2016-01

Reporte de formación complementaria en área de diseño de circuitos integrados digitales

Güereña-Morán, Alejandro

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

MAESTRÍA EN DISEÑO ELECTRÓNICO

Modalidad: **"Formación Complementaria en Área de Concentración"**
Área de Concentración: **"Diseño de Circuitos Integrados Digitales"**

Trabajo recepcional que para obtener el grado de

MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: Alejandro Güereña Morán

Asesores: Dr. José Luis Pizano Escalante
Dr. Luis Rizo Dominguez
Dr. Zabdiel Brito Brito

Tlaquepaque, Jalisco. Enero 2016.

MAESTRO EN INGENIERÍA (2016) ITESO

Maestría en Diseño Electrónico Tlaquepaque, Jal., México

ÁREA DE CONCENTRACIÓN: **“Diseño de Circuitos Integrados Digitales”**

AUTOR: Alejandro Güereña Morán

Ingeniero en Tecnologías Electrónicas (ITESM, Jalisco, México)

Especialista en Diseño de Circuitos Integrados (ITESO, Jalisco, México)

REVISORES: Dr. José Luis Pizano Escalante

Dr. Luis Rizo Dominguez

Dr. Zabdiel Brito Brito

NÚMERO DE PÁGINAS: iii, 120

Contenido

Introducción

En el mundo actual encontramos sistemas digitales en la mayoría de los dispositivos electrónicos que usamos, ya que permiten procesar, transmitir, recibir y almacenar señales digitales. Estas señales digitales representan información que el ser humano interpreta.

El estudio de los circuitos integrados digitales permite al alumno entender e implementar el flujo de elaboración de sistemas digitales contenidos dentro de circuito integrado. Este flujo consiste en varias etapas complejas, que van desde la especificación de un sistema digital en un documento de texto, su diseño de arquitectura, diseño RTL ("Register Transfer Level"), diseño físico, fabricación y todas las etapas de verificación, validación y pruebas, donde cada etapa es clave para poder implementar un circuito integrado digital.

En la actualidad, las empresas electrónicas aumentan la calidad de los circuitos integrados digitales, que desarrollan gracias a la investigación y mejoras en la implementación de cada una de las etapas del flujo de elaboración de un circuito integrado. Como alumno y empleado de una empresa electrónica que participa en este flujo, es clave estudiar día a día el conjunto de información, metodologías, lenguajes y herramientas involucradas en estas etapas.

Las materias cursadas para esta área fueron Diseño de Sistemas Digitales, Diseño de Circuitos Integrados Digitales, Verificación de Sistemas Digitales y Pruebas de Circuitos Integrados. Cada una de estas materias representa el estudio de una de las etapas que conforman el desarrollo de un circuito integrado digital.

1. Resumen de Proyectos Realizados

1.1. Proyecto I (Diseño de Sistemas Digitales)

1.1.1 Introducción

El proyecto realizado para la materia de Diseño de Sistemas Digitales consistió en diseñar un sistema digital que involucre un procesador MIPS32 básico con comunicación con controladores de interface RS-232 y de LCD para la tarjeta de prototipos Spartan-3E SK, con el objetivo de recibir un dato serial de la PC y procesarlo para ser desplegado en una pantalla LCD.

1.1.2 Antecedentes

En este curso se analizó los distintos elementos combinacionales y secuenciales que conforman los módulos digitales, así como su implementación a nivel RTL, con el objetivo de generar módulos como controladores, memorias y procesadores que fueron utilizados en este proyecto. También se aprendió sobre lenguajes de descripción de hardware que permiten al alumno describir y conectar módulos digitales para realizar una tarea de procesamiento de información.

1.1.3 Solución desarrollada

El diseño de sistemas digitales comienza con la implementación de la especificación que contiene la descripción funcional del sistema. La calidad de la descripción de la especificación y el análisis de este, tiene impacto sobre el diseño del sistema digital.

Para este proyecto, la especificación describe un sistema conformado por un controlador RS-232 que recibe el valor de una tecla presionada en una PC. El controlador verifica la trama serial recibida y enviar el valor de la tecla a un procesador MIPS32, cuya función será detectar qué tecla fue presionada, convertirla a mayúscula e indicar al controlador de LCD que debe ser

desplegada. El controlador LCD será utilizado para recibir el valor del procesador y desplegarlo en una pantalla LCD.

Una vez determinada la especificación, se definió la arquitectura de cada uno de los controladores y del procesador. Esto consistió en escoger e interconectar todos los elementos combinacionales y secuenciales necesarios, así como diseñar las máquinas de estados para controlar y sincronizar los distintos estados de los controladores y procesador. La arquitectura diseñada fue modular, de tal manera que se realizaron módulos separados para los dos controladores y el procesador y luego fueron interconectados. Una vez definida la arquitectura, se describió el RTL usando Verilog.

También se diseñó el programa a ejecutarse por el procesador, que tiene la función de cargar los valores transmitidos por el controlador RS-232 en registros, que son utilizados por el procesador para realizar comparaciones durante la ejecución del programa. Estas comparaciones permitirán detectar sí la pantalla LCD esta inicializada, sí existe un dato nuevo, o si el controlador LCD no está ocupado; además de realizar la lógica para detectar que una letra recibida es minúscula y convertirla en mayúscula. Una vez almacenados los valores en registros, se ejecuta una serie de pasos definidos por instrucciones para la ejecución de la aplicación.

1.1.4 Análisis de resultados

Dado que este proyecto fue un sistema modular, se implementaron módulos de los controladores de RS-232, LCD y el procesador para luego ser interconectados. Esto permitió realizar camas de pruebas o “test-benches” individuales para cada módulo y verificar su funcionamiento en simulación. Una vez verificados los módulos, estos fueron interconectados y se valida el sistema completo en simulación, donde se implementa una cama de prueba que inserte valores al controlador de RS-232, representando la tecla presionada y observando los valores de salida del controlador de LCD. La herramienta de ModelSim fue utilizada para el análisis de los resultados de verificación.

Una vez validado funcionalmente, el diseño en RTL es descargado en el FPGA y se valida a nivel usuario, usando teclado, interfaz RS-232 y LCD de la tarjeta de prototipos Spartan-3E SK.

1.1.5 Conclusiones

El diseño de sistemas digitales trae consigo el reto de definir la arquitectura óptima para la implementación de una aplicación. Se tiene que determinar cuáles son los elementos digitales adecuados que cumplan las especificaciones definidas. Esto lleva a tener retos como definir el número óptimo de bits de cada señal, el número de estados de una máquina de estados, la cantidad de registros usados, cantidad de señales de control etc… Al trabajar en una empresa que desarrolla sistema digitales, los diseñadores enfrentan día a día los retos de definir una arquitectura y escoger los elementos digitales más óptimos para implementarlos, donde se tienen en cuenta compromisos de potencia, interconexión, sincronización, frecuencia, implementación, verificación entre otros.

1.2. Proyecto II (Diseño de Circuitos Integrados Digitales)

1.2.1 Introducción

El proyecto realizado para la materia de Diseño de Circuitos Integrados Digitales consistió en la optimización de los tiempos de "setup" y "hold" de un flip flop tipo D Maestro-Esclavo, con el objetivo de demostrar su impacto en velocidad, área y consumo de potencia. También se diseñó un contador anillo módulo 8 utilizando este flip flop, del cual se analizó su desempeño.

1.2.2 Antecedentes

El análisis, diseño y prueba de los flip flops es parte fundamental del diseño de la lógica digital secuencial. Dentro del curso se aprendió los distintos parámetros de diseño de un flip flop así como las distintas arquitecturas para construir un flip flop. Estos son usados para implementar máquinas de estado, contadores, memorias, registros de corrimiento y otros circuitos secuenciales.

La optimización de los tiempos de "setup" y "hold" un flip flop es esencial para entender la relación entre los distintos tiempos que involucran un flip flop y el resto de factores (frecuencia, potencia consumida…) que impactan el desempeño de un circuito secuencial.

1.2.3 Solución desarrollada

Para optimizar los tiempos de “setup” y “hold”, se detectó los transistores que influyen en la velocidad del flip flop, mediante el aumento del doble de dimensión de cada uno de manera individual. Una vez aumentado cada flip flop y midiendo el impacto que este aumento individual tiene sobre los tiempos de “setup” y “hold” del circuito, se determinó los transistores cuyas dimensiones deben de ser modificadas para aumentar la velocidad del circuito. Al aumentar el doble del tamaño inicial (2µm a 4µm) de estos transistores, se logro aumentar de 1193 MHz a 2197 MHz la frecuencia del circuito.

En el diseño del layout del flip flop se consideró tener una simetría entre las señales de relojes (“clk” y “clk” negado), de tal forma que tengan el mismo tiempo de propagación por el circuito. También se sobrepusieron las difusiones comunes entre los transistores y se diseña el anillo contador de tal manera que la salida y entrada de cada uno de los flip flop estén lo más cerca posible, ya que al tener líneas de metal de menor longitud, se logra reducir el factor de capacitancias parásitas, que aumentan los tiempos de propagación del circuito.

El layout del contador de anillo módulo 8, consistió en poner cuatro flip flops en línea y colocar abajo del cuarto flip flop, el quinto flip flop. Luego, se insertó el resto de flip flops en línea de tal forma que la salida del último flip flop este lo más cerca posible de la entrada del primero, Colocar cuatro flip flops arriba y el resto abajo generó el compromiso de dos conexiones de metal mayores que el resto de las demás (entrada y salida del 4to-5to y 8vo-1ro par de flip flops). Sin embargo, estas conexiones son menores que tener los ocho flip flops en línea e insertar una conexión de metal de longitud mayor para conectar el primer flip flop con el último, ya que esto generaría una capacitancia significativa, provocando un tiempo de propagación alto.

1.2.4 Análisis de resultados

Los tiempos de “setup” y “hold” se redujeron al incrementar las dimensiones de los transistores identificados, pero existió un incremento en los tiempos de propagación del flip flop. Esto es un factor importante para el desempeño de un circuito secuencial, ya que aunque se reduzcan los tiempos de “setup” y “hold”, se incrementaron los tiempos de propagación, los cuales influyen significativamente en la velocidad de un circuito secuencial.

Al aumentar el tiempo de propagación de las señales de salida de cada flip flop de 1766 ps a 1937 ps del contador, y siendo esta señal, la entrada del siguiente flip flop, el desempeño del contador se ve afectado de manera considerable. Los tiempos de "setup" y "hold" son menores para los flip flops del contador optimizado, reduciendo el tiempo de setup de 1360 ps a 973 ps y de hold de 933 ps a 752 ps, pero la frecuencia máxima real disminuye debido a los mayores tiempos de propagación, pasando de 428 MHz a 423 MHz. La potencia y el área del contador optimizado son mayores que un contador no optimizado, al tener transistores con el doble de dimensión, pasando de transistores de 2µm a 4µm y aumentando la potenicia de 1600µW a 1713µW.

1.2.5 Conclusiones

La metodología aplicada logró reducir los tiempos "setup" y "hold" del flip flop, pero esto no garantizó una mejora notable en la frecuencia de operación del contador de anillo, debido al aumento de los tiempos de propagación. Esto representó una lección en el diseño de circuitos secuenciales, donde se observó el compromiso que reducir tiempos de "setup" y "hold", causan un impacto en los tiempos de propagación, de tal manera que la frecuencia del circuito es impactada, así como la potencia consumida, que es mayor en un circuito optimizado.

En el diseño de circuitos integrados siempre existe el compromiso de afectar un factor de desempeño cuando optimizamos otro factor. Por lo tanto, al diseñar debemos entender las relaciones entre los distintos factores de desempeño del circuito para obtener la mejor relación entre factores y maximizar el desempeño de este.

1.3. Proyecto II (Verificación de Sistemas Digitales)

1.3.1 Introducción

El proyecto realizado para la materia de Verificación de Sistemas Digitales consistió en implementar un plan y ambiente de verificación para la unidad despachadora de un Microprocesador de 32 bits implementada a nivel RTL. System Verilog fue usado como lenguaje de Verificación.

1.3.2 Antecedentes

La verificación pre-silicio permite probar que un sistema digital es una correcta y completa representación funcional de su especificación [Sucar-Moreno-01]. Durante el curso de verificación se estudiaron las distintas etapas para implementar la verificación de un sistema digital (planificación, implementación, simulación y debugging), así como el diseño e implementación a nivel código de un ambiente de verificación.

1.3.3 Solución desarrollada

Para la verificación de la unidad despachadora, se analizó a detalle su especificación para definir un plan de verificación. El plan de verificación definió la estrategia de verificación como de Caja Gris y usar un Modelo de Referencia para compararse con el RTL. Las áreas funcionales que se verificaron fueron la lectura, decodificación, despacho a colas, cálculo de direcciones y lógica de detención.

El diseño e implementación del ambiente de verificación consistió en definir, diseñar e implementar los elementos de modelo de referencia, interfaces, "drivers", monitores, checadores y puntos de cobertura. Todos estos elementos fueron codificados en un ambiente de verificación basado en clases, donde son instanciados como objetos en un "testbench" o clase ambiente. Dentro de esta clase ambiente, la comunicación de los distintos elementos se realizó mediante referencias a objetos y estos objetos se comunican con el RTL y Modelo de Referencia mediante interfaces virtuales.

1.3.4 Análisis de resultados

Para el análisis de resultados se desarrollaron escenarios de pruebas con el objetivo verificar la correcta y completa funcionalidad del RTL. Una clase que define y permite configurar los escenarios de prueba fue realizada e instanciada dentro de la clase ambiente, permitiendo la comunicación con los demás elementos del ambiente. Elementos aleatorios y dirigidos fueron utilizados para definir los distintos escenarios de pruebas.

1.3.5 Conclusiones

Entender la especificación a detalle es clave para definir un plan de verificación que permita checar todos los escenarios para garantizar la correcta y completa funcionalidad de un sistema.

Diseñar e implementar el ambiente de verificación mediante clases e instanciarlos dentro de una clase ambiente, permite una fácil integración y comunicación de los distintos elementos de verificación. Todo esto aumenta la calidad del ambiente de verificación, permitiéndole al verificador encontrar más errores dentro de un sistema y reducir los errores del ambiente.

2. Conclusiones

Los proyectos realizados en el área de circuitos integrados digitales representaron una grande experiencia en el diseño y verificación de sistemas digitales. En los distintos cursos se aprendió los conceptos y el flujo de las distintas etapas involucradas en la elaboración de un circuito integrado, desde su diseño RTL, diseño nivel transistor y la verificación de los cuales participó en la industria electrónica. En cada proyecto se puso en práctica los conocimientos aprendidos, desarrollando código en Verilog, System Verilog y usando herramientas como Cadence Virtuoso, ModelSim, Questa Sim, que son usadas en la industria.

Paradigmas y retos fueron afrontados durante la realización de los proyectos, contestando preguntas como ¿qué compromisos enfrentó al optimizar un parámetro de diseño?, ¿cuál arquitectura es la óptima para cumplir las especificaciones de un sistema?, ¿qué debería de verificar?, ¿cuándo he terminado de verificar?, ¿quién verifica al verificador? y otras. Estas preguntas pudieron ser contestadas con los conocimientos obtenidos durante la maestría y durante la implementación de estos proyectos, que fueron diseñados por tutores que son integrantes de la industria. Esto ha sido clave para mi desarrollo profesional dentro de la empresa electrónica INTEL, donde verifico el RTL de un procesador digital de gráficos y estos compromisos y se retos se presentan día a día, tomando decisiones para afrontarlos. La efectividad y calidad de las decisiones para afrontar estos compromisos y retos son claves para mostrar resultados y entregar un producto de alta calidad.

Enfatizo en la importancia de lidiar con los retos de la verificación de sistemas digitales en la industria, donde un ambiente y sus respectivos elementos de verificación tienen que cumplir con un alto nivel de calidad, que pueda ser implementado con los recursos disponibles, que carezca de errores en su implementación y ejecución y que permita detectar errores en pre-silicio, con el objetivo de otorgar al cliente un circuito integrado carente de errores y en los tiempos planeados.

Apéndices

A. PROYECTO FINAL DE DISEÑO DE SISTEMAS DIGITALES

Proyecto Final
Diseño de Sistemas Digitales

Ing. Alejandro Güereña Morán *(Autor)*
Especialidad en Diseño de CI.
ITESO
Tlaquepaque, México
md679705@iteso.mx

Ing. Marcos I. Bolaños Valerio *(Autor)*
Especialidad en Diseño de CI.
ITESO
Tlaquepaque, México

***Resumen*— El diseño de un sistema digital que involucre un procesador MIPS32 básico es presentado. Los controladores de interface RS-232 y de LCD para la tarjeta de prototipos Spartan-3E SK se comunican con el procesador para realizar una aplicación, donde se recibe un dato serial de la PC y se procesa para ser desplegado en una pantalla LCD.**

El diseño es implementado con el lenguaje de descripción de hardware Verilog. Las simulaciones y su implementación en FPGA se realizaron con ModelSim y Xilinx-ISE respectivamente.

I. Introducción

Los sistemas digitales consisten en la interconexión de módulos de circuitos o celdas digitales que realizan una tarea específica de procesamiento de información [1]. En este proyecto se implementa dicha interconexión de celdas digitales implementadas durante el curso. Estas celdas son los controladores de RS-232, LCD y un procesador MIPS32 básico.

A. Abreviaciones y Acrónimos:

Las siguientes abreviaciones y acrónimos son usados
en este documento.

ASCII	American Standard Code for Information Interchange
FPGA	Field Programmable Gate Array
LCD	Liquid Crystal Display
MIPS32	Microprocessor without Interlocked Pipeline Stages

Tabla 1. Abreviaciones y Acrónimos

II. Funcionamiento

La aplicación a realizar con el sistema digital implementado, consiste en recibir una letra enviada por el usuario desde la Hyperterminal de una PC. El controlador de interface RS-232 debe de recibir los datos seriales provenientes de la PC, detectando los bits de inicio, datos, paridad y paro, para realizar el análisis de la trama recibida vía RS-232. Una vez analizada la trama serial, se debe de detectar que no exista error en ella para enviar el dato al procesador MIPS32. El registro30 (R30), del procesador representa la entrada del procesador, que es donde se recibe el dato de 8 bits proveniente del controlador de RS-232. Para indicar que se recibió un nuevo dato para desplegarse en LCD, el dato enviado al registro de entrada, contiene la información en un bit para indicar que un dato nuevo fue recibido, además del dato de 8 bits a procesar.

El procesador MIPS32 ejecutará un programa que espera que una letra sea recibida. Una vez recibida, el programa checará si la letra es minúscula o mayúscula. Si la letra es minúscula, esta será convertida por el procesador a mayúscula y enviada al controlador de LCD para que sea desplegada. Si la letra recibida es mayúscula, se envía directamente al controlador de LCD, sin realizar alguna conversión. Para enviar

el dato al controlador de LCD, se utiliza el registro 31 (R31) del procesador, donde se envía el dato en ASCII a desplegar en LCD. Para indicar que un nuevo dato debe de desplegarse en el LCD, el dato enviado por el registro de salida, contiene información en un bit para indicar que un dato nuevo debe de ser desplegado, además del dato de 8 bits a desplegar.

El controlador de LCD tiene la función de inicializar y configurar correctamente la pantalla LCD de la tarjeta de prototipos Spartan-3E SK. También realiza el despliegue de un dato proveniente del procesador, una vez que se indique que existe un dato nuevo a desplegar.

El diagrama a bloques general del sistema digital a que representa su funcionamiento y la interconexión entre los bloques es:

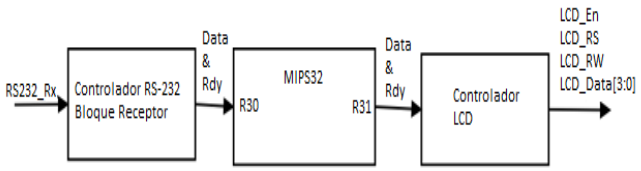


Figura 1. Diagrama a bloques del sitema.

A. Código RTL del Top Module:

- Ver Anexo de códigos RTL del proyecto:
  - proyectoFinal.v –página #10.

III. Funcionamiento del controlador RS-232

Se implementa en la tarjeta de prototipos Spartan-3E SK, un controlador de comunicación asíncrona (UART) que pueda recibir información de manera serial, usando el protocolo RS232. Los parámetros del enlace de comunicación son: 9600 bps, 8 bits de datos, paridad par, 1 bit de paro y sin control de flujo.

A. Arquitectura:

La arquitectura del controlador RS-232 consiste en la recepción de una trama de entrada. Las salidas de esta arquitectura, indican si un dato nuevo ha sido recibido (DataRdy), el dato recibido (Data_Rx) y si existe errores, tanto en la recepción de datos (Err_Rx) o en el bit de paridad (Err_Par).

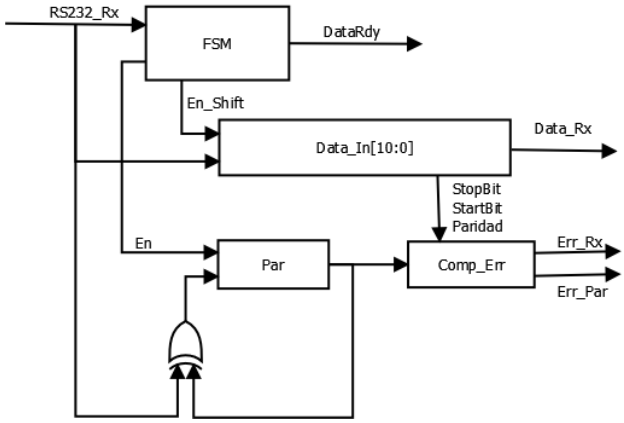


Figura 2. Arquitectura del controlador RS-232.

B. FSM:

La máquina de estados del controlador RS-232 define el funcionamiento de la recepción de un dato serial.

En el Receptor, primero se espera recibir el Start bit. Una vez que se detecta el Start Bit, que es un 0, se pasa al estado de bitStart. En este estado, se espera a contar 8 ciclos de reloj, debido a que se está sobre muestreando 16 veces más rápido que la frecuencia del baud rate. Esto con el objetivo de posicionarse justamente a la mitad de la recepción de los datos y almacenar el dato de una manera segura.

Después de esto, se reciben los 8 bits de datos, recibiendo primero el bit menos significativo. Una vez que se reciben los datos, se recibe el bit de paridad y el stop bit. Una vez recibida la trama, se calcula la paridad de los datos recibidos y se compara con la paridad recibida. Si no coinciden, existe un error, por lo tanto, se activan los leds del FPGA y la señal de DataRdy no se activa, indicando que no existe un nuevo dato recibido. Caso contrario, si la paridad es correcta, DataRdy sí se activa, indicando que el dato recibido es correcto y debe de ser procesado.

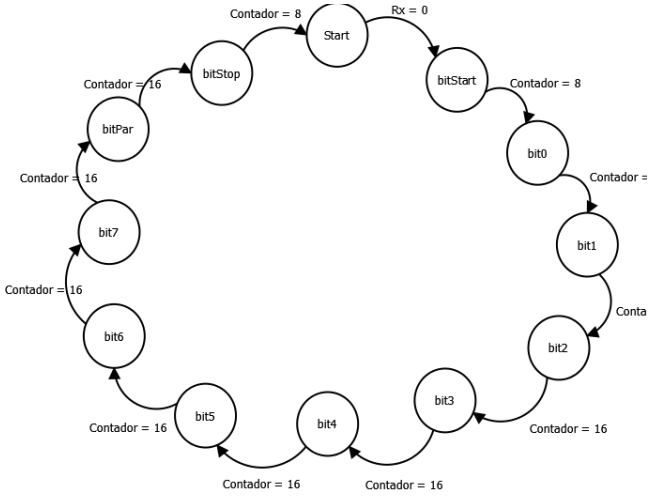


Figura 3. FSM del controlador RS-232.

C. Código RTL:

- Ver Anexo de códigos RTL del proyecto:
  - RS232_Rx.v –página #11.

IV. Funcionamiento del controlador del LCD

Se implementa en la tarjeta de prototipos Spartan-3E SK, un controlador de LCD. Este controlador envía las instrucciones y datos necesarios para la correcta inicialización y configuración de la pantalla y para el despliegue de letras en la pantalla. Esta pantalla LCD de la tarjeta usada, consta de una interface de 4 bits para datos, por lo tanto, el controlador debe de ser diseñado para este formato de 4 bits.

A. Arquitectura:

La arquitectura del controlador de LCD consiste en la inicialización de la pantalla, cumpliendo los tiempos mínimos de espera para cada instrucción. Los comandos para la inicialización de la pantalla, representan el bloque de inicialización del LCD de la arquitectura. Los comandos para la configuración de la pantalla, representan el bloque de configuración del LCD de la arquitectura. Los tiempos mínimos que determinan el valor de un contador, cntDelay, se determinan en estos dos bloques mencionados. Estos dos bloques interactúan entre sí, para inicializar la pantalla primero, y luego configurarla, enviando las instrucciones a los bloques de Interaccion LCD y Pulse Enable LCD. Una vez terminada la inicialización y configuración, las señales de InitFinish y CfgFinish se activan, indicando que el LCD puede recibir datos para desplegar. Los bloques de Interaccion LCD y Pulse Enable LCD, envían las instrucciones o datos al LCD, enviando primero 4 bits, y luego los otros 4, cumpliendo con los tiempos mínimos del toggle de la señal de Enable.

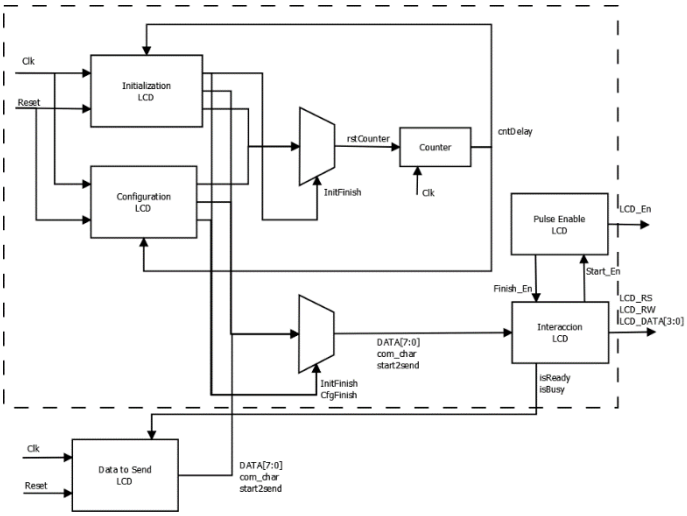


Figura 4. Arquitectura del controlador del LCD.

B. FSM:

La máquina de estados del controlador de LCD define el funcionamiento de la inicialización y configuración de la pantalla, así como el envío de datos a desplegar en ella.

El bloque de inicialización del LCD, manda las instrucciones necesarias al LCD para inicializara, con el objetivo de tenerla lista para enviar las instrucciones de configuración. Los tiempos mínimos por cada instrucción son cumplidos.

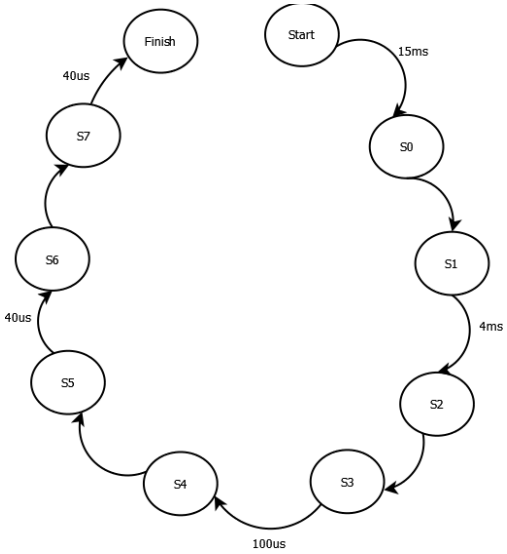


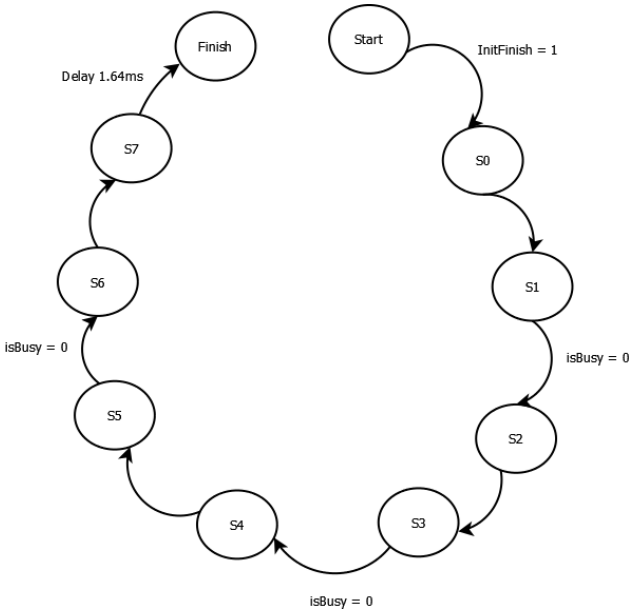
Figura 5. FSM del controlador del LCD. Bloque Initialization LCD.

Tabla 2. Valores del FSM del controlador del LCD. Bloque Initialization LCD.

El bloque de configuración del LCD, manda las instrucciones necesarias al LCD para configurarlo. Se indica que la pantalla será manejada con instrucciones o datos de 4 bits. Los tiempos mínimos por cada instrucción son cumplidos.

Estado	rstCounter	Start2send	Data	InitFinish
Start	1	0	8'h00	0
S0	0	0	8'h00	0
S1	1	1	8'h30	0
S2	0	0	8'h30	0
S3	1	1	8'h30	0
S4	0	0	8'h30	0
S5	1	1	8'h30	0
S6	0	0	8'h30	0
S7	1	1	8'h20	0
Finish	0	0	8'h00	1

Tabla 3. Valores del FSM del controlador del LCD. Bloque Configuration LCD.



Estado	Data	Start2send	rstCounter	CfgFinish
Start	8´h00	0	0	0
S0	FunctionSet	1	0	0
S1	FunctionSet	0	0	0
S2	EntryMode	1	0	0
S3	EntryMode	0	0	0
S4	Display_On/Off	1	0	0
S5	Display_On/Off	0	0	0
S6	ClrDisplay	1	1	0
S7	ClrDisplay	0	0	0
Finish	8'h00	0	0	1

Figura 6. FSM del controlador del LCD. Bloque Configuration LCD.

La máquina de control implementada para el toggle de la señal Enable, tiene como función activar el bit de la señal de Enable, esperar el tiempo mínimo y luego desactivar este bit para cada 4 bits enviados al LCD.

