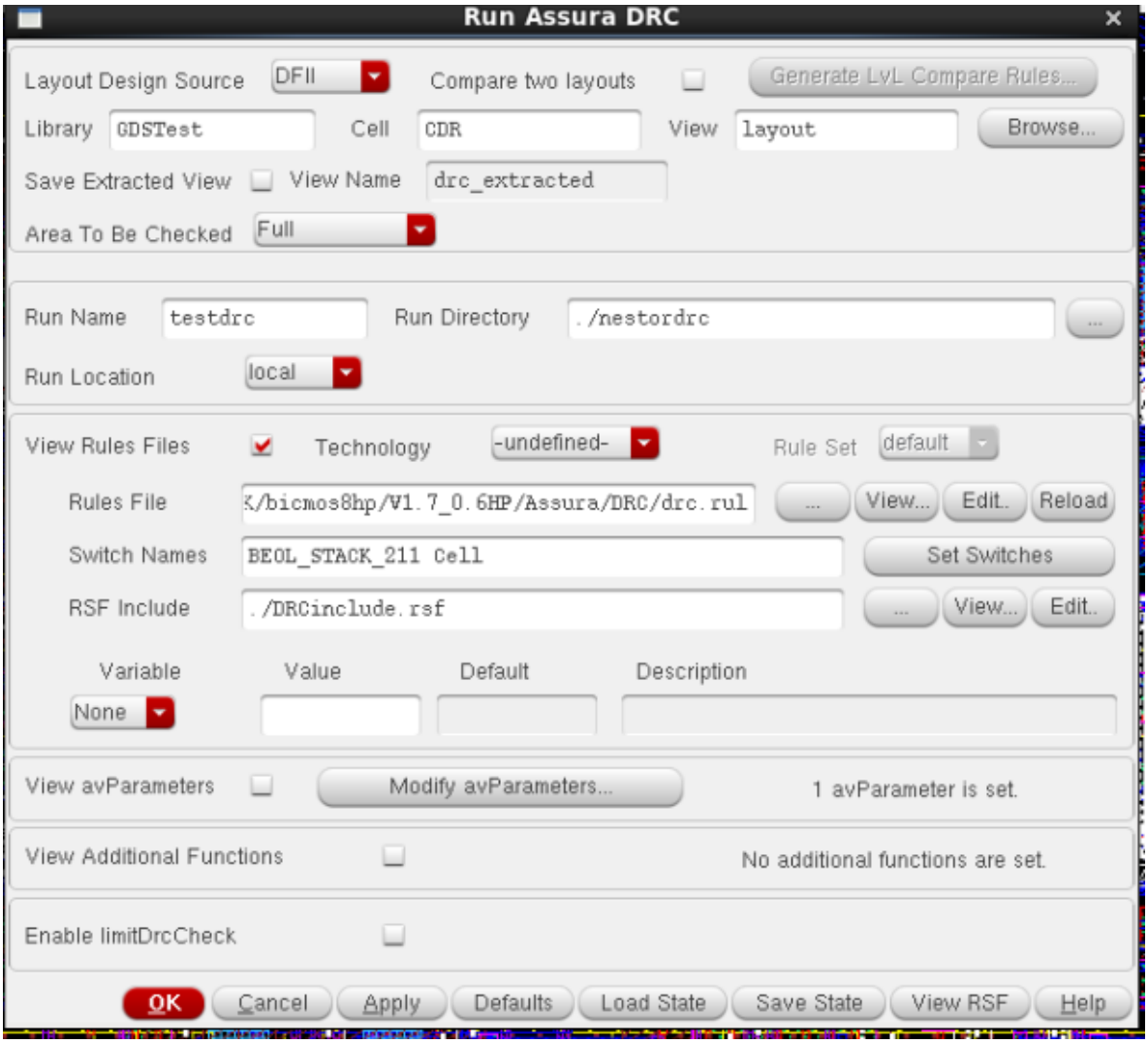


Figura 115: Parámetros para ejecutar análisis DRC

Se espera a que termine la ejecución del análisis y aparecerá una ventana como la de la Figura 116.

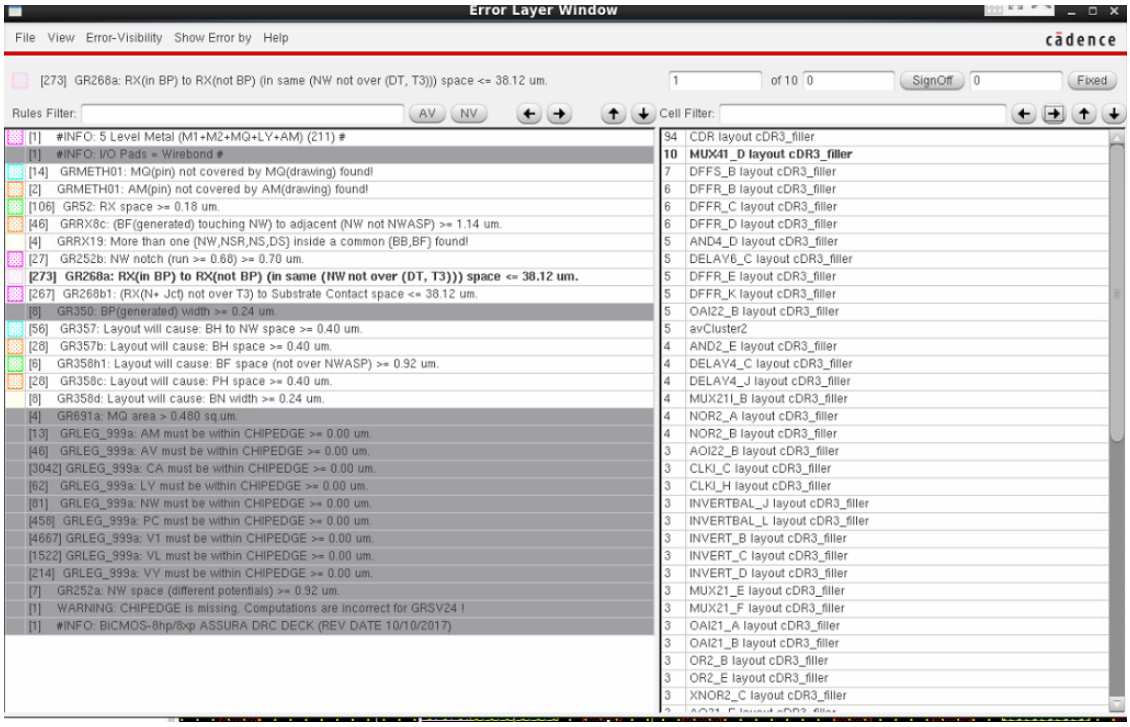


Figura 116: Reporte de resultados del análisis DRC

3.5.2 Ejecución de análisis LVS

Para correr un análisis LVS se hace lo siguiente:

En el menú Assura se eleige la opción run LVS. Como se muestra en la Figura 117, en la opción Switch Names se eligen las opciones: “NO_SUBC_IN_GRLOGIC” y “resimulate_extracted”. Además, se debe cambiar la extensión de los archivos en las opciones “Compare Rules” y “Binding files” a “.vldb”. Después se presiona el botón “Netlisting options” y aparecerá una ventana como la de la Figura 118.

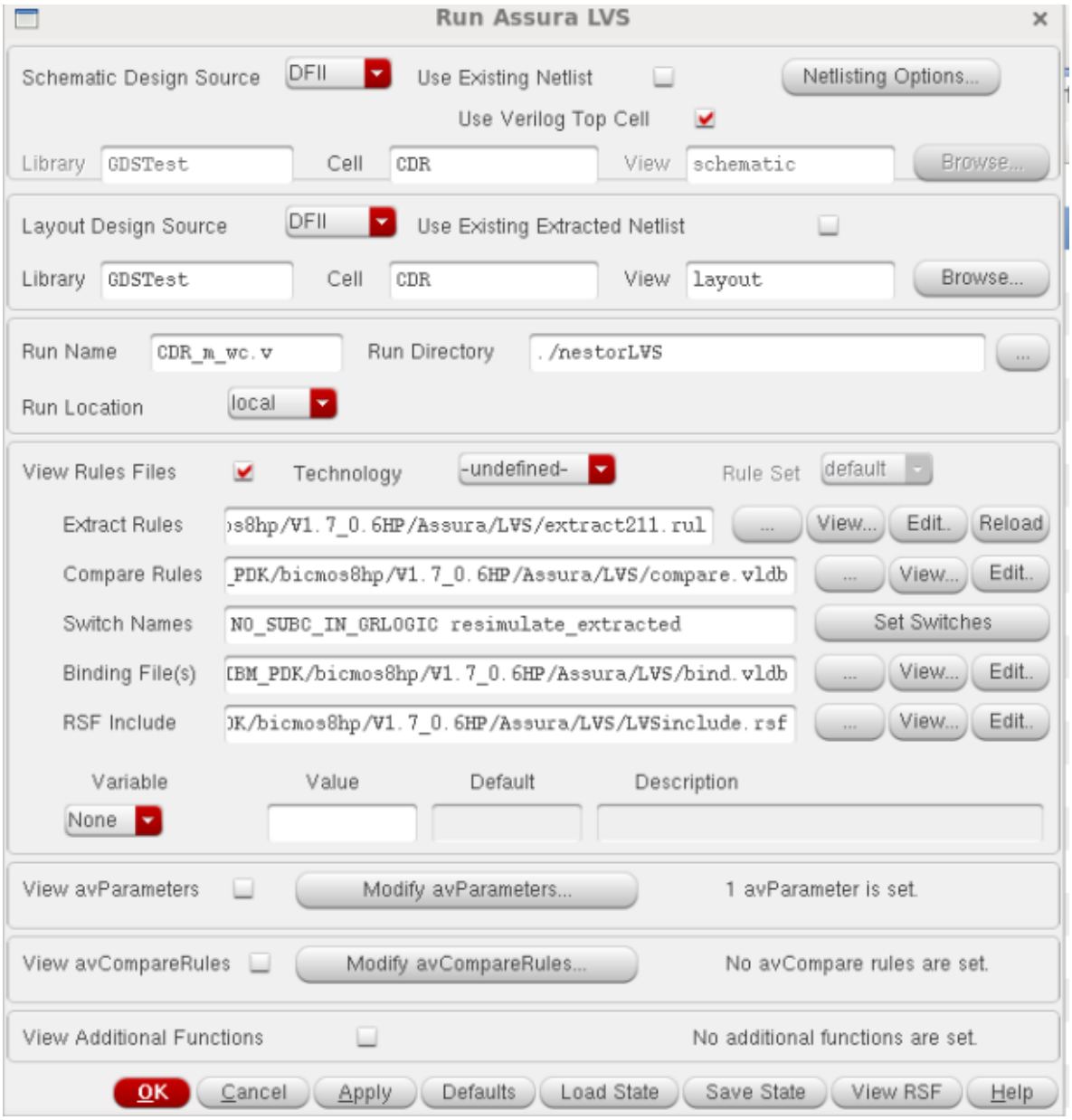


Figura 117: Parámetros para ejecución de LVS

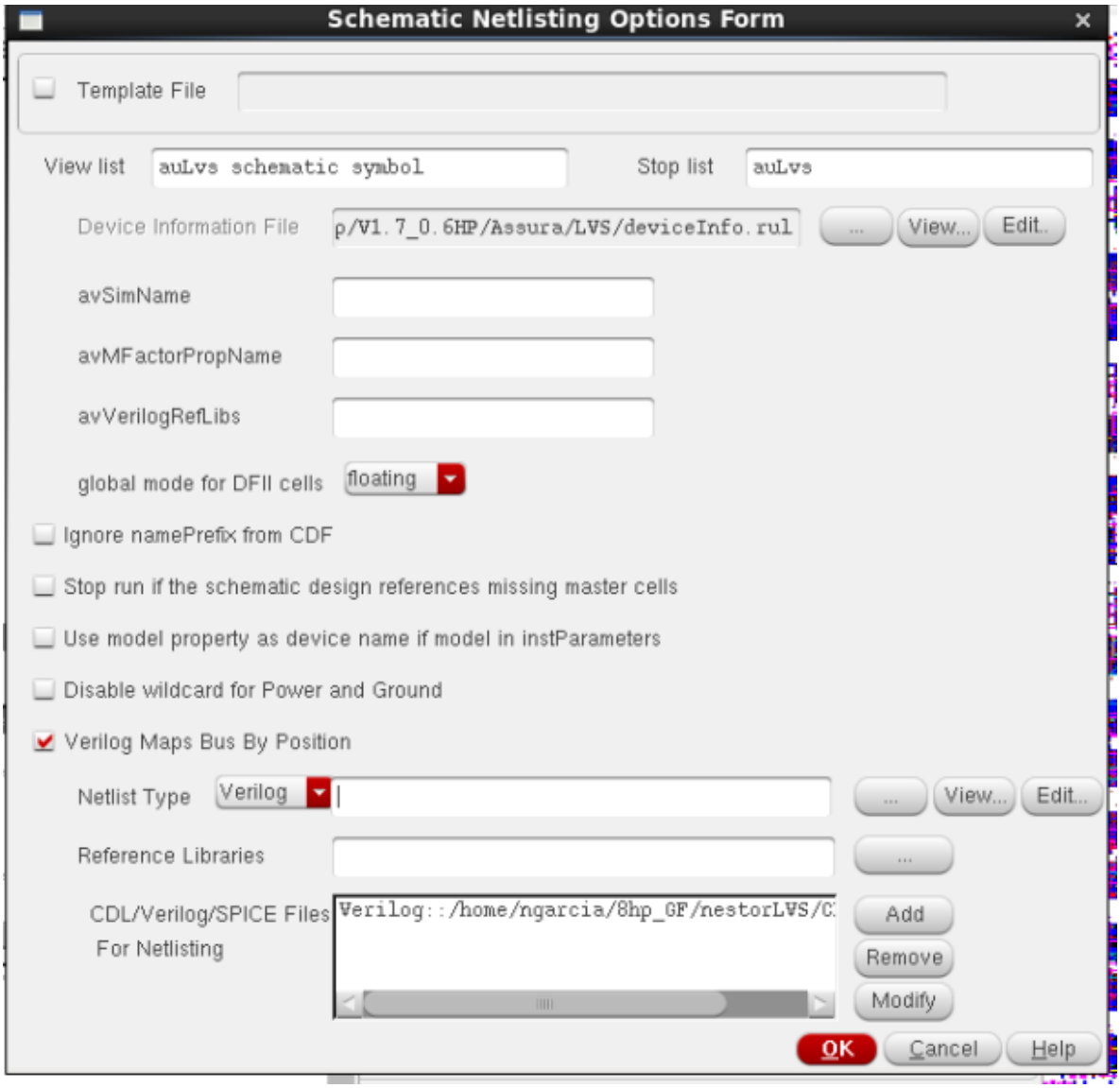


Figura 118: Ventana de Netlisting options

Se elige “Verilog” en la opción “Netlist type”, después se presiona el botón con 3 puntos suspensivos para buscar el archivo netlist; y se presiona el botón “Add”. Se debe elegir el netlist exportado de Encounter. Se da click en el botón OK.

Posteriormente, se regresará a la ventana “Run Assura LVS” en donde se debe seleccionar la opción “Use Verilog Top Cell” como en la Figura 119. Y se presiona el botón “Apply”.