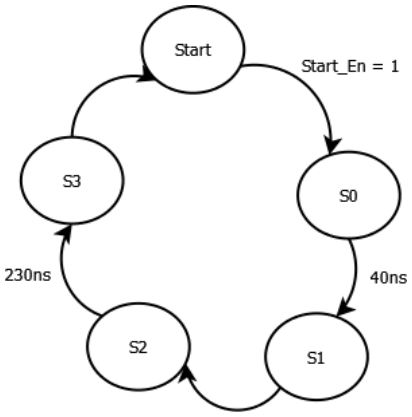


Figura 7. FSM del controlador del LCD. Bloque Pulse Enable LCD.

Estado	LCD_En	rstCounter
Start	0	1
S0	0	0
S1	1	1
S2	1	0
S3	0	0

Tabla 4. Valores del FSM del controlador del LCD. Bloque Pulse Enable LCD.

La máquina de control implementada para enviar un dato de 8 bits a la pantalla, tiene como función enviar primero los 4 bits más significativos, activando el toggle de la señal de Enable. Una vez que termina el toggle de la señal de Enable, se envían los 4 bits menos significativos, activando de nuevo el toggle de la señal de Enable.

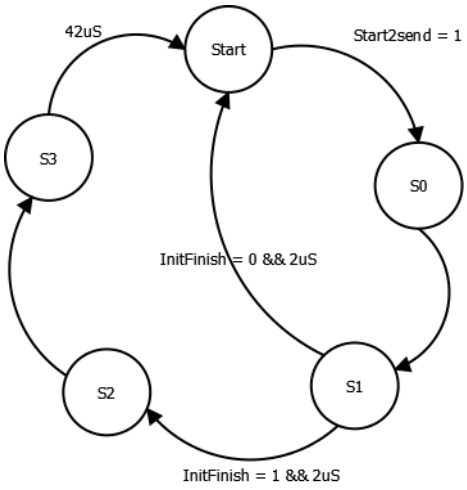


Figura 8. FSM del controlador del LCD. Bloque Interaccion LCD.

Estado	Data	rstCounter	Start_En	isBusy
Start	MSB[7:4]	0	0	0
S0	MSB[7:4]	1	1	1
S1	MSB[7:4]	0	0	1
S2	LSB[3:0]	1	1	1
S3	LSB[3:0]	0	0	1

Tabla 5. Valores del FSM del controlador del LCD. Bloque Interaccion LCD.

C. Código RTL:

- Ver Anexo de códigos RTL del proyecto:
  - InitLCD.v –página # 14.
  - ConfigLCD.v –página # 15.
  - LCD_Driver_4bits.v –página # 17.
  - send_char_command.v –página # 18.
  - counter.v –página # 20.

v. Funcionamiento del procesador

El microprocesador MIPS32 básico multi-ciclo realiza todas las fases de procesamiento de una instrucción en varios ciclos. Consta de los siguientes tipos de instrucciones, que definen los ciclos de procesamiento de una instrucción.

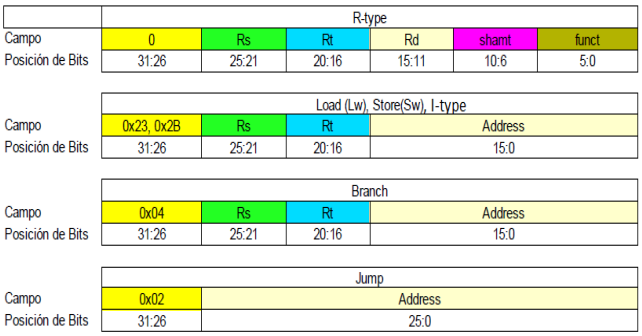


Figura 9. Tipos de instrucciones del MIPS32.

La instrucción ADDI, de tipo I, conserva el siguiente formato:

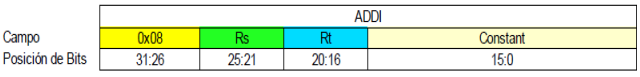


Figura 10. Instrucción ADDI del MIPS32.

Para poder ejecutar este tipo de instrucciones, se necesitan dos grandes bloques, la ruta de datos y la ruta de control. En este caso, la ruta de control es una maquina de estados, la cual en su versión original sólo contiene 10 estados, y en el "peor" de los casos, o la instrucción mas tardada, tiene que pasar por 5 estados, lo cual, para este caso, se traduce en 5 ciclos de reloj, y en el mejor de los casos, la instrucción más rápida en ejecutarse lo hace en sólo 3 ciclos de reloj.

A. Arquitectura:

La arquitectura del MIPS32 básico multi-ciclo consta de distintos registros, una memoria, ALU, lógica de control y lógica combinacional para controlar el funcionamiento del microprocesador. Los registros presentes en la arquitectura, son el contador de programa (PC), el registro de instrucciones (inst_reg), el registro de datos de memoria (Mem_Data_Reg), el registro de salida de la ALU (ALU_Out), registros A y B (A y B) y un banco de registros que está compuesto de 32 registros de 32 bits.

Consta de una unidad de control, representada por una máquina de estados, que controla las diferentes señales que van a los registros y a la lógica combinacional de la arquitectura. Esta unidad de control, recibe como entradas los bits 31 a 26 del registro de instrucciones para checar el tipo de instrucción a ejecutar, y también los bits del 0 al 5 del mismo registro, para detectar cuando se ejecuta una operación de tipo shift, ya que unos de sus operandos dependen del campo shamt, donde uno de los campos de registro fuente no se usa.

También se tiene una lógica de control para la ALU, que decodifica el valor de la función proveniente de la máquina de estados, para especificar a la ALU la correcta operación a ejecutara según la instrucción procesada.

B. FSM:

La máquina de estados implementada para el MIMPS32. Esta máquina varía respecto a la vista en el curso, debido a que para acceder a un dato de la memoria, se necesita un ciclo extra de reloj, ya que la memoria es síncrona. También se agregan estados para la instrucción ADDI, cuya estructura de su

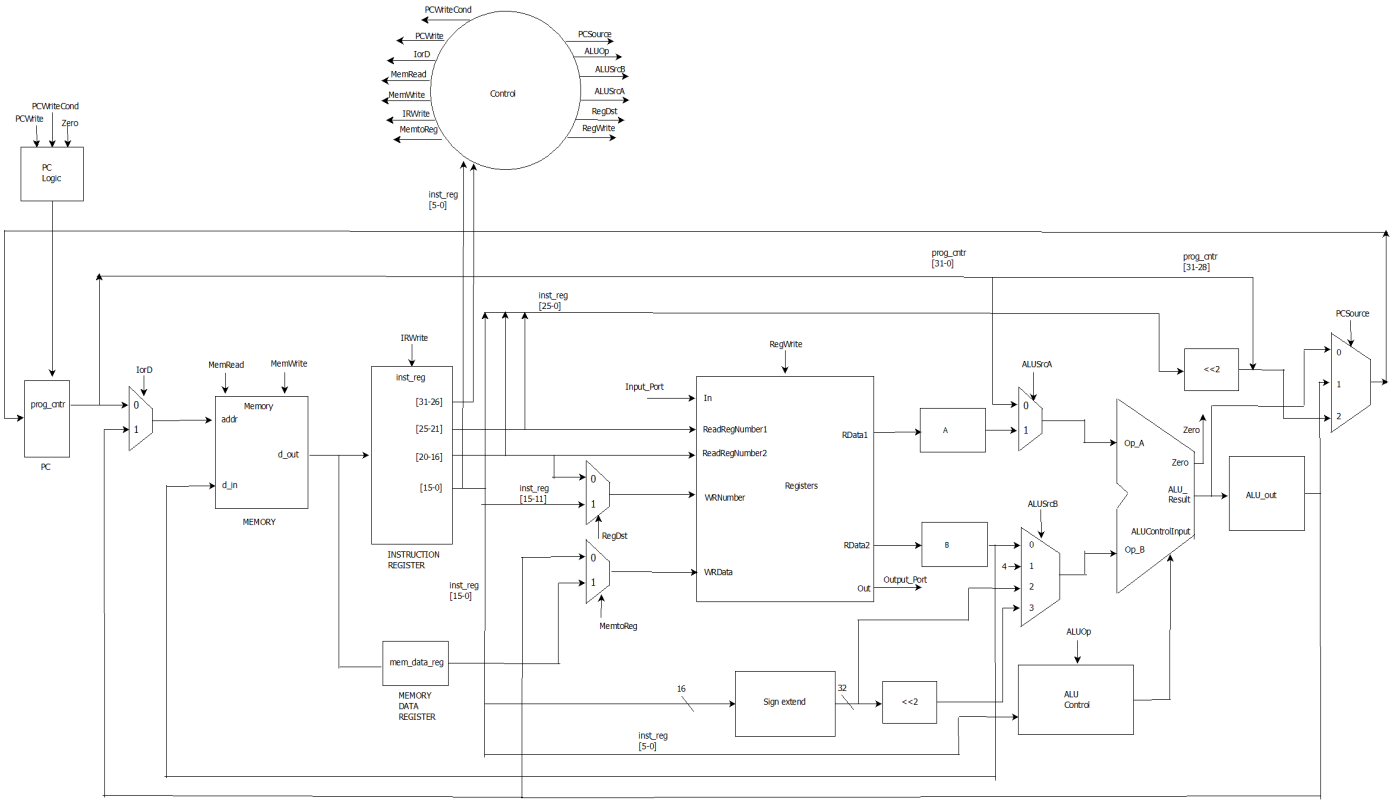


Figura 11. Arquitectura de MIPS32

frame es de tipo I, como los son las instrucciones LOAD/STORE.
La máquina de estados tiene sus transiciones de acuerdo al valor de Op, que representan los bits 31 a 26 del registro de instrucciones (inst_reg). Existe un estado mealy, S6, donde el valor de una salida es función tanto del estado como de la entrada, donde el valor de los bits 5 a 0 del registro de instrucciones (func), determinarán el valor de la señal ALUScrB, para identificar las instrucciones que realizan una operación de corrimiento de las demás de tipo R. Las transiciones entre estados donde no se indica alguna señal de entrada, cambian con la señal de reloj.

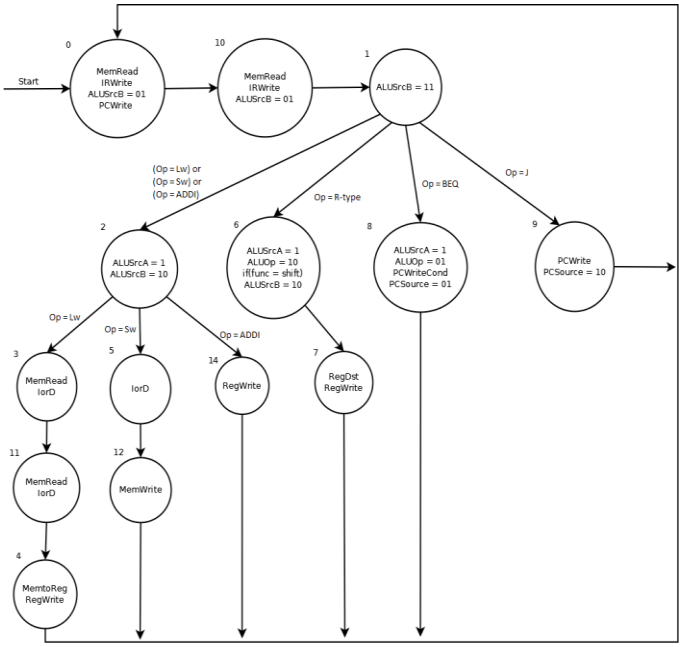


Figura 12. FSM del Control del MIPS32.

C. Programa a ejecutarse:

El programa a ejecutarse, tiene la función de cargar las constantes en registros que se usarán para realizar comparaciones durante la ejecución del programa. Estas comparaciones permitirán detectar sí la pantalla LCD esta inicializada, sí existe un dato nuevo recibido, si el controlador LCD no está ocupado, además de de realizar la lógica para detectar que una letra recibida es minúscula. Una vez almacenadas las constantes en registros, se ejecuta una serie de pasos definidos por instrucciones para la ejecución de la aplicación.

1.- Revisar que isReady sea 1, indicando que la LCD ya se inicializo.
2.- Revisar que isBusy_LCD, sea cero, indicando que el LCD no está ocupado.
3.- Si data_rdy_uart es 1, quedarse en esa misma localidad.
4.- Si data_rdy_uart es cero, quedarse en esa misma localidad.
5.- Hacer una and lógica para quedarse sólo con el dato de la UART.
6.- Comparar el dato con 0x61, si es menor, se manda a LCD, si no, hacer paso siguiente
7.- Comparar el dato con 0x7A, si es mayor, se manda a LCD, si no, hacer paso siguiente
8.- Al dato recibido, restarle 0x20, para convertir la letra en mayúscula
9.- Mandar a LCD
10.- Repetir desde el paso 2

- Ver Anexo de códigos almacenador en memoria del MIPS32 del proyecto:
  - página # 8.

D. Código de MIPS 32:

- Ver Anexo de códigos RTL del proyecto:
  - TopLevel_PCMIPS.v –página #21.
  - Datapath_PCMIPS.v –página #22.
  - Control_PCMIPS.v –página #28.

VI. Síntesis en FPGA

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,350	9,312	25%	
Number of 4 input LUTs	5,710	9,312	61%	
Number of occupied Slices	3,442	4,656	73%	
Number of Slices containing only related logic	3,442	3,442	100%	
Number of Slices containing unrelated logic	0	3,442	0%	
Total Number of 4 input LUTs	5,763	9,312	61%	
Number used as logic	5,710			
Number used as a route-thru	53			
Number of bonded IOBs	12	232	5%	
Number of BUFGMUXs	2	24	8%	
Average Fanout of Non-Clock Nets	4.63			

Figura 13. Recursos Utilizados del FPGA.

Se utilizaron un total de 2350 Sliced Flip Flops y 5710 LUTs de 4 entradas.

```
Clock to Setup on destination clock clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clk            |   14.508|         |         |         |
---------------+---------+---------+---------+---------+
```

Figura 14. Resultados del sintetizador sobre señal de Reloj.

La máxima frecuencia de operación fue dada por el sintetizador en ns. Esta frecuencia de operación o mínimo período es de 68.927MHz – 14.508ns respectivamente.

VII. Conclusiones

Este proyecto representa la integración de varios elementos vistos durante el curso, el cual permiten realizar un sistema digital. Se realizó una aplicación en específico, según el programa que insertamos dentro de la memoria del procesador, aunque se pueden realizar varios programas para distintas aplicaciones, el detalle es especificar el programa adecuado para realizarlo.

Durante el curso se aprendieron los distintos elementos que conforman los sistemas digitales y como programar estos en un lenguaje de descripción de hardware. Esto permite describir el hardware que queremos implementar y que una herramienta de síntesis, pueda implementar físicamente el circuito que se realizo en un FPGA. Esto representa una enorme ventaja, ya que se pueden realizar cambios al RTL del hardware e implementarse de manera rápida en un FPGA, con la desventaja de que el diseño queda limitado a los recursos y estructura del FPGA.

Con este proyecto, se comprobó físicamente el funcionamiento del procesador MIPS realizado durante el curso, que utiliza una memoria síncrona, lo cual provocó la modificación de la máquina de estados vista en el curso estados para que pudiera funcionar correctamente.

VIII. Referencias

[1] Mariano Aguirre, “Conceptos de Diseño Estructurado,” Notas de Curso de Diseño de Sistemas Digitales. ITESO, México

ANEXOS:

Código almacenado en memoria del MIPS32:

Etiquetas	Address		Instrucción	Operandos	Hexadecimal
	Dec	**Hex**			
	0	0	ADDI	R0, R5, 0x0061	20
	1	1			05
	2	2			00
	3	3			61
	4	4	ADDI	R0, R6, 0x007B	20
	5	5			06
	6	6			00
	7	7			7B
	8	8	ADDI	R0, R7, 0x0001	20
	9	9			07
	10	A			00
	11	B			01
	12	C	SLL	R7, R8, 5	00
	13	D			E0
	14	E			41
	15	F			40
	16	10	SLL	R8, R9, 4	01
	17	11			00
	18	12			49
	19	13			00
	20	14	SLL	R9, R3, 20	01
	21	15			20
	22	16			1D
	23	17			00
	24	18	SLL	R3, R2, 1	00
	25	19			60
	26	1A			10
	27	1B			40
	28	1C	SLL	R2, R1, 1	00
	29	1D			40
	30	1E			08
	31	1F			40
	32	20	ADDI	R0, R4, 0x00FF	20
	33	21			04
	34	22			00
	35	23			FF
Regresa	36	24	AND	R30, R3, R10	03
	37	25			C3

	38	26			50
	39	27			24
	40	28	BEQ	R10, R0, Regresa	11
	41	29			40
	42	2A			FF
	43	2B			FE
Espera	44	2C	AND	R30,R2, R10	03
	45	2D			C2
	46	2E			50
	47	2F			24
	48	30	BEQ	R10, R2, Espera	11
	49	31			42
	50	32			FF
	51	33			FE
Espera2	52	34	AND	R30, R1, R10	03
	53	35			C1
	54	36			50
	55	37			24
	56	38	BEQ	R10, R1, Espera2	11
	57	39			41
	58	3A			FF
	59	3B			FE
Espera3	60	3C	AND	R30, R1, R10	03
	61	3D			C1
	62	3E			50
	63	3F			24
	64	40	BEQ	R10, R0, Espera3	11
	65	41			40
	66	42			FF
	67	43			FE
	68	44	AND	R30, R4, R29	03
	69	45			C4
	70	46			E8
	71	47			24
	72	48	Set on Less	R29, R5, R10	03
	73	49			A5
	74	4A			50
	75	4B			2A
	76	4C	BEQ	R10, R7, Manda	11
	77	4D			47
	78	4E			00
	79	4F			03
	80	50	Set On Less	R29, R6, R10	03
	81	51			A6
	82	52			50

	83	53			2A
	84	54	BEQ	R10, R0, Manda	11
	85	55			40
	86	56			00
	87	57			01
	88	58	SUB	R29, R8, R29	03
	89	59			A8
	90	5A			E8
	91	5B			22
Manda	92	5C	OR	R29, R9, R31	03
	93	5D			A9
	94	5E			F8
	95	5F			25
	96	60	AND	R29, R4, R31	03
	97	61			A4
	98	62			F8
	99	63			24
	100	64	J	Espera	08
	101	65			00
	102	66			00
	103	67			0B

Códigos RTL del proyercto final:

TOP MODULE:

proyectoFinal.v

```
//module proyectoFinal(clk, rst, LCD_RS, LCD_RW, LCD_E, LCD_DATA, SW, ROT_CENTER, LED);
module proyectoFinal(clk, rst, LCD_RS, LCD_RW, LCD_E, LCD_DATA, par_err, err_rx, RS232_Rx);
//module proyectoFinal(clk, rst, LCD_RS, LCD_RW, LCD_E, LCD_DATA, RS232_Rx, LED);
	input clk, rst;
	//input ROT_CENTER;
	input RS232_Rx;
	//input [3:0] SW;
	output LCD_RS, LCD_RW, LCD_E;
	output [3:0] LCD_DATA;
	//output [7:0] LED;
	output par_err, err_rx;

	wire data_rdy_uart;
	wire [7:0] data_out_uart;
	wire isBusy_LCD;
	wire isReady_LCD;
	wire [31:0] Port_Out_MIPS;
//	wire Rx_ready;
	//assign LED[7] = ~ROT_CENTER;
	//assign LED[6:0] = Port_Out_MIPS[6:0];

	/*UART_Rx_Driver UART_Rx(
		.Clk(clk),
```

```
		.Arst(rst),
		.Rx_Serial_Data(RS232_Rx),
		.Rx_Ready(Rx_ready),
		.Rx_Data_Received(data_rdy_uart),
		.Rx_Data(data_out_uart),
		.Error_Leds(LED)
	);*/

	RS232_Rx UART_RX(
		.RS232_Rx(RS232_Rx),
		.data_rdy(data_rdy_uart),
		.clk(clk),
		.rst(rst),
		.datap_out(data_out_uart),
		.par_err(par_err),
		.err_rx(err_rx)
	);

	TopLevel_PCMIPS MIPS(
		.clk(clk),
		.rst(rst),
		.Port_in({data_rdy_uart,isBusy_LCD,isReady_LCD,21'H000000, data_out_uart}),
		//.Port_in({~ROT_CENTER,isBusy_LCD,isReady_LCD,21'H000000, 1'b0, 1'b1, SW[3],
2'b00, SW[2:0]}),
		.Port_out(Port_Out_MIPS)
	);

	LCD_Driver_4bits LCD(
		.LCD_RS(LCD_RS),
		.LCD_RW(LCD_RW),
		.LCD_E(LCD_E),
		.LCD_Data(LCD_DATA),
		.Data_in(Port_Out_MIPS[7:0]),
		.com_char(1'b1),
		.isBusy(isBusy_LCD),
		.start2send(Port_Out_MIPS[9]),
		.reset(rst),
		.clock(clk),
		.isReady(isReady_LCD)
	);

endmodule
```

CONTROLADOR RS-232

RS232_Rx.v

```
module RS232_Rx(RS232_Rx, data_rdy, clk, rst, datap_out, par_err, err_rx);
	input RS232_Rx, clk, rst;
	output [7:0] datap_out;
	output reg data_rdy;
	output err_rx;
	output par_err;

	parameter bitStart                          = 4'b0001;
	parameter bitPar                                    = 4'b0010;
	parameter bitStop                           = 4'b0011;
	parameter Start                                     = 4'b0100;
	parameter waitFinish                        = 4'b0111;
	parameter bit0                                      = 4'b1000;
	parameter bit1                                      = 4'b1001;
	parameter bit2                                      = 4'b1010;
	parameter bit3                                      = 4'b1011;
	parameter bit4                                      = 4'b1100;
	parameter bit5                                      = 4'b1101;
	parameter bit6                                      = 4'b1110;
	parameter bit7                                      = 4'b1111;
```

```
	parameter BaudRate_115200_x16 = 8'hA3;

	reg clk_int;
	reg [4:0] NoCycles;
	reg [7:0] countBaudRate;
	reg [10:0] datap_out_reg;
	reg rstNoCycles;
	reg shift;
	reg par_calc;
	reg enable_par_calc;
	reg rst_par;

	wire rstNC;

	reg [3:0] NXT_STATE;
	reg [3:0] CUR_STATE;

	always@(posedge clk, posedge rst) begin
		if(rst) begin
			countBaudRate <= 8'b0;
			clk_int <= 1'b0;
		end
		else begin
			countBaudRate <= countBaudRate + 1'b1;
			if(countBaudRate == BaudRate_115200_x16) begin
				clk_int <= ~clk_int;
				countBaudRate <= 8'b0;
			end
		end
	end

	always@(posedge clk_int, posedge rst)begin
		if(rst) datap_out_reg <= 11'h400;
		else if(shift) datap_out_reg <= {RS232_Rx, datap_out_reg[10:1]};
	end

	always@(posedge clk_int, posedge rst) begin
		if(rst) CUR_STATE <= Start;
		else CUR_STATE <= NXT_STATE;
	end

	always@(posedge clk_int) begin
		if(rstNC) NoCycles <= 5'b0;
		else NoCycles <= NoCycles + 1'b1;
	end

	always@(posedge clk_int, posedge rst_par) begin     // Cálculo de paridad
		if(rst_par) par_calc <= 1'b0;
		else if(enable_par_calc) par_calc <= par_calc ^ RS232_Rx;
	end


	always@(*) begin
		NXT_STATE = CUR_STATE;
		data_rdy = 1'b0;
		rstNoCycles = 1'b0;
		enable_par_calc = 1'b0;
		shift = 1'b0;
		rst_par = 1'b0;
		case (CUR_STATE)
			Start: begin
				data_rdy = 1'b1;
				if(RS232_Rx == 0) begin NXT_STATE = bitStart; rstNoCycles = 1'b1;
rst_par = 1'b1; end
			end

			bitStart: begin
				if(NoCycles == 5'b00111) begin NXT_STATE = bit0; rstNoCycles =
1'b1; shift = 1'b1; end
```

```
                        end

                        bit0: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bit1; rstNoCycles =
1'b1; shift = 1'b1; enable_par_calc = 1'b1; end
                        end

                        bit1: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bit2; rstNoCycles =
1'b1; shift = 1'b1; enable_par_calc = 1'b1; end
                        end

                        bit2: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bit3; rstNoCycles =
1'b1; shift = 1'b1; enable_par_calc = 1'b1; end
                        end

                        bit3: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bit4; rstNoCycles =
1'b1; shift = 1'b1; enable_par_calc = 1'b1; end
                        end

                        bit4: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bit5; rstNoCycles =
1'b1; shift = 1'b1; enable_par_calc = 1'b1; end
                        end

                        bit5: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bit6; rstNoCycles =
1'b1; shift = 1'b1; enable_par_calc = 1'b1; end
                        end

                        bit6: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bit7; rstNoCycles =
1'b1; shift = 1'b1; enable_par_calc = 1'b1; end
                        end

                        bit7: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bitPar; rstNoCycles =
1'b1; shift = 1'b1; enable_par_calc = 1'b1; end
                        end

                        bitPar: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = bitStop; rstNoCycles =
1'b1; shift = 1'b1; end
                        end

                        bitStop: begin
                                if(NoCycles == 5'b01111) begin NXT_STATE = Start; rstNoCycles =
1'b1; shift = 1'b1; end
                        end

                        /*waitFinish: begin
                                if(NoCycles == 5'b00011) begin NXT_STATE = Start; data_rdy = 1'b1;
end
                        end*/

                        default: begin
                                NXT_STATE = Start;
                        end
                endcase
        end

        assign datap_out = datap_out_reg[8:1];
        assign err_rx = datap_out_reg[0] | ~datap_out_reg[10];
        assign par_err = datap_out_reg[9] ^ par_calc;
        assign rstNC = rst | rstNoCycles;

endmodule
```

CONTROLADOR DEL LCD:

InitLCD.v

```
module InitLCD(clk, rst, data, start2send, InitFinish, com_char,timeOut,rstCounter);
	input clk,rst;
	input [19:0] timeOut;
	output reg [7:0] data;
	output com_char;
	output reg start2send, InitFinish, rstCounter;

	parameter Start	= 4'b0000;
	parameter S0   = 4'b0001;
	parameter S1   = 4'b0010;
	parameter S2   = 4'b0011;
	parameter S3   = 4'b0100;
	parameter S4   = 4'b0101;
	parameter S5   = 4'b0110;
	parameter S6   = 4'b0111;
	parameter S7   = 4'b1000;
	parameter S8   = 4'b1001;
	parameter finish = 4'b1010;

	reg[3:0] CUR_STATE;
	reg[3:0] NXT_STATE;

	always@(posedge clk, posedge rst) begin
		if(rst) CUR_STATE <= Start;
		else CUR_STATE <= NXT_STATE;
	end

	always@(*) begin
		NXT_STATE = CUR_STATE;
		InitFinish = 1'b0;
		rstCounter = 1'b0;
		start2send = 1'b0;
		case (CUR_STATE)
			Start: begin
				rstCounter = 1'b1; data = 8'b0;
				NXT_STATE = S0;
			end

			S0: begin
				data = 8'b0;
				if(timeOut == 750000) NXT_STATE = S1;
			end

			S1: begin
				rstCounter = 1'b1; data = 8'h30; start2send = 1'b1;
				NXT_STATE = S2;
			end

			S2: begin
				data = 8'h30;
				if(timeOut == 205100) NXT_STATE = S3;
			end

			S3: begin
				rstCounter = 1'b1; data = 8'h30; start2send = 1'b1;
				NXT_STATE = S4;
			end

			S4: begin
				data = 8'h30;
				if(timeOut == 5100) NXT_STATE = S5;
			end

			S5: begin
```

```
                        rstCounter = 1'b1; data = 8'h30; start2send = 1'b1;
                        NXT_STATE = S6;
                end

                S6: begin
                        data = 8'h30;
                        if(timeOut == 2100) NXT_STATE = S7;
                end

                S7: begin
                        rstCounter = 1'b1; data = 8'h20; start2send = 1'b1;
                        NXT_STATE = S8;
                end

                S8: begin
                        data = 8'h20;
                        if(timeOut == 2100) NXT_STATE = finish;
                end

                finish:  begin
                        data = 8'b0; InitFinish = 1'b1;
                        NXT_STATE = finish;
                end

                default: begin
                        rstCounter = 1'b1; data = 8'b0;
                        NXT_STATE = Start;
                end
        endcase
end

assign com_char =1'b0;

endmodule
```

ConfigLCD.v

```
module ConfigLCD(clk, rst, data, start2send, CfgFinish,
com_char,isBusy,InitFinish,timeOut,rstCounter);
        input clk,rst,isBusy,InitFinish;
        input [17:0] timeOut;
        output reg [7:0] data;
        output reg start2send, CfgFinish, rstCounter;
        output com_char;

        parameter Function_Set = 8'h28;
        parameter Entry_Mode_Set = 8'h06;
        parameter Display_on_off = 8'h0F;
        parameter Clear_Display = 8'h01;

        parameter Start       = 4'b0000;
        parameter S0   = 4'b0001;
        parameter S1   = 4'b0010;
        parameter S2   = 4'b0011;
        parameter S3   = 4'b0100;
        parameter S4   = 4'b0101;
        parameter S5   = 4'b0110;
        parameter S6   = 4'b0111;
        parameter S7   = 4'b1000;
        parameter finish = 4'b1001;

        reg[3:0] CUR_STATE;
        reg[3:0] NXT_STATE;

        always@(posedge clk, posedge rst) begin
                if(rst) CUR_STATE <= Start;
                else CUR_STATE <= NXT_STATE;
        end
```

```
	always@(*) begin
		NXT_STATE = CUR_STATE;
		rstCounter = 1'b0;
		CfgFinish = 1'b0;
		start2send = 1'b0;
		case (CUR_STATE)
			Start: begin
				data = 8'b0;
				if(InitFinish) NXT_STATE = S0;
			end

			S0: begin
				data = Function_Set; start2send = 1'b1;
				NXT_STATE = S1;
			end

			S1: begin
				data = Function_Set; start2send = 1'b0;
				if(~isBusy) NXT_STATE = S2;
			end

			S2: begin
				data = Entry_Mode_Set; start2send = 1'b1;
				NXT_STATE = S3;
			end

			S3: begin
				data = Entry_Mode_Set;
				if(~isBusy) NXT_STATE = S4;
			end

			S4: begin
				data = Display_on_off; start2send = 1'b1;
				NXT_STATE = S5;
			end

			S5: begin
				data = Display_on_off;
				if(~isBusy) NXT_STATE = S6;
			end

			S6: begin
				rstCounter = 1'b1; data = Clear_Display; start2send = 1'b1;
				NXT_STATE = S7;
			end

			S7: begin
				data = Clear_Display;
				if(timeOut == 82100) NXT_STATE = finish;
			end

			finish:  begin
				data = 8'b0; CfgFinish = 1'b1;
				NXT_STATE = finish;
			end

			default: begin
				data = 8'b0;
				NXT_STATE = Start;
			end
		endcase
	end

	assign com_char = 1'b0;

endmodule
```