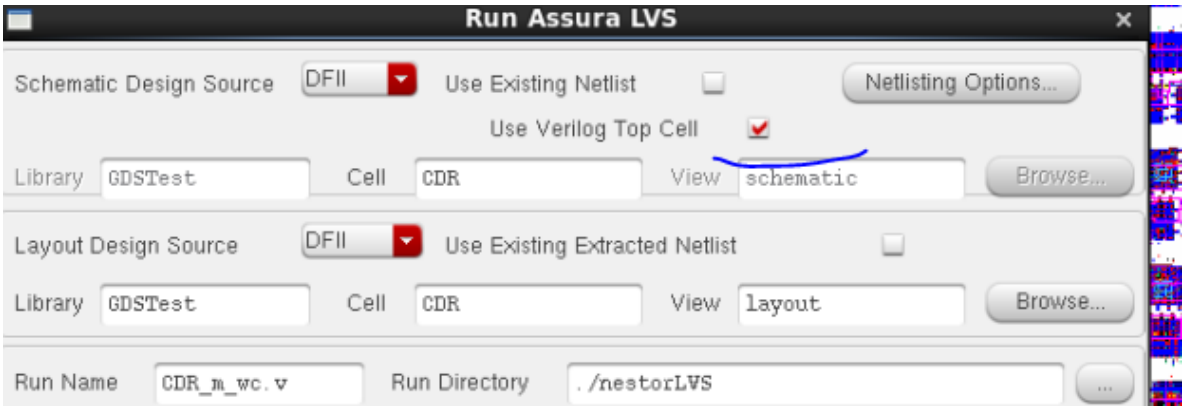


Figura 119: Siguientes pasos para LVS

Después de presionar "Apply", se espera a que concluya el análisis LVS y se obtiene un reporte como el mostrado en la Figura 120.

Figura 120: Reporte LVS

Si se da click en "ok" aparece otra ventana (Figura 121) que permite hacer debugging a los errores.

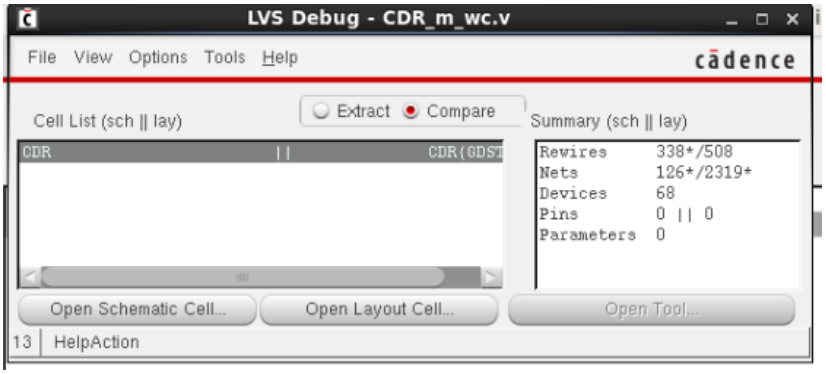


Figura 121: Ventana para hacer debug al reporte LVS

3.6. Conclusiones

Para este circuito CDR se pudo cumplir con la metodología de diseño de un sistema en chip. Se describieron los módulos que lo conforman así como los resultados de la síntesis lógica así como la síntesís física realizada con herramientas CAD empleadas en la industria.

La verificación del diseño se realizó con diferentes herramientas entre las que estuvieron el software Modelsim, para el código verilog antes de la síntesis lógica; mientras que los netlist obtenidos para diferentes esquinas (con transistores lentos y transistores típicos) se verificaron con simulaciones en Simvision y usando los mismos archivos testbech.

Para la etapa de verificación, hubo problemas al simular el jitter, pues la primera implementación de éste consistía en extender el tiempo de los bits con un valor aleatorio cuyo rango estaba entre 0, 5%, 10% y 15% del periodo de la señal del dato. Sin embargo, esto provocaba un jitter acumulado que dificultaba observar el efecto de la correción del dato. En la implementación final, el jitter implementado consistió en extender el tiempo de bit sólo de los 1 lógicos del dato en ambos bordes, tanto al inicio del bit, como al final de éste. Además, se usó un tren de pulsos que consistió en unos y ceros consecutivos, de tal forma que hubiera muchas transiciones y se pudiera estresar al sistema CDR.

Como se describió en los capítulos anteriores, el jitter incluído provoca que haya distintas transiciones en la fase elegida para recuperar el dato y se pudo observar que el sistema se adapta al jitter recuperando la información. Sin embargo, no se pudo agregar al CDR una etapa para cuantificar los errores en la recuperación de los datos, por lo que queda para trabajo a futuro.

Por otro lado, para el desarrollo del sistema CDR se usó un diseño modular que permitió optimizar algunos de los módulos independientemente como los comparadores, los contadores y el detector de bordes.

Una técnica que resultó efectiva para mejorar el rendimiento de algunos módulos como los comparadores, fue el dividir la lógica combinacional e incluirle flip flops de tal forma que se le permita al sistema procesar la información con varios ciclos de reloj del sistema (800MHz). No obstante, a pesar de estos arreglos, se pueden mejorar más al módulo que determina la ventana de análisis de 16 datos pues es el que presenta el peor path de timing. Además, se puede cambiar la ventana de análisis a 8, 32 ó 64 datos y ver como cambian los resultados.

También se observó que los constraints pueden cambiar el análisis estático de tiempo, y dar diferentes resultados de timing. Al especificar restricciones a la mayor cantidad de módulos del sistema, se pueden obtener resultados más realistas del desempeño final que tendría el sistema ya fabricado. Se hizo una especificación extensiva de estos la cual se presenta en el "Apéndice O".

Por otro lado, para la síntesis física, se observó que el área del floorplan elegida para el CDR, así como el número de stripes verticales y su separación de estos al borde del core, afecta en los resultados de timing de setup y hold. Esto hace que cambie el routing y placement de las celdas estándar. Ante los problemas de timing que se observaron en la síntesis físca, los comandos implementados para optimizar al diseño sí mostraron ser útiles, pues redujeron el tiempo tanto de setup como de hold, no al punto de tener slack positivo, pero mejoró sustancialmente.

Para la síntesis física no se agregaron más módulos al floorplan, asì que el siguiente paso sería agregar el módulo generador de datos pseudo aleatorios al CDR, pues al tener cercanos a ambos módulos, la degradación en la propagación de la señal de dato será menor. Además, de así se podrá probar la operación del CDR con un dato más realista.

También se deberá probar al CDR con un dato de entrada con frecuencias diferentes a 200MHz, y observar si éste puede recuperar los datos sin errores.

En esta implementación no se agregaron pines de entrada y salida, pues la mayoría de los puertos de este módulo serán internos en el sistema en chip integrado.

En cuanto a la verificación de la síntesis física, la exportación del diseño presentó un reto que retrasó el flujo de diseño pues al haber trabajado con una nueva tecnología, todos los archivos necesarios, no estaban listos. Aunado a esto, cuando se logró exportar el diseño a Virtuoso, no se

pudieron resolver todos los problemas de DRC obtenidos ante lo cual se debe revisar aún qué Switches de análisis son los más adecuados para las celdas estándar.

Este documento pretende ser una guía que facilite la escritura del documento de la tesis o del ensayo. Puedo ser utilizado para trabajos de nivel licenciatura o de nivel posgrado. El documento es revisado con cierta frecuencia y se procura mejorarlo siempre que sea posible. Si el lector encuentra alguna inconsistencia o error, o bien encuentra alguna forma de mejorar la implementación de alguna funcionalidad del documento, mucho le agradeceré su retroalimentación (Néstor García Hernández es717249@iteso.mx).

4. Apéndices

A. Código en verilog: Script para síntesis lógica

```
#*********************************************************************************************
# Name: CDR.tcl
# Description: This scripts creates the logical synthesis.
#   It can so logical synthesis for the worst case (slow transistors) or
#   typical case (typical timing transistors). Please uncomment the lines with the
#   text : "uncomment for Worst case"
#
# Version: 1.0
# Autor: Nétor Damián García
# Fecha: Primavera 2018
#*********************************************************************************************/

#### 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 CDR
set SYN_EFF medium

###set MAP_EFF medium

#set name_folders "CDR_adaptive_wc_v108_v140_t125_5LM" ;   #uncomment for Worst case
set name_folders "CDR_adaptive_typ_v108_v140_t125_5LM" ;
set MAP_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set _OUTPUTS_PATH "outputs_${name_folders}${DATE}"
set _REPORTS_PATH "reports_${name_folders}${DATE}"
set _LOG_PATH "logs_${name_folders}${DATE}"
##set ET_WORKDIR <ET work directory>

################# BiCMOS Libraries  ##################

# ********* Typical timing libs ************

set timing_library {typ_v150_t025/PnomV1p50T025_STD_CELL_8HP_12T.lib
/opt/libs/IBM_PDK/bicmos8hp/v.20160727/synopsys/typ_v150_v150_t25/IBM_CMOS8HP_BASE_WB_IO_TYP_V150_V150
_T25.lib}

# ********* Worst case timing libs ************

#uncomment for Worst case
#set timing_library {slow_v108_t125/PwcV1p08T125_STD_CELL_8HP_12T.lib
/opt/libs/IBM_PDK/bicmos8hp/v.20160727/synopsys/slow_v108_v140_t125/IBM_CMOS8HP_BASE_WB_IO_SLOW_V108_
V140_T125.lib}
```

```
#from folder v.20171220
set my_lef_library {/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/bicmos8hp_5AM_21_tech.lef
/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/BICMOS8HP_SC_1P2V_12T_RVT.lef
/opt/libs/IBM_PDK/bicmos8hp/v.20160727/lef/CMOS8HP_BASE_WB_IO_5LM.lef}

########## BiCMOS STD Cell Libraries path ####################

#from folder v.20171220 (usar este)
set_attribute lib_search_path {/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/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 ultra_global_mapping true /
##set_attribute iopt_ultra_optimization true /
##set_attribute wireload_mode <value> /
set_attribute information_level 7 /
################ Command to respect hierarchies ##############
set_attribute auto_ungroup none /
################ Command to ignore hierarchies ##############
#set_attribute auto_ungroup both /
#Avoid using scan flip flops
set_attr use_scan_seqs_for_non_dft false /
#####################################################################
## Library setup
#####################################################################
set_attribute library $timing_library
## PLE
set_attribute lef_library $my_lef_library /
## 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 -v2001 { ../CDR.v ../Phase_selector.v ../mux2to1.v ../detect_edge.v ../not_module.v ../Flipflop_data.v
../CounterEvents.v ../Allow_updateSignalSelector.v ../Mux8_to_1.v ../Flipflop_D.v ../CounterwFlag_P.v ../One_Shot.v
../Comparador_maj.v ../Xor_2ndStage2.v ../xor_3.v  ../clockGenerator_FFneg.v ../Top_comp.v ../Comp_Block.v} ;
elaborate $DESIGN
puts "Runtime & Memory after 'read_hdl'"
timestat Elaboration
check_design -unresolved
#####################################################################
## Constraints Setup
#####################################################################
read_sdc {../CDR_newConstraint.sdc}
puts "The number of exceptions is [llength [find /designs/$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}"
}

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

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

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

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 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 power > $_REPORTS_PATH/${DESIGN}_power.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}/.

report timing -lint -verbose
#report timing -paths [specify_paths -from clk_p0]
#report timing -lint -verbose

##quit
```

B.Código en Verilog: Módulo top de CDR

```
/*********************************************************************************
Name: CDR.v
Description: This module

Parameters:

*   BIT_COUNT: Number of bits for the variables which count edges detections
*   Bits_ph:

Inputs:
*   original_data:
*   sampling_clk0:  signal 0º for sampling
*   sampling_clk1:  signal 45º for sampling
*   sampling_clk2:  signal 90º for sampling
*   sampling_clk3:  signal 135º for sampling
*   sampling_clk4:  signal 180º for sampling
*   sampling_clk5:  signal 225º for sampling
*   sampling_clk6:  signal 270º for sampling
*   sampling_clk7:  signal 315º for sampling
*   reset:              reset signal
*   enable_cdr:     enable signal

Outputs:
*   [Bits_ph-1:0] best_sampling_clk_selector:   keep the best sampling clock selector from updater decision
*   clk_recovered:      clock recovered, which comes from the clock generator module
*   recovered_data:     result of sampling with the best selected clock

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
*********************************************************************************/

module CDR
```

```
#(
   parameter BIT_COUNT=4,
   parameter Bits_ph=3

)
(
   input   original_data,
   input   sampling_clk0,
   input   sampling_clk1,
   input   sampling_clk2,
   input   sampling_clk3,
   input   sampling_clk4,
   input   sampling_clk5,
   input   sampling_clk6,
   input   sampling_clk7,
   input   reset,
   input   enable_cdr,
   output [Bits_ph-1:0] best_sampling_clk_selector,
   output clk_recovered,
   output recovered_data );

wire [Bits_ph-1:0] selector_final_tmp;
wire [Bits_ph-1:0] selector_to_FF;
wire best_sampling_clk;  /* This signal is the selected phase which passes through the Phase multiplexer  */
wire dataforSampling;

//wire [BIT_COUNT-1:0]res_max_counting;

wire [BIT_COUNT-1:0] counter8_1;
wire [BIT_COUNT-1:0] counter1_2;
wire [BIT_COUNT-1:0] counter2_3;
wire [BIT_COUNT-1:0] counter3_4;
wire [BIT_COUNT-1:0] counter4_5;
wire [BIT_COUNT-1:0] counter5_6;
wire [BIT_COUNT-1:0] counter6_7;
wire [BIT_COUNT-1:0] counter7_8;

wire [BIT_COUNT-1:0] counter8_1_tmp;
wire [BIT_COUNT-1:0] counter1_2_tmp;
wire [BIT_COUNT-1:0] counter2_3_tmp;
wire [BIT_COUNT-1:0] counter3_4_tmp;
wire [BIT_COUNT-1:0] counter4_5_tmp;
wire [BIT_COUNT-1:0] counter5_6_tmp;
wire [BIT_COUNT-1:0] counter6_7_tmp;
wire [BIT_COUNT-1:0] counter7_8_tmp;

wire captureSelectedClk;

wire [Bits_ph-1:0] new_selection; /* Signal used to propagate the new proposed phase selection */
wire [Bits_ph-1:0] previous_selected_phase; /*Wire to pass the previous selected phase to the adaptive module*/
wire flag_to_resetcounting; /* Pulse signal from Edge Detector module, used to clear the counters and store the decision from
phase updater module */
wire pulsetoSaveprev; /* Clock signal used to store the decision from Phase-updater module to save-pevious-phase FF */

not_module
#(
   .NBIT(1'B1)
)NOT_M1
(
   .in(sampling_clk4),
```

```
    .out(negp4)
);

not_module
#(
    .NBIT(1'B1)
)NOT_pulsetoSaveprev
(
    .in(flag_to_resetcounting),
    .out(pulsetoSaveprev)
);

detect_edge Edge_detector
(
    .reset(reset),
    .data(original_data),
    .clk_p0(sampling_clk0),
    .clk_p45(sampling_clk1),
    .clk_p90(sampling_clk2),
    .clk_p135(sampling_clk3),
    .clk_p180(sampling_clk4),
    .clk_p225(sampling_clk5),
    .clk_p270(sampling_clk6),
    .clk_p315(sampling_clk7),
    .flag_to_startcounting_decision(captureSelectedClk),
    .flag_to_resetcounting(flag_to_resetcounting),
    .counter8_1(counter8_1_tmp),
    .counter1_2(counter1_2_tmp),
    .counter2_3(counter2_3_tmp),
    .counter3_4(counter3_4_tmp),
    .counter4_5(counter4_5_tmp),
    .counter5_6(counter5_6_tmp),
    .counter6_7(counter6_7_tmp),
    .counter7_8(counter7_8_tmp) );



//############################### FF entre edge_detector ############################

/* @brief: Flip flop stage to pass the data from the Edge Detector to the Phase Selector module */

Flipflop_data
#(
    .Nbit(BIT_COUNT)
)FF_edgePhaseSel81
(
    .input_data(counter8_1_tmp),
    .clk(negp4),
    .rst(reset),
    .output_data(counter8_1)
);

Flipflop_data
#(
    .Nbit(BIT_COUNT)
)FF_edgePhaseSel12
(
    .input_data(counter1_2_tmp),
    .clk(negp4),
    .rst(reset),
```

```
    .output_data(counter1_2)
);
Flipflop_data
#(
    .Nbit(BIT_COUNT)
)FF_edgePhaseSel23
(
    .input_data(counter2_3_tmp),
    .clk(negp4),
    .rst(reset),
    .output_data(counter2_3)
);

Flipflop_data
#(
    .Nbit(BIT_COUNT)
)FF_edgePhaseSel34
(
    .input_data(counter3_4_tmp),
    .clk(negp4),
    .rst(reset),
    .output_data(counter3_4)
);
Flipflop_data
#(
    .Nbit(BIT_COUNT)
)FF_edgePhaseSel45
(
    .input_data(counter4_5_tmp),
    .clk(negp4),
    .rst(reset),
    .output_data(counter4_5)
);

Flipflop_data
#(
    .Nbit(BIT_COUNT)
)FF_edgePhaseSel56
(
    .input_data(counter5_6_tmp),
    .clk(negp4),
    .rst(reset),
    .output_data(counter5_6)
);
Flipflop_data
#(
    .Nbit(BIT_COUNT)
)FF_edgePhaseSel67
(
    .input_data(counter6_7_tmp),
    .clk(negp4),
    .rst(reset),
    .output_data(counter6_7)
);

Flipflop_data
#(
    .Nbit(BIT_COUNT)
)FF_edgePhaseSel78
(
```

```
    .input_data(counter7_8_tmp),
    .clk(negp4),
    .rst(reset),
    .output_data(counter7_8)
);

//############################## FF stage ends ##############################


/* @brief: Module which selects the best sampling clock taking into account the edge detections */

Phase_selector
#(
    .BIT_COUNT(BIT_COUNT),
    .Bits_ph(Bits_ph)
)phase_select_mod
(
    .reset(reset),
    .enable(enable_cdr),
    .clk(negp4),
    .counter8_1(counter8_1),
    .counter1_2(counter1_2),
    .counter2_3(counter2_3),
    .counter3_4(counter3_4),
    .counter4_5(counter4_5),
    .counter5_6(counter5_6),
    .counter6_7(counter6_7),
    .counter7_8(counter7_8),
    .selector_final(selector_final_tmp)
);

/* @brief: Module to change dynamically the phase selection depending on the jitter*/

Allow_updateSignalSelector
#(
  .Bits_ph(Bits_ph)
)allow_updateCount
(
    .clk(negp4),
    .reset(reset),
    .previous_decision(previous_selected_phase) ,
    .selector_final_tmp(new_selection),
    .selector_final(best_sampling_clk_selector)   ///Selector final
);

/* @brief: Flip flop to store the previous selected phase */

Flipflop_data
#(
    .Nbit(Bits_ph)
)save_previousSel
(
    .input_data(best_sampling_clk_selector),
    .clk(pulsetoSaveprev),
    .rst(reset),
    .output_data(previous_selected_phase)
);

/* @brief: Flip flop to store the new proposed phase selection*/
```

```
Flipflop_data
#(
   .Nbit(Bits_ph)
)save_Newselection
(
   .input_data(selector_final_tmp),
   .clk(captureSelectedClk),
   .rst(reset),
   .output_data(new_selection)
);

/* @brief :Multiplexer which allows to select one clock phase to sample the data */

Mux8_to_1 mux_phase
(
   // Input Ports
   .Selector(best_sampling_clk_selector ),
   .Data_0(sampling_clk0),
   .Data_1(sampling_clk1),
   .Data_2(sampling_clk2),
   .Data_3(sampling_clk3),
   .Data_4(sampling_clk4),
   .Data_5(sampling_clk5),
   .Data_6(sampling_clk6),
   .Data_7(sampling_clk7),

   // Output Ports
   .Mux_Output(best_sampling_clk)
);


wire clk_recovered_tmp;  /* Signal to pass the divided selected phase, so it can be centered while sampling the input data*/

/* @brief :Module which divides the selected phase, so it can be centered for the sampling */

clockGenerator_FFneg clkgenT
(
   .clk_selected(best_sampling_clk),
   .reset(reset),
   .clk_recovered(clk_recovered_tmp)
);

/* @brief : Not gate used to invert the generated clock from selected phase. This is needed to have the rising edge more centered
in the input data */

not_module
#(
   .NBIT(1'B1)
)NOT_samplclk
(
   .in(clk_recovered_tmp),
   .out(clk_recovered)
);

/* @brief : Flip flop which samples the input data with the selected phase*/

Flipflop_data
#(
   .Nbit(1'b1)
)FF_recovered
```

```
(
   .input_data(original_data),
   .clk(clk_recovered),
   .rst(reset) ,
   .output_data(recovered_data)
);


/*Log Function*/
function integer CeilLog2;
    input integer data;
    integer i,result;
    begin
       for(i=0; 2**i < data; i=i+1)
          result = i + 1;
       CeilLog2 = result; //se debe usar el nombre de la funcion, que será la salida
    end
endfunction

endmodule
```

C. Código en Verilog: Módulo detector de bordes

```
/////////////////////////////////////////////////////////////////////////////////
Name: detect_edge.v
Description: This module detects data transitions (rising edges or falling edges).
There are 8 sampling signals which are separated 45º from each other.
The samples obtained by every clock signal are compared with a Xor stage to produce
pulses, which indicate where the transition/edge of the signal was detected first.
Then every transition pulse is counted in a counting stage.
The counting results will be passed to the Phase selector module.

Parameters

*       NUM_COUNT :    Maximum expected number of detected edges in a analysis window (16 data)
*       BIT4NUM_COUNT :        Number of needed bits for the counters to store the expected detected edges
*       NUMBER_COUNTING_4DECISION :          Number of 800MHz clock cycles which will form the analysis
window (for 16 data + 3 more clk cycles)
*       BITS_COUNTING_CYCLES_4DECISION : Number of bits which the 16 data counter will need

Inputs:

*       reset :    reset signal
*       data :             original data, either from exterior or from pseudorandom data generator
*       enable :  enable signal
*       clk_p0 :  sampling clock 0º
*       clk_p45 :          sampling clock 45º
*       clk_p90 :          sampling clock 90º
*       clk_p135 :         sampling clock 135º
*       clk_p180 :         sampling clock 180º
*       clk_p225 :         sampling clock 225º
*       clk_p270 :         sampling clock 270º
*       clk_p315 :         sampling clock 315º

Outputs:
*       flag_to_startcounting_decision :        Signals which
```

```
*           flag_to_resetcounting :                 Signal which resets the counters
*           [BIT4NUM_COUNT-1:0] counter8_1 : Counter for the 8 to 1 edge detection
*           [BIT4NUM_COUNT-1:0] counter1_2 : Counter for the 1 to 2 edge detection
*           [BIT4NUM_COUNT-1:0] counter2_3 : Counter for the 2 to 3 edge detection
*           [BIT4NUM_COUNT-1:0] counter3_4 : Counter for the 3 to 4 edge detection
*           [BIT4NUM_COUNT-1:0] counter4_5 : Counter for the 4 to 5 edge detection
*           [BIT4NUM_COUNT-1:0] counter5_6 : Counter for the 5 to 6 edge detection
*           [BIT4NUM_COUNT-1:0] counter6_7 : Counter for the 6 to 7 edge detection
*           [BIT4NUM_COUNT-1:0] counter7_8 : Counter for the 7 to 8 edge detection

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************************************/
module detect_edge
#(

   parameter NUM_COUNT = 16, /*for the counters*/
   parameter BIT4NUM_COUNT=CeilLog2(NUM_COUNT),
   parameter NUMBER_COUNTING_4DECISION=67, //sample window 64/4=16
   parameter BITS_COUNTING_CYCLES_4DECISION=CeilLog2(NUMBER_COUNTING_4DECISION)
)
(
            input reset,
            input data,
            input clk_p0,
            input clk_p45,
            input clk_p90,
            input clk_p135,
            input clk_p180,
            input clk_p225,
            input clk_p270,
            input clk_p315,
            output flag_to_startcounting_decision,
            output flag_to_resetcounting,
            output [BIT4NUM_COUNT-1:0] counter8_1,
            output [BIT4NUM_COUNT-1:0] counter1_2,
            output [BIT4NUM_COUNT-1:0] counter2_3,
            output [BIT4NUM_COUNT-1:0] counter3_4,
            output [BIT4NUM_COUNT-1:0] counter4_5,
            output [BIT4NUM_COUNT-1:0] counter5_6,
            output [BIT4NUM_COUNT-1:0] counter6_7,
            output [BIT4NUM_COUNT-1:0] counter7_8
);


/**************** Signals for the first data sampling stage, which will be taken as the most recent sample  *********/
wire New_samplePh0;
wire New_samplePh45;
wire New_samplePh90;
wire New_samplePh135;
wire New_samplePh180;
wire New_samplePh225;
wire New_samplePh270;
wire New_samplePh315;
/**************** Signals for the second data sampling stage, which will be taken as the stored previous sample *********/

wire Prev_sample_Ph0;
wire Prev_sample_Ph45;
wire Prev_sample_Ph90;
```

```
wire Prev_sample_Ph135;
wire Prev_sample_Ph180;
wire Prev_sample_Ph225;
wire Prev_sample_Ph270;
wire Prev_sample_Ph315;

wire neg_clk_p4; /* Clock from Phase 4 negated */
wire neg_clk_p5; /* Clock from Phase 5 negated */

/* Signals which pass the first stage of Xor operations */
wire Xor_8_1;
wire Xor_1_2;
wire Xor_2_3;
wire Xor_3_4;
wire Xor_4_5;
wire Xor_5_6;
wire Xor_6_7;
wire Xor_7_8;
wire or_reset; /* signal to reset the edge counters */
wire [BITS_COUNTING_CYCLES_4DECISION-1:0] counter_4decision;/* signal which stores the counting of the analysis
window of 16 data*/

wire pulse_2ndStage;                    /* Signal to pass the clock to the second stage of flip flops */
/* The second stage will capture the data with the rising edge of the last phase (315º).
The only exception will be the last flip flop, which will use the "neg_clk_p4" clock */

/**************** NOT GATES to generate capture clocks ****************/

not_module
#(
   .NBIT(1'B1)
)NOT_M1
(
   .in(clk_p225),
   .out(neg_clk_p5)
);

not_module
#(
   .NBIT(1'B1)
)NOT_M2
(
   .in(clk_p180),
   .out(neg_clk_p4)
);
/************* First stage of sampling Flip flops  ********************/

Flipflop_data
#(
   .Nbit(1'b1)
)FF_p0
(
   .input_data(data),
   .clk(clk_p0),
   .rst(reset),
   .output_data(New_samplePh0)
);

Flipflop_data
#(
```

```
    .Nbit(1'b1)
)FF_p45
(
    .input_data(data),
    .clk(clk_p45),
    .rst(reset),
    .output_data(New_samplePh45)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_p90
(
    .input_data(data),
    .clk(clk_p90),
    .rst(reset),
    .output_data(New_samplePh90)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_p135
(
    .input_data(data),
    .clk(clk_p135),
    .rst(reset),
    .output_data(New_samplePh135)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_p180
(
    .input_data(data),
    .clk(clk_p180),
    .rst(reset),
    .output_data(New_samplePh180)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_p225
(
    .input_data(data),
    .clk(clk_p225),
    .rst(reset),
    .output_data(New_samplePh225)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_p270
(
    .input_data(data),
    .clk(clk_p270),
```

```
    .rst(reset),
    .output_data(New_samplePh270)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_p315
(
    .input_data(data),
    .clk(clk_p315),
    .rst(reset),
    .output_data(New_samplePh315)
);
/**########################## First stage ends ######################*/

/************* Second stage of sampling Flip flops(previous sample) ******************/
assign pulse_2ndStage = clk_p315;

Flipflop_data
#(
    .Nbit(1'b1)
)FF_0_2nd
(
    .input_data(New_samplePh0),
    .clk(pulse_2ndStage),
    .rst(reset),
    .output_data(Prev_sample_Ph0)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_45_2nd
(
    .input_data(New_samplePh45),
    .clk(pulse_2ndStage),
    .rst(reset),
    .output_data(Prev_sample_Ph45)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_90_2nd
(
    .input_data(New_samplePh90),
    .clk(pulse_2ndStage),
    .rst(reset),
    .output_data(Prev_sample_Ph90)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_135_2nd
(
    .input_data(New_samplePh135),
    .clk(pulse_2ndStage),
    .rst(reset),
```

```
    .output_data(Prev_sample_Ph135)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_180_2nd
(
    .input_data(New_samplePh180),
    .clk(pulse_2ndStage),
    .rst(reset),
    .output_data(Prev_sample_Ph180)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_225_2nd
(
    .input_data(New_samplePh225),
    .clk(pulse_2ndStage),
    .rst(reset),
    .output_data(Prev_sample_Ph225)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_270_2nd
(
    .input_data(New_samplePh270),
    .clk(pulse_2ndStage),
    .rst(reset),
    .output_data(Prev_sample_Ph270)
);

Flipflop_data
#(
    .Nbit(1'b1)
)FF_315_2nd
(
    .input_data(New_samplePh315),
    .clk(neg_clk_p4),
    .rst(reset),
    .output_data(Prev_sample_Ph315)  /* Negative slack */
);

/************* Second sampling stage ends  **************/

wire previous_Xor8;

Flipflop_data
#(
    .Nbit(1'b1)
)save_previous_Xor8
(
    .input_data(Prev_sample_Ph315),
    .clk(neg_clk_p4),
    .rst(reset),
    .output_data(previous_Xor8)
```

```
);

//################################## Etapa FF entre módulos de Xor ##################################

/* @brief: This module performs the first stage of Xor operations in a synchronous way  */

Xor_2ndStage2 xor_2(
        .reset(reset),
        .clk(neg_clk_p5),
        .X8ant(previous_Xor8),
        .X1(Prev_sample_Ph0),
        .X2(Prev_sample_Ph45),
        .X3(Prev_sample_Ph90),
        .X4(Prev_sample_Ph135),
        .X5(Prev_sample_Ph180),
        .X6(Prev_sample_Ph225),
        .X7(Prev_sample_Ph270),
        .X8(Prev_sample_Ph315),
        .Comp_X8ant_1(Xor_8_1),
        .Comp_X1_2(Xor_1_2),
        .Comp_X2_3(Xor_2_3),
        .Comp_X3_4(Xor_3_4),
        .Comp_X4_5(Xor_4_5),
        .Comp_X5_6(Xor_5_6),
        .Comp_X6_7(Xor_6_7),
        .Comp_X7_8(Xor_7_8) );


CounterwFlag_P cont_to16data
(
        .clk(neg_clk_p4),
        .reset(reset),
        .flag(flag_to_startcounting_decision)
);

One_Shot onesht
(
        .clk(neg_clk_p4),
        .reset(reset) ,
        .Start(flag_to_startcounting_decision),
        .Shot(flag_to_resetcounting)
);
assign or_reset = reset & flag_to_resetcounting;  /*Signal needed to clear the counters since the beginning*/

CounterEvents counter_8_1
(
        .eventtoC(Xor_8_1),
        .clk(clk_p180),
        .reset(or_reset),
        .counter(counter8_1)
);

CounterEvents counter_1_2
(
        .eventtoC(Xor_1_2),
        .clk(clk_p180),
        .reset(or_reset),
        .counter(counter1_2)
);
```

```
CounterEvents counter_2_3
(
        .eventtoC(Xor_2_3),
        .clk(clk_p180),
        .reset(or_reset),
        .counter(counter2_3)
);
CounterEvents counter_3_4
(
        //.eventtoC(xor3_4_w_4_5),
        .eventtoC(Xor_3_4),
//      .clk(neg_clk_p4),
        .clk(clk_p180),
        .reset(or_reset),
        .counter(counter3_4)
);
CounterEvents counter_4_5
(
        .eventtoC(Xor_4_5),
        .clk(clk_p180),
        .reset(or_reset),
        .counter(counter4_5)
);
CounterEvents counter_5_6
(
        .eventtoC(Xor_5_6),
        .clk(clk_p180),
        .reset(or_reset),
        .counter(counter5_6)
);
CounterEvents counter_6_7
(
        .eventtoC(Xor_6_7),
        .clk(clk_p180),
        .reset(or_reset),
        .counter(counter6_7)
);
CounterEvents counter_7_8
(
        .eventtoC(Xor_7_8),
        .clk(clk_p180),
        .reset(or_reset),
        .counter(counter7_8)
);
/*****************************************************************************************/
 /*Log Function*/
    function integer CeilLog2;
      input integer data;
      integer i,result;
      begin
        for(i=0; 2**i < data; i=i+1)
          result = i + 1;
        CeilLog2 = result;
      end
   endfunction
/*****************************************************************************************
endmodule
```