LCD_Driver_4bits.v

```
module LCD_Driver_4bits(LCD_RS, LCD_RW, LCD_E, LCD_Data, Data_in, com_char, isBusy, start2send,
reset, clock, isReady);
	input com_char, start2send,reset,clock;
	input [7:0] Data_in;
	output LCD_RS, LCD_RW, LCD_E, isBusy, isReady;
	output [3:0] LCD_Data;

	wire rst_counterBig_CfgLCD;
	wire rst_counterBig_InitLCD;
	wire rst_counterSmall_int;
	wire [19:0] counterBig_int;
	wire [3:0] counterSmall_int;
	wire [7:0] data_in_CfgLCD;
	wire [7:0] data_in_InitLCD;
	wire start2send_CfgLCD;
	wire start2send_InitLCD;
	wire com_char_CfgLCD;
	wire com_char_InitLCD;
	wire [1:0] sel_data;
	wire sel_rst_counterBig;
	wire isBusy_int;
	wire start_En;

	reg rst_counterBig_int;
	reg com_char_int;
	reg start2send_int;
	reg [7:0] data_in_int;

	always@(*) begin
		case(sel_rst_counterBig)
			1'b0: rst_counterBig_int = rst_counterBig_InitLCD;
			1'b1: rst_counterBig_int = rst_counterBig_CfgLCD;
			default: rst_counterBig_int = 1'b0;
		endcase
	end

	always@(*) begin
		case(sel_data)
			2'b00: begin data_in_int = data_in_InitLCD; com_char_int =
com_char_InitLCD; start2send_int = start2send_InitLCD; end
			2'b01: begin data_in_int = data_in_CfgLCD; com_char_int = com_char_CfgLCD;
start2send_int = start2send_CfgLCD; end
			2'b10: begin data_in_int = 8'h00; com_char_int = 1'b1; start2send_int =
1'b0; end
			2'b11: begin data_in_int = Data_in; com_char_int = com_char; start2send_int
= start2send; end
			default: begin data_in_int = 8'h00; com_char_int = 1'b1; start2send_int =
1'b0; end
		endcase
	end

	ConfigLCD Config(
		.clk(clock),
		.rst(reset),
		.data(data_in_CfgLCD),
		.start2send(start2send_CfgLCD),
		.CfgFinish(sel_data[1]),
		.com_char(com_char_CfgLCD),
		.isBusy(isBusy_int),
		.InitFinish(sel_data[0]),
		.timeOut(counterBig_int[17:0]),
		.rstCounter(rst_counterBig_CfgLCD)
	);

	InitLCD Initialization(
		.clk(clock),
		.rst(reset),
		.data(data_in_InitLCD),
		.start2send(start2send_InitLCD),
		.InitFinish(sel_data[0]),
```

```
		.com_char(com_char_InitLCD),
		.timeOut(counterBig_int),
		.rstCounter(rst_counterBig_InitLCD)
	);

	send_Char_Command toSend(
		.data_in(data_in_int),
		.com_char(com_char_int),
		.isBusy(isBusy_int),
		.start2send(start2send_int),
		.clk(clock),
		.rst(reset),
		.LCD_RS(LCD_RS),
		.LCD_RW(LCD_RW),
		.LCD_DATA(LCD_Data),
		.InitFinish(sel_data[0]),
		.Start_En(start_En)
	);

	Pulse_E_LCD PulseForEnLCD(
		.str(start_En),
		.LCD_E(LCD_E),
		.timeOut(counterSmall_int),
		.rstCounter(rst_counterSmall_int),
		.clk(clock),
		.rst(reset)
	);

	counterSmall contadorSmall(
		.clk(clock),
		.rst(rst_counterSmall_int),
		.count(counterSmall_int)
	);

	counterBig contadorBig(
		.clk(clock),
		.rst(rst_counterBig_int),
		.count(counterBig_int)
	);

	assign sel_rst_counterBig = sel_data[0];
	assign isReady = sel_data[0] & sel_data[1];
	assign isBusy = isBusy_int;
endmodule
```

send_char_command.v

```
module send_Char_Command(data_in, com_char, isBusy, start2send, clk, rst, LCD_RS, LCD_RW,
LCD_DATA,InitFinish,Start_En);
	input [7:0] data_in;
	input com_char, start2send, clk, rst, InitFinish;
	output reg isBusy, Start_En;
	output LCD_RS, LCD_RW;
	output reg [3:0] LCD_DATA;

	parameter Start = 3'b000;
	parameter S0 = 3'b001;
	parameter S1 = 3'b010;
	parameter S2 = 3'b011;
	parameter S3 = 3'b100;

	reg[2:0] CUR_STATE;
	reg[2:0] NXT_STATE;

	reg [11:0] count;
	reg rst_cnt;
```

```
	always@(posedge clk, posedge rst) begin
		if(rst) CUR_STATE <= Start;
		else CUR_STATE <= NXT_STATE;
	end

	always@(posedge clk, posedge rst_cnt) begin
		if(rst_cnt) count <= 12'h000;
		else count <= count + 1'b1;
	end

	always@(*) begin
		isBusy = 1'b1;
		Start_En = 1'b0;
		rst_cnt = 1'b0;
		NXT_STATE = CUR_STATE;
		case(CUR_STATE)
			Start: begin
				isBusy = 1'b0;
				LCD_DATA = data_in[7:4];
				if(start2send) NXT_STATE = S0;
			end

			S0: begin
				LCD_DATA = data_in[7:4];
				Start_En = 1'b1;
				rst_cnt = 1'b1;
				NXT_STATE = S1;
			end

			S1: begin
				LCD_DATA = data_in[7:4];
				if(count == 100) begin
					if(InitFinish) NXT_STATE = S2;
					else NXT_STATE = Start;
				end
			end

			S2: begin
				LCD_DATA = data_in[3:0];
				Start_En = 1'b1;
				rst_cnt = 1'b1;
				NXT_STATE = S3;
			end

			S3: begin
				LCD_DATA = data_in[3:0];
				if(count == 2100) NXT_STATE = Start;
			end

			default: begin
				NXT_STATE = Start;
				LCD_DATA = 4'b0;
			end
		endcase
	end

	assign LCD_RW = 1'b0;
	assign LCD_RS = com_char;

endmodule

module Pulse_E_LCD(str, LCD_E, timeOut, rstCounter, clk, rst);
	input str, clk, rst;
	input [3:0] timeOut;
	output reg rstCounter, LCD_E;

	parameter Start	= 3'b000;
	parameter S0	= 3'b001;
	parameter S1	= 3'b010;
	parameter S2	= 3'b011;
	parameter S3	= 3'b100;
```

```
        reg[2:0] CUR_STATE;
        reg[2:0] NXT_STATE;

        always@(posedge clk, posedge rst) begin
                if(rst) CUR_STATE <= Start;
                else CUR_STATE <= NXT_STATE;
        end

        always@(*) begin
                NXT_STATE = CUR_STATE;
                LCD_E = 1'b0;
                rstCounter = 1'b0;
                case (CUR_STATE)
                        Start: begin
                                rstCounter = 1'b1;
                                if(str) NXT_STATE = S0;
                        end

                        S0: begin
                                if(timeOut == 2) NXT_STATE = S1;
                        end

                        S1: begin
                                LCD_E = 1'b1; rstCounter = 1'b1;
                                NXT_STATE = S2;
                        end

                        S2: begin
                                LCD_E = 1'b1;
                                if(timeOut == 12) NXT_STATE = S3;
                        end

                        S3: begin
                                NXT_STATE = Start;
                        end

                        default: begin
                                NXT_STATE = Start;
                        end
                endcase
        end
endmodule
```

counter.v

```
module counterBig(clk, rst, count);
        input clk, rst;
        output reg [19:0] count;

        always@(posedge clk, posedge rst) begin
                if(rst) count <= 20'b0;
                else count <= count + 1;
        end
endmodule

module counterSmall(clk, rst, count);
        input clk, rst;
        output reg [3:0] count;

        always@(posedge clk, posedge rst) begin
                if(rst) count <= 4'b0;
                else count <= count + 1;
        end
endmodule
```

MIPS32:

TopLevel_PCMIPS.v

```
module TopLevel_PCMIPS(clk, rst, Port_in, Port_out);
	input clk, rst;
	input [31:0] Port_in;
	output [31:0] Port_out;

	wire IorD, MemRead, MemWrite, IRWrite, RegDst, RegWrite, ALUSrcA, MemtoReg, PCWriteCond,
PCWrite;
	wire [1:0] ALUSrcB, ALUOp, PCSource;
	wire [5:0] OP, func;

	Datapath_PCMIPS Datapath(
		.IorD(IorD),
		.MemRead(MemRead),
		.MemWrite(MemWrite),
		.IRWrite(IRWrite),
		.RegDst(RegDst),
		.RegWrite(RegWrite),
		.ALUSrcA(ALUSrcA),
		.ALUSrcB(ALUSrcB),
		.MemtoReg(MemtoReg),
		.ALUOp(ALUOp),
		.PCSource(PCSource),
		.PCWriteCond(PCWriteCond),
		.PCWrite(PCWrite),
		.OP(OP),
		.func(func),
		.Port_in(Port_in),
		.Port_out(Port_out),
		.clk(clk),
		.rst(rst)
	);

	Control_PCMIPS Control(
		.IorD(IorD),
		.MemRead(MemRead),
		.MemWrite(MemWrite),
		.IRWrite(IRWrite),
		.RegDst(RegDst),
		.RegWrite(RegWrite),
		.ALUSrcA(ALUSrcA),
		.ALUSrcB(ALUSrcB),
		.MemtoReg(MemtoReg),
		.ALUOp(ALUOp),
		.PCSource(PCSource),
		.PCWriteCond(PCWriteCond),
		.PCWrite(PCWrite),
		.OP(OP),
		.func(func),
		.clk(clk),
		.rst(rst)
	);

endmodule
```

Datapath_PCMIPS.v

```
module Datapath_PCMIPS(IorD, MemRead, MemWrite, IRWrite, RegDst, RegWrite, ALUSrcA, ALUSrcB,
MemtoReg, ALUOp, PCSource, PCWriteCond, PCWrite, OP, func, Port_in, Port_out, clk, rst);
	input clk, rst;
```

```
	input IorD, MemRead, MemWrite, IRWrite, RegDst, RegWrite, ALUSrcA, MemtoReg, PCWriteCond,
PCWrite;
	input [1:0] ALUSrcB, ALUOp, PCSource;
	output [5:0] OP, func;
	output [31:0] Port_out;
	input [31:0] Port_in;

	reg [31:0] inst_reg;

	wire MemWrite, MemRead;
	reg [31:0] mem_Addrs;		// Falta ajustar los no. de bits
	reg [31:0] mem_Addrs_reg;
	wire [31:0] mem_data;
	reg [7:0] mem_array [0:127];

	reg [31:0] Mem_Data_Reg;

	reg [31:0] Prog_Cntr;
	reg [31:0] Prog_Cntr_Wire;
	wire load_Prog_Cntr;
	reg [27:0] ShfLft2_IR;

	reg [4:0] WR_Reg;
	reg [31:0] WR_Data_Reg;
	reg [31:0] RD_Data_Reg1, RD_Data_Reg2;

	reg [31:0] A, B;
	reg [31:0] Op_A, Op_B, ALU_Result;
	reg Zero;

	wire [15:0] in_sign_ext;
	reg [31:0] out_sign_ext, ShfLft2;
	reg [31:0] ALUOut;
	reg [3:0] ALU_Ctrl;

	reg [31:0] Register [0:31];

	/*************************************************************/
	/*				Sign Extend
	*/
	/*************************************************************/
	assign in_sign_ext = inst_reg[15:0];

	always@(in_sign_ext) begin
		if(in_sign_ext[15]) out_sign_ext = {16'hFFFF,in_sign_ext};
		else out_sign_ext = {16'h0000,in_sign_ext};
	end

	/*************************************************************/
	/*				Registers
	*/
	/*************************************************************/
	always@(posedge clk, posedge rst) begin
		if(rst) begin
			Register[0] <= 32'H00000000;
			Register[1] <= 32'H00000000;
			Register[2] <= 32'H00000000;
			Register[3] <= 32'H00000000;
			Register[4] <= 32'H00000000;
			Register[5] <= 32'H00000000;
			Register[6] <= 32'H00000000;
			Register[7] <= 32'H00000000;
			Register[8] <= 32'H00000000;
			Register[9] <= 32'H00000000;
			Register[10] <= 32'H00000000;
			Register[11] <= 32'H00000000;
			Register[12] <= 32'H00000000;
			Register[13] <= 32'H00000000;
			Register[14] <= 32'H00000000;
```

```
                Register[15] <= 32'H00000000;
                Register[16] <= 32'H00000000;
                Register[17] <= 32'H00000000;
                Register[18] <= 32'H00000000;
                Register[19] <= 32'H00000000;
                Register[20] <= 32'H00000000;
                Register[21] <= 32'H00000000;
                Register[22] <= 32'H00000000;
                Register[23] <= 32'H00000000;
                Register[24] <= 32'H00000000;
                Register[25] <= 32'H00000000;
                Register[26] <= 32'H00000000;
                Register[27] <= 32'H00000000;
                Register[28] <= 32'H00000000;
                Register[29] <= 32'H00000000;
                Register[30] <= 32'H00000000;
                Register[31] <= 32'H00000000;
        end
        else if(RegWrite)
                Register[WR_Reg] <= WR_Data_Reg;
end

assign Port_out = Register[31];

always@(inst_reg) begin
        if(inst_reg[25:21]==30) RD_Data_Reg1 = Port_in;
        else if(inst_reg[25:21]) RD_Data_Reg1 = Register[inst_reg[25:21]];
        else RD_Data_Reg1 = 32'H00000000;
end

always@(inst_reg) begin
        if(inst_reg[20:16]==30) RD_Data_Reg2 = Port_in;
        else if(inst_reg[20:16]) RD_Data_Reg2 = Register[inst_reg[20:16]];
        else RD_Data_Reg2 = 32'H00000000;
end


/*************************************************************/
/*                           Shift Left 2
*/
/*************************************************************/
always@(out_sign_ext) begin
        ShfLft2 = {out_sign_ext[29:0],2'b00};
end

always@(inst_reg) begin
        ShfLft2_IR = {inst_reg[23:0],2'b00};
end

/*************************************************************/
/*                           Program Counter
        */
/*************************************************************/
always@(posedge clk, posedge rst) begin
        if(rst) Prog_Cntr <= 32'H00000000;
        else if(load_Prog_Cntr) Prog_Cntr <= Prog_Cntr_Wire;
end

assign load_Prog_Cntr = (Zero & PCWriteCond) | PCWrite;

/*************************************************************/
/*                           Instruction Register
        */
/*************************************************************/
always@(posedge clk, posedge rst) begin
        if(rst) inst_reg <= 32'H00000000;
        else if(IRWrite) inst_reg <= mem_data;
end

assign OP = inst_reg[31:26];
assign func = inst_reg[5:0];
```

```
/************************************************************/
/*                          Data Register
*/
/************************************************************/
always@(posedge clk, posedge rst) begin
        if(rst) Mem_Data_Reg <= 32'H00000000;
        else Mem_Data_Reg <= mem_data;
end

/************************************************************/
/*                          ALUOut Register
        */
/************************************************************/
always@(posedge clk, posedge rst) begin
        if(rst) ALUOut <= 32'H00000000;
        else ALUOut <= ALU_Result;
end

/************************************************************/
/*                                  A Register
*/
/************************************************************/
always@(posedge clk, posedge rst) begin
        if(rst) A <= 32'H00000000;
        else A <= RD_Data_Reg1;
end

/************************************************************/
/*                                  B Register
*/
/************************************************************/
always@(posedge clk, posedge rst) begin
        if(rst) B <= 32'H00000000;
        else B <= RD_Data_Reg2;
end

/************************************************************/
/*                                  ALU
*/
/************************************************************/
parameter R = 2'b10;
parameter M = 2'b00;
parameter BEQ = 2'b01;

parameter ADD = 6'H20;
parameter SUB = 6'H22;
parameter AND = 6'H24;
parameter OR  = 6'H25;
parameter XOR = 6'H26;
parameter NOR = 6'H27;
parameter SLT = 6'H2A;                                  // Set on less than
parameter SRL = 6'H02;
parameter SLL = 6'H00;

parameter AND_ALU = 4'H0;
parameter  OR_ALU = 4'H1;
parameter ADD_ALU = 4'H2;
parameter NOR_ALU = 4'H3;
parameter XOR_ALU = 4'H4;
parameter SUB_ALU = 4'H6;
parameter SLT_ALU = 4'H7;
parameter SRL_ALU = 4'H8;
parameter SLL_ALU = 4'H9;

always@(*) begin
        casex({ALUOp, inst_reg[5:0]})
                {M, 6'HXX}:                     ALU_Ctrl = ADD_ALU;
                {BEQ, 6'HXX}:  ALU_Ctrl = SUB_ALU;
                {R, ADD}:                       ALU_Ctrl = ADD_ALU;
```

```
                        {R, SUB}:                       ALU_Ctrl = SUB_ALU;
                        {R, AND}:                       ALU_Ctrl = AND_ALU;
                        {R,  OR}:                       ALU_Ctrl = OR_ALU;
                        {R, SLT}:                       ALU_Ctrl = SLT_ALU;
                        {R, NOR}:                       ALU_Ctrl = NOR_ALU;
                        {R, XOR}:                       ALU_Ctrl = XOR_ALU;
                        {R, SLL}:                       ALU_Ctrl = SLL_ALU;
                        {R, SRL}:                       ALU_Ctrl = SRL_ALU;
                        default: ALU_Ctrl = 4'b1111;
                endcase
        end

        always@(*) begin
                case(ALU_Ctrl)
                        AND_ALU: ALU_Result = Op_A & Op_B;
                         OR_ALU: ALU_Result = Op_A | Op_B;
                        ADD_ALU: ALU_Result = Op_A + Op_B;
                        SUB_ALU: ALU_Result = Op_A - Op_B;
                        SLT_ALU: ALU_Result = (Op_A < Op_B) ? 1'b1 : 1'b0;  // "set on less than"
que pex con esto??
                        NOR_ALU: ALU_Result = ~(Op_A | Op_B);
                        XOR_ALU: ALU_Result = Op_A ^ Op_B;
                        SLL_ALU:       ALU_Result = Op_A << Op_B[10:6];
                        SRL_ALU:       ALU_Result = Op_A >> Op_B[10:6];
                        default: ALU_Result = 32'h00000000;
                endcase
        end

        always@(*)begin
                if(ALU_Result) Zero = 1'b0;
                else Zero = 1'b1;
        end

        /*************************************************************/
        /*                                  Multiplexores
        */
        /*************************************************************/
        always@(*)begin
                case (RegDst)
                        1'b0: WR_Reg = inst_reg[20:16];
                        1'b1: WR_Reg = inst_reg[15:11];
                        default: WR_Reg = 5'H00;
                endcase
        end

        always@(*)begin
                case (MemtoReg)
                        1'b0: WR_Data_Reg = ALUOut;
                        1'b1: WR_Data_Reg = Mem_Data_Reg;
                        default: WR_Data_Reg = 32'H00000000;
                endcase
        end

        always@(*)begin
                case (ALUSrcA)
                        1'b0: Op_A = Prog_Cntr;
                        1'b1: Op_A = A;
                        default: Op_A = 32'H00000000;
                endcase
        end

        always@(*)begin
                case (IorD)
                        1'b0: mem_Addrs = Prog_Cntr;
                        1'b1: mem_Addrs = ALUOut;
                        default: mem_Addrs = 32'H00000000;
                endcase
        end

        always@(*) begin
                case (ALUSrcB)
```

```
                2'b00: Op_B = B;
                2'b01: Op_B = 32'H00000004;
                2'b10: Op_B = out_sign_ext;
                2'b11: Op_B = ShfLft2;
                default: Op_B = 32'H00000000;
        endcase
end

always@(*) begin
        case (PCSource)
                2'b00: Prog_Cntr_Wire = ALU_Result;
                2'b01: Prog_Cntr_Wire = ALUOut;
                2'b10: Prog_Cntr_Wire = {Prog_Cntr[31:28],ShfLft2_IR};
                default: Prog_Cntr_Wire = 32'H00000000;
        endcase
end

/*************************************************************/
/*                                  Memoria
*/
/*************************************************************/
always@(posedge clk, posedge rst) begin
        if(rst) mem_Addrs_reg <= 32'H00000000;
        else mem_Addrs_reg <= mem_Addrs;
end

always@(posedge clk, posedge rst) begin
        if(rst) begin
                mem_array[0] <= 8'h20;
                mem_array[1] <= 8'h05;
                mem_array[2] <= 8'h00;
                mem_array[3] <= 8'h61;
                mem_array[4] <= 8'h20;
                mem_array[5] <= 8'h06;
                mem_array[6] <= 8'h00;
                mem_array[7] <= 8'h7B;
                mem_array[8] <= 8'h20;
                mem_array[9] <= 8'h07;
                mem_array[10] <= 8'h00;
                mem_array[11] <= 8'h01;
                mem_array[12] <= 8'h00;
                mem_array[13] <= 8'hE0;
                mem_array[14] <= 8'h41;
                mem_array[15] <= 8'h40;
                mem_array[16] <= 8'h01;
                mem_array[17] <= 8'h00;
                mem_array[18] <= 8'h49;
                mem_array[19] <= 8'h00;
                mem_array[20] <= 8'h01;
                mem_array[21] <= 8'h20;
                mem_array[22] <= 8'h1D;
                mem_array[23] <= 8'h00;
                mem_array[24] <= 8'h00;
                mem_array[25] <= 8'h60;
                mem_array[26] <= 8'h10;
                mem_array[27] <= 8'h40;
                mem_array[28] <= 8'h00;
                mem_array[29] <= 8'h40;
                mem_array[30] <= 8'h08;
                mem_array[31] <= 8'h40;
                mem_array[32] <= 8'h20;
                mem_array[33] <= 8'h04;
                mem_array[34] <= 8'h00;
                mem_array[35] <= 8'hFF;
                mem_array[36] <= 8'h03;
                mem_array[37] <= 8'hC3;
                mem_array[38] <= 8'h50;
                mem_array[39] <= 8'h24;
                mem_array[40] <= 8'h11;
```

```
mem_array[41] <= 8'h40;
mem_array[42] <= 8'hFF;
mem_array[43] <= 8'hFE;
mem_array[44] <= 8'h03;
mem_array[45] <= 8'hC2;
mem_array[46] <= 8'h50;
mem_array[47] <= 8'h24;
mem_array[48] <= 8'h11;
mem_array[49] <= 8'h42;
mem_array[50] <= 8'hFF;
mem_array[51] <= 8'hFE;
mem_array[52] <= 8'h03;
mem_array[53] <= 8'hC1;
mem_array[54] <= 8'h50;
mem_array[55] <= 8'h24;
mem_array[56] <= 8'h11;
mem_array[57] <= 8'h41;
mem_array[58] <= 8'hFF;
mem_array[59] <= 8'hFE;
mem_array[60] <= 8'h03;
mem_array[61] <= 8'hC1;
mem_array[62] <= 8'h50;
mem_array[63] <= 8'h24;
mem_array[64] <= 8'h11;
mem_array[65] <= 8'h40;
mem_array[66] <= 8'hFF;
mem_array[67] <= 8'hFE;
mem_array[68] <= 8'h03;
mem_array[69] <= 8'hC4;
mem_array[70] <= 8'hE8;
mem_array[71] <= 8'h24;
mem_array[72] <= 8'h03;
mem_array[73] <= 8'hA5;
mem_array[74] <= 8'h50;
mem_array[75] <= 8'h2A;
mem_array[76] <= 8'h11;
mem_array[77] <= 8'h47;
mem_array[78] <= 8'h00;
mem_array[79] <= 8'h03;
mem_array[80] <= 8'h03;
mem_array[81] <= 8'hA6;
mem_array[82] <= 8'h50;
mem_array[83] <= 8'h2A;
mem_array[84] <= 8'h11;
mem_array[85] <= 8'h40;
mem_array[86] <= 8'h00;
mem_array[87] <= 8'h01;
mem_array[88] <= 8'h03;
mem_array[89] <= 8'hA8;
mem_array[90] <= 8'hE8;
mem_array[91] <= 8'h22;
mem_array[92] <= 8'h03;
mem_array[93] <= 8'hA9;
mem_array[94] <= 8'hF8;
mem_array[95] <= 8'h25;
mem_array[96] <= 8'h03;
mem_array[97] <= 8'hA4;
mem_array[98] <= 8'hF8;
mem_array[99] <= 8'h24;
mem_array[100] <= 8'h08;
mem_array[101] <= 8'h00;
mem_array[102] <= 8'h00;
mem_array[103] <= 8'h0B;
mem_array[104] <= 8'h00;
mem_array[105] <= 8'h00;
mem_array[106] <= 8'h00;
mem_array[107] <= 8'h00;
mem_array[108] <= 8'h00;
mem_array[109] <= 8'h00;
mem_array[110] <= 8'h00;
mem_array[111] <= 8'h00;
```

```
                        mem_array[112] <= 8'h00;
                        mem_array[113] <= 8'h00;
                        mem_array[114] <= 8'h00;
                        mem_array[115] <= 8'h00;
                        mem_array[116] <= 8'h00;
                        mem_array[117] <= 8'h00;
                        mem_array[118] <= 8'h00;
                        mem_array[119] <= 8'h00;
                        mem_array[120] <= 8'h00;
                        mem_array[121] <= 8'h00;
                        mem_array[122] <= 8'h00;
                        mem_array[123] <= 8'h00;
                        mem_array[124] <= 8'h00;
                        mem_array[125] <= 8'h00;
                        mem_array[126] <= 8'h00;
                        mem_array[127] <= 8'h00;
                end
                else if(MemWrite) begin
                        mem_array[mem_Addrs_reg[6:0]] <= B[31:24];
                        mem_array[mem_Addrs_reg[6:0]+1] <= B[23:16];
                        mem_array[mem_Addrs_reg[6:0]+2] <= B[15:8];
                        mem_array[mem_Addrs_reg[6:0]+3] <= B[7:0];
                end
        end

        assign mem_data[31:24] = mem_array[mem_Addrs_reg[6:0]];
        assign mem_data[23:16] = mem_array[mem_Addrs_reg[6:0] + 1];
        assign mem_data[15:8] = mem_array[mem_Addrs_reg[6:0] + 2];
        assign mem_data[7:0] = mem_array[mem_Addrs_reg[6:0] + 3];

endmodule
```

Control_PCMIPS.v

```
module Control_PCMIPS(IorD, MemRead, MemWrite, IRWrite, RegDst, RegWrite, ALUSrcA, ALUSrcB,
MemtoReg, ALUOp, PCSource, PCWriteCond, PCWrite, OP, func, clk, rst);
        input clk, rst;
        input [5:0] OP;
        input [5:0] func;

        output reg IorD, MemRead, MemWrite, IRWrite, RegDst, RegWrite, ALUSrcA, MemtoReg,
PCWriteCond, PCWrite;
        output reg [1:0] ALUSrcB, ALUOp, PCSource;

        reg [3:0] NXT_STATE;
        reg [3:0] CUR_STATE;

        parameter S0 = 4'H0;
        parameter S1 = 4'H1;
        parameter S2 = 4'H2;
        parameter S3 = 4'H3;
        parameter S4 = 4'H4;
        parameter S5 = 4'H5;
        parameter S6 = 4'H6;
        parameter S7 = 4'H7;
        parameter S8 = 4'H8;
        parameter S9 = 4'H9;
        parameter SA = 4'HA;
        parameter SB = 4'HB;
        parameter SC = 4'HC;
        parameter SD = 4'HD;

        parameter LW = 6'H23;
        parameter SW = 6'H2B;
        parameter ADDI = 6'H08;
        parameter R = 6'H00;
```

```
parameter BEQ = 6'H04;
parameter J = 6'H02;

always@(posedge clk, posedge rst)begin
	if (rst) CUR_STATE <= S0;
	else CUR_STATE <= NXT_STATE;
end

always@(*) begin
	IorD = 1'b0;
	MemRead = 1'b0;
	MemWrite = 1'b0;
	IRWrite = 1'b0;
	RegDst = 1'b0;
	RegWrite = 1'b0;
	ALUSrcA = 1'b0;
	MemtoReg = 1'b0;
	PCWriteCond = 1'b0;
	PCWrite = 1'b0;
	ALUSrcB = 2'b00;
	ALUOp = 2'b00;
	PCSource = 2'b00;
	NXT_STATE = CUR_STATE;
	case(CUR_STATE)
		S0: begin
			MemRead = 1'b1;
			IRWrite = 1'b1;
			ALUSrcB = 2'b01;
			PCWrite = 1'b1;
			NXT_STATE = SA;
		end

		S1: begin
			ALUSrcB = 2'b11;
			case(OP)
				LW: NXT_STATE = S2;
				SW: NXT_STATE = S2;
				ADDI: NXT_STATE = S2;
				R: NXT_STATE = S6;
				BEQ: NXT_STATE = S8;
				J: NXT_STATE = S9;
			endcase
		end

		S2: begin
			ALUSrcA = 1'b1;
			ALUSrcB = 2'b10;
			case(OP)
				LW: NXT_STATE = S3;
				SW: NXT_STATE = S5;
				ADDI: NXT_STATE = SD;
			endcase
		end

		S3: begin
			MemRead = 1'b1;
			IorD = 1'b1;
			NXT_STATE = SB;
		end

		S4: begin
			RegWrite = 1'b1;
			MemtoReg = 1'b1;
			NXT_STATE = S0;
		end

		S5: begin
			IorD = 1'b1;
			NXT_STATE = SC;
		end
```

```
                        S6: begin
                                ALUSrcA = 1'b1;
                                ALUOp = 2'b10;
                                if(func == 6'H02 || func == 6'H00) ALUSrcB = 2'b10;
                                NXT_STATE = S7;
                        end

                        S7: begin
                                RegDst = 1'b1;
                                RegWrite = 1'b1;
                                NXT_STATE = S0;
                        end

                        S8: begin
                                ALUSrcA = 1'b1;
                                ALUOp = 2'b01;
                                PCWriteCond = 1'b1;
                                PCSource = 2'b01;
                                NXT_STATE = S0;
                        end

                        S9: begin
                                PCWrite = 1'b1;
                                PCSource = 2'b10;
                                NXT_STATE = S0;
                        end

                        SA: begin
                                MemRead = 1'b1;
                                IRWrite = 1'b1;
                                ALUSrcB = 2'b01;
                                NXT_STATE = S1;
                        end

                        SB: begin
                                MemRead = 1'b1;
                                IorD = 1'b1;
                                NXT_STATE = S4;
                        end

                        SC: begin
                                MemWrite = 1'b1;
                                NXT_STATE = S0;
                        end

                        SD: begin
                                RegWrite = 1'b1;
                                NXT_STATE = S0;
                        end
                        default: NXT_STATE = S0;
                endcase
        end

endmodule
```

B. PROYECTO FINAL DE DISEÑO DE CIRCUITOS INTEGRADOS DIGITALES

ITESO

Universidad Jesuita
de Guadalajara

Diseño de circuitos integrados Digitales I

Proyecto
“Optimización de los tiempos de setup y hold de flip flops”

Alejandro Güereña Morán
679705

Tabla de Contenidos

3. 1.- Objetivo:

- Demostrar que realizando la optimización de los tiempos de setup y hold de flip flops dentro de un sistema impacta y en qué medida, el desempeño de un circuito constituido por Flip Flops.