D. Código en verilog: módulo selector de reloj

```
/**********************************************************************************
Name: Phase_selector.v
Description: This module receives edge detection countings and decides which is the
phase which had more edge detections.

Inputs:
*   reset:  reset signal
*   enable: enable signal
*   clk:    clock signal
*   [BIT_COUNT-1:0] counter8_1: Edge detection counting
*   [BIT_COUNT-1:0] counter1_2: Edge detection counting
*   [BIT_COUNT-1:0] counter2_3: Edge detection counting
*   [BIT_COUNT-1:0] counter3_4: Edge detection counting
*   [BIT_COUNT-1:0] counter4_5: Edge detection counting
*   [BIT_COUNT-1:0] counter5_6: Edge detection counting
*   [BIT_COUNT-1:0] counter6_7: Edge detection counting
*   [BIT_COUNT-1:0] counter7_8: Edge detection counting

Outputs:
*           [Bits_ph-1:0] selector_final: selector of the phase with more edge detections

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************************/

module Phase_selector
#(
   parameter BIT_COUNT = 5,
   parameter Bits_ph=3,
            parameter val_m1_0_reg = 0,
            parameter val_m1_1_reg = 1,
            parameter val_m2_0_reg = 2,
            parameter val_m2_1_reg = 3,
            parameter val_m3_0_reg = 4,
            parameter val_m3_1_reg = 5,
            parameter val_m4_0_reg = 6,
            parameter val_m4_1_reg = 7
)
(
            input reset,
            input clk,
            input [BIT_COUNT-1:0] counter8_1,
            input [BIT_COUNT-1:0] counter1_2,
            input [BIT_COUNT-1:0] counter2_3,
            input [BIT_COUNT-1:0] counter3_4,
            input [BIT_COUNT-1:0] counter4_5,
            input [BIT_COUNT-1:0] counter5_6,
            input [BIT_COUNT-1:0] counter6_7,
            input [BIT_COUNT-1:0] counter7_8,
            output [Bits_ph-1:0] selector_final,
            output [BIT_COUNT-1:0]res_max_counting
);

//wire [BIT_COUNT-1:0]res_max_counting;
wire [Bits_ph-1:0] selector_final_tmp;
wire [BIT_COUNT-1:0]res_max_counting_tmp;
```

```
wire [Bits_ph-1:0] s1_m;
wire [Bits_ph-1:0] s2_m;
wire [Bits_ph-1:0] s3_m;
wire [Bits_ph-1:0] s4_m;
wire [Bits_ph-1:0] s5_m;
wire [Bits_ph-1:0] s6_m;

wire [BIT_COUNT-1:0]res_mux1;
wire [BIT_COUNT-1:0]res_mux2;
wire [BIT_COUNT-1:0]res_mux3;
wire [BIT_COUNT-1:0]res_mux4;
wire [BIT_COUNT-1:0]res_mux5;
wire [BIT_COUNT-1:0]res_mux6;

wire [Bits_ph-1:0] s3_m_tmp;
wire [BIT_COUNT-1:0]res_mux3_tmp;
wire [Bits_ph-1:0] s4_m_tmp;
wire [BIT_COUNT-1:0]res_mux4_tmp;

wire [Bits_ph-1:0] s1_m_tmp;
wire [BIT_COUNT-1:0]res_mux1_tmp;
wire [Bits_ph-1:0] s2_m_tmp;
wire [BIT_COUNT-1:0]res_mux2_tmp;
wire [Bits_ph-1:0] s5_m_tmp;
wire [BIT_COUNT-1:0]res_mux5_tmp;
wire [Bits_ph-1:0] s6_m_tmp;
wire [BIT_COUNT-1:0]res_mux6_tmp;


Top_comp
#(
   .Nbits(BIT_COUNT),
   .Bits_ph(Bits_ph)
)comp1
(
   .reset(reset),
   .clk(clk),
   .signal1(counter8_1),
   .signal2(counter1_2),
   .sx_m1(3'd0),
   .sx_m2(3'd1),
   .res_phase(s1_m),
   .res_max(res_mux1)
);

Top_comp
#(
   .Nbits(BIT_COUNT),
   .Bits_ph(Bits_ph)
)comp2
(
   .reset(reset),
   .clk(clk),
   .signal1(counter2_3),
   .signal2(counter3_4),
   .sx_m1(3'd2),
   .sx_m2(3'd3),
   .res_phase(s2_m),
   .res_max(res_mux2)
```

```
);


Top_comp
#(
   .Nbits(BIT_COUNT),
   .Bits_ph(Bits_ph)
)comp3
(
   .reset(reset),
   .clk(clk),
   .signal1(counter4_5),
   .signal2(counter5_6),
   .sx_m1(3'd4),
   .sx_m2(3'd5),
   .res_phase(s3_m),
   .res_max(res_mux3)
);

Top_comp
#(
   .Nbits(BIT_COUNT),
   .Bits_ph(Bits_ph)
)comp4
(
   .reset(reset),
   .clk(clk),
   .signal1(counter6_7),
   .signal2(counter7_8),
   .sx_m1(3'd6),
   .sx_m2(3'd7),
   .res_phase(s4_m),
   .res_max(res_mux4)
);


Top_comp
#(
   .Nbits(BIT_COUNT),
   .Bits_ph(Bits_ph)
)comp5
(
   .reset(reset),
   .clk(clk),
   .signal1(res_mux1),
   .signal2(res_mux2),
   .sx_m1(s1_m),
   .sx_m2(s2_m),
   .res_phase(s5_m),
   .res_max(res_mux5)
);


Top_comp
#(
   .Nbits(BIT_COUNT),
   .Bits_ph(Bits_ph)
)comp6
(
   .reset(reset),
```

```
    .clk(clk),
    .signal1(res_mux3),
    .signal2(res_mux4),
    .sx_m1(s3_m),
    .sx_m2(s4_m),
    .res_phase(s6_m),
    .res_max(res_mux6)
);

Top_comp
#(
    .Nbits(BIT_COUNT),
    .Bits_ph(Bits_ph)
)comp7
(
    .reset(reset),
    .clk(clk),
    .signal1(res_mux5),
    .signal2(res_mux6),
    .sx_m1(s5_m),
    .sx_m2(s6_m),
    .res_phase(selector_final),
    .res_max(res_max_counting)
);

endmodule
```

E.Código en verilog: Módulo adaptivo a jitter

```
/*********************************************************************************
Name: Allow_updateSignalSelector.v
Description: This module
Inputs:
*           clk :
*           reset :
*           [Bits_ph-1:0] previous_decision :
*           [Bits_ph-1:0] selector_final_tmp :
Outputs:
*           [Bits_ph-1:0] selector_final:
Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
*********************************************************************************/
module Allow_updateSignalSelector
#(
            /* The following parameters are aimed to create constants which will be used
                        to define the phase selector
             */
            parameter Bits_ph=3,
            parameter ph0_cnst = 3'd0,
            parameter ph1_cnst = 3'd1,
            parameter ph2_cnst = 3'd2,
            parameter ph3_cnst = 3'd3,
            parameter ph4_cnst = 3'd4,
            parameter ph5_cnst = 3'd5,
            parameter ph6_cnst = 3'd6,
```

```
        parameter ph7_cnst = 3'd7
)
(
        input clk,
        input reset,
        input [Bits_ph-1:0] previous_decision,
        input [Bits_ph-1:0] selector_final_tmp,
        output [Bits_ph-1:0] selector_final
);
reg [Bits_ph-1:0]selector_final_reg=0;
always@(posedge clk or negedge reset)begin
   if(reset==1'b0)begin
        selector_final_reg = {Bits_ph{1'b0} };
   end else begin
            case(selector_final_tmp)
                3'd0:
                   begin
                      if( previous_decision == ph1_cnst|| previous_decision == ph7_cnst || previous_decision == ph0_cnst )
begin
                         //mantener el anterior
                         selector_final_reg = previous_decision;
                      end else begin
                         //actualizar
                         selector_final_reg = ph0_cnst;
                      end
                   end
                3'd1:
                   begin
                      if( previous_decision == ph2_cnst || previous_decision == ph0_cnst || previous_decision == ph1_cnst)
begin
                         //mantener el anterior
                         selector_final_reg = previous_decision;
                      end else begin
                         //actualizar
                         selector_final_reg = ph1_cnst;
                      end
                   end
                3'd2:
                   begin
                      if( previous_decision == ph3_cnst  || previous_decision == ph1_cnst || previous_decision == ph2_cnst )
begin
                         //mantener el anterior
                         selector_final_reg = previous_decision;
                      end else begin
                         //actualizar
                         selector_final_reg = ph2_cnst;
                      end
                   end
                3'd3:
                   begin
                      if( previous_decision == ph4_cnst || previous_decision == ph5_cnst ||previous_decision == ph1_cnst ||
previous_decision == ph3_cnst ) begin
                         //mantener el anterior
                         selector_final_reg = previous_decision;
                      end else begin
                         //actualizar
                         selector_final_reg = ph3_cnst;
                      end
                   end
                3'd4:
```

```
                        begin
                            if( previous_decision == ph5_cnst  || previous_decision == ph3_cnst|| previous_decision == ph4_cnst)
begin
                                //mantener el anterior
                                selector_final_reg = previous_decision;
                            end else begin
                                //actualizar
                                selector_final_reg = ph4_cnst;
                            end
                        end
                    3'd5:
                        begin
                    if( previous_decision == ph6_cnst || previous_decision == ph4_cnst || previous_decision == ph5_cnst) begin

                                //mantener el anterior
                                selector_final_reg = previous_decision;
                            end else begin
                                //actualizar
                                selector_final_reg = ph5_cnst;
                            end
                        end
                    3'd6:
                        begin
                            if( previous_decision == ph7_cnst || previous_decision == ph5_cnst || previous_decision == ph6_cnst)
begin
                                //mantener el anterior
                                selector_final_reg = previous_decision;
                            end else begin
                                //actualizar
                                selector_final_reg = ph6_cnst;
                            end
                        end
                    3'd7:
                        begin
                                if( previous_decision == ph0_cnst || previous_decision == ph6_cnst|| previous_decision ==
ph7_cnst) begin
                                //mantener el anterior
                                selector_final_reg = previous_decision;
                            end else begin
                                //actualizar
                                selector_final_reg = ph7_cnst;
                            end
                        end
                    default:
                    begin
                        //mantener el anterior
                        selector_final_reg = previous_decision;
                    end
            endcase
    end
end

assign selector_final = selector_final_reg;

endmodule
```

F. Código en verilog: Módulo top para bloque comparador de conteos

```
/**********************************************************************************
Name: Top_comp.v
Description: This is the top module for the comparator block

Parameter:
*   Nbits: Number of bits for the counters (input signals)
*   Bits_ph: Number of bits for the selector signal
Inputs:
*           clk : clock signal
*           reset : reset signal

*   input [Nbits-1:0] signal1: Signal 1 to be compared
*   input [Nbits-1:0] signal2: Signal 2 to be compared
*   input [Bits_ph-1:0]sx_m1:  Signal to be multiplexed.This is the output if signal 1 is bigger than signal 2
*   input [Bits_ph-1:0]sx_m2:  Signal to be multiplexed.This is the output if signal 2 is bigger than signal 2

Outputs:
*   output [Bits_ph-1:0] res_phase:
*   output [Nbits-1:0]res_max:

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************************/

module Top_comp
#(
   parameter Nbits=5,
   parameter Bits_ph=3
)
(
   input reset,
   input clk,

   //Signals for the mux which passes the counting
   input [Nbits-1:0] signal1,
   input [Nbits-1:0] signal2,
   //Signals for the mux which selects the phase
   input [Bits_ph-1:0]sx_m1,
   input [Bits_ph-1:0]sx_m2,
   //Output of mux to select the phase
   output [Bits_ph-1:0] res_phase,
   //Output with the max number
   output [Nbits-1:0]res_max
);

wire C_out;
wire C_outnot;

Comp_Block c1
(
   .a(signal1) ,
   .b(signal2) ,
   .clk(clk) ,
   .reset(reset) ,
   .C_out(C_out)
```

```
);

assign C_outnot = !C_out;
wire [Nbits-1:0]res_maxtmp;
wire [Bits_ph-1:0] res_phasetmp;
wire [Bits_ph-1:0]sx_m1tmp;
wire [Bits_ph-1:0]sx_m2tmp;
wire [Nbits-1:0] signal1tmp;
wire [Nbits-1:0] signal2tmp;
///// flip flops to update data


wire orClk3;
//////////flips flops to generate a pulse 3 times slower
wire clko1,clko2,clko3;

wire ffd1_q;
wire qneg_ffd1;
wire ffd2_q,ffd3_q;
wire qneg_ffd2;

wire and1;
wire clkneg;


Flipflop_D
#(
        .Nbit(1)
)FF_delay1
(
        .input_data(and1),
        .clk(clk),
        .rst(reset),
        .q(ffd1_q)
);

not_module
#(
   .NBIT(1'B1)
)NOT_del1
(
   .in(ffd1_q),
   .out(qneg_ffd1)
);

assign and1 = qneg_ffd1 & qneg_ffd2;

Flipflop_D
#(
        .Nbit(1)
)FF_delay2
(
        .input_data(ffd1_q),
        .clk(clk),
        .rst(reset),
        .q(ffd2_q )
);

not_module
#(
```

```
    .NBIT(1'B1)
)NOT_del2
(
    .in(ffd2_q),
    .out(qneg_ffd2)
);

not_module
#(
    .NBIT(1'B1)
)NOT_del3
(
    .in(clk),
    .out(clkneg)
);

Flipflop_D
#(
            .Nbit(1)
)FF_delay3
(
            .input_data(ffd2_q),
            .clk(clkneg),
            .rst(reset),
            .q(ffd3_q )
);
assign orClk3 = ffd3_q | ffd2_q;

Flipflop_data
#(
    .Nbit(Bits_ph)
)FF_Sxinmux1
(
    .input_data(sx_m1),
    .clk(orClk3),
    .rst(reset),
    .output_data(sx_m1tmp)
);

Flipflop_data
#(
    .Nbit(Bits_ph)
)FF_Sxinmux2
(
    .input_data(sx_m2),
    .clk(orClk3),
    .rst(reset),
    .output_data(sx_m2tmp)
);

Flipflop_data
#(
    .Nbit(Nbits)
)FF_Datainmux1
(
    .input_data(signal1),
    .clk(orClk3),
    .rst(reset),
    .output_data(signal1tmp)
);
```

```
Flipflop_data
#(
   .Nbit(Nbits)
)FF_Datainsmux2
(
   .input_data(signal2),
   .clk(orClk3),
   .rst(reset),
   .output_data(signal2tmp)
);

mux2to1
#(
   .Nbit(Bits_ph),
)sx_m
(
            .mux_sel(C_outnot),
            .data1(sx_m1tmp),
            .data2(sx_m2tmp),
            .Data_out(res_phasetmp)
);


mux2to1
#(
            .Nbit(Nbits)
)res_mux_x
(
            .mux_sel(C_outnot),
            .data1(signal1tmp),
            .data2(signal2tmp),
            .Data_out(res_maxtmp)
);

Flipflop_data
#(
   .Nbit(Nbits)
)FF_Datamux
(
   .input_data(res_maxtmp),
   .clk(orClk3),
   .rst(reset),
   .output_data(res_max)
);

Flipflop_data
#(
   .Nbit(Bits_ph)
)FF_Sxmux
(
   .input_data(res_phasetmp),
   .clk(orClk3),
   .rst(reset),
   .output_data(res_phase)
);

endmodule
```

G. Código Verilog: Comparator block

```
/**********************************************************************
Name: Comp_Block.v
Description: This is comparator block whose combinational logic was divided by flip flops

Parameter:
*   Nbits: Number of bits for the counters (input signals)
*   Bits_ph: Number of bits for the selector signal
Inputs:
*   input [4:0] a: Signal 1 to be compared
*   input [4:0] b: Signal 2 to be compared
*   clk : clock signal
*   reset : reset signal

Outputs:
*   output C_out: This is the signal resulted from the comparison

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************/

module Comp_Block
(
   input [4:0] a,
   input [4:0] b,
   input clk,
   input reset,
   output C_out);

wire n1;
wire n2;
wire n3;
wire n4;
wire n5;
wire n6;
wire n7;
wire n8;
wire [6:0]concat;
wire [6:0]concat_out;

assign n1 = !a[2];
assign n2 = !a[3];
assign n3 = !b[0];
assign n4 = !a[1];
assign n5 = !a[4];

assign n6 = ~((n4)|(b[1]));

assign n7 = ~( (n3)&(a[0]) );
assign n8 = !n6;

assign tmp1 = ~((b[2]) & ( n1) );
assign tmp2 = ~(b[3] | n2 );
assign tmp3 = ~(b[3] & n2);
assign tmp4 = ~(b[2] | n1);
assign tmp5 = ~(n4 & b[1]);
assign tmp6 = ~(n7 & n8) ;
assign concat = {n5,tmp6,tmp5,tmp4,tmp3,tmp2,tmp1 };
```

```
Flipflop_data
#(
   .Nbit(7)
)FF_tmp1
(
   .input_data(concat),
   .clk(clk),
   .rst(reset),
   .output_data(concat_out)
);

/************
Second stage
***********/
wire Dos_w1;
wire Dos_w2;
wire Dos_w3;
wire Dos_w4;
wire Dos_w5;
wire Dos_w6;
wire Dos_w7;
wire Dos_w8;
wire Dos_w9;
assign Dos_w1 = ~(concat_out[0] | concat_out[1] );
assign Dos_w2 = !concat_out[2];
assign Dos_w3 = ~(concat_out[1] |concat_out[3]);
assign Dos_w4 = ~(concat_out[4] & concat_out[5]);
assign Dos_w5 = ~(b[4] | concat_out[6]);
assign Dos_w6 = ~(Dos_w1 | Dos_w2 );
assign Dos_w7 = ~(Dos_w3 & Dos_w4);
assign Dos_w8 = !Dos_w5;
assign Dos_w9 = ~(Dos_w6 & Dos_w7);

/************
Third stage
***********/

wire [3:0]concat2;
wire [3:0] concat2_out;
assign concat2  = { Dos_w8,Dos_w9,n5, b[4] };
Flipflop_data
#(
   .Nbit(4)
)FF_tmp2
(
   .input_data(concat2),
   .clk(clk),
   .rst(reset),
   .output_data(concat2_out)
);

wire Tres_w1;
wire Tres_w2;
wire Tres_w3;
wire Tres_w4;
assign Tres_w1 = ~(concat2_out[0] & concat2_out[1]);
assign Tres_w2 = ~(concat2_out[2] & concat2_out[3]);
assign Tres_w3 = ~(Tres_w1 & Tres_w2);
assign Tres_w4 = ~Tres_w3 ;
```

```
Flipflop_data
#(
   .Nbit(1)
)FF_tmp3
(
   .input_data(Tres_w4),
   .clk(clk),
   .rst(reset),
   .output_data(C_out)
);

endmodule
```

H. CÓDIGO EN VERILOG: DETECTOR DE TRANSICIONES

```
/**********************************************************************************************
Name: Xor_2ndStage2.v
Description: This module performs Xor operations in a synchronous way
Inputs:
*   reset:  reset signal
*   clk:    clock signal
*   X8ant:  signal from saved sample of 315º phase
*   X1:     signal 1 from old sample of flip flop stage
*   X2:     signal 2 from old sample of flip flop stage
*   X3:     signal 3 from old sample of flip flop stage
*   X4:     signal 4 from old sample of flip flop stage
*   X5:     signal 5 from old sample of flip flop stage
*   X6:     signal 6 from old sample of flip flop stage
*   X7:     signal 7 from old sample of flip flop stage
*   X8:     signal 8 from old sample of flip flop stage
Outputs:
*   Comp_X8ant_1:   Result from xor operation between signal 8 and 1
*   Comp_X1_2:  Result from xor operation between signal 1 and 2
*   Comp_X2_3:  Result from xor operation between signal 2 and 3
*   Comp_X3_4:  Result from xor operation between signal 3 and 4
*   Comp_X4_5:  Result from xor operation between signal 4 and 5
*   Comp_X5_6:  Result from xor operation between signal 5 and 6
*   Comp_X6_7:  Result from xor operation between signal 6 and 7
*   Comp_X7_8:  Result from xor operation between signal 7 and 8
Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************************************/
module Xor_2ndStage2
(
   input reset,
   input clk,
   input X8ant,
   input X1,
   input X2,
   input X3,
   input X4,
```

```
    input X5,
    input X6,
    input X7,
    input X8,
    //outputs
    output Comp_X8ant_1,
    output Comp_X1_2,
    output Comp_X2_3,
    output Comp_X3_4,
    output Comp_X4_5,
    output Comp_X5_6,
    output Comp_X6_7,
    output Comp_X7_8);

reg X8ant_1_reg;
reg X1_2_reg;
reg X2_3_reg;
reg X3_4_reg;
reg X4_5_reg;
reg X5_6_reg;
reg X6_7_reg;
reg X7_8_reg;

    always@(posedge clk or negedge reset) begin

        if(reset==1'b0)begin
            X8ant_1_reg = 1'b0;
            X1_2_reg = 1'b0;
            X2_3_reg = 1'b0;
            X3_4_reg = 1'b0;
            X4_5_reg = 1'b0;
            X5_6_reg = 1'b0;
            X6_7_reg = 1'b0;
            X7_8_reg = 1'b0;
        end else begin
           X8ant_1_reg = X8ant ^ X1;
            X1_2_reg = X1 ^ X2;
            X2_3_reg = X2 ^ X3;
             X3_4_reg = X3 ^ X4;
            X4_5_reg = X4 ^ X5;
            X5_6_reg = X5 ^ X6;
            X6_7_reg = X6 ^ X7;
            X7_8_reg = X7 ^ X8;

        end
    end

    assign Comp_X8ant_1 = X8ant_1_reg;
    assign Comp_X1_2 = X1_2_reg;
    assign Comp_X2_3 = X2_3_reg;
    assign Comp_X3_4 = X3_4_reg;
    assign Comp_X4_5 = X4_5_reg;
    assign Comp_X5_6 = X5_6_reg;
    assign Comp_X6_7 = X6_7_reg;
    assign Comp_X7_8 = X7_8_reg;

endmodule
```

I. CÓDIGO EN VERILOG: DIVISOR DE RELOJ

```
/**********************************************************************************
Name: clockGenerator.v
Description: This module divides the input frequency by two, and this new
frequency which is slower, will be used to sample the input data

Inputs:
*   clk_selected:   input selected clock phase
*   reset:          reset signal

Outputs:
*   clk_recovered:  Divided clock

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************************/

module clockGenerator
(
   input clk_selected,
   input reset,
   output clk_recovered
);

wire ffd1_q;
wire qneg_ffd1;
wire ffd2_q;
wire qneg_ffd2;

not_module
#(
   .NBIT(1'B1)
)NOT_FF1
(
   .in(ffd1_q),
   .out(qneg_ffd1)
);

not_module
#(
   .NBIT(1'B1)
)NOT_FF2
(
   .in(ffd2_q),
   .out(qneg_ffd2)
);

Flipflop_D
#(
   .Nbit(1)
)FFD1
(
   .input_data(qneg_ffd1),
   .clk(clk_selected),
   .rst(reset),
   .q(ffd1_q)
```

```
);

Flipflop_D
#(
   .Nbit(1)
)FFD2
(
   .input_data(qneg_ffd2),
   .clk(ffd1_q),
   .rst(reset),
   .q(ffd2_q)
);

assign clk_recovered = ffd2_q;

endmodule
```

J. CÓDIGO EN VERILOG: Contador de detección de transiciones

```
/**********************************************************************************
Name: CounterEvents.v
Description: This module counts every transition detection, it means, whenever the signal "eventtoC" changes from 0 to 1, the count increases.

Inputs:
*   eventtoC: trigger signal to increase counting
*   clk : clock signal
*   reset : reset signal

Outputs:
*   [NBITS-1:0] counter: counter accumulator

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************************/

module CounterEvents
#(
   // Parameter Declarations
   parameter MAXIMUM_VALUE = 32,
   //parameter TARGET= MAXIMUM_VALUE/2,
   parameter NBITS = CeilLog2(MAXIMUM_VALUE)
)
(
   // Input Ports
      input eventtoC,
   input clk,
   input reset,
   // Output Ports
   output[NBITS-1:0] counter
);

reg [NBITS-1:0] counter_reg=0;

/**********************************************************************************/
```

```
   //always@(negedge clk or negedge reset) begin
   always@(posedge clk or negedge reset) begin
      if (reset == 1'b0)
         counter_reg <= {NBITS{1'b0}};
      else begin
            if(eventtoC == 1'b1) begin

               if(counter_reg == MAXIMUM_VALUE-1 )begin
                  counter_reg <= {NBITS{1'b0}};
               end else begin
                  counter_reg <= counter_reg + 1'b1;
               end
            end
      end
   end

/*****************************************************************************************/
assign counter = counter_reg;
/***********

/*****************************************************************************************/

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

/*****************************************************************************************/

endmodule
```

K. Código en verilog: módulo contador de 16 datos

```
/*************************************************************************
Name: CounterwFlag_P.v
Description: This module will be used to count 16 data and generate a pulse
Parameter:
*   MAXIMUM_VALUE = Number of data samples to count
*   NBITS = Number of bits required to count to the maximum value
Inputs:
*   clk :   clock signal
*   reset : reset signal

Outputs:
*   flag:   signal which indicates the counting reached the maximum value
*   counter:    signal which stores the counting value

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
```

```
**************************************************************************/

module CounterwFlag_P
#(
   // Parameter Declarations
   parameter MAXIMUM_VALUE = 67, //64 samples (16 data) + 2 cycles to let the max counter propagate through the output
flip flop
   parameter NBITS = CeilLog2(MAXIMUM_VALUE)
)
(
   // Input Ports
   input clk,
   input reset,
   // Output Ports
   output flag,
);

reg MaxValue_Bit=1'b0;
reg flag_reg=0;
reg [NBITS-1:0] counter_reg=0;

/*****************************************************************************************************/
   always@(posedge clk or negedge reset) begin
      if (reset == 1'b0) begin
         counter_reg <= {NBITS{1'b0}};
      end else begin
               if(counter_reg == MAXIMUM_VALUE-1)begin
                  counter_reg <= { NBITS{1'b0} };
               end else begin
                  counter_reg <= counter_reg + 1'b1;
               end
      end
   end
/*************************************************************************/

always@(counter_reg)begin
   if(counter_reg == 0 || counter_reg ==1)begin
      flag_reg =1'b1;
   end else
      flag_reg =1'b0;
end

assign flag = flag_reg;
//assign counter = counter_reg;
/*************************************************************************/
/*************************************************************************/
 /*Log Function*/
    function integer CeilLog2;
      input integer data;
      integer i,result;
      begin
        for(i=0; 2**i < data; i=i+1)
          result = i + 1;
        CeilLog2 = result;
      end
   endfunction
/*************************************************************************/

endmodule
```

L.Código en Verilog: Módulo multiplexor 8 a 1

```
/**********************************************************************************
Name: Mux8_to_1.v
Description: This module is an 8 to 1 multiplexer
Inputs:
*   Selector:   Selector signal
*   Data_0:     Input signal 0
*   Data_1:     Input signal 1
*   Data_2:     Input signal 2
*   Data_3:     Input signal 3
*   Data_4:     Input signal 4
*   Data_5:     Input signal 5
*   Data_6:     Input signal 6
*   Data_7:     Input signal 7
Outputs:
*   Mux_Output: Selected signal

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************************/
module Mux8_to_1
(
   // Input Ports
   input [2:0]Selector,
   input Data_0,
   input Data_1,
   input Data_2,
   input Data_3,
   input Data_4,
   input Data_5,
   input Data_6,
   input Data_7,
   // Output Ports
   output reg Mux_Output
);
 always@(Selector,
     Data_0,
     Data_1,
     Data_2,
     Data_3,
     Data_4,
     Data_5,
     Data_6,
     Data_7
 ) begin
   case(Selector)
      3'd0 : Mux_Output= Data_0;
      3'd1 : Mux_Output= Data_1;
      3'd2 : Mux_Output= Data_2;
      3'd3 : Mux_Output= Data_3;
      3'd4 : Mux_Output= Data_4;
      3'd5 : Mux_Output= Data_5;
      3'd6 : Mux_Output= Data_6;
      3'd7 : Mux_Output= Data_7;
    default: Mux_Output= Data_0;
   endcase
 end
endmodule
```

M. Código en Verilog: Multiplexor 2 a 1

```
/*********************************************************************************
Name: mux2to1.v
Description: This module is a 2 to 1 multiplexer

Inputs:
*   input mux_sel:  Mux selector
*   input [Nbit-1:0] data1: input signal 1
*   input [Nbit-1:0] data2: input signal 2

Outputs:
*   Data_out:   selected signal

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
*********************************************************************************/
module mux2to1
#(
   parameter Nbit=6
)
(
   input mux_sel,
   input [Nbit-1:0] data1,
   input [Nbit-1:0] data2,
   output [Nbit-1:0] Data_out
);

reg [Nbit-1:0] mux_output;
always@(mux_sel,data1,data2)begin
   if(mux_sel==1'b0)begin
      mux_output = data1;
   end else begin
      mux_output = data2;
   end
end

assign Data_out = mux_output;
endmodule
```

N. Código en Verilog: Módulo generador de pulso para reiniciar contadores

```
/**********************************************************************************************
Name: One_Shot.v
Description: This module generates a pulse to reset the counters stage

Inputs:
*   clk : clock signal
*   reset : reset signal
*   Start : signal which indicates the start of the operation

Outputs:
*   Shot :  Reset pulse for the counters

Version: 1.0
Autor: Nétor Damián García
Fecha: Primavera 2018
**********************************************************************************************/
module One_Shot
#(
   parameter Waiting_Shot = 3'd0,
   parameter Shot_State = 3'd1,
   parameter Shot_State2 = 3'd2,
   parameter Shot_State3 = 3'd3,
   parameter Waiting_Not_Shot = 3'd4
)
(
   // Input Ports
   input clk,
   input reset,
   input Start,

   // Output Ports
   output Shot
);
reg [2:0] state;
reg Shot_reg;
wire Not_Start;
assign Not_Start = Start;
/*-------------------------------------------------------------------------------------------*/
/*Asignacion de estado*/

//always@(negedge clk or negedge reset)
always@(posedge clk or negedge reset)
begin

if(reset == 1'b0) begin
      state<=Waiting_Shot;
end
else begin
      case(state)

      Waiting_Shot:
         if(Not_Start == 1'b1)
            state <= Shot_State;
```

```
            else
               state <= Waiting_Shot;

         Shot_State:
               state <= Shot_State2;
         Shot_State2:
               state <= Shot_State3;
         Shot_State3:
               state <= Waiting_Not_Shot;
         Waiting_Not_Shot:
            if (Not_Start == 1'b1)
               state <= Waiting_Not_Shot;
            else
               state <= Waiting_Shot;

         default:
               state <= Shot_State;

         endcase
      end
end//end always
/*-------------------------------------------------------------------------------------------*/
/*Salida de control, proceso combintorio*/
always@(state)
begin
   case(state)
      Waiting_Shot:
         begin
            Shot_reg=1'b1;
         end

      Shot_State:
         begin
            Shot_reg=1'b1;
         end
      Shot_State2:
         begin
            Shot_reg=1'b0;
         end
      Shot_State3:
         begin
            Shot_reg=1'b0;
         end
      Waiting_Not_Shot:
         begin
            Shot_reg=1'b1;
         end

   default:
      begin
            Shot_reg=1'b1;
      end
   endcase
end

assign Shot = Shot_reg;

endmodule
```