4. 2.- Introducción:

- Los circuitos secuenciales tienen la característica de que su salida depende de las entradas actuales, así como de las entradas previas. Los flip flops son el elemento fundamental de la lógica digital secuencial. Representan elementos básicos de memoria al ser capaces de permanecer en un estado lógico, cambiando su salida sólo en flancos de reloj.

- Se usan para implementar máquinas de estado, contadores, memorias, registros de corrimiento y otros circuitos secuenciales.

- Los flip flops pueden ser diseñados y reproducidos con la lógica de compuertas estándares NAND/NOR/NOT, pero su diseño a nivel transistor permite una implementación que utilice menos componentes.

- El diseño de de un flip flop impacta en el desempeño de los circuitos secuenciales, ya que las dimensiones y topologías que se usen para el diseño de éstos, afectarán en parámetros como consumo de potencia, área de circuito y velocidad.

- Las entradas de datos y señal de reloj de un flip flop necesitan satisfacer restricciones básicas de tiempo, para garantizar el correcto funcionamiento del flip flop. Estas restricciones de tiempo entre la señal de entrada y la señal de reloj se conocen como tiempo de setup y tiempo de hold. El incumplimiento de estas restricciones de tiempo, causan que nuestro circuito secuencial tenga un funcionamiento incorrecto.

- El tiempo de setup (tsu) se define como el mínimo tiempo requerido para que el valor del dato de entrada se encuentre definido antes de la transición de reloj.

- El tiempo de hold (thold) se define como el mínimo tiempo requerido en que el valor del dato de entrada debe permanecer retenido antes de la transición de reloj.

5. 3.- Desarrollo:

1. Elegir una topología de flip flop tipo D a estudiar.
2. Realizar el diseño funcional a nivel esquemático
3. Realizar el layout del flip flop (No realizar ajustes post layout)
4. Elegir un circuito que esté integrado por flip fllops, ejemplo: contador o shift register (de 4 a 10 flip flops). Circuito elegido: *Contador de anillo módulo 8.*
5. Realizar el layout del circuito utilizando el flip flop elegido.
6. Realizar pruebas de desempeño, velocidad.
7. Optimizar el mismo flip flop reduciendo los tiempos de set y hold.
8. Realizar el mismo circuito utilizando el flip flop mejorado y realizar las mismas pruebas.
9. Comparar circuitos en velocidad, área, potencia

Este documento primero presenta la implementación de un flip flop D, mostrando su desempeño pre-layout, post-layout y pre-layout optimizado. Posteriormente, se presenta un circuito implementado con este flip flop, mostrando desempeño pre-layout, post-layout y pre-layout optimizado.

5.1. 3.1 – Especificaciones de diseño:

Las especificaciones de este proyecto se muestran en la Tabla 1:

Tech	Amis 0.6µm CMOS

Vdd	3 V
CLoad	50 f F
fin (clk)	100 MHz
tramp in	100 ps
Wp/Wn	4.05 µm /1.95 µm
Ln	0.6µm
Tappering	1

Tabla 1. Especificaciones del diseño.

6. 4.- Flip Flop D:

En el desarrollo del Flip Flop D, primero se analizan dos propuestas de flip flop y se escoge una según sus características. Una vez definido el flip flop a usar dentro de un circuito digital, se construye el layout y se realizan ajustes para optimizar sus tiempos de setup y hold. El procedimiento de obtener los distintos parámetros obtenidos para flip flop y contador son presentados en el primer análisis de flip flops.

6.1. 4.1 – Topología Flip Flop D:

La topología de flip flop D que será usada en este proyecto es la arquitectura “Power PC” [1], añadiendo lógica de Reset y Set. Esta topología se muestra en la figura 1.

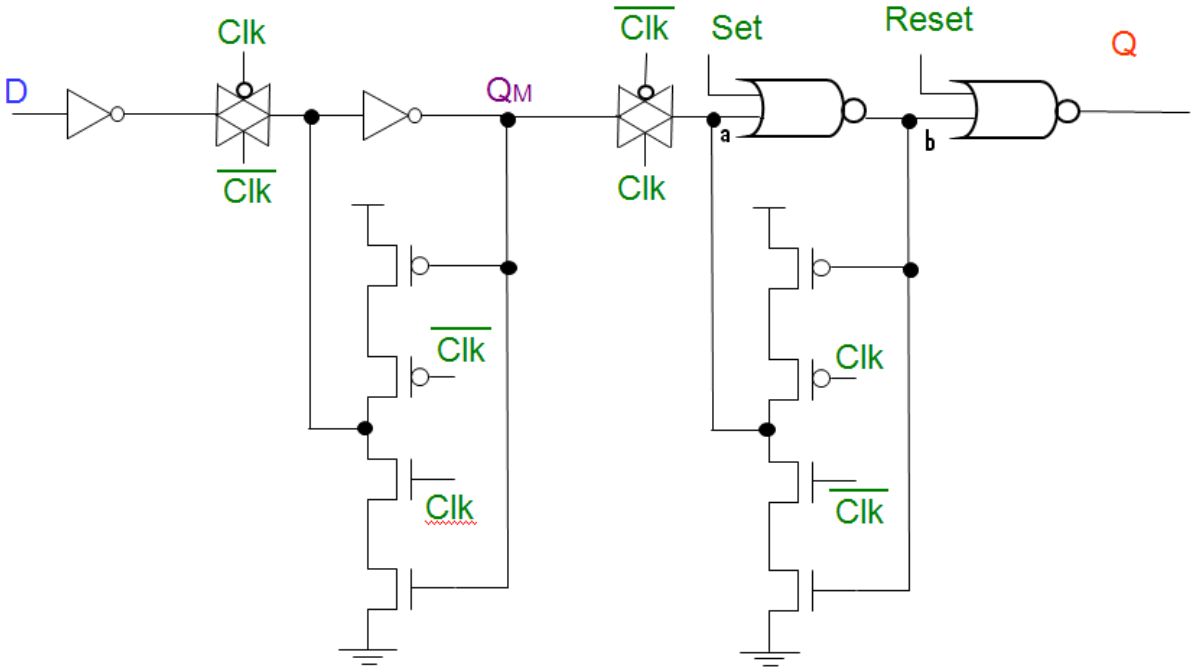


Figura 1. Topología Power PC.

Esta topología contiene: dos compuertas de transmisión que se activan en flancos opuestos de la señal de reloj; dos inversores y dos latches que mantienen el estado anterior antes de que se presente un nuevo flanco de subida de reloj. Las compuertas NOR, sirven para forzar un 0 en la salida, cuando la señal de Reset este activa en alto, sin importar el valor de la señal de Set. Si la señal Reset es desactivada y Set activada en alto, se obtiene un 1 lógico en la salida. Cuando las dos señales estén desactivadas, las compuertas NOR se comportan como inversores, siendo este el funcionamiento normal de flip flop Power PC. En la tabla 2, se muestra la tabla lógica del comportamiento de las señales de Reset y Set.

Reset	**Set**	**b**	**Q**
1	X	X	0
0	1	0	1
0	0	~a	a

Tabla 2. Lógica Reset y Set.

Para la implementación de este proyecto, es necesario agregar la lógica de Set y Reset al flip-flop D debido al circuito digital que se escogió. Se implementaron dos lógicas de Set y Reset.

Una es la mostrada en la figura 1, que consta de dos compuertas NOR en la etapa esclavo del flip flop. La otra lógica consta dos compuertas NOR en la etapa esclavo y dos compuertas NAND en la etapa maestro, sustituyendo los inversores.
La ventaja de insertar lógica en la etapa maestro es garantizar que no existan datos inválidos en la salida, producto de datos almacenados anteriormente en la etapa maestro. El compromiso radica en la velocidad, ya que al insertar lógica extra, tenemos mayor tiempo de propagación de la entrada a la salida, causando una menor velocidad de operación del circuito respecto a sólo tener lógica en la etapa esclavo.

6.2. 4.2 – Circuito Flip Flop D Pre-Layout:

Para este proyecto se analizaron dos flip flops tipo D PowerPC:

1- Flip Flop D con lógica Reset-Set en nodo esclavo.
2- Flip Flop D con lógica Reset-Set en nodo maestro y esclavo.

A continuación, se muestra el esquemático de los dos flip flop implementados. Uno con lógica de Reset-Set en etapa esclavo, (figura 2) y otro con lógica Reset-Set en etapas maestro y esclavo (figura 3).
Se observa en la figura 3, que al agregar una lógica más compleja al flip flop, el número de transistores incrementa. Se tienen 8 transistores de diferencia entre los dos flip flops.

Las dimensiones de los transistores para los flip flops no ajustados son los mostrados en la tabla 1.

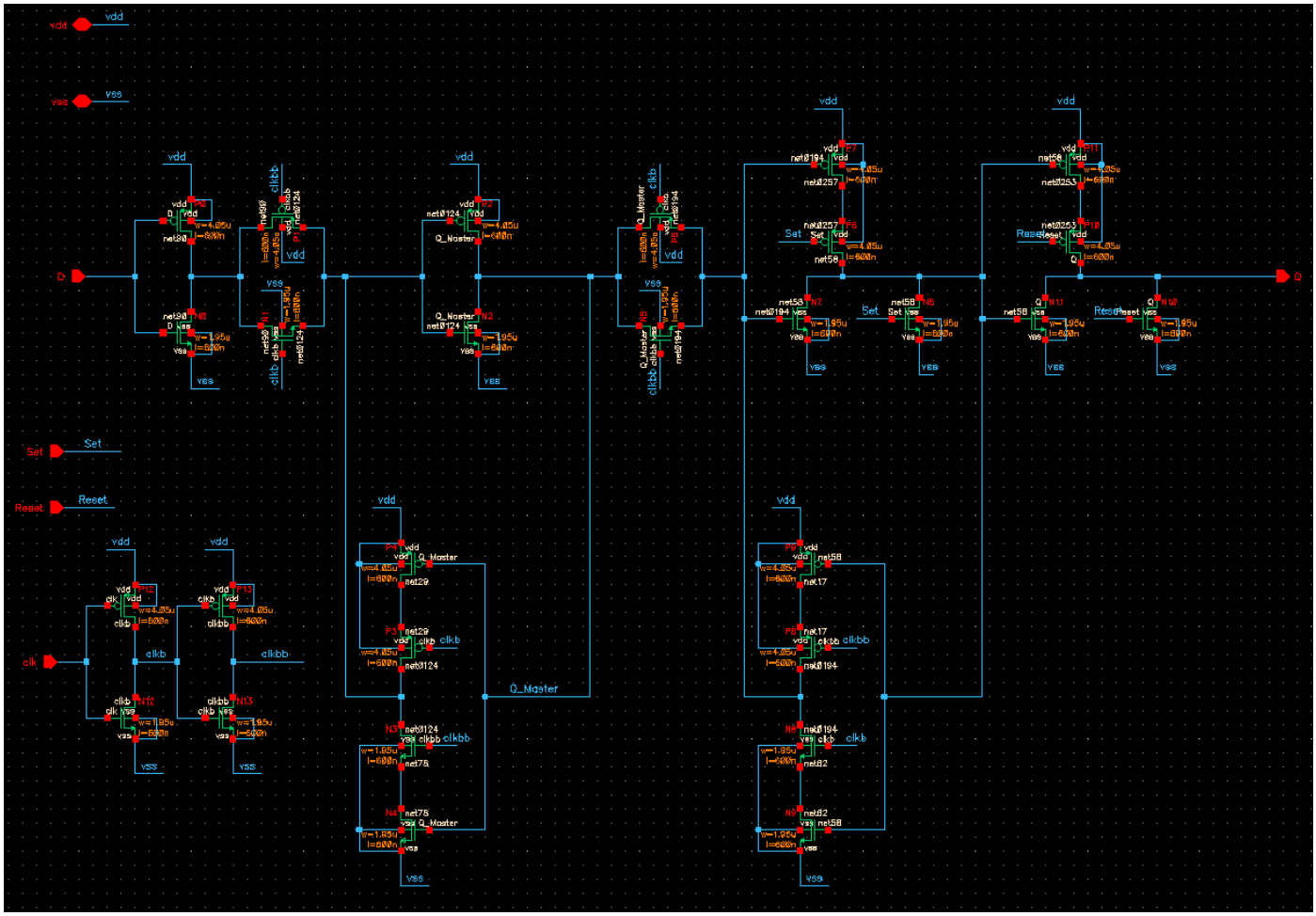


Figura 2. Esquemático Flip Flop D, Lógica Reset-Set en etapa esclavo.

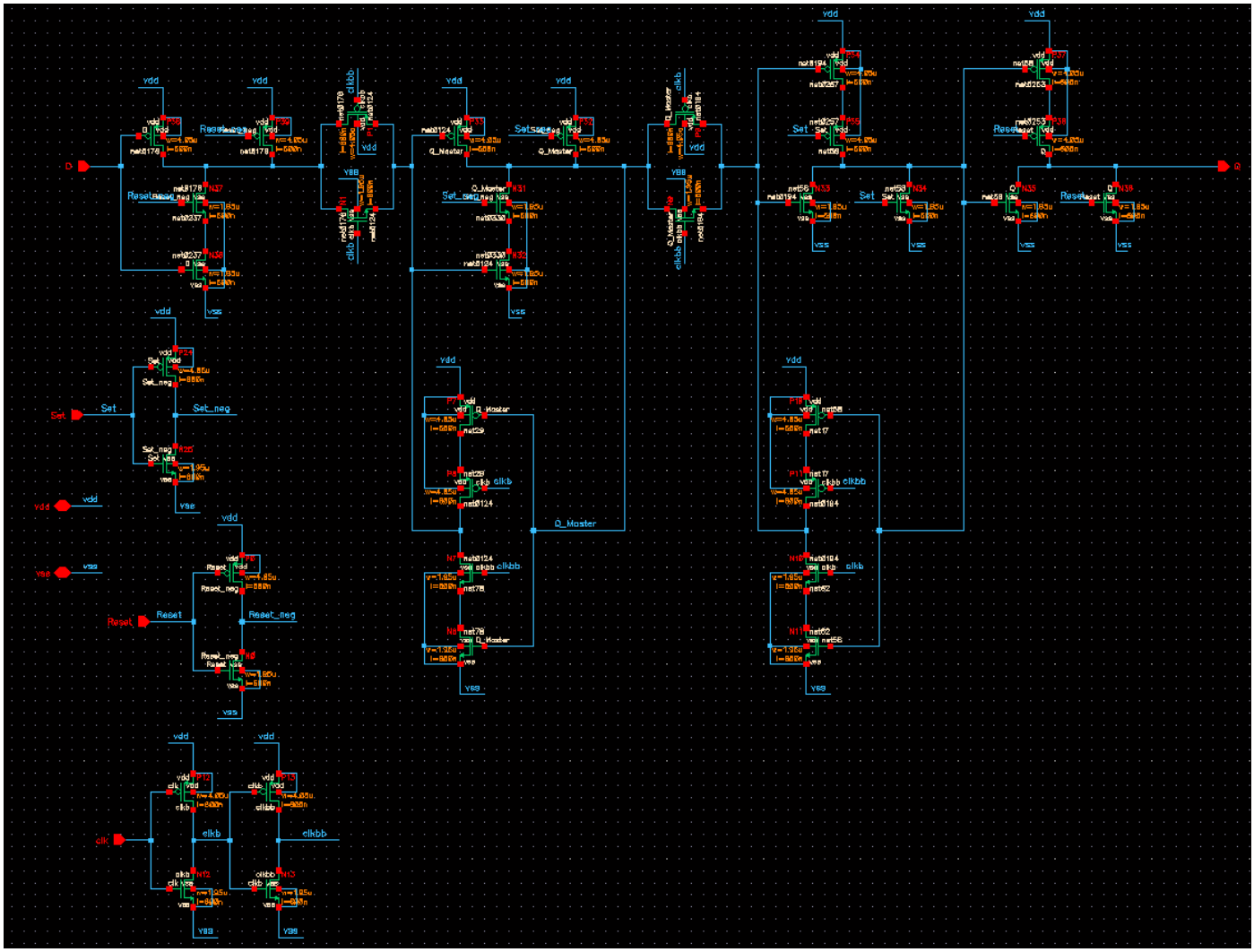


Figura 3. Esquemático Flip Flop D, Lógica Reset-Set en etapa maestro y esclavo.

Para comprobar el funcionamiento y realizar la caracterización del flip flop, se construye en la herramienta Cadence, el siguiente banco de pruebas, mostrado en la figura 4.

El banco de pruebas está constituido por los siguientes elementos:

- Fuente de voltaje DC de alimentación para vdd = 3V.
- Fuente de voltaje DC de alimentación para vss = 0V.
- Fuente de voltaje vpwl para generar señal de Reset.
- Fuente de voltaje vpwl para generar señal de Set.
- Fuente de voltaje de pulsos para generar la señal de entrada D, con las siguientes características (La señal de entrada D se mantiene al doble del período de la señal de reloj):
  - Delay Time 0s, 3ns.
  - Fall Time y Rise Time = 100 ps
  - Ancho de pulso = Ancho de pulso de reloj / 2.

- Periódo = Período del reloj / 2

- Fuente de voltaje de pulsos para generar la señal de entrada clk, con las siguientes características:
  - Delay Time = 0s
  - Fall Time y Rise Time = 100 ps
  - Ancho de pulso = Variable.
  - Periódo = Variable.
- Capacitor de carga CL = 50f F.

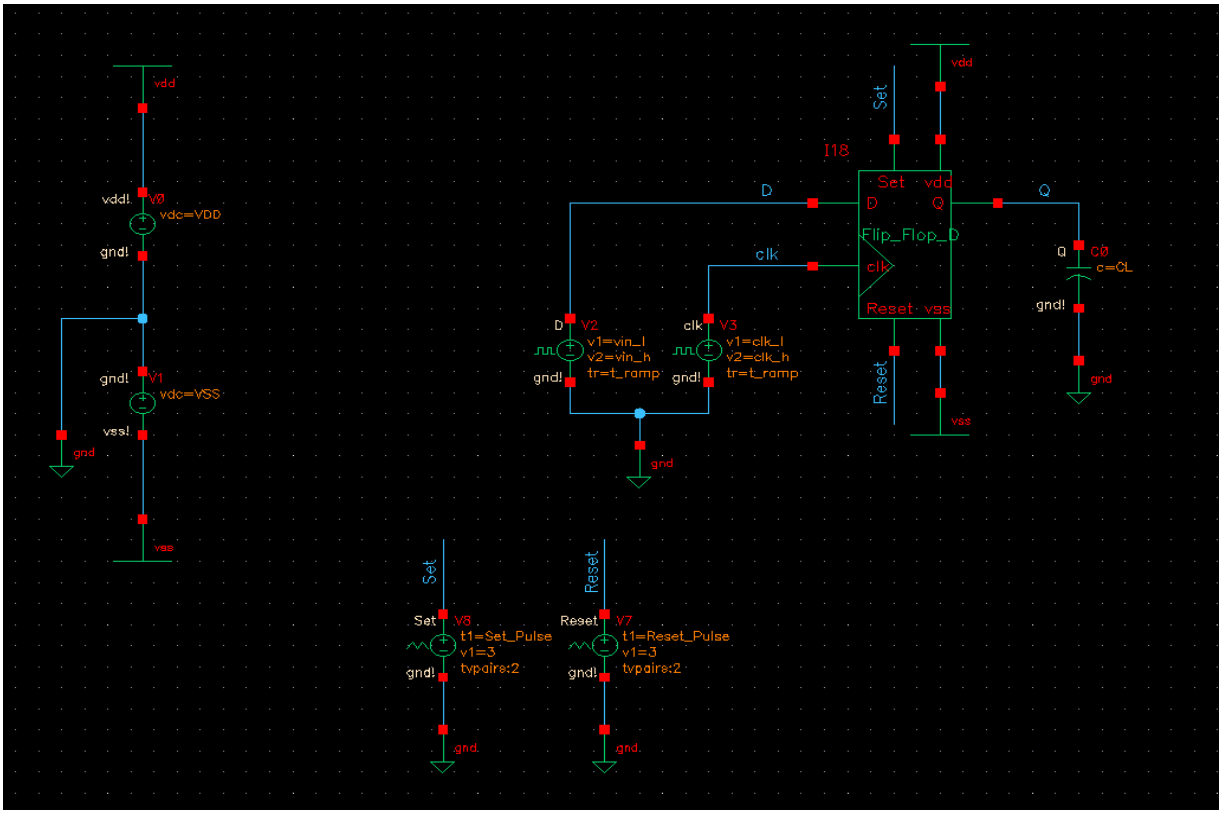


Figura 4. Banco de prueba para funcionamiento y caracterización de Flip Flop.

6.2.1 4.2.1 – Funcionamiento circuito Flip Flop D:

A continuación se muestra en funcionamiento de los dos flip flops con diferentes lógicas de Reset-Set. Se muestran simulaciones donde se verifique la diferencia en funcionamiento entre los flip flops.

La figura 5 muestra el funcionamiento del flip flop D con lógica Reset-Set en sólo la etapa esclavo. Se observa que el nodo maestro Q_M, no es afectado por las señales de Reset y Set, ya que obtiene el valor de la señal de entrada D en los flancos negativos de la señal de

reloj. Cuando la señal de Set es desactivada y la señal de reloj se encuentra en alto, la señal del nodo maestro pasa a la etapa esclavo y el valor rápidamente se refleja en la salida Q. En este flip flop, el valor del nodo maestro va siendo almacenado y puede generar un valor no deseado en la señal de salida Q al desactivar la señal de Set y Reset.

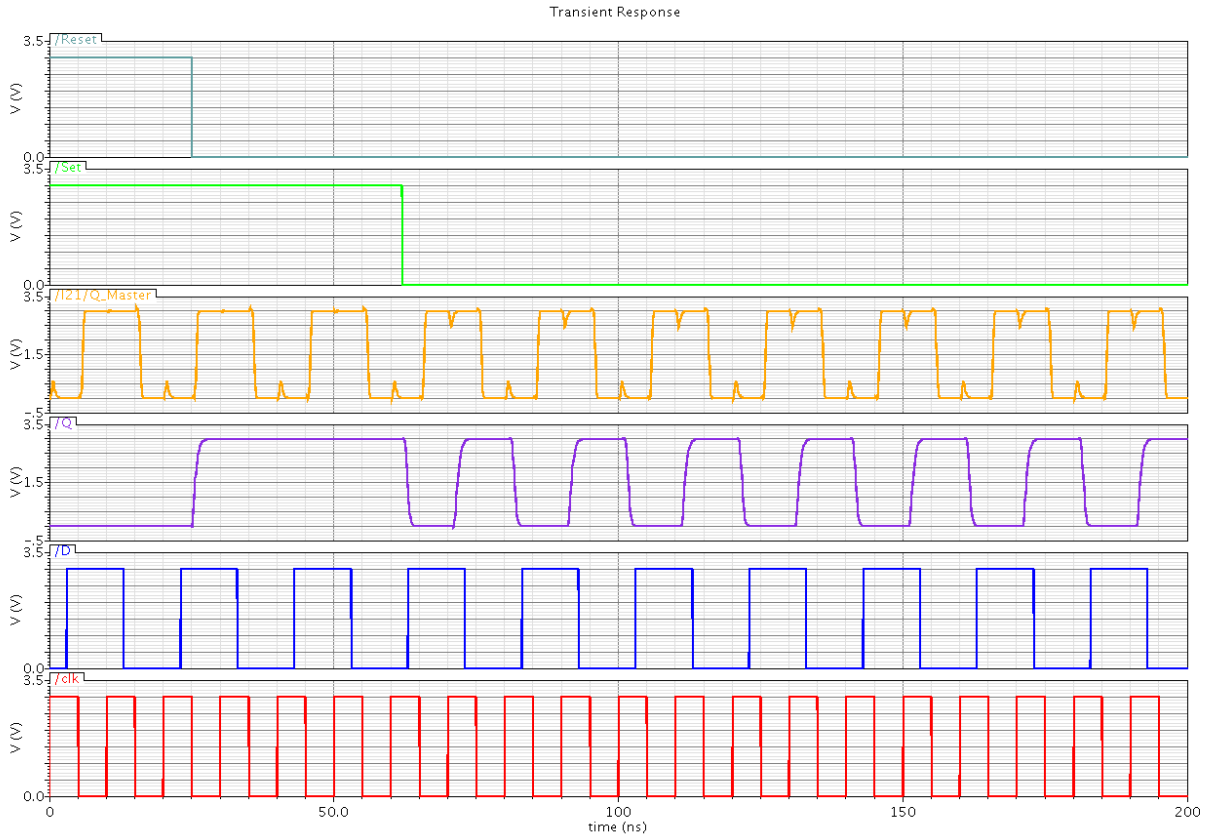


Figura 5. Funcionamiento Flip Flop D, lógica Reset-Set en etapa esclavo.

La figura 6 muestra el funcionamiento del flip flop D con lógica Reset-Set en las etapas maestro esclavo. A diferencia del flip flop anterior, se observa que el nodo maestro Q_M es afectado por las señales de Reset y Set, ya que se mantiene en alto cuando estas son activadas. Cuando la señal de Set es desactivada, el flip flop opera normalmente, ya que el valor del nodo maestro va a capturar el valor de la señal de entrada D cuando exista un flanco negativo de la señal de reloj, como es la operación normal del flip flop.

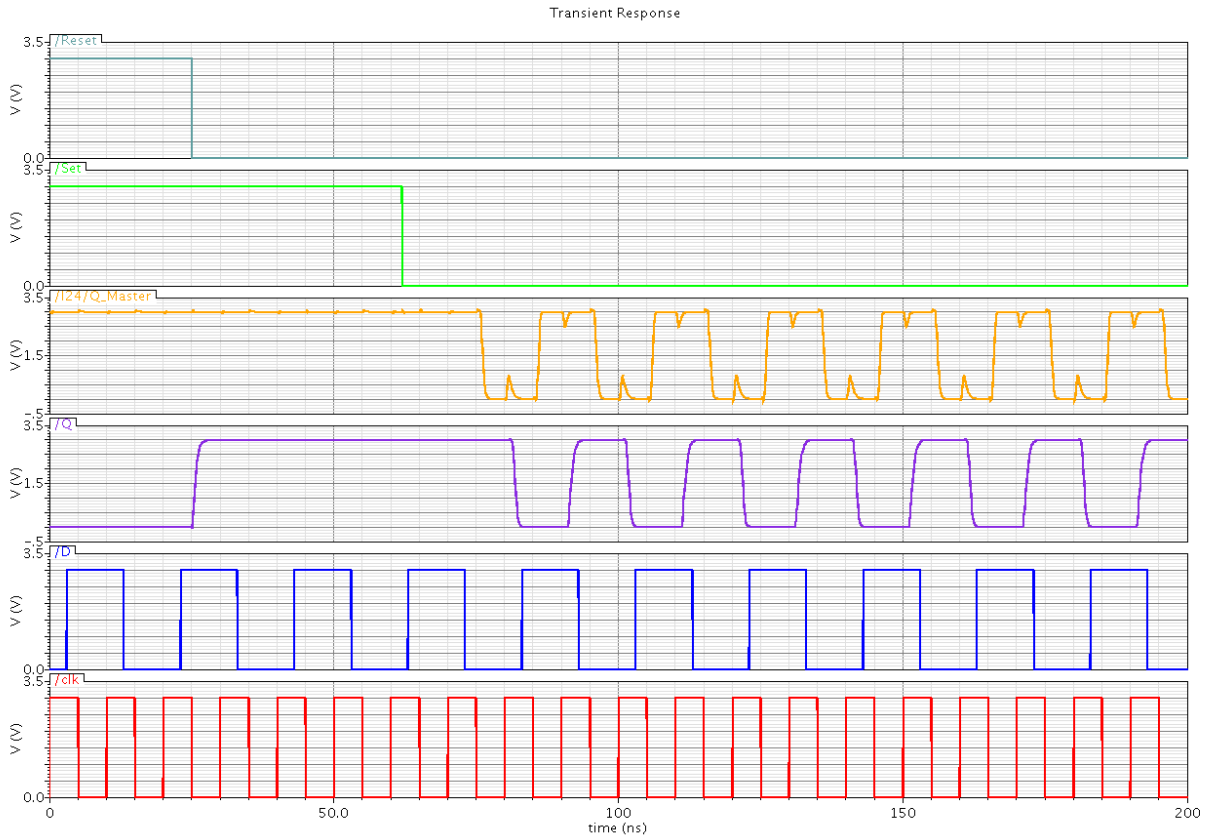


Figura 6. Funcionamiento Flip Flop D, lógica Reset-Set en etapa maestro y esclavo.

6.2.2 4.2.2 – Caracterización de tiempos de rampa, propagación y jitter de circuito Flip Flop D pre-layout:

4.2.2.1 –Tiempos de propagación, TpHL, TpLH de circuito Flip Flop D pre-layout:

Los tiempos de propagación nos indican el tiempo que transcurre entre el flanco de subida de reloj y la transición de la señal de salida Q a 1.5V.
Se obtuvieron los tiempos de propagación TpHL y TpLH usando la función “delay” de la calculadora de Cacende.

En la tabla 3 se muestran los tiempos medidos de propagación del flip flop con lógica Reset-Set en nodo esclavo y del flip flop con lógica Reset-Set en nodo maestro y esclavo.

Flip Flop Reset-Set en nodo esclavo		**Flip Flop Reset-Set en nodo maestro y esclavo**	
TpHL (ps)	**TpLH (ps)**	**TpHL (ps)**	**TpLH (ps)**
1756	1692	1850	1682

Tabla 3. TpHL y TpLH Flip Flop D pre-layout.

4.2.2.2 –Tiempos de rampa, Trise, Tfall de circuito Flip Flop D pre-layout:

Los tiempos de fall y rise de la señal de salida Q, fueron obtenidos usando la función delay de la herramienta calculadora de Cadence.
El tiempo de rise se obtiene midiendo el tiempo que tarda la señal de salida Q en ascender de 0.3 V a 2.7 V, valores que representan el 10% y 90% del voltaje total de la señal.
El tiempo de fall se obtiene midiendo el tiempo que tarda la señal de salida Q en descender de 2.7 V a 0.3 V, valores que representan el 90% y 10% del voltaje total de la señal.

En la tabla 4 se muestran los tiempos medidos de rampa del flip flop con lógica Reset-Set en nodo esclavo y del flip flop con lógica Reset-Set en nodo maestro y esclavo.

Flip Flop Reset-Set en nodo esclavo		**Flip Flop Reset-Set en nodo maestro y esclavo**	
Trise (ps)	**Tfall (ps)**	**Trise (ps)**	**Tfall (ps)**
1219	910	1240	911

Tabla 4. Trise y Tfall Flip Flop D pre-layout.