O. Código Verilog: Constraints para síntesis lógica

```
## CDR Timing Constraints
## ITESO University
## Nestor Garcia Hdez.


#the following parameters will be considered in whole design
set_time_unit -picoseconds
set_load_unit -femtofarads
#current_design CDR

#### Clock definition for the input clks from PLL (800MHz)  ###

define_clock -name 800MHz_CLK0 -period 1250  -rise 0 -fall 50 -domain 1 -divide_period 1
/designs/CDR/ports_in/sampling_clk0;  #0
define_clock -name 800MHz_CLK45 -period 1250 -rise 12 -fall 62 -domain 2 -divide_period 1
/designs/CDR/ports_in/sampling_clk1;   #45
define_clock -name 800MHz_CLK90 -period 1250 -rise 25 -fall 75  -domain 3 -divide_period 1
/designs/CDR/ports_in/sampling_clk2;  #90
define_clock -name 800MHz_CLK135 -period 1250 -rise 37 -fall 87 -domain 4 -divide_period 1
/designs/CDR/ports_in/sampling_clk3 ; #135
define_clock -name 800MHz_CLK180 -period 1250 -rise 50 -fall 0 -domain 5 -divide_period 1
/designs/CDR/ports_in/sampling_clk4  ; #180
define_clock -name 800MHz_CLK225 -period 1250 -rise 62 -fall 12 -domain 6 -divide_period 1
/designs/CDR/ports_in/sampling_clk5 ; #225
define_clock -name 800MHz_CLK270 -period 1250 -rise 75 -fall 25 -domain 7 -divide_period 1
/designs/CDR/ports_in/sampling_clk6 ; #270
define_clock -name 800MHz_CLK315 -period 1250 -rise 87 -fall 37 -domain 8 -divide_period 1
/designs/CDR/ports_in/sampling_clk7 ; #315

############### 1st stage of FF ############################

define_clock -name FF_1st_clk_p0   -period 1250   -domain 1 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_p0/pins_in/clk]
define_clock -name FF_1st_clk_p45   -period 1250  -domain 2  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_p45/pins_in/clk]
define_clock -name FF_1st_clk_p90   -period 1250   -domain 3 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_p90/pins_in/clk]
define_clock -name FF_1st_clk_p135   -period 1250  -domain 4 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_p135/pins_in/clk]
define_clock -name FF_1st_clk_p180   -period 1250  -domain 5 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_p180/pins_in/clk]
define_clock -name FF_1st_clk_p225   -period 1250  -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_p225/pins_in/clk]
define_clock -name FF_1st_clk_p270   -period 1250   -domain 7 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_p270/pins_in/clk]
define_clock -name FF_1st_clk_p315   -period 1250  -domain 8  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_p315/pins_in/clk]

############### 2nd stage of FF ############################

define_clock -name FF_2nd_clk_0 -period 1250 -domain 8 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_0_2nd/pins_in/clk]
define_clock -name FF_2nd_clk_45 -period 1250 -domain 8 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_45_2nd/pins_in/clk]
define_clock -name FF_2nd_clk_90 -period 1250 -domain 8 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_90_2nd/pins_in/clk]
```

```
define_clock -name FF_2nd_clk_135 -period 1250 -domain 8 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_135_2nd/pins_in/clk]
define_clock -name FF_2nd_clk_180 -period 1250 -domain 8 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_180_2nd/pins_in/clk]
define_clock -name FF_2nd_clk_225 -period 1250 -domain 8 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_225_2nd/pins_in/clk]
define_clock -name FF_2nd_clk_270 -period 1250 -domain 8 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_270_2nd/pins_in/clk]



################## Xor stage-Neg 225° domain 6 (1 cycle to launch and capture)######

#Create a negated clock from phase 6 clock

#define_clock -name xorclk -period 1250 -rise 12 -fall 62 -domain 6
/designs/CDR_LFSR/instances_hier/cdr_module/Edge_detector/instances_hier/xor_2/pins_in/clk

define_clock -name xorclk1  -period 1250 -rise 12 -fall 62 -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/xor_2/instances_seq/X1_2_reg_reg/pins_in/CLK]
define_clock -name xorclk2  -period 1250 -rise 12 -fall 62 -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/xor_2/instances_seq/X2_3_reg_reg/pins_in/CLK]
define_clock -name xorclk3  -period 1250 -rise 12 -fall 62 -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/xor_2/instances_seq/X3_4_reg_reg/pins_in/CLK]
define_clock -name xorclk4  -period 1250 -rise 12 -fall 62 -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/xor_2/instances_seq/X4_5_reg_reg/pins_in/CLK]
define_clock -name xorclk5  -period 1250 -rise 12 -fall 62 -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/xor_2/instances_seq/X5_6_reg_reg/pins_in/CLK]
define_clock -name xorclk6  -period 1250 -rise 12 -fall 62 -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/xor_2/instances_seq/X6_7_reg_reg/pins_in/CLK]
define_clock -name xorclk7  -period 1250 -rise 12 -fall 62 -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/xor_2/instances_seq/X7_8_reg_reg/pins_in/CLK]
define_clock -name xorclk8  -period 1250 -rise 12 -fall 62 -domain 6 [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/xor_2/instances_seq/X8ant_1_reg_reg/pins_in/CLK]

##########  Neg 180 clock domain 5, 180deg ########################################

define_clock -name not_180clk -period 1250 -rise 0 -fall 50  -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/cont_to16data/pins_in/clk]

####for second stage FF only 315 Flip flop
define_clock -name FF_2nd_clk_315 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/FF_315_2nd/pins_in/clk]

####for save previous Xor 8 Flip flop (probably this can use the 315 clock )

define_clock -name prevXor8clk -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/save_previous_Xor8/pins_in/clk]

####for counting stage

define_clock -name CountingStageclk12 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/counter_1_2/pins_in/clk]
define_clock -name CountingStageclk23 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/counter_2_3/pins_in/clk]
define_clock -name CountingStageclk34 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/counter_3_4/pins_in/clk]
define_clock -name CountingStageclk45 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/counter_4_5/pins_in/clk]
```

```
define_clock -name CountingStageclk56 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/counter_5_6/pins_in/clk]
define_clock -name CountingStageclk67 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/counter_6_7/pins_in/clk]
define_clock -name CountingStageclk78 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/counter_7_8/pins_in/clk]
define_clock -name CountingStageclk81 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/Edge_detector/instances_hier/counter_8_1/pins_in/clk]

####for flip flops stage between detect_edge module and phase selector mode

define_clock -name FF_edgePhaseSelclk12 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/FF_edgePhaseSel12/pins_in/clk]
define_clock -name FF_edgePhaseSelclk23 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/FF_edgePhaseSel23/pins_in/clk]
define_clock -name FF_edgePhaseSelclk34 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/FF_edgePhaseSel34/pins_in/clk]
define_clock -name FF_edgePhaseSelclk45 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/FF_edgePhaseSel45/pins_in/clk]
define_clock -name FF_edgePhaseSelclk56 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/FF_edgePhaseSel56/pins_in/clk]
define_clock -name FF_edgePhaseSelclk67 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/FF_edgePhaseSel67/pins_in/clk]
define_clock -name FF_edgePhaseSelclk78 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/FF_edgePhaseSel78/pins_in/clk]
define_clock -name FF_edgePhaseSelclk81 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/FF_edgePhaseSel81/pins_in/clk]
#-------------------------------------------------------------------------------------------
## *************   for Phase selector mode start ******************************

define_clock -name Phase_select_modClk -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/pins_in/clk]
define_clock -name Datainmux1_clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[0]/pins_in/CLK]
define_clock -name Datainmux1_clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[1]/pins_in/CLK]
define_clock -name Datainmux1_clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[2]/pins_in/CLK]
define_clock -name Datainmux1_clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[3]/pins_in/CLK]
define_clock -name Datainmux1_clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[4]/pins_in/CLK]

#####mux2

define_clock -name Datainmux2_clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[0]/pins_in/CLK]
define_clock -name Datainmux2_clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[1]/pins_in/CLK]
define_clock -name Datainmux2_clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[2]/pins_in/CLK]
```

```
define_clock -name Datainmux2_clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[3]/pins_in/CLK]
define_clock -name Datainmux2_clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[4]/pins_in/CLK]

########FFDatamux

define_clock -name FFDatamux_clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datamux/instances_seq/output_data_re
g[0]/pins_in/CLK]
define_clock -name FFDatamux_clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datamux/instances_seq/output_data_re
g[1]/pins_in/CLK]
define_clock -name FFDatamux_clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datamux/instances_seq/output_data_re
g[2]/pins_in/CLK]
define_clock -name FFDatamux_clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datamux/instances_seq/output_data_re
g[3]/pins_in/CLK]
define_clock -name FFDatamux_clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Datamux/instances_seq/output_data_re
g[4]/pins_in/CLK]
define_clock -name FF_Sxinmux_clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[0]/pins_in/CLK]
define_clock -name FF_Sxmux_comp1clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp1/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
0]/pins_in/CLK]

###Comp2

define_clock -name FF_Datainmux1clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux1clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux1clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux1clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux1clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[4]/pins_in/CLK]

#####FF_Datainmux2

define_clock -name FF_Datainmux2clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux2clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux2clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[2]/pins_in/CLK]
```

```
define_clock -name FF_Datainmux2clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux2clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[4]/pins_in/CLK]

####Datainsmux2

define_clock -name FFDatamux2_clk0   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datamux/instances_seq/output_data_re
g[0]/pins_in/CLK]
define_clock -name FFDatamux2_clk1   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datamux/instances_seq/output_data_re
g[1]/pins_in/CLK]
define_clock -name FFDatamux2_clk2   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datamux/instances_seq/output_data_re
g[2]/pins_in/CLK]
define_clock -name FFDatamux2_clk3   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datamux/instances_seq/output_data_re
g[3]/pins_in/CLK]
define_clock -name FFDatamux2_clk4   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Datamux/instances_seq/output_data_re
g[4]/pins_in/CLK]
define_clock -name FF_2Sxinmux1_clk0   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Sxinmux1/instances_seq/output_data_r
eg[1]/pins_in/CLK]
define_clock -name FF_2Sxmux2_clk1   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[0]/pins_in/CLK]
define_clock -name FF_Sxmux_clk0   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
0]/pins_in/CLK]

define_clock -name FF_SxmuxComp2_clk1   -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp2/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
1]/pins_in/CLK]

#Comp3

define_clock -name FF_SxmuxComp3_clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
0]/pins_in/CLK]
define_clock -name FF_SxmuxComp3_clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
2]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp3clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp3clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp3clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp3clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[3]/pins_in/CLK]
```

define_clock -name FF_Datainmux1_comp3clk4 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainmux1/instances_seq/output_data_reg[4]/pins_in/CLK]

#####FF_Datainmux2

define_clock -name FF_Datainmux2_comp3clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp3clk1 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp3clk2 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp3clk3 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp3clk4 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[4]/pins_in/CLK]

####Datainsmux2

define_clock -name FFDatamux2_comp3clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datamux/instances_seq/output_data_reg[0]/pins_in/CLK]
define_clock -name FFDatamux2_comp3clk1 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datamux/instances_seq/output_data_reg[1]/pins_in/CLK]
define_clock -name FFDatamux2_comp3clk2 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datamux/instances_seq/output_data_reg[2]/pins_in/CLK]
define_clock -name FFDatamux2_comp3clk3 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datamux/instances_seq/output_data_reg[3]/pins_in/CLK]
define_clock -name FFDatamux2_comp3clk4 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Datamux/instances_seq/output_data_reg[4]/pins_in/CLK]
define_clock -name FF_3Sxinmux1_clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Sxinmux1/instances_seq/output_data_reg[2]/pins_in/CLK]
define_clock -name FF_3Sxmux2_clk1 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp3/instances_hier/FF_Sxinmux2/instances_seq/output_data_reg[0]/pins_in/CLK]

############### Comp4

define_clock -name FF_4Sxinmux1_clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Sxinmux1/instances_seq/output_data_reg[1]/pins_in/CLK]
define_clock -name FF_4Sxmux2_clk1 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Sxinmux2/instances_seq/output_data_reg[0]/pins_in/CLK]
define_clock -name FF_SxmuxComp4_clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Sxmux/instances_seq/output_data_reg[0]/pins_in/CLK]
define_clock -name FF_SxmuxComp4_clk1 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Sxmux/instances_seq/output_data_reg[1]/pins_in/CLK]

```
define_clock -name FF_Datainmux1_comp4clk0 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp4clk1 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp4clk2 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp4clk3 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp4clk4 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[4]/pins_in/CLK]

#####FF_Datainmux2

define_clock -name FF_Datainmux2_comp4clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp4clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp4clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp4clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp4clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[4]/pins_in/CLK]

####Datainsmux2

define_clock -name FFDatamux2_comp4clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datamux/instances_seq/output_data_re
g[0]/pins_in/CLK]
define_clock -name FFDatamux2_comp4clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datamux/instances_seq/output_data_re
g[1]/pins_in/CLK]
define_clock -name FFDatamux2_comp4clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datamux/instances_seq/output_data_re
g[2]/pins_in/CLK]
define_clock -name FFDatamux2_comp4clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datamux/instances_seq/output_data_re
g[3]/pins_in/CLK]
define_clock -name FFDatamux2_comp4clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp4/instances_hier/FF_Datamux/instances_seq/output_data_re
g[4]/pins_in/CLK]

###Comp5

define_clock -name FF_Datainmux1_comp5clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp5clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[1]/pins_in/CLK]
```

```
define_clock -name FF_Datainmux1_comp5clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp5clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp5clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainmux1/instances_seq/output_data
_reg[4]/pins_in/CLK]
define_clock -name FF_Sxinmux1_comp5clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Sxinmux1/instances_seq/output_data_r
eg[0]/pins_in/CLK]
define_clock -name FF_Sxinmux2_comp5clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[0]/pins_in/CLK]
define_clock -name FF_Sxinmux2_comp5clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[1]/pins_in/CLK]
define_clock -name FF_Sxmux_comp5clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
0]/pins_in/CLK]
define_clock -name FF_Sxmux_comp5clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
1]/pins_in/CLK]

#####FF_Datainmux2

define_clock -name FF_Datainmux2_comp5clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp5clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp5clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp5clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp5clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datainsmux2/instances_seq/output_dat
a_reg[4]/pins_in/CLK]

####Datainsmux2

define_clock -name FFDatamux2_comp5clk0 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datamux/instances_seq/output_data_re
g[0]/pins_in/CLK]
define_clock -name FFDatamux2_comp5clk1 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datamux/instances_seq/output_data_re
g[1]/pins_in/CLK]
define_clock -name FFDatamux2_comp5clk2 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datamux/instances_seq/output_data_re
g[2]/pins_in/CLK]
define_clock -name FFDatamux2_comp5clk3 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datamux/instances_seq/output_data_re
g[3]/pins_in/CLK]
define_clock -name FFDatamux2_comp5clk4 -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp5/instances_hier/FF_Datamux/instances_seq/output_data_re
g[4]/pins_in/CLK]
```

Comp6

define_clock -name FF_Datainmux1_comp6clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainmux1/instances_seq/output_data_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp6clk1 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainmux1/instances_seq/output_data_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp6clk2 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainmux1/instances_seq/output_data_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp6clk3 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainmux1/instances_seq/output_data_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux1_comp6clk4 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainmux1/instances_seq/output_data_reg[4]/pins_in/CLK]

#####FF_Datainmux2

define_clock -name FF_Datainmux2_comp6clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[0]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp6clk1 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[1]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp6clk2 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[2]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp6clk3 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[3]/pins_in/CLK]
define_clock -name FF_Datainmux2_comp6clk4 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datainsmux2/instances_seq/output_data_reg[4]/pins_in/CLK]

####Datainsmux2

define_clock -name FFDatamux2_comp6clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datamux/instances_seq/output_data_reg[0]/pins_in/CLK]
define_clock -name FFDatamux2_comp6clk1 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datamux/instances_seq/output_data_reg[1]/pins_in/CLK]
define_clock -name FFDatamux2_comp6clk2 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datamux/instances_seq/output_data_reg[2]/pins_in/CLK]
define_clock -name FFDatamux2_comp6clk3 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datamux/instances_seq/output_data_reg[3]/pins_in/CLK]
define_clock -name FFDatamux2_comp6clk4 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Datamux/instances_seq/output_data_reg[4]/pins_in/CLK]
define_clock -name FF_Sxinmux1_Comp6_clk0 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Sxinmux1/instances_seq/output_data_reg[0]/pins_in/CLK]
define_clock -name FF_Sxinmux1_Comp6_clk2 -period 1250 -rise 0 -fall 50 -domain 5 [get_pins /designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Sxinmux1/instances_seq/output_data_reg[2]/pins_in/CLK]

```
define_clock -name FF_Sxinmux2_Comp6_clk0 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[0]/pins_in/CLK]
define_clock -name FF_Sxinmux2_Comp6_clk1 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[1]/pins_in/CLK]
define_clock -name FF_Sxmux_Comp6_clk0 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
0]/pins_in/CLK]
define_clock -name FF_Sxmux_Comp6_clk1 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
1]/pins_in/CLK]
define_clock -name FF_Sxmux_Comp6_clk2 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp6/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
2]/pins_in/CLK]

#Comp 7

define_clock -name FF_Sxinmux1_Comp7_clk0 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp7/instances_hier/FF_Sxinmux1/instances_seq/output_data_r
eg[0]/pins_in/CLK]
define_clock -name FF_Sxinmux1_Comp7_clk1 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp7/instances_hier/FF_Sxinmux1/instances_seq/output_data_r
eg[1]/pins_in/CLK]
define_clock -name FF_Sxinmux2_Comp7_clk0 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp7/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[0]/pins_in/CLK]
define_clock -name FF_Sxinmux2_Comp7_clk1 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp7/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[1]/pins_in/CLK]
define_clock -name FF_Sxinmux2_Comp7_clk2 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp7/instances_hier/FF_Sxinmux2/instances_seq/output_data_r
eg[2]/pins_in/CLK]
define_clock -name FF_Sxmux_Comp7_clk0 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp7/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
0]/pins_in/CLK]
define_clock -name FF_Sxmux_Comp7_clk1 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp7/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
1]/pins_in/CLK]
define_clock -name FF_Sxmux_Comp7_clk2 -period 1250 -rise 0 -fall 50 -domain 5   [get_pins
/designs/CDR/instances_hier/phase_select_mod/instances_hier/comp7/instances_hier/FF_Sxmux/instances_seq/output_data_reg[
2]/pins_in/CLK]

#---------------------------------------

############### for adaptive update phase module
define_clock -name allowupdclk -period 1250 -rise 0 -fall 50 -domain 5  [get_pins
/designs/CDR/instances_hier/allow_updateCount/pins_in/clk]


##################################### end ###################################


################## Other Flip flops domain 11 ###################################

## This will use as clock the signal "flag_to_reset_counting" but negated
define_clock -name save_selecClk1 -period 83750 -rise 3 -fall 6 -domain 11 /designs/CDR/instances_hier/save_previousSel/clk

#This will use as clock the signal "flag_to_startcounting_decision". It is a pulse generated after 16 data
define_clock -name save_selecClk3 -period 83750 -rise 0 -fall 3 -domain 11 /designs/CDR/instances_hier/save_Newselection/clk
```

```
################################### end ######################################


################## Mux-phase domain 15 ########################################
#Define the multiple clocks which can come from the input of the multiplexer

create_clock -name muxin1 -period 1250  [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected]
create_clock -name muxin2 -period 1250  [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] -add
create_clock -name muxin3 -period 1250  [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] -add
create_clock -name muxin4 -period 1250  [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] -add
create_clock -name muxin5 -period 1250  [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] -add
create_clock -name muxin6 -period 1250  [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] -add
create_clock -name muxin7 -period 1250  [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] -add
create_clock -name muxin8 -period 1250  [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] -add

create_generated_clock -name clkgen1 \
            -divide_by 4 -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
            -master_clock muxin1 [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]]

create_generated_clock -name clkgen2 \
            -divide_by 4 -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
            -master_clock muxin2 [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]] -add

create_generated_clock -name clkgen3 \
            -divide_by 4 -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
            -master_clock muxin3 [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]] -add

create_generated_clock -name clkgen4 \
            -divide_by 4 -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
            -master_clock muxin4 [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]] -add

create_generated_clock -name clkgen5 \
            -divide_by 4 -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
            -master_clock muxin5 [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]] -add

create_generated_clock -name clkgen6 \
            -divide_by 4 -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
            -master_clock muxin6 [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]] -add

create_generated_clock -name clkgen7 \
            -divide_by 4 -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
            -master_clock muxin7 [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]] -add

create_generated_clock -name clkgen8 \
            -divide_by 4 -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
            -master_clock muxin8 [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]] -add


################## Clock divider module domain 12-13 ############################

### clock from "mux_phase" Module to clockGenerator (clock divider) module  ###

create_generated_clock \
           -name c1_genClk \
           -source [get_pins /designs/CDR/instances_hier/clkgenT/pins_in/clk_selected] \
           -divide_by 2 -invert [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD1/pins_out/q[0]]

### clock from internal clock divider module. This signal has 400MHz freq  ###
```

```
create_generated_clock \
        -name clk_div200 \
        -source [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD1/pins_out/q[0]] \
        -divide_by 2 -invert [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]]


#Negated clock for the sampling flip flop

create_generated_clock \
        -name neg4recovering \
        -source [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD2/pins_out/q[0]] \
        -invert \
         [get_pins /designs/CDR/instances_hier/FF_recovered/pins_in/clk]


################## Sampling original data domain 14 ##############################

### clock for the Sampling Flip flop (FF_recovered), it contains 4 clock cycles from original 800MHz clock. It comes from the
clkgenT module ###
create_generated_clock \
        -name divClk_FFrecovered \
        -source [get_pins /designs/CDR/instances_hier/clkgenT/instances_hier/FFD1/pins_out/q[0]] \
         [get_pins /designs/CDR/instances_hier/FF_recovered/pins_in/clk]

################################### end ########################################

# Input delay definition: This is the delay coming from outside the design. Here, it's defined at 10% of clock period.

external_delay -clock [find / -clock 800MHz_CLK0] -input 125 -name IDelay [find /des* -port ports_in/*]
external_delay -clock [find / -clock 800MHz_CLK0] -input 125 -name IDelay [find -port /designs/CDR/ports_in/original_data]
external_delay -clock [find / -clock 800MHz_CLK0] -input 125 -name IDelay [find -port designs/CDR/reset]

# Output delay definition: This is the delay going outside the design.  Here, it's defined at 10% of clock period.

external_delay -clock [find / -clock 800MHz_CLK0] -output 125 -name ODelay [find /des* -port ports_out/*]


# Driving cell definition

set_attribute external_driver [find [find / -libcell BUFFER_D] -libpin Z] { /designs/CDR/ports_in/* }

# Considering a pad output buffer POC2A load

set_attribute external_pin_cap 31 /designs/CDR/ports_out/*
```

P. Código Verilog: Script para realizar síntesis física

```
# /*********************************************************************
# Description: This script creates the physical synthesis in EDI Encounter for the CDR
# Version: 1.0
# Autor: Nétor Damián García
# Fecha: Primavera 2018
#*********************************************************************/

######Specifying the floorplan ################

floorPlan -site CORE -s 207 207 11 11 11 11
uiSetTool select
getIoFlowFlag
fit
uiSetTool ruler
setDesignMode -process 130
clearGlobalNets
globalNetConnect VDD -type pgpin -pin VDD -inst * -module {}
globalNetConnect VDD -type tiehi -pin VDD -inst * -module {}
globalNetConnect VSS -type pgpin -pin VSS -inst * -module {}
globalNetConnect VSS -type tielo -pin VSS -inst * -module {}
set sprCreateIeRingNets {}
set sprCreateIeRingLayers {}
set sprCreateIeRingWidth 1.0
set sprCreateIeRingSpacing 1.0
set sprCreateIeRingOffset 1.0
set sprCreateIeRingThreshold 1.0
set sprCreateIeRingJogDistance 1.0

######Creating Power Ring ################

addRing -skip_via_on_wire_shape Noshape -skip_via_on_pin Standardcell -center 1 -stacked_via_top_layer AM -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 4 -spacing 1 -offset 0.2

zoomIn

fit

######Creating horizontal stripes ################

sroute -connect { blockPin padPin padRing corePin floatingStripe } -layerChangeRange { M1 AM } -blockPinTarget {
nearestTarget } -padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -corePinTarget { firstAfterRowEnd } -
floatingStripeTarget { blockring padring ring stripe ringpin blockpin followpin } -allowJogging 1 -crossoverViaLayerRange {
M1 AM } -nets { VDD VSS } -allowLayerChange 1 -blockPin useLef -targetViaLayerRange { M1 AM }

###### Doing verifications ################

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 }
verify_drc -report CDR.drc_powerring.rpt -limit 1000
verifyConnectivity -type all -error 1000 -warning 50
saveDesign CDR_powerring_corrected.enc
```

```
######Creating vertical stripes ################

set sprCreateIeStripeNets {}
set sprCreateIeStripeLayers {}
set sprCreateIeStripeWidth 10.0
set sprCreateIeStripeSpacing 2.0
set sprCreateIeStripeThreshold 1.0

addStripe -skip_via_on_wire_shape Noshape -block_ring_top_layer_limit MQ -max_same_layer_jog_length 4 -
padcore_ring_bottom_layer_limit M1 -number_of_sets 2 -skip_via_on_pin Standardcell -stacked_via_top_layer AM -
padcore_ring_top_layer_limit MQ -spacing 1 -xleft_offset 65 -xright_offset 65 -merge_stripes_value 0.2 -layer M2 -
block_ring_bottom_layer_limit M1 -width 3 -nets {VDD VSS} -stacked_via_bottom_layer M1

############ Doing the placement ##############

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
setDrawView place

############# Clock tree synthesis ############
setCTSMode -engine ck

createClockTreeSpec -bufferList {BUFFER_C BUFFER_D BUFFER_E BUFFER_F BUFFER_H BUFFER_I BUFFER_J
BUFFER_K BUFFER_L BUFFER_M BUFFER_N BUFFER_O CLKI_C CLKI_D CLKI_E CLKI_F CLKI_H CLKI_I CLKI_K
CLKI_M CLKI_O CLKI_Q CLK_C CLK_D CLK_E CLK_F CLK_H CLK_I CLK_K CLK_M CLK_O CLK_Q DELAY4_C
DELAY4_F DELAY4_J DELAY6_C DELAY6_F DELAY6_J DELAY6_M INVERTBAL_C INVERTBAL_D
INVERTBAL_E INVERTBAL_F INVERTBAL_H INVERTBAL_J INVERTBAL_L INVERT_A INVERT_B INVERT_C
INVERT_D INVERT_E INVERT_F INVERT_H INVERT_I INVERT_J INVERT_K INVERT_L INVERT_M INVERT_N
INVERT_O} -file ../Clock.ctstch
clockDesign -specFile ../Clock.ctstch -outDir clock_report -fixedInstBeforeCTS

uiSetTool select
displayClockTree -skew -allLevel -clkRouteOnly

############# Running 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

###### Doing verifications ################

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 }
verify_drc -report CDR.drc_nanoroute.rpt -limit 1000
```

```
verifyConnectivity -type all -error 1000 -warning 50
verifyProcessAntenna -reportfile CDR.antenna.rpt -error 1000

######## Running timing reports after nanorouting ###############

redirect -quiet {set honorDomain [getAnalysisMode -honorClockDomains]} > /dev/null
timeDesign -postCTS -pathReports -drvReports -slackReports -numPaths 50 -prefix CDR_postCTS -outDir timingReports
redirect -quiet {set honorDomain [getAnalysisMode -honorClockDomains]} > /dev/null
timeDesign -postCTS -hold -pathReports -slackReports -numPaths 50 -prefix CDR_postCTS -outDir timingReports
redirect -quiet {set honorDomain [getAnalysisMode -honorClockDomains]} > /dev/null
timeDesign -postRoute -pathReports -drvReports -slackReports -numPaths 50 -prefix CDR_postRoute -outDir timingReports
redirect -quiet {set honorDomain [getAnalysisMode -honorClockDomains]} > /dev/null
timeDesign -postRoute -hold -pathReports -slackReports -numPaths 100 -prefix CDR_postRoute -outDir timingReports
redirect -quiet {set honorDomain [getAnalysisMode -honorClockDomains]} > /dev/null
timeDesign -postRoute -hold -pathReports -slackReports -numPaths 50 -prefix CDR_postRoute -outDir timingReports
setDesignMode -process 130

source ../opt_post_cts.tcl

###### Doing verifications ################

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 }
verify_drc -report CDR.drc_optimization.rpt -limit 1000
verifyConnectivity -type all -error 1000 -warning 50
```

Q. Código Verilog: Globals for CDR

```
#####################################################################
#  Generated by:      Cadence Encounter 14.27-s035_1
#  OS:                Linux x86_64(Host ID fv00)
#  Generated on:      Wed Jun 20 18:29:11 2018
#  Design:
#  Command:           save_global CDR.globals
#####################################################################
#
# Version 1.1
#

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}
```

```
set init_lef_file {/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/bicmos8hp_5AM_21_tech.lef
/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/lef/BICMOS8HP_SC_1P2V_12T_RVT.lef
/opt/libs/IBM_PDK/bicmos8hp/v.20160727/lef/CMOS8HP_BASE_WB_IO_5LM.lef}
set init_mmmc_file {../CDR_corrected.view}
set init_pwr_net {VDD}
set init_verilog {../CDR_m_wc.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 ;}
```

R. Código Verilog: Views for the CDR

```
# /*****************************************************************************
# Description: This is the view file for the physical synthesis
# Version: 1.0
# Autor: Nétor Damián García
# Fecha: Primavera 2018
#*****************************************************************************/

# Version:1.0 MMMC View Definition File
# Do Not Remove Above Line
create_rc_corner -name typ_rc_corner -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 wc_rc_corner -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_timing_lib -timing
{/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/synopsys/typ_v150_t025/PnomV1p50T025_STD_CELL_8HP_12T.lib
/opt/libs/IBM_PDK/bicmos8hp/v.20160727/synopsys/typ_v150_v150_t25/IBM_CMOS8HP_BASE_WB_IO_TYP_V150_V150
_T25.lib}
create_library_set -name wc_timing_lib -timing
{/opt/libs/IBM_PDK/bicmos8hp/v.20160727/synopsys/slow_v108_v140_t125/IBM_CMOS8HP_BASE_WB_IO_SLOW_V108
_V140_T125.lib
/media/Ext/libs/IBM_PDK/bicmos8hp/v.20171220/synopsys/slow_v108_t125/PwcV1p08T125_STD_CELL_8HP_12T.lib}
create_constraint_mode -name typ_constraint_mode -sdc_files {../CDR_m_typ.sdc}
create_constraint_mode -name wc_constraint_mode -sdc_files {../CDR_m_wc.sdc}
create_delay_corner -name typ_delay_corner -library_set {typ_timing_lib} -rc_corner {typ_rc_corner}
create_delay_corner -name wc_delay_corner -library_set {wc_timing_lib} -rc_corner {wc_rc_corner}
create_analysis_view -name typ_analysis_view -constraint_mode {typ_constraint_mode} -delay_corner {typ_delay_corner}
create_analysis_view -name wc_analysis_view -constraint_mode {wc_constraint_mode} -delay_corner {wc_delay_corner}
set_analysis_view -setup {wc_analysis_view} -hold {typ_analysis_view}
```