4.2.2.3 –Jitter de circuito Flip Flop D pre-layout:

El Jitter es un importante parámetro que se debe de medir en un flip flop, ya que representa la variación temporal de una señal respecto a la transición de referencia de la señal. Hay varios factores que puede ocasionar esta variación, como temperatura y ruido, por eso es necesario diseñar un fliop flop que reduzca esta variación.

Para obtener el jitter del flip flop implementado, se usa el banco de pruebas de la figura 4, realizando un análisis transitorio que muestre 20 ciclos de la señal de salida.
Una vez teniendo el análisis transitorio, se hace doble click sobre la etiqueta de “time”, en la parte centrar inferior de la gráfica mostrada por Cadence, accediendo a los atributos del eje de tiempo, como se muestra en siguiente figura:

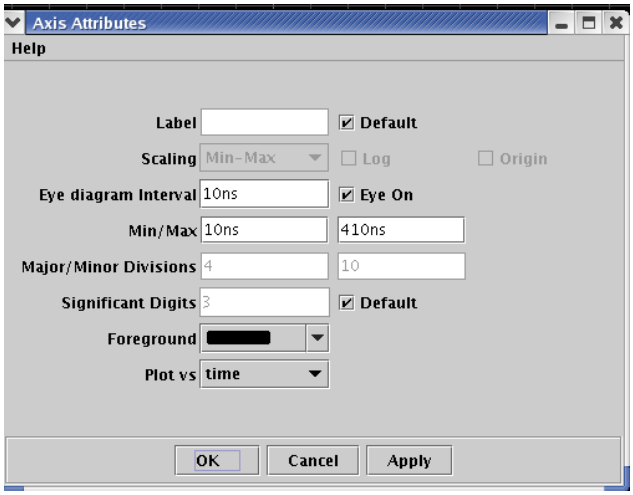


Figura 7. Obtención de diagrama de Ojo.

De la figura anterior, se selecciona la opción "Eye On" y se define el intervalo del diagrama del ojo. Este intervalo corresponde a 10ns, que es el período de la señal de reloj. Se escoge un mínimo y un máximo valor de tal forma que se tengan 20 ciclos de la señal de salida evitando el primer ciclo, debido a que este ciclo puede presentar un funcionamiento inválido o un retardo propagación mayor debido a que es el primer ciclo.

Para obtener le jitter mediante el diagrama de ojo, se mide la mayor diferencia que existe entre la señales de bajada y de subida del diagrama de ojo a 1.5 V.

En las figuras 8 y 9 se muestra el jitter de la salida Q, obtenido del flip flop con lógica Reset-Set en nodo esclavo y del flip flop con lógica Reset-Set en nodo maestro y esclavo.

En la tabla 5, se muestra una comparación de los valores obtenido de Jitter para los Flip Flops.

Flip Flop Reset-Set en nodo esclavo	**Flip Flop Reset-Set en nodo maestro y esclavo**
Jitter (ps)	**Jitter (ps)**
64.52	167.8

Tabla 5. Trise y Tfall Flip Flop D pre-layout.

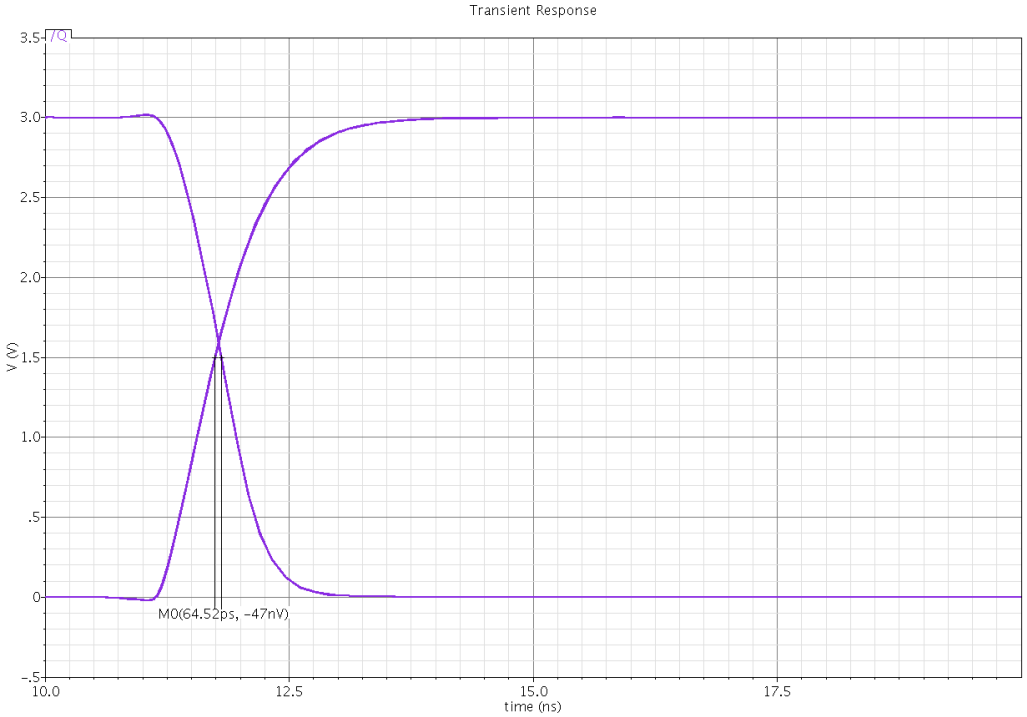


Figura 8. Diagrama de Ojo Flip Flop D, Lógica Reset-Set en etapa esclavo.

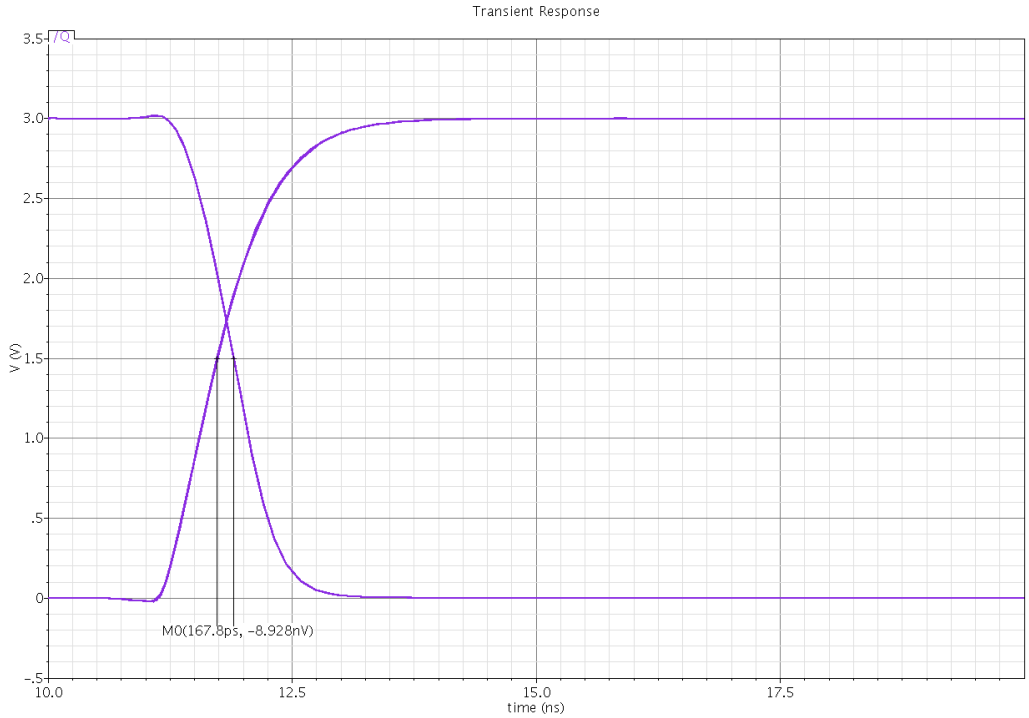


Figura 9. Diagrama de Ojo Flip Flop D, Lógica Reset-Set en etapa maestro y esclavo.

6.2.3 4.2.3 – Caracterización de setup y hold del circuito Flip Flop D pre-layout:

Para obtener los tiempos de setup y hold, se implementa un banco de pruebas que consta de una fuente de pulsos por puntos "vpwl". Esta fuente permite definir en qué instantes de tiempo queremos un voltaje específico. Este banco de pruebas es igual al presentado en la figura 4, sustituyendo la fuente de voltaje por pulsos de la señal de entrada D, por una fuente "vpwl"

El tiempo de setup se obtiene modificando el tiempo en que el valor de entrada D se encuentra definido antes del flanco de subida de la señal de reloj.

El tiempo de hold se obtiene modificando el tiempo en que el valor de entrada D permanece retenido antes o después del flanco de subida de la señal de reloj.

La configuración de la fuente de pulsos por puntos "vpwl" se muestra a continuación, para los cuatro análisis de tiempos, tsu 0->1, tsu 1->0, thold 0->1 y thold 1->0. Los tiempos fijos definidos deben de garantizar que el flip flop funcione correctamente. Sólo los tiempos que varían según el tiempo de set o hold constan de una variable.

CDF Parameter	Value	Display
Number of pairs of points	2	off
AC magnitude		off
AC phase		off
Time 1	(100n+tsu) s	off
Voltage 1	0 V	off
Time 2	(100n+tsu+t_ramp) s	off
Voltage 2	3 V	off
Delay time		off

Figura 10. Configuración de "vpwl" para tsu 0->1.

CDF Parameter	Value	Display
Number of pairs of points	2	off
AC magnitude		off
AC phase		off
Time 1	(100n+tsu) s	off
Voltage 1	3 V	off
Time 2	(100n+tsu+t_ramp) s	off
Voltage 2	0 V	off
Delay time		off

Figura 11. Configuración de "vpwl" para tsu 1->0.

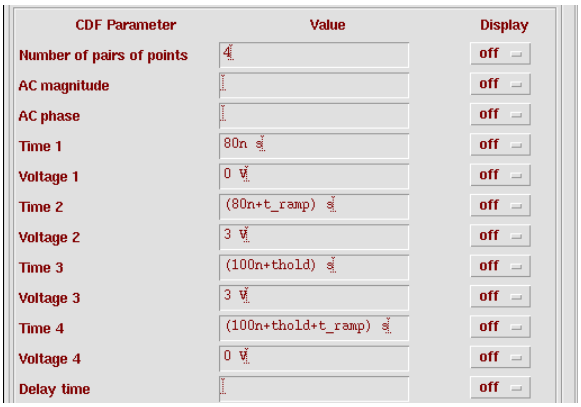


Figura 12. Configuración de “vpwl” para thold 0->1.

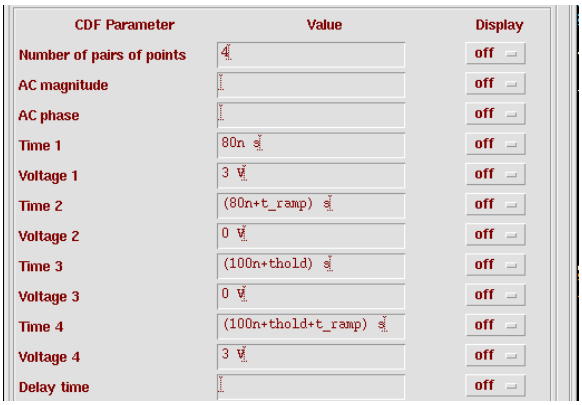


Figura 13. Configuración de “vpwl” para thold 1->0.

La configuración de la fuente vpwl genera una gráfica como la mostrada en la figura 14.

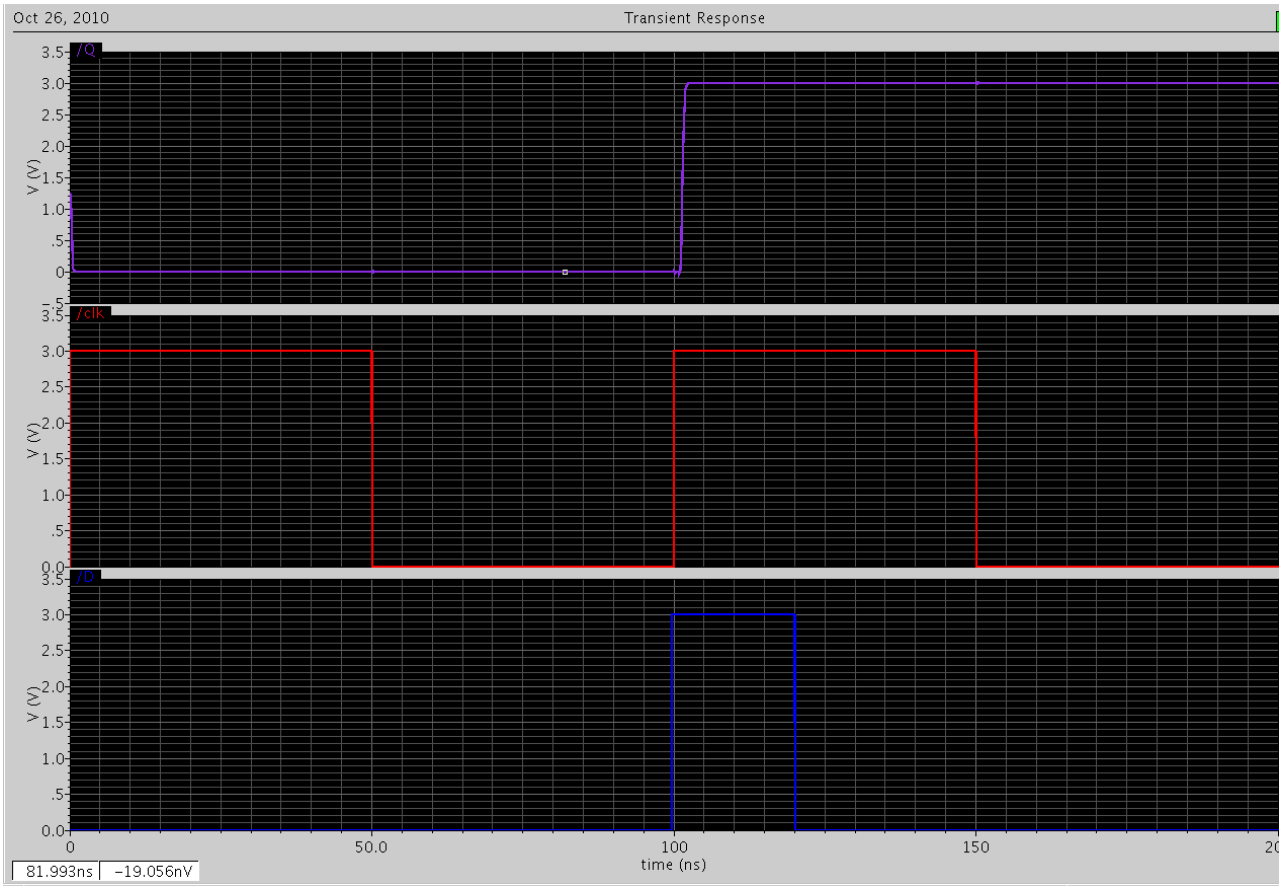


Figura 14. Gráfica para tsu 0->1.

De la figura 14, se observa que modificando los valores de tsu para la señal de entrada Q, se observará sí la señal Q de salida funciona correctamente.
Para obtener los valores de setup y hold de manera rápida, se usa la función “delay” de la calculadora, donde al modificar la variable tsu o thold, se obtiene el tiempo de propagación entre el flanco de subida de reloj y la señal de salida Q. Si se violan los tiempos de setup o hold, no va a existir un tiempo de propagación entre la señal de reloj y la señal Q. Con este tiempo de propagación y el valor dado a las variables tsu y thold, se obtienen los valores de tsu y thold. Se realiza un análisis para métrico como la figura 15, para obtener de manera gráfica las curvas de tsu y thold. Las curvas obtenidas en Cadence, son almacenadas en un archivo .cvs para generar las curvas en el programa Microsoft Office Excel, para presentar varias curvas en la misma gráfica, graficando un mínimo de 100 puntos por curva, donde el eje Y indica el tiempo de propagación entre la señal de reloj “clk” y la señal de salida Q, mientras que el eje X indica el tiempo de propagación entre la señal de reloj “clk” y la señal de entrada D.

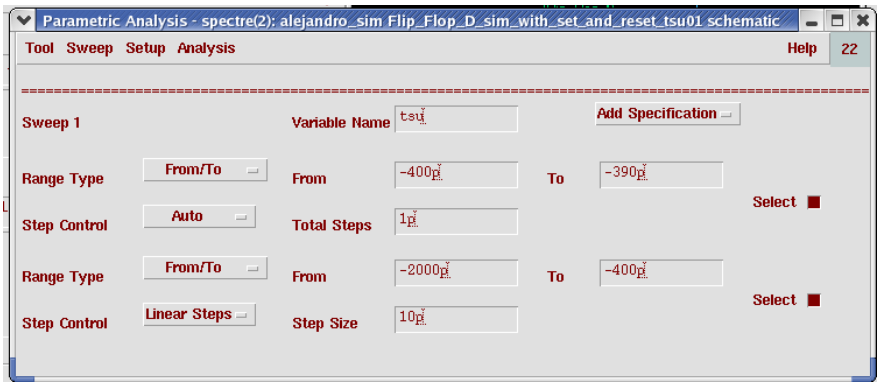


Figura 15. Análisis para métrico para tiempos de Setup y Hold.

Las gráficas de tsu y thold obtenidas para el flip flop con lógica Reset-Set en nodo esclavo y el flip flop con lógica Reset-Set en nodo maestro y esclavo se muestran a continuación:

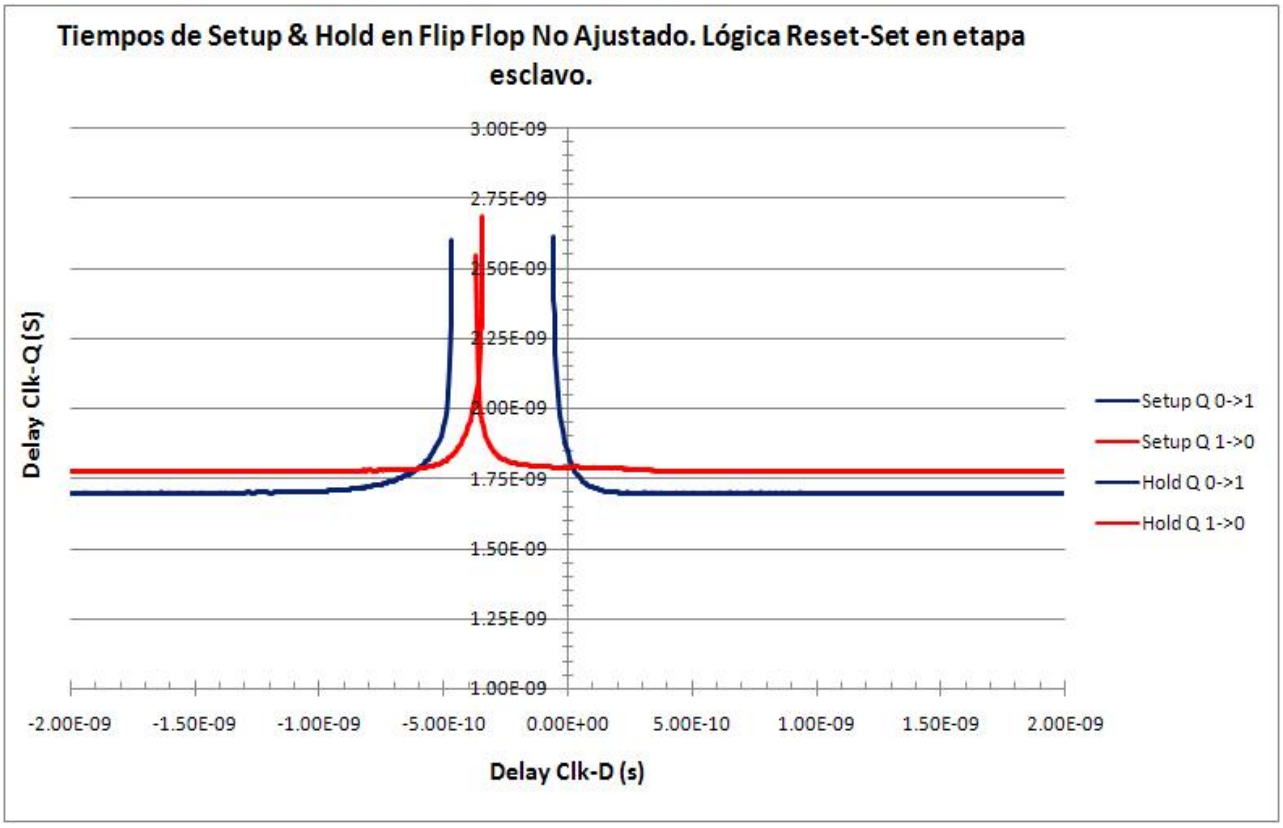


Figura 16. Tiempos de Setup y Hold para Flip Flop D. Lógica Reset-Set en etapa esclavo.

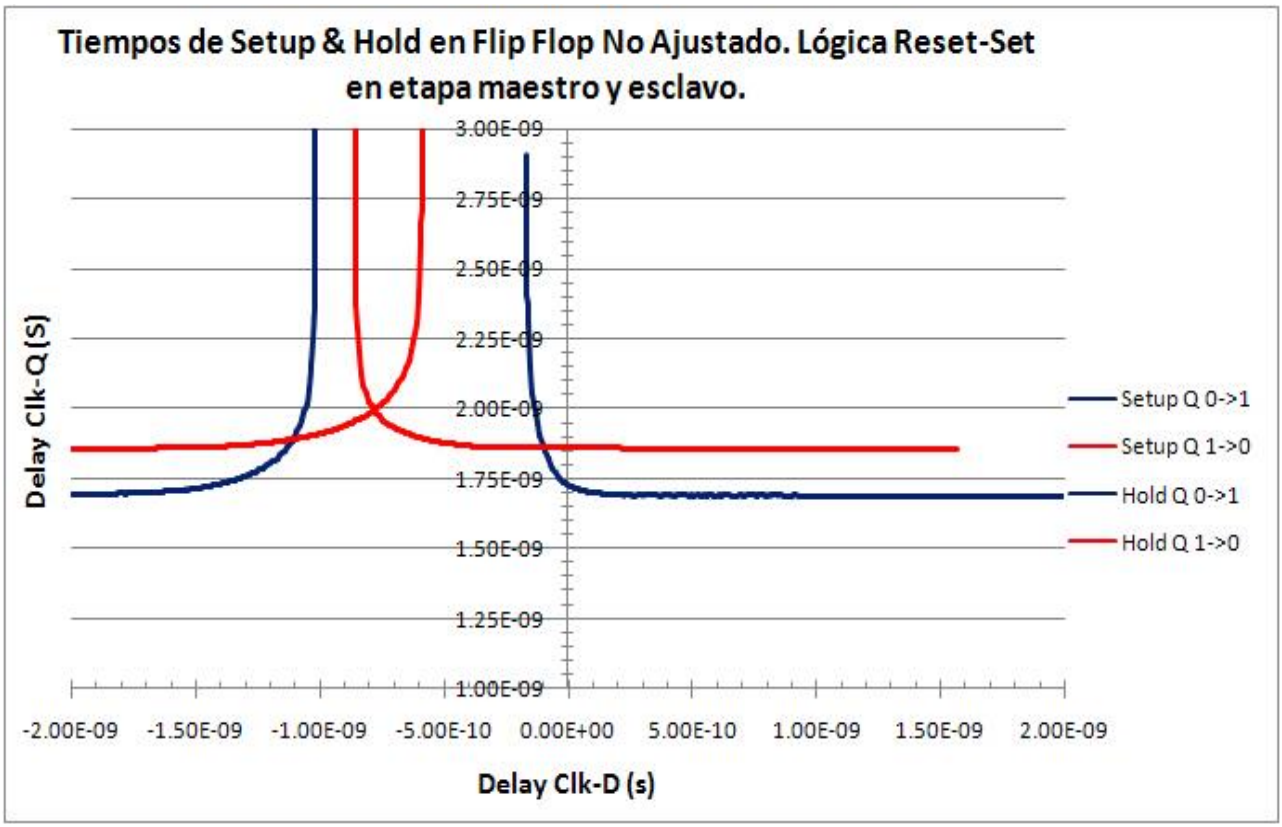


Figura 17. Tiempos de Setup y Hold para Flip Flop D. Lógica Reset-Set en etapa maestro y esclavo.

En la siguiente tabla se insertan los valores obtenidos para tiempos de setup, hold y los tiempos de propagación de la señal de salida en setup y hold.

	Setup		**Hold**	
	FF. Nodo Esclavo	**FF. Nodo Maestro y Esclavo**	**FF. Nodo Esclavo**	**FF. Nodo Maestro y Esclavo**
	Q 0 ->1			
Clk -> Q (ps)	2597	3612	2609	2908
Clk -> D (ps)	-468	-1015	-57	-168
	Q 1 ->0			
Clk -> Q (ps)	2685	3076	2542	3164
Clk -> D (ps)	-341	-585	-370	-856

Tabla 6. Tiempos de Setup, Hold y propagación para Flip Flop D con lógica Reset-Set, sin ajustes.

Los tiempos de propagación son mayores para el flip flop con lógica en nodo maestro y esclavo, necesitando la entrada de dato mucho antes de la señal de reloj que el otro flip flop, debido a los transistores adicionales que tiene este flip flop.

6.2.4 4.2.4 – Potencia del Flip Flop de circuito Flip Flop D pre-layout:

Se realizan mediciones de potencia máxima y RMS del circuito con las siguientes características:

- 20 ciclos de la señal de salida Q.
- Frecuencia de reloj a 100MHz
- Capacitor de carga de 50f F.

Para obtener la potencia máxima, se utiliza la calculadora de Cadence, que nos permite multiplicar la corriente de nuestro Flip Flop, por el voltaje de la fuente de alimentación de 3 V.
Para calcular el valor de potencia RMS, se utiliza la función rms de la calculadora y nos otorga el valor RMS de la potencia de nuestro circuito.

En la siguiente tabla se muestran los valores de potencias máximas y potencias RMS:

Potencia / Flip Flop	**FF. Nodo Esclavo**	**FF. Nodo Maestro y Esclavo**
Potencia Máxima (µW)	1004	1040
Potencia RMS (µW)	310.8	314.9

Tabla 7. Potencia Máxima y potencia RMS para Flip Flop D con lógica Reset-Set, sin ajustes.

La potencia entre los dos flip flops es similar, aunque es ligeramente mayor para el flip lop que contiene más transistores.

6.2.5 4.2.5 – Frecuencia Máxima de circuito Flip Flop D pre-layout:

La frecuencia máxima individual de un Flip Flop, se encuentra determinada por sus tiempos de setup, hold y de propagación.
La suma de los tiempos mayores de setup y hold, nos otorgan el mínimo período o la máxima frecuencia con la que opera nuestro Flip Flop. También se debe de tomar en cuenta que el menor tiempo de propagación de la señal de salida Q, respecto a la señal de reloj, influye en la frecuencia máxima, ya que al violar este tiempo de propagación, el Flip Flop puede tener un comportamiento erróneo.
Se observó durante el desarrollo del proyecto, que la suma de los tiempos de setup y hold, otorgan frecuencias altas, que al probarlas, el Flip Flop no funcionó.
La metodología para encontrar la frecuencia máxima de operación, fue observar el tiempo de propagación en los tiempos de setup y hold, y con de ese tiempo, calcular la frecuencia. A partir de esa frecuencia, se prueba nuestro Flip Flop. Sí el Flip Flop funciona con esa frecuencia, esta se va aumentando hasta que deje de funcionar el circuito. Se puede realizar un análisis para-métrico variando la señal de reloj y observar hasta que frecuencia deja de operar el Flip Flop. En la figura 18 se muestra un ejemplo de este análisis para métrico.

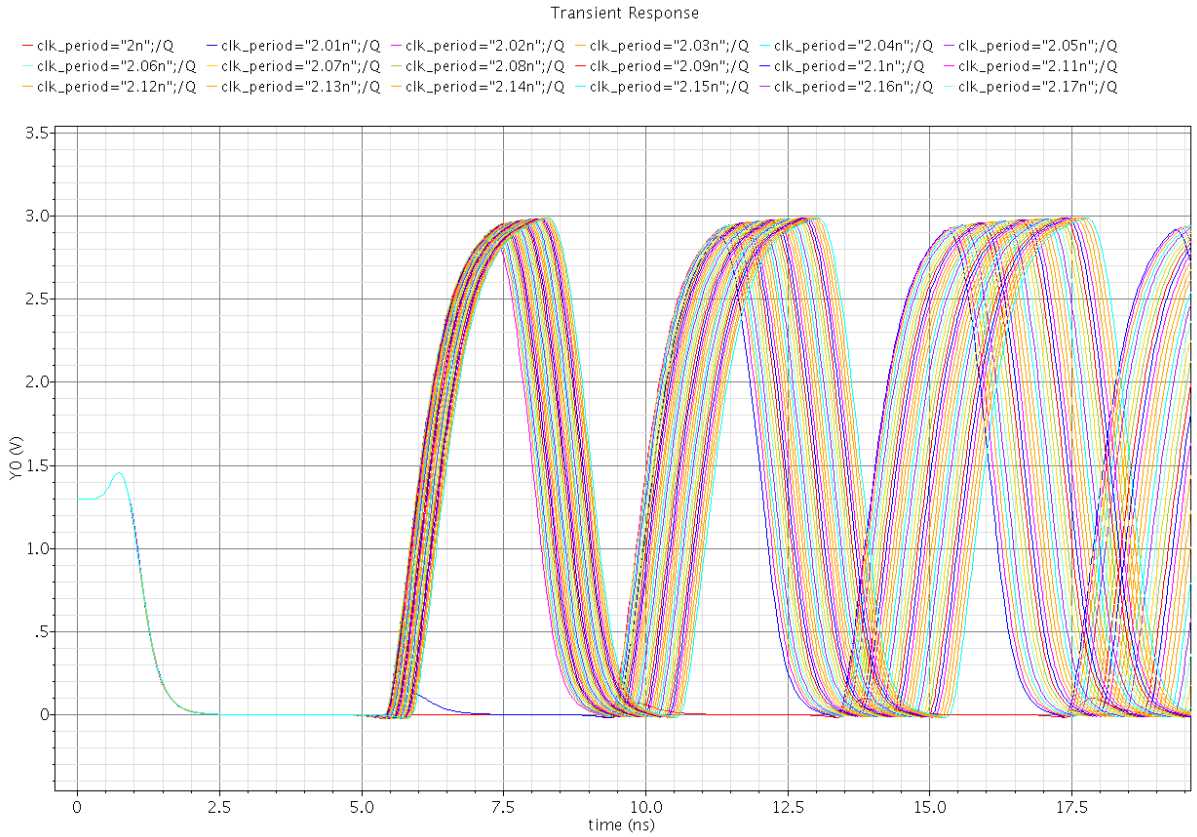


Figura 18. Análisis para métrico para obtener la frecuencia máxima de operación del Flip Flop.