S. Código Verilog: Optimization script for the physical synthesis

```
# ITESO University
# This script shoud be sourced affter the
# .ctstch file has been modified for completing constraints

setDesignMode -process 130

Puts "Timing the design before CTS"

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

timeDesign -preCTS -prefix preCTS_setup
timeDesign -preCTS -prefix preCTS_hold;#  -hold

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

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

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

Puts "Optimizing for DRV"
optDesign -postCTS -drv

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

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

Puts "Optimizing for Setup"
optDesign -postCTS

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

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

Puts "Routing the Design"
setNanoRouteMode -quiet -timingEngine {}
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
```

```
setNanoRouteMode -quiet -routeWithTimingDriven false
setNanoRouteMode -quiet -routeWithSiDriven false
routeDesign -globalDetail

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

T.Código en Verilog: Testbench para simular CDR

```
`timescale 1 ps / 1 ps

module CDR_TB;

        localparam NUM_SAMPLES=2;
        localparam NBIT_NSAMPLES=CeilLog2(NUM_SAMPLES);
        localparam BIT_COUNT=6;
        localparam Bits_ph=3;


        //localparam clk_200MHz=20010; //200MHz
        //Respecto a 800MHz
        //needed delays in pico seconds


        localparam clk_p0=1250;  //800MHz 0°
        localparam clk_p1=156;  //800MHz 45°
        localparam clk_p2=312;  //800MHz 90°
        localparam clk_p3=468;  //800MHz 135°
        localparam clk_p4=625;  //800MHz 180°
        localparam clk_p5=782;  //800MHz 225°
        localparam clk_p6=938;  //800MHz 270°
        localparam clk_p7=1094;  //800MHz 315°

        localparam NUM_SIGN=8;
        localparam jitter =100;

        reg reset=0;
//      reg enable=0;
        reg data_nojitter=0;
        reg data=0;

        reg phase_0=1;
        reg phase_1=1;
        reg phase_2=1;
        reg phase_3=1;
        reg phase_4=1;
        reg phase_5=1;
        reg phase_6=1;
        reg phase_7=1;
        wire [2:0] decision_final;
        wire clk_recovered;
```

```
        wire recovered_data;


initial begin

        $recordfile("CDR_results_Jitter_5P_Contadores_Comp_New.trn");
        $recordvars;
end

CDR cdr_module
(
//      .original_data(data_nojitter),
        .original_data(data),
        .sampling_clk0(phase_0),
        .sampling_clk1(phase_1),
        .sampling_clk2(phase_2),
        .sampling_clk3(phase_3),
        .sampling_clk4(phase_4),
        .sampling_clk5(phase_5),
        .sampling_clk6(phase_6),
        .sampling_clk7(phase_7),
        .reset(reset),
        //.enable_cdr(enable),
        .best_sampling_clk_selector(decision_final),
        .clk_recovered(clk_recovered),
        .recovered_data(recovered_data)
);

  /*************** +/- 20% ***************/
  //Random value for 100MHz : +/- 2000ps
  localparam range_jitter_100M_20 = 2000;
  //Random value for 200MHz : +/- 1000ps
  //localparam range_jitter_200M_20 = 500;   //10%
  localparam range_jitter_200M_20 = 250;   //5%
  /*************** +/- 30% ***************/
  //Random value for 100MHz : +/- 3000ps
  localparam range_jitter_100M_30 = 3000 ;
  //Random value for 200MHz : +/- 1500ps
  localparam range_jitter_200M_30 = 1500;

        //localparam Period=10000; //For 100MHz
        //localparam Period=50000; //For 200MHz

        //sin jitter
   localparam Period_100= 10000; //10000 debe ser la mitad; //For 100MHz
   localparam Period_200=  5000;//5000 debe ser la mitad; //For 200MHz
        //localparam Period_200=  4750;//5000 debe ser la mitad; //For 200MHz

        //reg [7:0] jitter_200M=0;



//Valores de jitter
reg [7:0] jitter_5P=63;
reg [7:0] jitter_10P=125;
reg [7:0] jitter_15P=188;
reg [7:0] jitter_20P=250;
reg [7:0] jitter_25P=313;
reg [7:0] jitter_30P=375;
```

```
//Dato sin jitter para comparaciones

initial begin
	#0
	//reset=0;
	//enable=0;
	#15
	//reset=1;
	//enable=1;

	//############ First data ############
	//data_nojitter =1;  //data without jitter
	//#(Period_200 ) //normal duration
	//############ First data ############

	//Agregando jitter al 0. El '1' durará menos
		#1000
	data =1;  //data without jitter
	#(Period_200 ) //normal duration
	repeat(100) begin
	   data_nojitter =0;
	   #(Period_200)

	   data_nojitter =1;
	   #(Period_200)
	  data_nojitter =0;
	end
		#5000000 $finish;
end



////////################################## Escenario 5: Agregando jitter al 1 en ambos lados, reduciendo tiempo de
'0'################################
///Periodo con jitter de 5%


initial begin
	#0
	reset=0;
	#1500
	reset=1;
	#1000

	//############ First data ############
		#1000
	data =1;  //data without jitter
	#(Period_200+jitter_5P ) //normal duration
	//############ First data ############

	//Agregando jitter al 0. El '1' durará menos

		repeat(50) begin
		   data =0;
		   #(Period_200-jitter_5P-jitter_5P)

		   data =1;
		   #(Period_200+jitter_5P+jitter_5P)
```

```
            data =0;
        end

        #100
        repeat(50) begin
                data =0;
                #(Period_200-jitter_5P-jitter_5P)

                data =1;
                #(Period_200+jitter_5P+jitter_5P)
                data =0;
    end
        #200
        repeat(50) begin
                data =0;
                #(Period_200-jitter_5P-jitter_5P)

                data =1;
                #(Period_200+jitter_5P+jitter_5P)
                data =0;
    end
        #300
        repeat(50) begin
                data =0;
                #(Period_200-jitter_5P-jitter_5P)

                data =1;
                #(Period_200+jitter_5P+jitter_5P)
                data =0;
    end

        #400
        repeat(50) begin
                data =0;
                #(Period_200-jitter_5P-jitter_5P)

                data =1;
                #(Period_200+jitter_5P+jitter_5P)
                data =0;
    end
        repeat(50)begin
        #Period_200
        data=1;
        end
        repeat(40)begin
        #Period_200
        data=0;
        end
         #5000000 $finish;
end



/*
///Periodo con jitter de 10%
initial begin
    #0
    reset=0;
    #1500
```

```
reset=1;

#1000
//############ First data ############
data =1;  //data without jitter
#(Period_200+jitter_10P ) //normal duration
//############ First data ############

//Agregando jitter al 0. El '1' durará menos

repeat(50) begin
   data =0;
   #(Period_200-jitter_10P-jitter_10P)

   data =1;
   #(Period_200+jitter_10P+jitter_10P)
   data =0;
end
      #100
      repeat(50) begin
   data =0;
   #(Period_200-jitter_10P-jitter_10P)

   data =1;
   #(Period_200+jitter_10P+jitter_10P)
   data =0;
end
      #200
      repeat(50) begin
   data =0;
   #(Period_200-jitter_10P-jitter_10P)

   data =1;
   #(Period_200+jitter_10P+jitter_10P)
   data =0;
end
      #300
      repeat(50) begin
   data =0;
   #(Period_200-jitter_10P-jitter_10P)

   data =1;
   #(Period_200+jitter_10P+jitter_10P)
   data =0;
end
      #400
      repeat(50) begin
   data =0;
   #(Period_200-jitter_10P-jitter_10P)

   data =1;
   #(Period_200+jitter_10P+jitter_10P)
   data =0;
end

      repeat(40)begin
      #Period_200
      data=1;
      end
      repeat(40)begin
```

```
            #Period_200
            data=0;
            end
             #5000000 $finish;
end
*/


///Periodo con jitter de 15%
/*
initial begin
      #0
      reset=0;

      #1500
      reset=1;

      #1000
      //############ First data ############
      data =1;  //data without jitter
      #(Period_200+jitter_15P ) //normal duration
      //############ First data ############

      //Agregando jitter al 0. El '1' durará menos

      repeat(50) begin
         data =0;

         #(Period_200-jitter_15P-jitter_15P)

         data =1;
         #(Period_200+jitter_15P+jitter_15P)
         data =0;
      end
            #100
            repeat(50) begin
         data =0;
         #(Period_200-jitter_15P-jitter_15P)

         data =1;
         #(Period_200+jitter_15P+jitter_15P)
         data =0;
      end
            #200
            repeat(50) begin
         data =0;
         #(Period_200-jitter_15P-jitter_15P)

         data =1;
         #(Period_200+jitter_15P+jitter_15P)
         data =0;
      end

            #300
            repeat(50) begin
         data =0;
         #(Period_200-jitter_15P-jitter_15P)

         data =1;
         #(Period_200+jitter_15P+jitter_15P)
```

```
        data =0;
    end


        #400
        repeat(50) begin
        data =0;
        #(Period_200-jitter_15P-jitter_15P)
        data =1;
        #(Period_200+jitter_15P+jitter_15P)
        data =0;
    end

        repeat(40)begin
        #Period_200
        data=1;
        end
        repeat(40)begin
        #Period_200
        data=0;
        end
         #5000000 $finish;
end
*/

/////############################## Escenario 6: Agregando jitter al 0 en ambos lados, reduciendo tiempo de '1'  ############
////////#########################################################################################################################

///Periodo con jitter de 5%

/*
initial begin
    #0
    reset=0;
    #1500
    reset=1;

    #1000
    //############ First data ############
    data =1;  //data without jitter
    #(Period_200) //normal duration
    //############ First data ############

    //Agregando jitter al 0. El '1' durará menos

    repeat(50) begin
        data =0;
        #(Period_200-jitter_5P-jitter_5P)

        data =1;
        #(Period_200+jitter_5P+jitter_5P)
        data =0;
    end
#100
    repeat(50) begin
        data =0;
        #(Period_200-jitter_5P-jitter_5P)

        data =1;
        #(Period_200+jitter_5P+jitter_5P)
```

```
        data =0;
    end
#200
    repeat(50) begin
        data =0;
        #(Period_200-jitter_5P-jitter_5P)

        data =1;
        #(Period_200+jitter_5P+jitter_5P)
        data =0;
    end
#300
    repeat(50) begin
        data =0;
        #(Period_200-jitter_5P-jitter_5P)

        data =1;
        #(Period_200+jitter_5P+jitter_5P)
        data =0;
    end
#400
    repeat(50) begin
        data =0;
        #(Period_200-jitter_5P-jitter_5P)

        data =1;
        #(Period_200+jitter_5P+jitter_5P)
        data =0;
    end
          repeat(40)begin
          #Period_200
          data=1;
          end
          repeat(40)begin
          #Period_200
          data=0;
          end
           #5000000 $finish;
end

*/

/*
initial begin
    #0
    reset=0;
    enable=0;
    #1500
    reset=1;
    #1000
    enable=1;

    //############ First data ############
    data =0;  //data without jitter
    #(Period_200+jitter_10P ) //normal duration
    //############ First data ############

    //Agregando jitter al 0. El '1' durará menos

    repeat(10) begin
```

```
        data =1;
        #(Period_200-jitter_10P-jitter_10P)

        data =0;
        #(Period_200+jitter_10P+jitter_10P)
        data =1;
    end
end
*/
/*
initial begin
    #0
    reset=0;
    enable=0;
    #1500
    reset=1;
    #1000
    enable=1;

    //############ First data ############
    data =0;  //data without jitter
    #(Period_200+jitter_15P ) //normal duration
    //############ First data ############

    //Agregando jitter al 0. El '1' durará menos

    repeat(10) begin
        data =1;
        #(Period_200-jitter_15P-jitter_15P)

        data =0;
        #(Period_200+jitter_15P+jitter_15P)
        data =1;
    end
end
*/


initial begin
        forever    #(clk_p0/2) phase_0=!phase_0;
end

initial begin
        #(clk_p1);
        forever    #(clk_p0/2) phase_1=!phase_1;
end

initial begin
        #(clk_p2);
        forever    #(clk_p0/2) phase_2=!phase_2;
end

initial begin
        #(clk_p3);
        forever    #(clk_p0/2) phase_3=!phase_3;
end

initial begin
        #(clk_p4);
        forever    #(clk_p0/2) phase_4=!phase_4;
```

```
end

initial begin
        #(clk_p5);
        forever    #(clk_p0/2) phase_5=!phase_5;
end

initial begin
        #(clk_p6);
        forever    #(clk_p0/2) phase_6=!phase_6;
end

initial begin
        #(clk_p7);
        forever    #(clk_p0/2) phase_7=!phase_7;
end


function integer CeilLog2;
        input integer data;
        integer i,result;
        begin
                for(i=0; 2**i < data; i=i+1)
                        result = i + 1;
                CeilLog2 = result; //se debe usar el nombre de la funcion, que será la salida
        end
endfunction

endmodule
```

5. Bibliografía

Analog Devices. (2017, 08). *ADN2917. Continuous Rate 8.5 Gbps to 11.3 Gbps Clock and Data Recovery IC with Integrated Limiting Amp/EQ.* Retrieved 07 01, 2018, from http://www.analog.com/media/en/technical-documentation/data-sheets/ADN2917.pdf

Bidaj, K., Begueret, J., & J., D. (2017). Generation of Colored Noise Patterns with Gaussian Jitter Distribution. IEEE Trans. of inst. and meas. vol. 66, No. 10.

Businessweek, B. (2016, 02 18). *The most important apple executive you've never heard of*. (Bloomberg) Retrieved 06 18, 2018, from https://www.bloomberg.com/features/2016-johny-srouji-apple-chief-chipmaker/

Gennum. (2007). *Clock and Data Recovery Solutions*. (Gennum) Retrieved 07 01, 2018, from http://data.datasheetlib.com/pdf1/68/45/684500/gennum-corporation-gn2004s_fdfe4b10d9.pdf

Keysight. (2018, 03 03). *N1078A Optical/Electrical Clock Recovery* . (Keysight) Retrieved 07 01, 2018, from https://www.keysight.com/en/pdx-2906605-pn-N1078A/optical-electrical-clock-recovery?nid=-33198.1238480&cc=MX&lc=eng

Macom. (2015, 12). *M21011-12/M21012-12.* Retrieved 07 01, 2018, from https://cdn.macom.com/datasheets/M21011-12%20-%20V4.pdf

Merritt, R. (2008, 04 23). *DoD may push back on Apple's P.A. Semi bid*. (EETimes) Retrieved 06 18, 2018, from https://www.eetimes.com/document.asp?doc_id=1168406

Nagaria, D. (2017, 02 01). *Beginners Guide To Clock Data Recovery*. Retrieved 07 12, 18, from https://protocol-debug.com/2017/02/01/beginners-guide-to-clock-data-recovery/

Proakis, J., & Manolakis, D. (1996). The sampling theorem. . In *Digital signal processing. Principles, algorithms and applications.* (p. 29). Prentice Hall.

Qualcomm. (2012, 06 18). *Qualcomm Acquires Summit Microelectronics*. (Qualcomm) Retrieved 06 18, 2018, from https://www.qualcomm.com/news/releases/2012/06/18/qualcomm-acquires-summit-microelectronics

Rivas-Villegas, R. (2016, 08). *Design, Implementation and Verification of a Deserializer Module for a SerDes Mixed Signal System on Chip in 130 nm CMOS Technology*. Retrieved 07 01, 2018, from http://hdl.handle.net/11117/3990

6. Glosario

GPS	Globa Positioning System
PLL	Módulo Phase Locked Loop
PFD	Detector de fase
VCO	Oscilador controlado por voltaje
CDR	Rrecuperador de datos y reloj
LFSR	Módulo generador de datos pseudo aleatorios
MHz	Mega Hertz
PLL	Módulo Phase Locked Loop
BER	tasa de error de bit
HDL	Hardware Description Language
RTL	Register transfer Level
CAD	Computer Aided Design
VHDL	Very High Speed Integrated Circuit Hardware Description Language