La siguiente tabla muestra las frecuencias máximas de operación del flip flop con lógica Reset-Set en nodo esclavo y del flip flop con lógica Reset-Set en nodo maestro y esclavo. La frecuencia máxima de operación teórica está representada por la suma de los tiempos máximos de setup y hold, mostrados en la tabla 8. La frecuencia máxima real, fue la que se obtuvo mediante las simulaciones.

Frecuencia / Flip Flop	**FF. Nodo Esclavo**	**FF. Nodo Maestro y Esclavo**
Frecuencia Máxima Teórica (MHz)	1193	534.4
Frecuencia Máxima Real (MHz)	581.3	497.5

Tabla 8. Frecuencia máxima para Flip Flop D con lógica Reset-Set, sin ajustes.

.

Se observa de la tabla 8, que la frecuencia de operación es mayor para el flip flop con lógica sólo en el nodo esclavo. Este dato confirma que los tiempos de propagación, rampa, jitter son menores para este flip flop, debido a que consta de menos transistores, por lo tanto, consta de una mayor velocidad. El compromiso es sacrificar una lógica segura de Reset-Set por velocidad, área y potencia. Para este proyecto, se utiliza el flip flop de lógica Reset-Set en nodo esclavo debido a su mayor frecuencia de operación.

6.2.6 4.3 – Circuito Flip Flop D Post-Layout:

Para este proyecto es necesario realizar el layout del flip flop con lógica de Reset-Set en nodo esclavo, para observar el análisis post-layout del circuito y compararlo con el análisis pre-layout.
El layout del flip flop es construido a partir de las dimensiones de la tabla 1.

6.2.7 4.3.1 – Layout de Flip Flop:

En la figura 19 se muestra el layout del flip flop con lógica de Reset-Set en nodo esclavo:

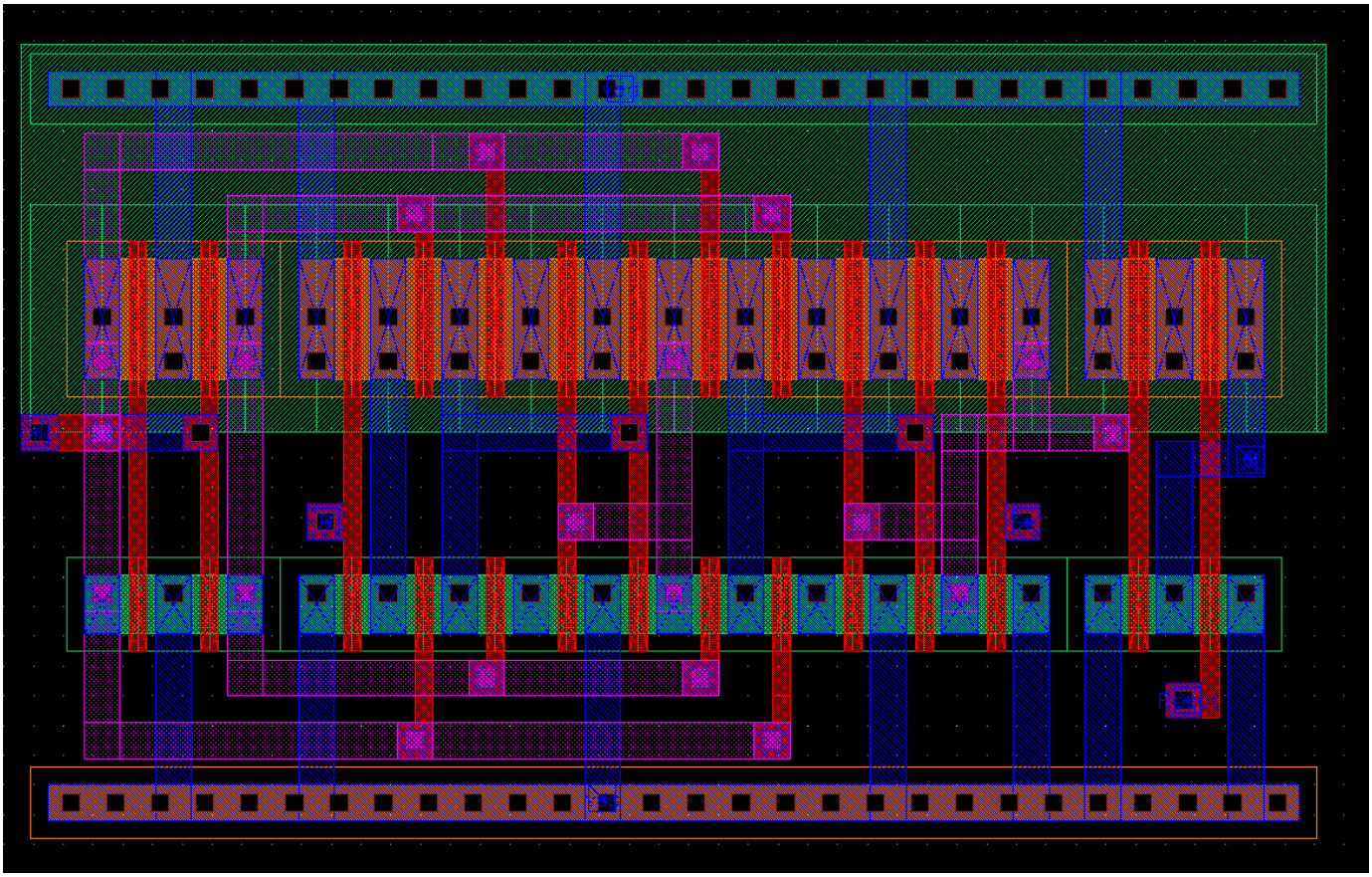


Figura 19. Layout de Flip Flop D con lógica de Reset-Set en nodo esclavo.

6.2.8

El área del layout del flip flop es de 43.8 µm en X y 26.7 µm en Y.
El análisis DRC y LVS del flip flop se presentan en la figura 20.

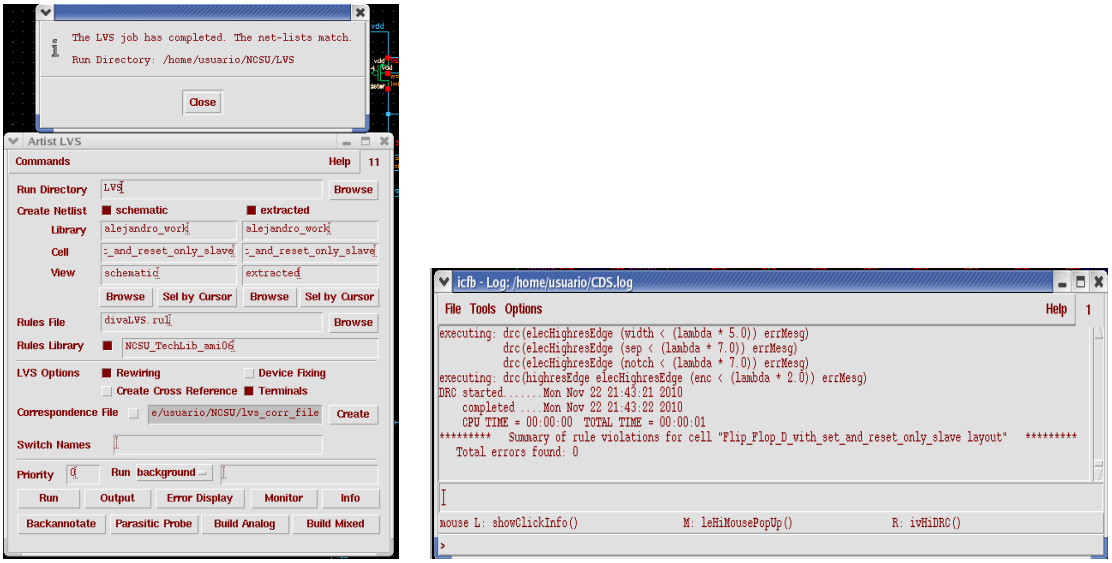


Figura 20. Análisis DRC y LVS de Flip Flop.

6.2.9 4.3.2 – Caracterización de tiempos de rampa, propagación y jitter de circuito Flip Flop D post-layout:

4.3.2.1 –Tiempos de propagación, TpHL, TpLH de circuito Flip Flop D post-layout:

En la tabla 9 se muestran los tiempos medidos de propagación del flip flop con lógica Reset-Set en nodo esclavo post-layout.

Flip Flop Post-layout	
TpHL (ps)	**TpLH (ps)**
1711	1583

Tabla 9. TpHL y TpLH de Flip Flop D post-layout.

4.3.2.2 –Tiempos de rampa, Trise, Tfall de circuito Flip Flop D post-layout:

En la tabla 10 se muestran los tiempos medidos de rampa del flip flop con lógica Reset-Set en nodo esclavo post-layout.

Flip Flop Post-layout	
Trise (ps)	**Tfall (ps)**
1168	810

Tabla 10. Trise y Tfall de Flip Flop D post-layout.

4.3.2.3 –Jitter de circuito Flip Flop D post-layout:

En la figura 21 se muestra el jitter de la salida Q, obtenido para flip flop con lógica Reset-Set en nodo esclavo post-layout.

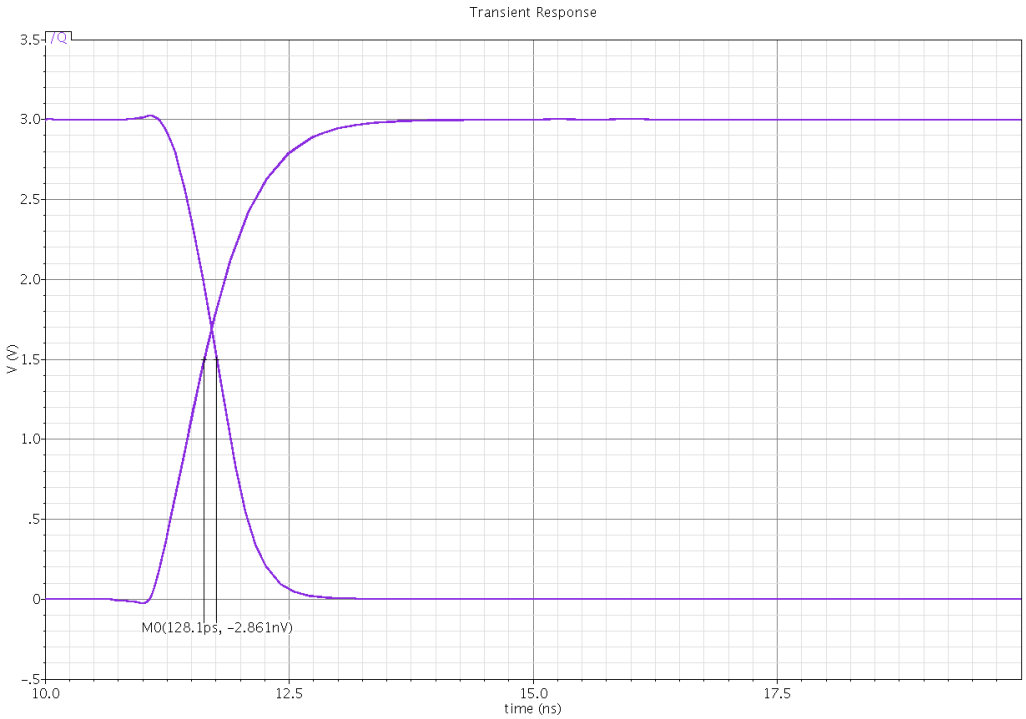


Figura 21. Diagrama de Ojo Flip Flop D, post-layout.

En la tabla 11, se muestra el valor obtenido de Jitter.

Flip Flop Post-layout
Jitter (ps)
128.1

Tabla 11. Jitter de Flip Flop D post-layout.

6.2.10 4.3.3 – Caracterización de setup y hold de circuito Flip Flop D post-layout:

Las gráficas de tsu y thold obtenidas para el flip flop con lógica Reset-Set en nodo esclavo post layout se muestran en la siguiente figura:

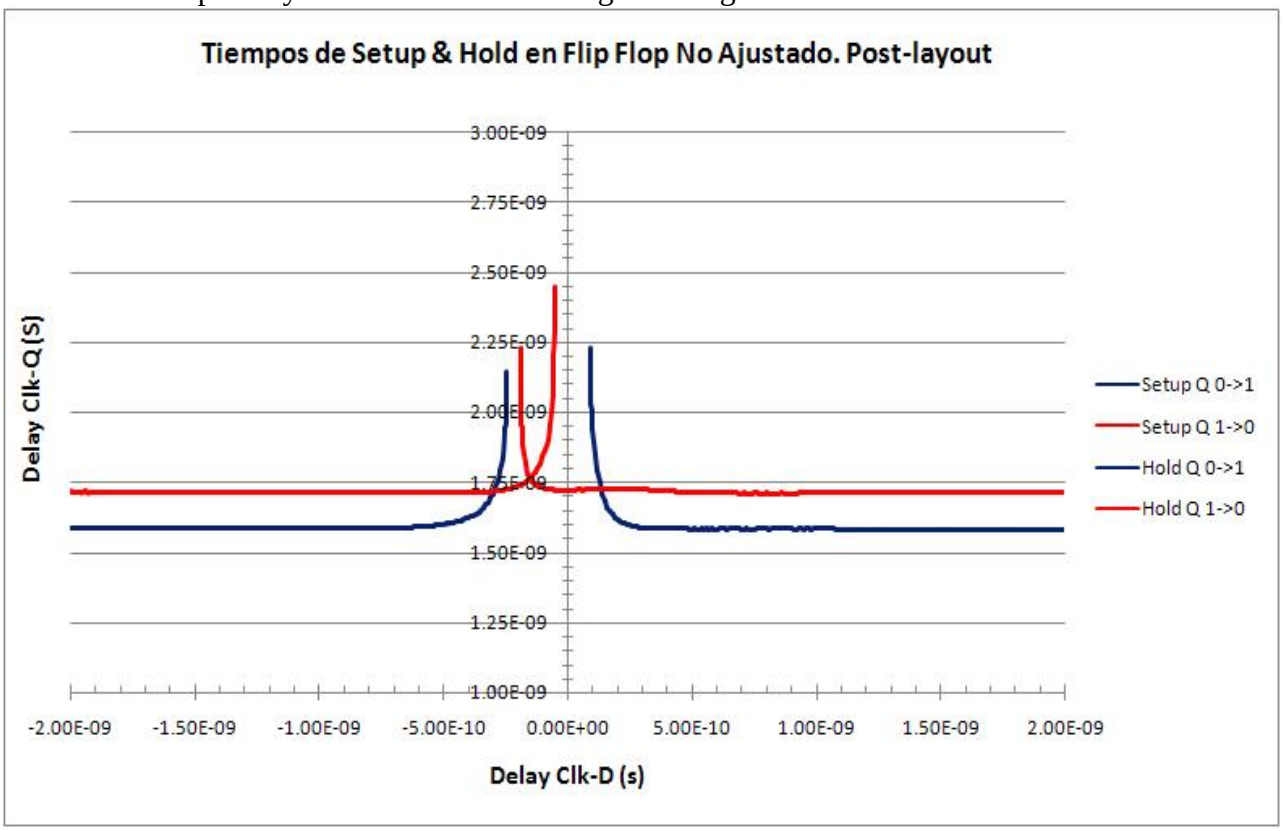


Figura 22. Tiempos de Setup y Hold para Flip Flop D post-layout.

En la siguiente tabla se insertan los valores obtenidos para tiempos de setup, hold y los tiempos de propagación de la señal de salida en setup y hold.

	Setup	**Hold**
	Flip Flop Post-layout	
	Q 0 ->1	
Clk -> Q (ps)	2142	2229
Clk -> D (ps)	-246	91
	Q 1 ->0	
Clk -> Q (ps)	2450	2228
Clk -> D (ps)	-53	-189

Tabla 12. Tiempos de Setup, Hold y propagación para Flip Flop D post-layout.

6.2.11 4.3.4 – Potencia del Flip Flop de circuito Flip Flop D post-layout:

En la siguiente tabla se muestran los valores de potencia máxima y potencia RMS:

Potencia / Flip Flop	**Flip Flop Post-layout**
Potencia Máxima (µW)	1114
Potencia RMS (µW)	315

Tabla 13. Potencia Máxima y potencia RMS para Flip Flop D post-layout.

6.2.12 4.3.5 – Frecuencia Máxima de circuito Flip Flop D post-layout:

La siguiente tabla muestra la frecuencia máxima real y teórica de operación del flip flop con lógica Reset-Set en nodo esclavo post layout.

Frecuencia / Flip Flop	**Flip Flop Post-layout**
Frecuencia Máxima Teórica (MHz)	2229
Frecuencia Máxima Real (MHz)	720

Tabla 14. Frecuencia máxima para Flip Flop D post-layout.

.

6.2.13 4.4 – Caracterización de tiempos de setup y hold al variar dimensiones de los transistores.

Después de definir que el Flip Flop con lógica de Reset-Set en nodo esclavo será usado para construir un circuito digital, se rediseño el flip flop mediante la obtención de los mínimos tiempos de setup y hold, incrementando las dimensiones de cada uno de los transistores del flip flop para observar que transistores influyen en la velocidad del flip flop [7]. Los transistores se incrementaron al doble del tamaño de manera individual. Los resultados de las mediciones se muestran en la tabla 15. Dimensiones de transistores en µm y tiempos en ps.

		N0	P0	N1	P1	N2	P2	N3-N4	P3-P4	N5	P5	N6-N7	P6-P7	N8-N9	P8-P9	N10-N11	
N0	1.95	4.05	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	
P0	4.05	4.05	7.95	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	
N1	1.95	1.95	1.95	4.05	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	
P1	4.05	4.05	4.05	4.05	7.95	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	
N2	1.95	1.95	1.95	1.95	1.95	4.05	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	
P2	4.05	4.05	4.05	4.05	4.05	4.05	7.95	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	
N3-N4	1.95	1.95	1.95	1.95	1.95	1.95	1.95	4.05	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	
P3-P4	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	7.95	4.05	4.05	4.05	4.05	4.05	4.05	4.05	
N5	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	4.05	1.95	1.95	1.95	1.95	1.95	1.95	
P5	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	7.95	4.05	4.05	4.05	4.05	4.05	
N6-N7	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	4.05	1.95	1.95	1.95	1.95	
P6-P7	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	7.95	4.05	4.05	4.05	
N9-N9	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	4.05	1.95	1.95	
P8-P9	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	7.95	4.05	
N10-N11	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	1.95	4.05	
P10-P11	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	4.05	
Setup 0 -> 1 (ps)	-468	-235	-542	-421	-587	-556	-435	-470	-602	-484	-479	-470	-472	-454	-468	-468	
Setup 1 -> 0 (ps)	-341	-367	-173	-336	-315	-171	-171	-464	-404	-363	-368	-348	-348	-329	-368	-341	
Hold 0 -> 1 (ps)	-57	-80	38	-67	-69	-29	-151	-164	-17	-51	-29	54	-49	-30	-46	-57	
Hold 1 -> 0 (ps)	-370	-162	-441	-338	-465	-481	-357	-348	-496	-371	-355	-367	-366	-365	-344	-370	

Tabla 15. Resultados de las mediciones de setup y hold al incrementar al doble el tamaño de los transistores.

En la siguiente tabla se muestran las diferencias respecto a los valores obtenidos variando las dimensiones de los transistores respecto al flip flop de dimensiones fijas. En color verde se muestran aquellas diferencias que disminuyen significativamente los tiempos de setup y hold y en color rojo las diferencias que aumentan significativamente los tiempos de setup y hold.

	N0	P0	N1	P1	N2	P2	N3-N4	P3-P4	N5	P5	N6-N7	P6-P7	N8-N9	P8-P9	N10-N11	P10P1
Setup 0 -> 1	233	-74	47	-119	-88	33	-2	-134	-16	-11	-2	-4	14	0	0	
Setup 1 -> 0	-26	168	5	26	170	170	-123	-63	-22	-27	-7	-7	12	-27	0	
Hold 0 -> 1	-23	95	-10	-12	28	-94	-107	40	6	28	111	8	27	11	0	
Hold 1 -> 0	208	-71	32	-95	-111	13	22	-126	-1	15	3	4	5	26	0	

Tabla 16. Diferencias entre los tiempos iniciales con respecto a los obtenidos variando las dimensiones.

Se prueban varias combinaciones de aumentar dimensiones de transistores conjuntamente, para observar el cambio de los tiempos de Setup y Hold. Se decide aumentar las dimensiones de los transistores que disminuyan los tiempos de Setup y Hold.

A continuación se muestra una tabla, donde se comparan los tiempos obtenidos al variar ciertos transistores conjuntamente.

	N0, N8, N9 = 4.05 µm	**N1, N8, N9 = 4.05 µm**	**N0, N1, N8, N9 = 4.05 µm**	**N6, N7, N8, N9 = 4.05 µm**	**N0, N6, N7, N8, N9 = 4.05 µm**
Setup 0 -> 1 (ps)	-222	-405	-146	-362	-223
Setup 1 -> 0 (ps)	-355	-325	-352	-336	-362
Hold 0 -> 1 (ps)	-52	-40	-62	195	-49
Hold 1 -> 0 (ps)	-156	-331	-103	-456	-153

Tabla 17. Comparación entre los tiempos de Setup y Hold al aumentar dimensiones de manera conjunta.

El caso de aumentar N0, N1, N8, N9 a 4.05 µm, otorga la mejor relación de tiempos de Setup y Hold de las varias combinaciones de dimensiones analizadas. A partir del aumento de estas dimensiones, se construirá el flip flop y se utilizará en el circuito digital integrado por flip flops D.

A continuación, se muestra la caracterización y obtención de parámetros del flip flop ajustado, aumentando las dimensiones de los transistores N0, N1, N8, N9 a 4.05 µm.

6.2.14 4.4.1 – Caracterización de tiempos de rampa, propagación y jitter de circuito Flip Flop D pre-layout ajustado:

4.4.1.1 –Tiempos de propagación, TpHL, TpLH de circuito Flip Flop D pre-layout ajustado:

En la tabla 18 se muestran los tiempos medidos de propagación del flip flop pre-layout ajustado.

Flip Flop Pre-layout ajustado	
TpHL (ps)	**TpLH (ps)**
1857	1833

Tabla 18. TpHL y TpLH de Flip Flop D pre-layout ajustado.

4.4.1.2 –Tiempos de rampa, Trise, Tfall de circuito Flip Flop D pre-layout ajustado:

En la tabla 19 se muestran los tiempos medidos de rampa del flip flop pre-layout ajustado.

Flip Flop Pre-layout ajustado	
Trise (ps)	**Tfall (ps)**
1207	903

Tabla 19. Trise y Tfall de Flip Flop D pre-layout ajustado.

4.4.1.3 –Jitter de circuito Flip Flop D pre-layout ajustado:

En las figuras 23 se muestra el jitter de la salida Q, obtenido para el flip flop pre-layout ajustado.

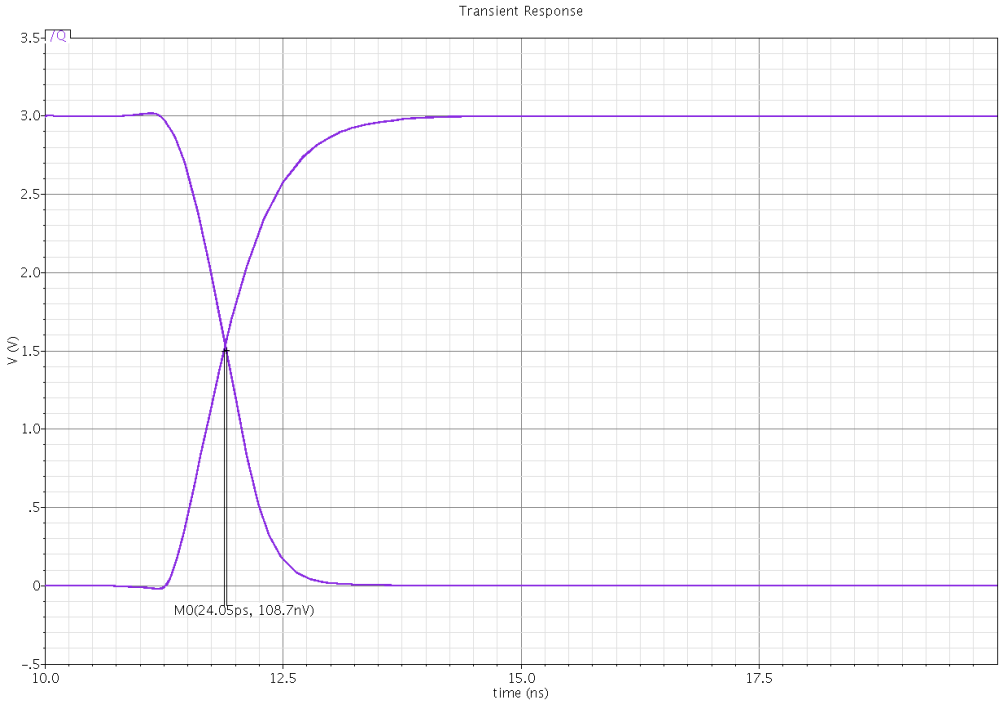


Figura 23. Diagrama de Ojo Flip Flop D pre-layout ajustado.

En la tabla 20, se muestra el valor obtenido de Jitter.

Flip Flop Pre-layout ajustado
Jitter (ps)
24.05

Tabla 20. Jitter de Flip Flop D pre-layout ajustado.

6.2.15 4.4.2 – Caracterización de setup y hold del circuito Flip Flop D pre-layout ajustado:

Las gráficas de tsu y thold obtenidas para el flip flop pre-layout ajustado se muestran en la siguiente figura:

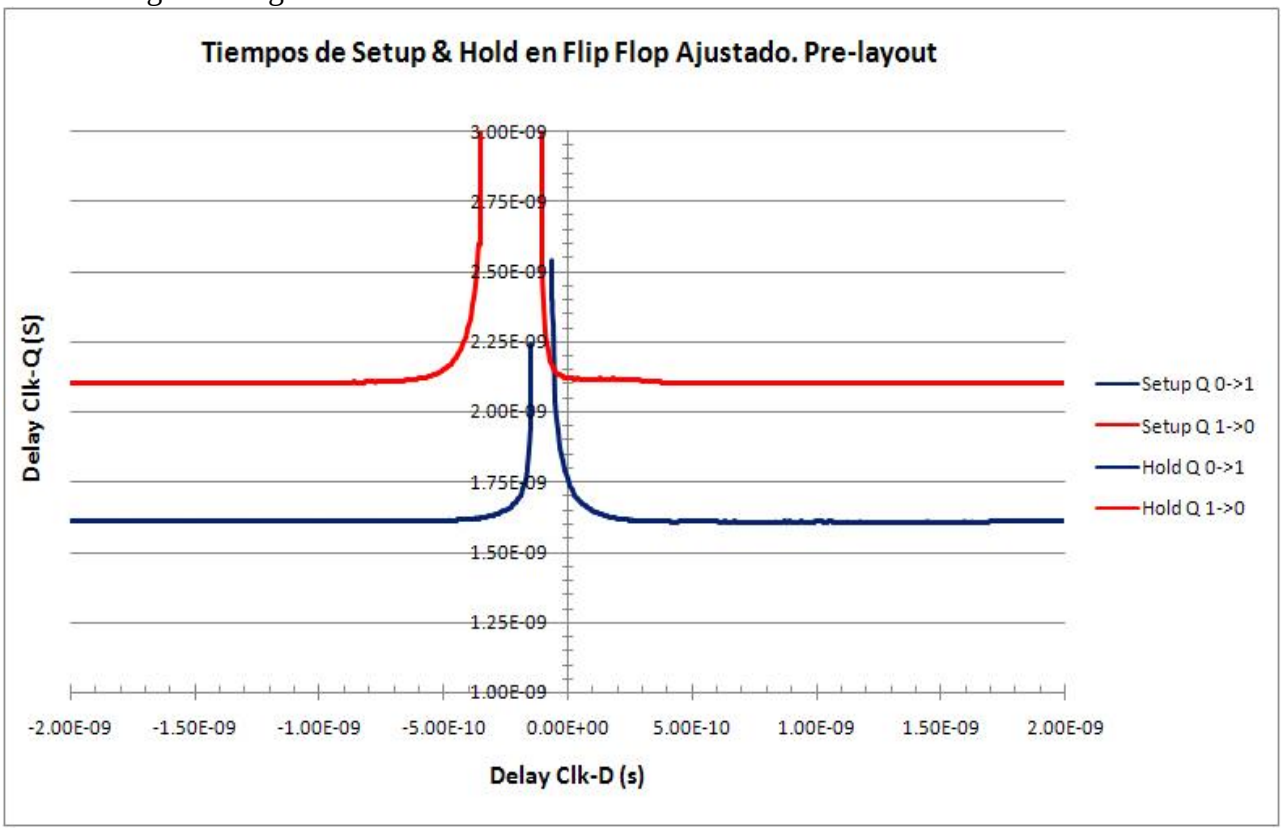


Figura 24. Tiempos de Setup y Hold para Flip Flop D pre-layout ajustado.

En la siguiente tabla se insertan los valores obtenidos para tiempos de setup, hold y los tiempos de propagación de la señal de salida en setup y hold.

	Setup	Hold
	Flip Flop Pre-layout ajustado	
	Q 0 ->1	
Clk -> Q (ps)	2243	2538
Clk -> D (ps)	-146	-62
	Q 1 ->0	
Clk -> Q (ps)	3500	3027
Clk -> D (ps)	-352	-103

Tabla 21. Tiempos de Setup, Hold y propagación para Flip Flop D pre-layout ajustado.

6.2.16 4.4.3 – Potencia del Flip Flop de circuito Flip Flop D pre-layout ajustado:

En la siguiente tabla se muestran los valores de potencia máxima y potencia RMS:

Potencia / Flip Flop	**Flip Flop Pre-layout ajustado**
Potencia Máxima (µW)	1334
Potencia RMS (µW)	351.9

Tabla 22. Potencia Máxima y potencia RMS para Flip Flop D pre-layout ajustado.

6.2.17 4.4.4 – Frecuencia Máxima de circuito Flip Flop D pre-layout ajustado:

La siguiente tabla muestra la frecuencia máxima real y teórica de operación del flip flop pre-layout ajustado:

Frecuencia / Flip Flop	**Flip Flop Pre-layout ajustado**
Frecuencia Máxima Teórica (MHz)	2197
Frecuencia Máxima Real (MHz)	580.3

Tabla 23. Frecuencia máxima para Flip Flop D pre-layout ajustado.

6.2.18 4.5 – Comparación entre Flip Flops D.

FF	TpHL (ps)	TpLH (ps)	Trise (ps)	Tfall (ps)	Jitter (ps)	Setup 0->1 (ps)	Setup 1->0 (ps)	Hold 0->1 (ps)	Hold 0->1 (ps)
No Ajustado	1756	1692	1219	910	64.52	-468	-341	-57	-370
Post-layout	1711	1583	1168	810	128.1	-246	-53	91	-189
Ajustado	1857	1833	1207	903	24.05	-146	-352	-62	-103

FF	Potencia Máxima (µW)	Potencia RMS (µW)	Frecuencia Máxima Teórica (MHz)	Frecuencia Máxima Real (MHz)
No Ajustado	1004	310.8	1193	581.3
Post-layout	1114	315	2229	720
Ajustado	1334	351.9	2197	580.3

Tabla 24. Tabla comparativa de resultados entre Flip Flops.

La tabla 24 nos muestra las diferencias que existen entre el flip flop con lógica Reset-Set en nodo esclavo. Se observa que el desempeño en frecuencia, tiempos de rampa y propagación en el análisis post-layout es mejor que en los casos pre-layout. Esto debido a que la herramienta de Cadence calcula el efecto de capacitancias debido a los perímetros y áreas de los transistores que se insertan en el esquemático. Estas capacitancias representan un análisis pesimista de las capacitancias parásitas que pueden ser mayores que las capacitancias parásitas extraídas del layout.
A pesar de reducir los tiempos de Setup y Hold del transistor, se tiene una frecuencia de operación similar al flip flop no ajustado. Esto debido a los tiempos de propagación que aumentaron al ajustar las dimensiones de algunos transistores. Estos tiempos de propagación influyen en gran medida en la frecuencia de operación del circuito, ya que si son violados, el circuito no opera.
Las potencias de los tres circuitos fueron similares, teniendo una mayor potencia en el circuito ajustado al tener transistores con dimensiones mayores a los demás.

6.2.19 5 – Contador de anillo módulo 8.

El circuito secuencial que se escogió para comparar su desempeño cuando se optimizan los tiempos de Setup y Hold, es el contador que se muestra en la figura 25. [2]
Este contador es un contador de anillo equivalente a un contador módulo 8. Cada salida Q de los 8 flip flops que constituyen el circuito, determinarán la cuenta. Tiene la característica

que la frecuencia de operación del contador está determinada por el tiempo de propagación de las señales de los flip flops.

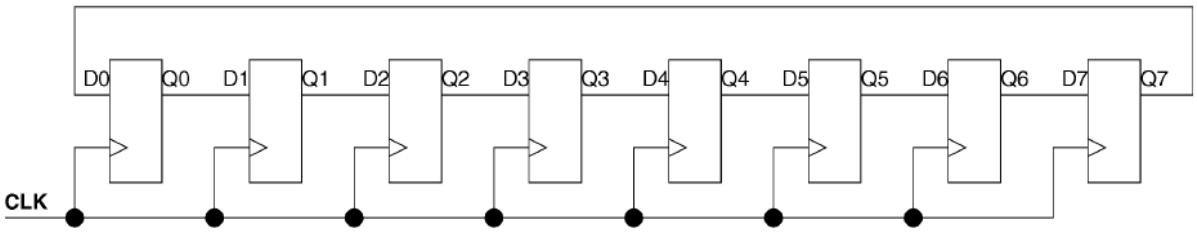


Figura 25. Contador de Anillo Módulo 8.

Este contador se denomina de anillo porque la salida del último flip flop, Q7, va conectada a la entrada del primer flip flop, D0, repitiéndose la secuencia de conteo. El contador es considerado de módulo 8, debido a que contiene 8 distintas etapas, donde cada flip flop genera una salida cada octavo período de la señal de reloj.
Debido a la ausencia de una señal de entrada externa, se implementaron las señales de Reset y Set en los flip flops.
El contador de anillo módulo 8 se implementa usando los flip flops D con lógica Reset-Set en nodo esclavo, realizando análisis pre-layout y post-layout. Luego, se implementó usando el flip flop D ajustado, realizando análisis pre-layout. Las mediciones de parámetros, se realizaron de la misma manera que un flip flop individual.

6.2.20 5.1 – Esquemáticos del Contador de anillo módulo 8.

El esquemático en Cadence del contador de anillo módulo 8 se muestra en la figura 26. El esquemático sigue la topología mostrada en la figura 25, conformado por 8 flip flops, donde cada salida Q flip flops, representa una salida del contador. Cada flip flop consta de una señal de Set diferente, pero la señal de Reset es común para todos los flip flops.

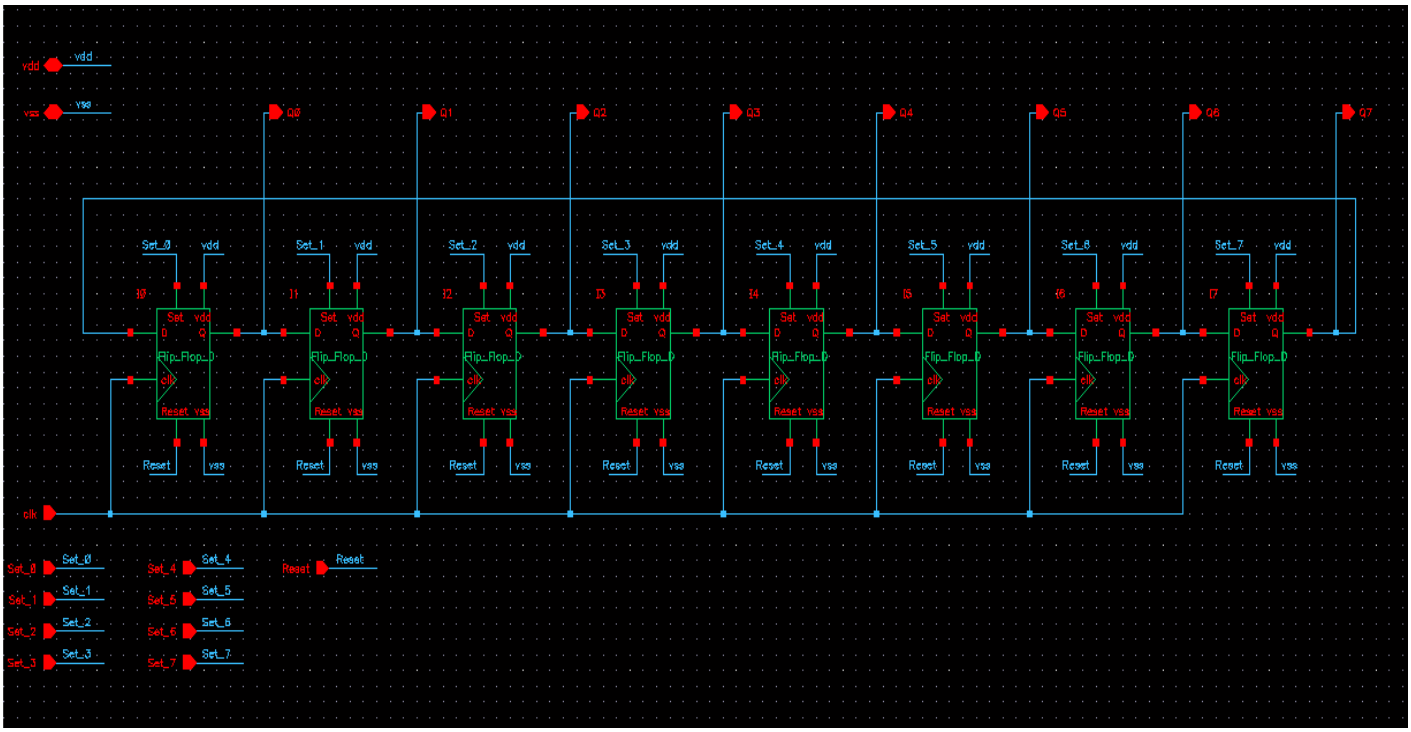


Figura 26. Esquemático de Contador de Anillo Módulo 8.

Para comprobar el funcionamiento y realizar la caracterización del contador de anillo, se construye en la herramienta Cadence, el siguiente banco de pruebas, mostrado en la figura 27.

El banco de pruebas está constituido por los siguientes elementos:

- Fuente de voltaje DC de alimentación para vdd = 3V.
- Fuente de voltaje DC de alimentación para vss = 0V.
- Fuente de voltaje vpwl para generar señal de Reset.
- Ocho fuentes de voltaje vpwl para generar señal de Set de cada flip flop del contador.
- Fuente de voltaje de pulsos para generar la señal de entrada clk, con las siguientes características:
  - Delay Time = 0s
  - Fall Time y Rise Time = 100 ps
  - Ancho de pulso = Variable
  - Periódo = Variable
- Capacitor de carga CL = 50f F.

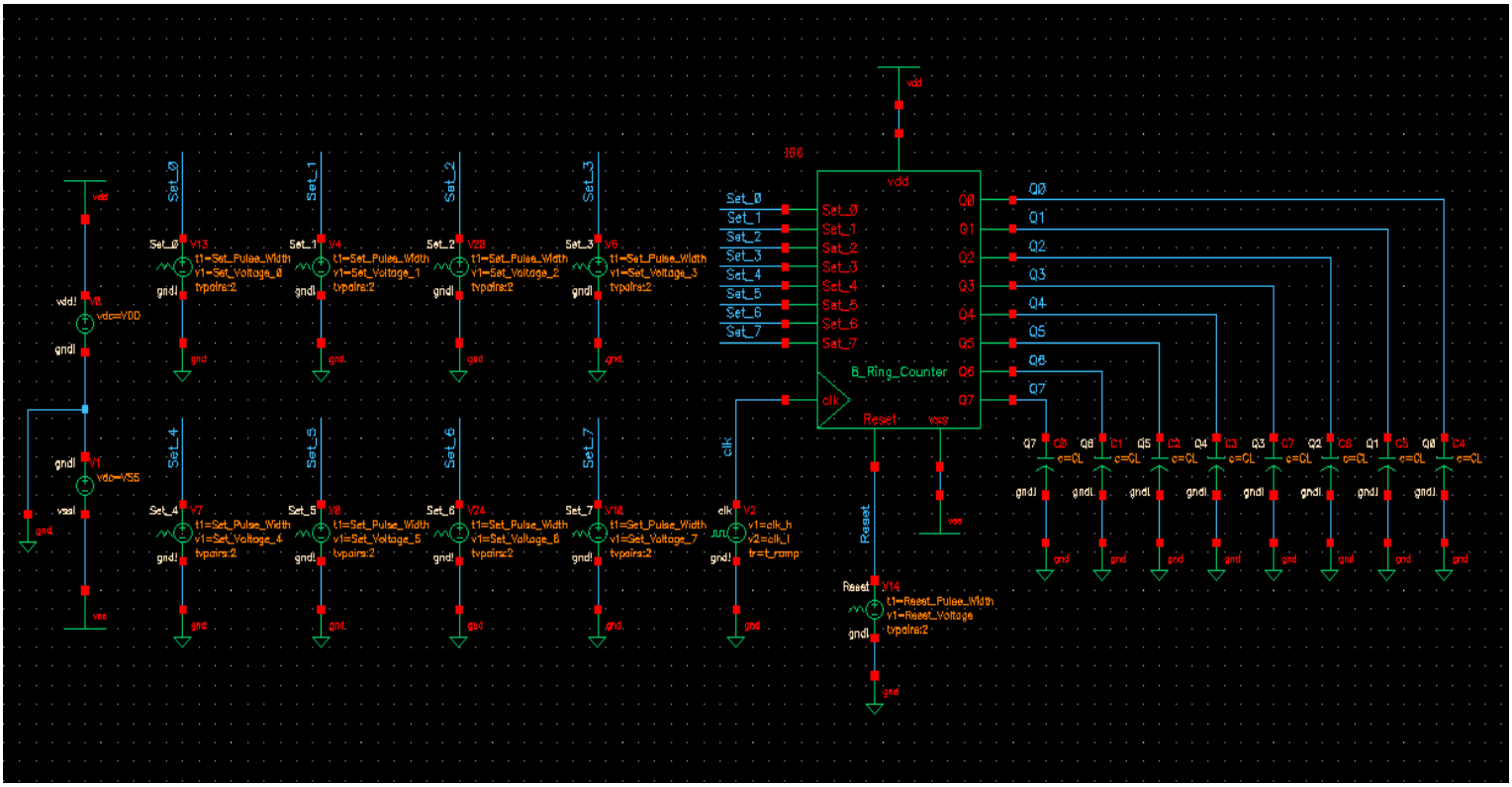


Figura 27. Banco de pruebas de Contador de Anillo Módulo 8.

6.2.21 5.2 – Layout del Contador de anillo módulo 8.

Para este proyecto se realiza el layout del contador de anillo módulo 8, para observar el análisis post-layout del circuito y compararlo con el análisis pre-layout. El layout del contador de anillo es construido a partir de las dimensiones del flip flop no ajustado.

El diseño de layout de registros y contadores sugiere que los flip flops que componen el circuito estén unos tras otro, en línea, de tal forma que la salida de un flip flop recorra una distancia menor a la entrada del siguiente flip flop, teniendo como ventaja que se reduce el efecto de capacitancia al insertar líneas de metal de menor longitud.

El contador de anillo propuesto no tiene lógica entre cada lfip flop, por lo tanto, se puede insertar los flip flop en líneas. El compromiso radica en que la salida del último flip flop es la entrada del primer flip flop, por lo tanto es necesaria una conexión entre estos dos flip flops.

Una opción de layout es poner los transistores en línea e insertar una conexión de metal de longitud grande para conectar el primer flip flop con el último. Esto generaría una capacitancia significativa y provocaría un tiempo de propagación alto en un flip flop.
El layout a diseñar, consiste en poner cuatro flip flops en línea y colocar abajo del cuarto flip flop, el quinto flip flop. Luego, se inserta el resto de flip flops en línea de tal forma que la salida del último flip flop este lo más cerca posible de la entrada del primero. Esto genera el compromiso de dos conexiones de metal entre las salidas y entradas de los flip flop mayores que el resto de las demás. Sin embargo, estas conexiones son menores que la propuesta anteriormente donde todos los transistores están en línea. Debido a que el el flip flop más lento de un circuito digital es el que determina la velocidad del circuito, se busca evitar un flip flop de respuesta lenta.
El área del layout del contador de anillo es de 178.5 µm en X y 52.290 µm en Y.
El layout del contador de anillo módulo 8 se muestra a continuación:

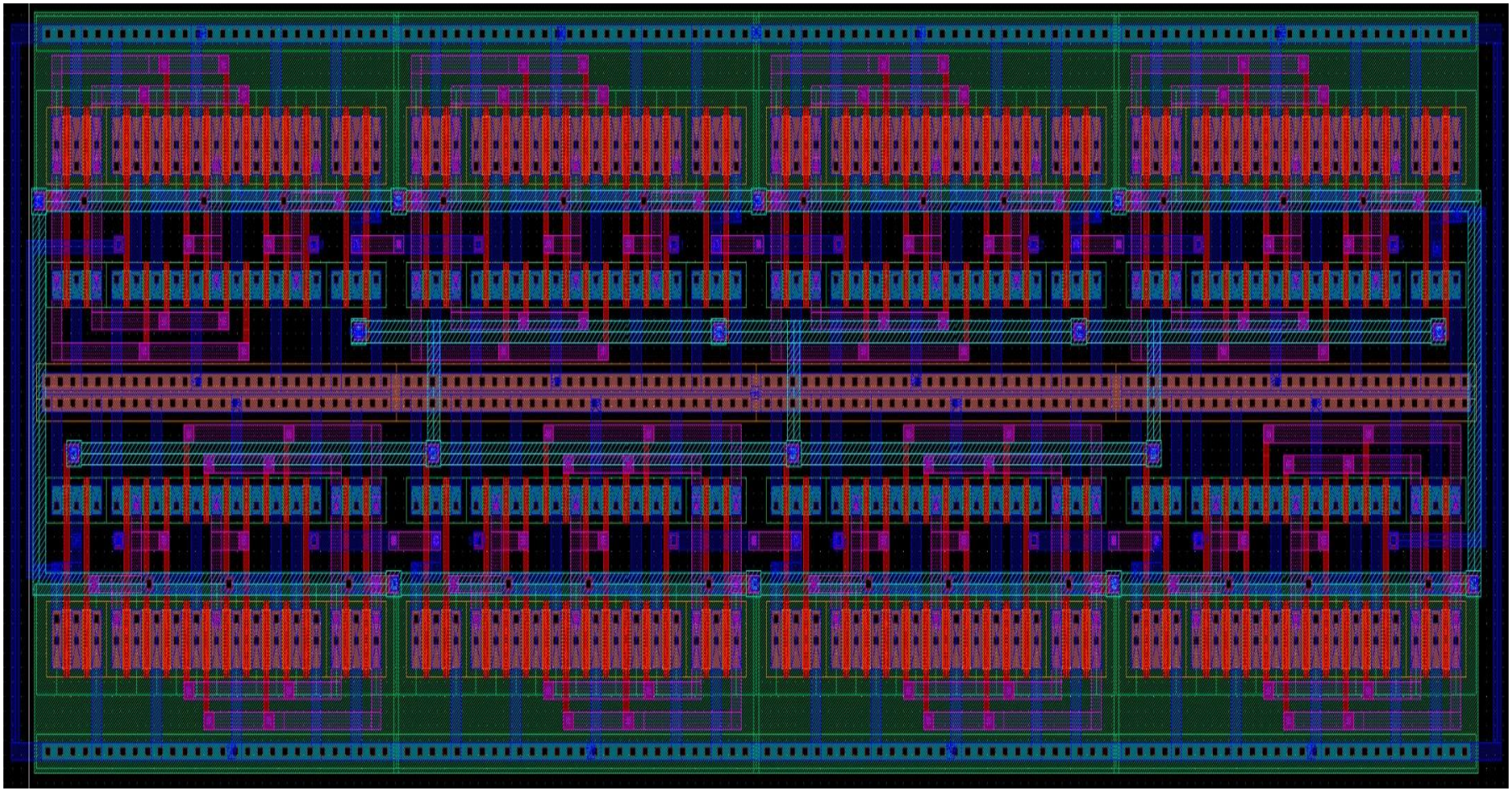


Figura 28. Layout del Contador de Anillo Módulo 8.

El análisis DRC y LVS del contador de anillo se presentan en la figura 29.

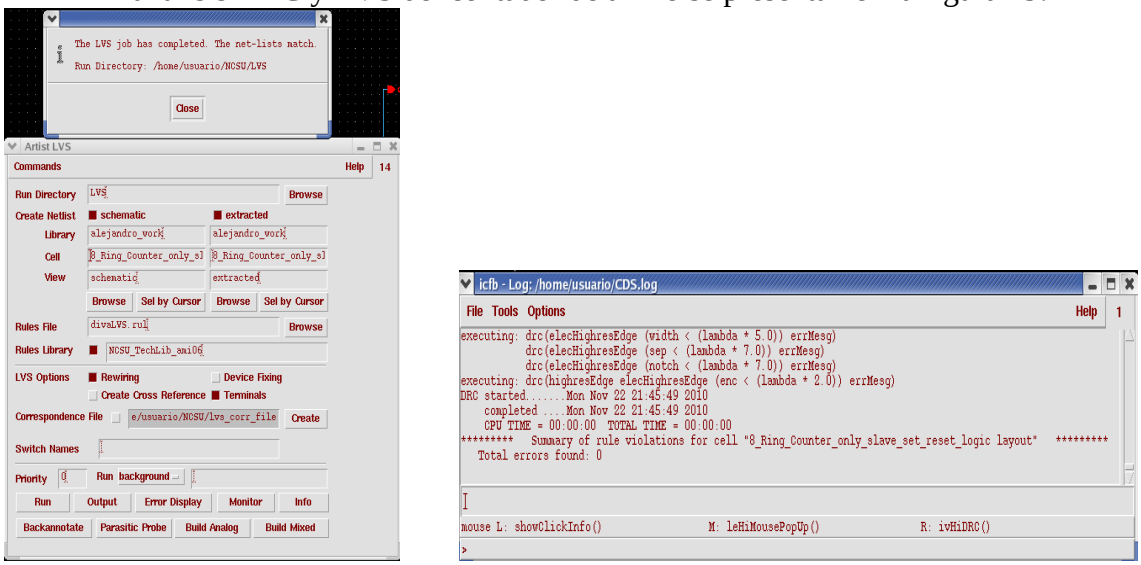


Figura 29. Análisis DRC y LVS del Contador de Anillo Módulo 8.

6.2.22 5.3 – Funcionamiento del Contador de anillo módulo 8.

El funcionamiento del contador de anillo módulo 8 se muestra en la figura 30. Se observa que la señal reset es activada al inicio, para reiniciar los valores de salida de los flip flop a 0. Luego, la señal de reset es desactivada y la seña de Set_0, que se encuentra activada, genera un 1 de salida en el flip flop 0. Al activar este flip flop, la señal de salida pasa al siguiente flip flop y el valor en el segundo flip flop es capturado en el siguiente flanco de reloj. El valor de salida de cada flip flop es capturado en el siguiente flip flop, hasta que 8 ciclos de reloj después, se genera un dato de entrada para el primer flip flop y se repite el conteo hasta que las señales de Reset o Set de los flip flop se activen.

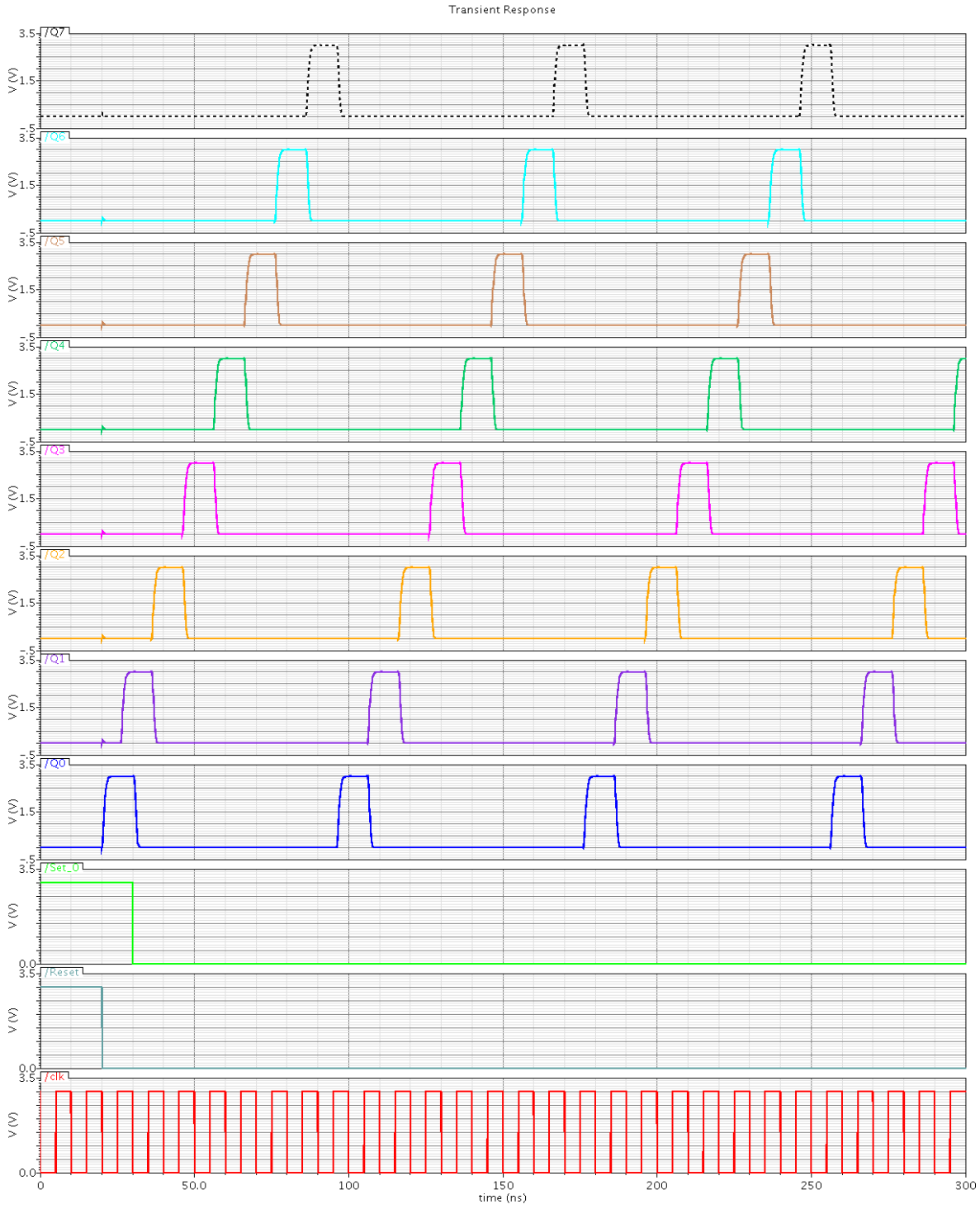


Figura 30. Funcionamiento del Contador de Anillo Módulo 8.

6.2.23 5.4 – Caracterización de tiempos de rampa y propagación del Contador de anillo módulo 8.

5.4.1 –Tiempos de propagación, TpHL, TpLH del Contador de anillo módulo 8:

En la tabla 25 se muestran los tiempos medidos de propagación de los flip flops que conforman el contador de anillo módulo 8 pre-layout sin ajustar y ajustado. Los tiempos para post-layout son presentados en la tabla 26, debido a que los tiempos de los flip flops pueden variar individualmente dependiendo su localización física dentro del layout.

Contador No ajustado		**Contador Ajustado**	
TpHL (ps)	**TpLH (ps)**	**TpHL (ps)**	**TpLH (ps)**
1800	1766	1924	1937

Tabla 25. TpHL y TpLH de Contador de Anillo Módulo 8 pre-layout.

Contador Post-Layout		
Flip Flop	**TpHL (ps)**	**TpLH (ps)**
I0	1823	1736
I1	1841	1737
I2	1841	1736
I3	1878	1785
I4	1823	1736
I5	1839	1734
I6	1839	1735
I7	1874	1782

Tabla 26. TpHL y TpLH de Contador de Anillo Módulo 8 post-layout.

5.4.2 –Tiempos de rampa, Trise, Tfall del Contador de anillo módulo 8:

En la tabla 27 se muestran los tiempos de rampa de los flip flops que conforman el contador de anillo módulo 8 pre-layout sin ajustar y ajustado. Los tiempos para post-layout son presentados en la tabla 28, individualmente por cada flip flop.

Contador No ajustado		**Contador Ajustado**	
Trise (ps)	**Tfall (ps)**	**Trise (ps)**	**Tfall (ps)**
1389	996.7	1460	1060

Tabla 27. Trise y Tfall del Contador de Anillo Módulo 8 pre-layout.

Contador Post-Layout		
Flip Flop	**Trise (ps)**	**Tfall (ps)**
I0	1399	963.6

I1	1403	964.3
I2	1403	963.6
I3	1506	1024
I4	1401	963.8
I5	1404	963.5
I6	1404	963.6
I7	1503	1020

Tabla 28. Trise y Tfall del Contador de Anillo Módulo 8 post-layout.

Los tiempos de rampa y propagación resultan ser mayores para los flip flops que se encuentra en las orillas, donde las conexiones de dato de salida del flip flop al dato de entrada del siguiente flip flop son de mayor longitud respecto a las otras.

6.2.24 5.5 – Caracterización de setup y hold del Contador de anillo módulo 8.

Las gráficas de tsu y thold obtenidas para los flip flops que conforman el contador de anillo módulo 8 pre-layout sin ajustar y ajustado se presentan a continuación:

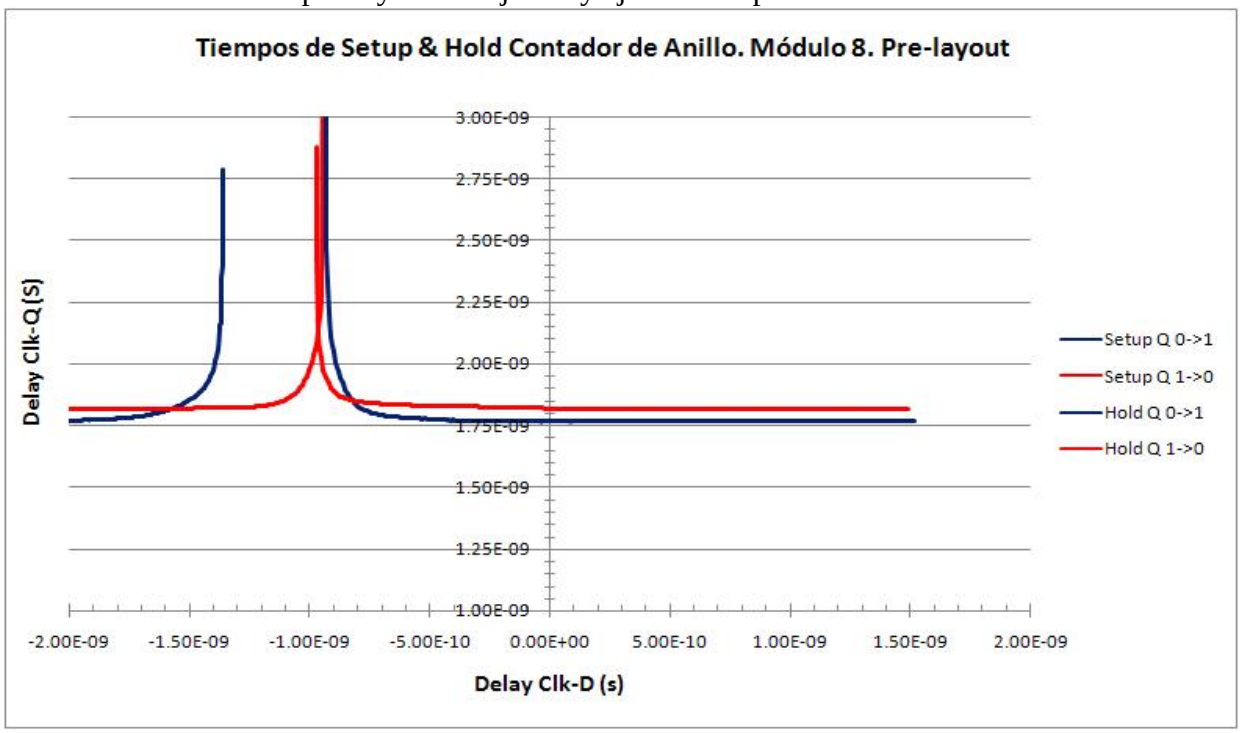


Figura 31. Tiempos de Setup y Hold del Contador de Anillo Módulo 8 pre-layout no ajustado.

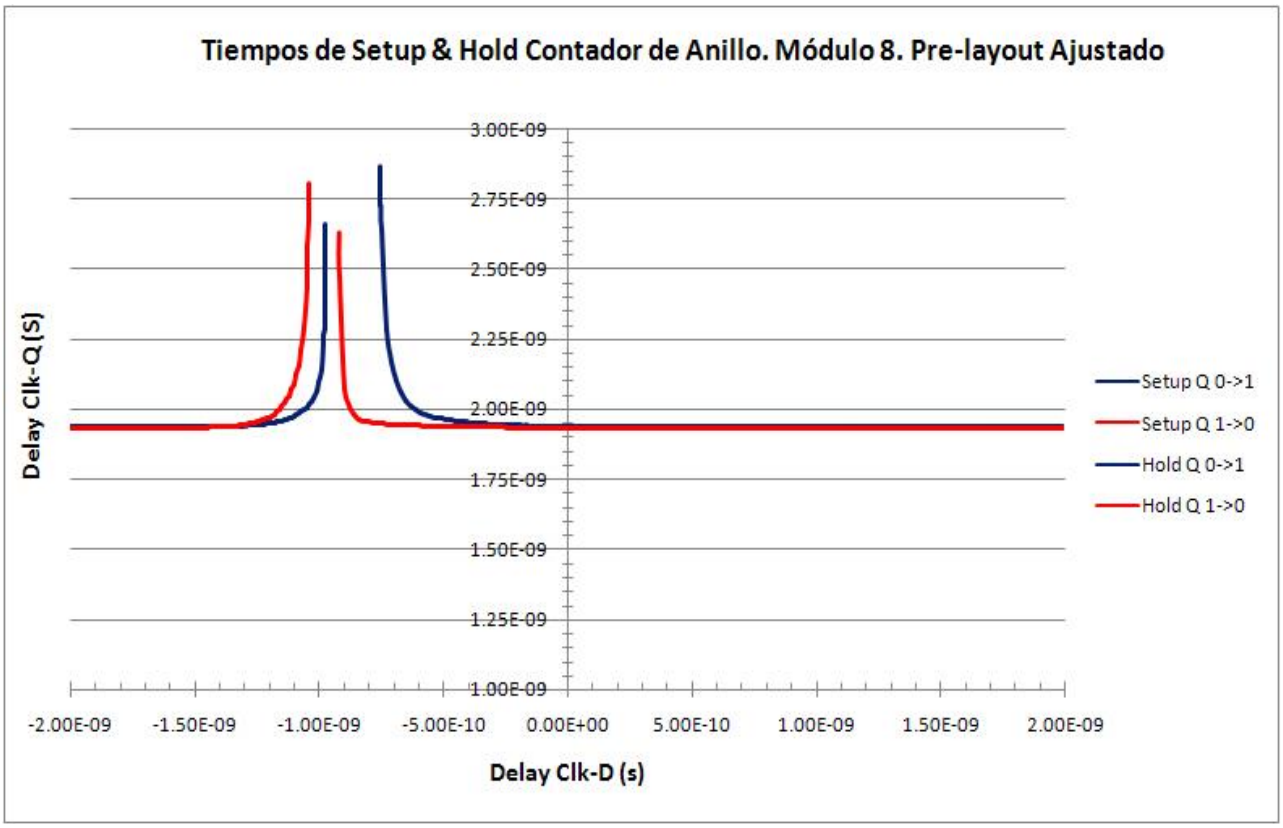


Figura 32. Tiempos de Setup y Hold del Contador de Anillo Módulo 8 pre-layout ajustado.

En la tabla 29 se insertan los valores obtenidos para tiempos de setup, hold y los tiempos de propagación de la señal de salida en setup y hold. Los tiempos individuales de cada flip flop del contador post-layout se muestran en la tabla 30.

	Setup		**Hold**	
	Contador No ajustado	**Contador Ajustado**	**Contador No ajustado**	**Contador Ajustado**
	Q 0 ->1			
Clk -> Q (ps)	2789	2660	3050	2868
Clk -> D (ps)	-1360	-973	-933	-752
	Q 1 ->0			
Clk -> Q (ps)	3609	2808	2878	2630
Clk -> D (ps)	-940	-1042	-972	-921

Tabla 29. Tiempos de Setup, Hold y propagación del Contador de Anillo Módulo 8 pre-layout.

		Setup		**Hold**	
Flip Flop	**Transición**	**Clk -> Q (ps)**	**Clk -> D (ps)**	**Clk -> Q (ps)**	**Clk -> D (ps)**

I0	**Q 0 ->1**	2335	-1212	2442	-507
	Q 1 ->0	2892	-669	2417	-1149
I1	**Q 0 ->1**	2452	-1142	2537	-457
	Q 1 ->0	2921	-617	3221	-1082
I2	**Q 0 ->1**	2452	-1142	2537	-457
	Q 1 ->0	2919	-617	3220	-1082
I3	**Q 0 ->1**	2502	-1142	2588	-457
	Q 1 ->0	2885	-617	3170	-1082
I4	**Q 0 ->1**	2335	-1212	2442	-507
	Q 1 ->0	2890	-669	2417	-1149
I5	**Q 0 ->1**	2680	-1141	2471	-459
	Q 1 ->0	2524	-620	2787	-1083
I6	**Q 0 ->1**	2613	-1143	2471	-459
	Q 1 ->0	2407	-621	2785	-1083
I7	**Q 0 ->1**	2665	-1143	250	-459
	Q 1 ->0	2557	-620	2790	-1083

Tabla 30. Tiempos de Setup, Hold y propagación del Contador de Anillo Módulo 8 post-layout.

Se observa que los flip flop con peores tiempos de setup y hold son el I0 e I4. Esto debido a que dependen de la señal de salida de I7 e I3 respectivamente, las cuales tiene mayores tiempos de propagación y rampa.

6.2.25 5.6 – Potencia del Contador de anillo módulo 8.

En la siguiente tabla se muestran los valores de potencia máxima y potencia RMS del contador de anillo módulo 8:

Potencia / Contador	**Contador No ajustado**	**Contador Post-Layout**	**Contador Ajustado**
Potencia Máxima (µW)	7736	12400	8231
Potencia RMS (µW)	1600	1999	1713

Tabla 31. Potencia Máxima y potencia RMS del Contador de Anillo Módulo 8.

6.2.26 5.7 – Frecuencia máxima del Contador de anillo módulo 8.

La siguiente tabla muestra la frecuencia máxima real y teórica de operación del contador de anillo módulo 8:

Frecuencia / Contador	Contador No ajustado	Contador Post-Layout	Contador Ajustado
Frecuencia Máxima Teórica (MHz)	428.8	509	423.5
Frecuencia Máxima Real (MHz)	328.9	364.95	289

Tabla 32. Frecuencia máxima del Contador de Anillo Módulo 8.

6.2.27 5.8 – Comparación entre Contadores de anillo módulo 8.

Contador	TpHL (ps)	TpLH (ps)	Trise (ps)	Tfall (ps)	Setup 0->1 (ps)	Setup 1->0 (ps)	Hold 0->1 (ps)	Hold 0->1 (ps)
No Ajustado	1800	1766	1389	996.7	-1360	-940	-933	-972
Post-layout	1878	1785	1506	1024	-1212	-669	-507	-1149
Ajustado	1924	1937	1460	1060	-973	-1042	-752	-921

Contador	Potencia Máxima (µW)	Potencia RMS (µW)	Frecuencia Máxima Teórica (MHz)	Frecuencia Máxima Real (MHz)
No Ajustado	7736	1600	428.8	328.9
Post-layout	12400	1999	509	364.95
Ajustado	8231	1713	423.5	289

Tabla 33. Tabla comparativa de resultados entre Contadores de Anillo Módulo 8.

6.2.28 De la tabla 33 se observa que el desempeño del post layout es mejor que los contadores pre-layout, por la misma razón de que Cadence realiza cálculos pesimistas de capacitancias de los esquemáticos.

Se observa que los tiempos de propagación del contador ajustado provocaron un desempeño más lento a pesar de tener tiempos de setup y hold menores. Los tiempos de propagación y rampa representan un factor clave en este tipo de circuitos, debido a que la entrada de un flip flop depende de la señal de salida del anterior. Si la señal de salida del flip flop anterior consta de un tiempo de rampa y propagación alto, el siguiente flip flop no podrá capturar el dato de entrada en el flanco positivo de la señal de reloj, debido a que no se ha alcanzado el voltaje de entrada necesario para provocar una transición de alto a bajo o de bajo a alto.

6.2.29 6 – Conclusiones

- Dentro del diseño de circuitos digitales se debe de tener en cuenta los tiempos de operación de los Flip Flops, ya que al ser violados, se presenta un funcionamiento incorrecto de nuestro circuito. Es necesario realizar una caracterización detallada del flip flop a usar dentro de un sistema secuencial, ya que podremos determinar la frecuencia máxima de operación de todo nuestro sistema. Aunque en este proyecto se ajustaron los tiempos de setup y hold, se experimentó con el problema del aumento de tiempos de propagación de las señales de salida. Estos tiempos de propagación representan tiempos significativos para la operación del circuito, ya que no se puede generar una señal en un flanco de reloj, si la señal de entrada tiene un tiempo de propagación alto.
- La metodología para reducir los tiempos de setup and hold del flip flop fue correcta, ya que se logró reducir estos tiempos, pero esto no garantizó una mejora en la frecuencia de operación del circuito. Como se mencionó en al descripción del contador de anillo, su frecuencia esta determinada por los tiempos de propagación de los flip flops que lo componen. Al mejorar los tiempos de setup and hold, se tuvo una mejora del jitter, pero los tiempos de propagación empeoraron. El compromiso en el diseño de circuitos secuenciales, radica en diseñar circuitos con buenos tiempos de setup y hold y que los tiempos de propagación no se incrementen.
- Se observó durante la caracterización del flip flop y del contador, que al aumentar la velocidad de nuestro circuito, tenemos un incremento significativo de la potencia consumida. Esto representa una de las relaciones más importantes que hay que tomar en cuenta al diseñar un circuito. Los circuitos digitales de alta velocidad que se usan hoy en día, tienen que compensar el aumento de la velocidad con el aumento de la potencia consumida, provocando que la tendencia actual sea diseñar circuitos de alta velocidad con la menor potencia consumida.

6.2.30 7 – Referencias

- [1] S. Heo and K. Asanović, “Load-Sensitive Flip-Flop Characterization”, *IEEE Workshop on VLSI*, Orlando, Florida, April 2001.
- [2] M. Stan, A. Tenca and M. Ercegovac, “Long and Fast Up/Down Counters”, *IEEE Transactions on Computers*, VOL. 47, NO. 7, July 2008.
- [3] V. Oklobdzija, “Multi-GHz Systems Clocking”, Fellow IEEE, Department of Electrical Engineering, University of California.
- [4] “Flip Flop Design with Option of Set/Reset, Clear or Enable”, Course NO: ECE 491, University of Tennessee, Spring 2007.
- [5] N. Nedovic, M. Aleksic and V. Oklobdzija, “Timing Characterization of Dual-Edge Triggered Flip-Flops”, Department of Electrical and Computer Engineering, University of California.
- [6] S. Kang, Y. Leblebici. “CMOS Digital Integrated Circuits”, Mc Graw Will.
- [7] M. Villafranco, “Metodología de caracterización y diseño de un Flip Flop D topología Power PC”, Curso de Diseño Digital I, ITESO, 2009.

C. PROYECTO FINAL DE VERIFICACION DE SISTEMAS DIGITALES

ITESO

Universidad Jesuita
de Guadalajara

Verificación de Sistemas Digitales

“Verificación de un Microprocesador”

Alejandro Güereña Morán	*679705*
Marcos Manuel Becerra Ulloa	*679780*
Marcos I. Bolaños Valerio	*100575*

Plan de Verificación para el Microprocesador

El plan de verificación tiene como objetivo definir qué y cómo se verificará la unidad despachadora o “dispatch” del Microprocesador de 32 bits,

Para este proyecto, el plan de verificación constará de los siguientes puntos.

- La estrategia de verificación
- Áreas funcionales

ESTRATEGIA DE VERIFICACIÓN:

La estrategia de verificación define el enfoque, la jerarquía, sincronización y comparación con modelo de referencia, que permita verificar el circuito digital con detalle, para la máxima detección de errores.

- **Enfoque de la Verificación Funcional:**

El enfoque de este proyecto se define como Caja Gris "Grey Box"). El despachador será verificado como caja gris, donde se verificarán las señales de entrada y salida del bloque despachador, así se verifican las señales internas y las interconexiones entre los bloques que lo conforman. Sin embargo, no se observa la totalidad de este, debido a que no se verifica todas las señales internas.

- **Sincronización y Comparación con Modelo de Referencia:**

-Sincronización: La sincronización consiste en adecuar los tiempos de retardo o “delay” puestos para la estimulación del RTL, con la obtención de resultados del modelo de referencia, de tal forma que se comparen los datos observados con los datos obtenidos del Modelo de Referencia en tiempo real de simulación.

-Comparación: En las tareas que ejecutan los checkers, se realizará la comparación entre los resultados observados del RTL con los resultados obtenidos del Modelo de Referencia, mediante sentencias de tipo IF-ELSE.

- **Jerarquía de Verificación de Hardware:**

Bloque/Cluster :

- "Dispatcher"

  Bloques Internos:

  - Tag Fifo
  - Rst_Mem
  - RegFile
  - Rst

Sistema: Microprocesador.

AREAS FUNCIONALES

Microprocesador:

- Ejecutar Programa
- Procesar instrucciones y operandos de 32 bits

Bloques Principales:

Dispatch

- Interactuar con IFQ, con el CDB y las cuatro colas de ejecución
  - Unidad de dispatch:
    - Leer instrucciones del IFQ.
    - Decodificar instrucciones provenientes de la IFQ. (Entrada directa)
    - Despachar instrucciones a sus respectivas colas de ejecución.
    - Calcular las direcciones de brinco para las instrucciones Jump y Branch.
    - Realizar lógica de detención de IFQ en caso de instrucciones Branch o en caso de colas llenas.

Ambiente de Verificación para el Microprocesador

El ambiente de verificación tiene define los elementos para controlar y observar las áreas funcionales de la unidad despachadora o “dispatch” del Microprocesador de 32 bits,

Para este proyecto, el ambiente de verificación constará de los siguientes puntos.

- Modelo de Referencia
- Interfaces
- Estimulación
- Monitores
- Checadores
- Cobertura

MODELO DE REFERENCIA

El modelo de referencia tiene como objetivo generar los resultados esperados con una abstracción de alto nivel, para comparar estos resultados con los resultados observados del RTL con monitores.

El modelo de referencia es puesto dentro de una clase, para luego realizar una instancia de este en el banco de pruebas, con el objetivo que las tareas donde se ejecutan checadores, obtengan los valores del modelo de referencia para su comparación.

INTERFACES

La conectividad de un módulo es hecha por medio de un grupo de puertos de entrada/salida, o simplemente interface. Las interfaces tienen como función abstraer toda la conectividad entre el DUV y framework. .

Las interfaces definidas para la unidad despachadora son definidas de acuerdo a las señales de entrada-salida y las señales internas.

Interfaces:

DispatchIf DisIf: Interface para señales de entrada y salida del despachador.
DispatchIntIf DisIntIf: Interface para señales internas del despachador.

Las señales y su nombre con interface, se presentan a continuación

clk		*DisIf.Clk*
reset		*DisIf.Reset*
ifq_pcout_plus4		*DisIf.IFQ_PCOUT_PLUS4*
ifq_inst		*DisIf.IFQ_INST*
ifq_empty		*DisIf.IFQ_EMPTY*
cdb_tag		*DisIf.CDB_TAG*
cdb_valid		*DisIf.CDB_VALID*
cdb_data		*DisIf.CDB_DATA*
cdb_branch	***Interface***	*DisIf.CDB_BRANCH*
cdb_branch_taken	→	*DisIf.CDB_BRANCH_TAKEN*
equeuels_ready		*DisIf.EQUEUELS_READY*
equeueint_ready		*DisIf.EQUEUEINT_READY*
equeuemult_ready		*DisIf.EQUEUEMULT_READY*
equeuediv_ready		*DisIf.EQUEUEDIV_READY*
debug_regfile_addr		*DisIf.DEBUG_REGFILE_ADDR*
ifq_ren		*DisIf.IFQ_REN*
ifq_branch_addr		*DisIf.IFQ_BRANCH_ADDR*

ifq_branch_valid		*DisIf.IFQ_BRANCH_VALID*
equeue_imm		*DisIf. EQUEUE_IMM*
equeue_rdtag		*DisIf.EQUEUE_RDTAG*
equeue_rstag		*DisIf.EQUEUE_RSTAG*
equeue_rttag		*DisIf.EQUEUE_RTTAG*
equeue_rsdata		*DisIf.EQUEUE_RSDATA*
equeue_rtdata		*DisIf.EQUEUE_RTDATA*
equeue_rsvalid		*DisIf.EQUEUE_RSVALID*
equeue_rtvalid		*DisIf.EQUEUE_RTVALID*
equeuels_opcode		*DisIf.EQUEUELS_OPCODE*
equeuels_en		*DisIf.EQUEUELS_EN*
equeueint_opcode		*DisIf.EQUEUEINT_OPCODE*
equeueint_en		*DisIf.EQUEUEINT_EN*
equeuemult_en		*DisIf.EQUEUEMULT_EN*
equeuediv_en		*DisIf.EQUEUEDIV_EN*
debug_regfile_data		*DisIf.DEBUG_REGFILE_DATE*
dispatch.state _r		DisIntIf.state_r
dispatch.inst_opcode		DisIntIf.inst_opcode

ESTIMULACIÓN

La estimulación consiste en generar tareas o funciones que estimulen la unidad despachadora. Las estimulaciones propuestas para la unidad despachadora, según las áreas funcionales propuestas, son las siguientes:

- Send_Instruction_Responder(Paquete):

Elemento reactivo de estimulación que espera la señal de ifq_ren del despachador se encuentre activa, para enviar una instrucción al despachador, así como activar la señal de ifq_empty, que notifica al despachador que existe una nueva instrucción.

Este elemento, permite cumplir con todas las áreas funcionales propuestas, ya que se necesita enviar instrucciones al despachador para verificar estas áreas. Consta de parámetro, una clase de tipo paquete, que pertenece a la capa de generación de estímulos, que son estructuras de datos para construir de manera automática y sencilla paquetes que representan las instrucciones a mandar, así como otras señales.

- Enable_Queues(Paquete): Elemento activo de estimulación que tiene como objetivo generar las señales de ready de las colas. Esta señal indica que una instrucción puede ser despachada a la cola cuya seña ready se encuentra activa. Este elemento es creado para el área funcional de despachar instrucciones a sus respectivas colas de ejecución. También contiene de un parámetro de clase tipo paquete, realizando una estructura que permita activar las señales ready de manera aleatoria o dirigida.

MONITORES

Los monitores consisten en observar señales, para generar eventos que nos permitirán analizar, para checadores o para responders. Los monitores propuestos para la unidad despachadora, según las áreas funcionales propuestas, son las siguientes:

- Run_Monitor(): Elemento pasivo que tiene como objetivo observar las siguientes señales para generar eventos según la áreas funcionales propuestas:

  ifq_ren: Señal que indica que el despachador puede recibir una instrucción.
  equeuels_ready: Señal que indica que la cola puede recibir una instrucción correspondiente a esa cola. (Cola de LS)
  equeueint_ready: Señal que indica que la cola puede recibir una instrucción correspondiente a esa cola. (Cola de enteros)
  equeuemult_ready: Señal que indica que la cola puede recibir una instrucción correspondiente a esa cola. (Cola de multiplicación)
  equeuediv_ready: Señal que indica que la cola puede recibir una instrucción correspondiente a esa cola. (Cola de división)
  ifq_branch_valid: Señal que indica que una instrucción branch o jump fue recibida.
- Las siguientes señales también son monitoreadas, para su análisis en checadores.

  Ifq_branch_addr: Contiene la dirección calculada para instrucción de tipo branch o jump.

equeuels_en: Una instrucción es despachada a la cola de ls.
equeueint_en: Una instrucción es despachada a la cola de enteros.
equeuemult_en: Una instrucción es despachada a la cola de multiplicación.
equeuediv_en: Una instrucción es despachada a la cola de división.

CHECADORES

Para evaluar las respuestas dentro de la simulación, se implementan checadores que tienen como objetivo analizar las funciones del Dispatch, usando los monitores de señales, para revisar sí estas se ejecutaron de manera correcta o incorrecta. Los checadores propuestos para la unidad despachadora, según las áreas funcionales propuestas, son las siguientes:

- Run_Checker_Decode(Modelo de Referencia): Elemento de análisis que comprueba que la decodificación de una instrucción. Mediante el monitoreo de señales internas del despachador, señales state_r e inst_opcode, y comparándolas con los valores obtenidos del modelo de referencia, se logra verificar si se decodificó de manera correcta la instrucción recibida.

- Run_Checker_Address_Calculation(Modelo de Referencia): Elemento de análisis que verifica el cálculo de la dirección para instrucciones de tipo branch o jump. Mediante el monitoreo de las señales ifq_branch_valid e ifq_branch_addr, se compara el valor de esta última señal, que almacena el valor de la dirección calculada por el despachador, con el valor calculado por el modelo de referencia, para verificar si se realizo el cálculo de la dirección de manera correcta.

- Run_Checker_Queues(Modelo de Referencia): Elemento de análisis que comprueba si una instrucción fue despachada a su correcta cola de ejecución. Con el monitoreo de las señales ifq_inst y equeue_en de todas las colas, se compara con el valor del modelo de referencia, que indica que cola debe de recibir el dato, para verificar si se despachó la instrucción a la correcta cola de ejecución.

- Run_Checker_IFQStop(): Elemento de análisis que comprueba sí la IFQ es detenida de enviar instrucciones, cuando se ejecuta una instrucción de tipo branch o cuando las colas de ejecución de se encuentran llenas. Con el monitoreo de las señales

ifq_ren, equeu_ready de todas las colas e ifq_branch_valid se verifica si la IFQ es detenida de enviar instrucciones cuando se detecta las condiciones para detener el envío.

COBERTURA

La cobertura definida para la unidad despachadora nos permite determinar que tanto se han ejercitado las áreas funcionales, definiendo puntos que se deben de ejercer, para alcanzar una cobertura completa.

"Dispatcher"

- Leer instrucción del IFQ
- Decodificar instrucciones de enteros y acceso a memorias
- Decodificar instrucciones de multiplicación
- Decodificar instrucciones de división
- Enviar instrucciones a la cola de enteros y acceso a memorias
- Enviar instrucciones a la cola de multiplicación
- Enviar instrucciones a la cola de división.
- Ejecutar instrucciones de Branch
- Ejecutar instrucciones de jump
- Tratar de despachar una instrucción cuando la cola está llena
- Tratar de despachar una instrucción cuando la cola no está llena

Desarrollo de Pruebas

ESCENARIOS

Escenario

El escenario propuesto para la verificación del bloque despachador a nivel Full Chip, se describe como el contenido de una clase, que se instancia como un objeto dentro del banco de pruebas.

Esta clase escenario contiene los elementos para la generación de estímulos, que consiste de una clase denominada BasePacket, donde se describen distintos tipos de estructuras de datos de alto nivel, para generar las instrucciones que el despachador va a procesar, así como las señales extras para que el despachador decodifique y despache las instrucciones.

El escenario consta de los elementos para generar paquetes aleatorios, con los requeridos constraints para la estimulación del despachador. Estos constraints sirven para realizar escenarios donde se evite enviar instrucciones inválidas (cuando no se requiere verificar el funcionamiento del despachador para instrucciones no definidas) y para realizar pruebas más específicas o dirigidas.

El escenario también cuenta con instancia de la clase generadora de estímulos, para ejecutar las tareas de estimulación del RTL cuando se instancie el objeto escenario en el banco de pruebas o test bench.

ELEMENTOS ALEATORIOS

Elementos aleatorios fueron definidos dentro de la clase de generación de estímulos. Esta clase de generación de estímulos, BasePacket, consta de estructuras de tipo rand o aleatorias, que con la función randomize(), se pueden definir como aleatorias.

Los campos de las tramas que forman las instrucciones, fueron especificadas dentro de la estructura, de tal manera que se pueda aleatorizar de forma individual cada uno de los campos de las instrucciones.

Las estructuras que determinan que tipo de instrucción se enviará al despachador, así como la cola de ejecución que está lista para recibir instrucciones también pueden son aleatorias.

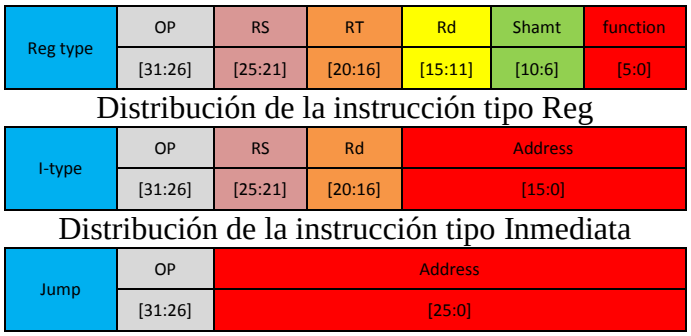


Figura. Campos Aleatorios de las Instrucciones

ELEMENTOS DIRIGIDOS

Los contraints representan elementos para realizar pruebas dirigidas, ya que al modificar las condiciones como los valores de estos constraints, se tienen pruebas digiridas de acuerdo a lo que específicamente se quiere verificar.

Se insertan constraints para el tipo de instrucciones a mandar a la unidad despachadoras.

Resultados

Entre los resultados obtenidos, se observó que existe error de diseño de la unidad despachadora, debido a que al enviar correctamente las señales de instrucción y de empty, no existe una decodificación, ya varias de las señales adquieren valores indefinidos, indicando que existe un problema de inicialización de dichas señales con el reset.

Las señales que almacenan los valores de los campos que conforman la instrucción, adquieren un valor, pero luego no existe una correcta decodificación, donde se observa que simplemente las señales jamás adquiere un valor definido, quedándose en valor indefinido.

Anexos

En esta sección de anexos, se insertan los archivos en lenguaje SystemVerilog, implementados para la verificación del despachador.

Interfaces.sv contiene las interfaces de conexión con el RTL, así como la instanciación del modulo de dispatcher.

Verification.sv contiene todo el ambiente de verificación a realizar, así como el escenario y el banco de pruebas a ejecutar. Lo ideal es separar cada clase en distintos archivos e incluirlos, pero se tuvo algunos problemas con ello de compilación.

Dispatch_RM.sv contiene la implementación a nivel comportamental del funcionamiento del despchador. A partir de los datos obtenidos de este modelo de referencia, se hacen comparaciones con los datos observados en el ambiente de verificación.

Interfaces.sv

```
// ====================================
// Dispatch Interface
// ====================================
interface DispatchIf(input Clk, Reset);

   //Signals

   /****** IFQ Connection Signals ******/

   //Input Signals
   logic[31:0] IFQ_PCOUT_PLUS4;
   logic[31:0] IFQ_INST;
   logic       IFQ_EMPTY;

   //Output Signals
   logic       IFQ_REN;
   logic[31:0] IFQ_BRANCH_ADDR;
   logic       IFQ_BRANCH_VALID;

   /****** CDB Connection Signals ******/

   //Input signals
   logic[5:0]  CDB_TAG;
   logic       CDB_VALID;
   logic[31:0] CDB_DATA;
   logic       CDB_BRANCH;
   logic       CDB_BRANCH_TAKEN;

   /****** Common Queues Connection Signals ******/


   //Output signals
   logic[15:0] EQUEUE_IMM;
   logic[5:0]  EQUEUE_RDTAG;
   logic[5:0]  EQUEUE_RSTAG;
   logic[5:0]  EQUEUE_RTTAG;
   logic[31:0] EQUEUE_RSDATA;
   logic[31:0] EQUEUE_RTDATA;
   logic       EQUEUE_RSVALID;
   logic       EQUEUE_RTVALID;

   /****** Load/Store Queues Connection Signals ******/

   //Input Signals
   logic       EQUEUELS_READY;

   //Output Signals
```

```
logic       EQUEUELS_OPCODE;
logic       EQUEUELS_EN;

/****** Integer Queues Connection Signals ******/

//Input Signals
logic       EQUEUEINT_READY;

//Output Signals
logic[3:0]  EQUEUEINT_OPCODE;
logic       EQUEUEINT_EN;

/****** Mult Queues Connection Signals ******/

//Input Signals
logic       EQUEUEMULT_READY;

//Output Signals
logic       EQUEUEMULT_EN;


/****** Div Queues Connection Signals ******/

//Input Signals
logic       EQUEUEDIV_READY;

//Output Signals
logic       EQUEUEDIV_EN;

/****** Debug RegFile Connection Signals ******/

//Input Signals
logic[4:0]  DEBUG_REGFILE_ADDR;

//Output Signals
logic[31:0] DEBUG_REGFILE_DATE;


/********Modports*******/


//Dispatch DUV
modport DUV(
input   Clk,
        Reset,
        IFQ_PCOUT_PLUS4,
```

```
        IFQ_INST,
        IFQ_EMPTY,
        CDB_TAG,
        CDB_VALID,
        CDB_DATA,
        CDB_BRANCH,
        CDB_BRANCH_TAKEN,
        EQUEUELS_READY,
        EQUEUEINT_READY,
        EQUEUEMULT_READY,
        EQUEUEDIV_READY,
        DEBUG_REGFILE_ADDR,
 output IFQ_REN,
        IFQ_BRANCH_ADDR,
        IFQ_BRANCH_VALID,
        EQUEUE_IMM,
        EQUEUE_RDTAG,
        EQUEUE_RSTAG,
        EQUEUE_RTTAG,
        EQUEUE_RSDATA,
        EQUEUE_RTDATA,
        EQUEUE_RSVALID,
        EQUEUE_RTVALID,
        EQUEUELS_OPCODE,
        EQUEUELS_EN,
        EQUEUEINT_OPCODE,
        EQUEUEINT_EN,
        EQUEUEMULT_EN,
        EQUEUEDIV_EN,
        DEBUG_REGFILE_DATE  );

//Controlability
modport TX(
input   Clk,
        Reset,
output  IFQ_PCOUT_PLUS4,
        IFQ_INST,
        IFQ_EMPTY,
        CDB_TAG,
        CDB_VALID,
        CDB_DATA,
        CDB_BRANCH,
        CDB_BRANCH_TAKEN,
        EQUEUELS_READY,
        EQUEUEINT_READY,
        EQUEUEMULT_READY,
        EQUEUEDIV_READY,
```

```
            DEBUG_REGFILE_ADDR );

   //Observability
   modport MON(
   input   Clk,
           Reset,
           IFQ_REN,
           IFQ_BRANCH_ADDR,
           IFQ_BRANCH_VALID,
           EQUEUE_IMM,
           EQUEUE_RDTAG,
           EQUEUE_RSTAG,
           EQUEUE_RTTAG,
           EQUEUE_RSDATA,
           EQUEUE_RTDATA,
           EQUEUE_RSVALID,
           EQUEUE_RTVALID,
           EQUEUELS_OPCODE,
           EQUEUELS_EN,
           EQUEUEINT_OPCODE,
           EQUEUEINT_EN,
           EQUEUEMULT_EN,
           EQUEUEDIV_EN,
           DEBUG_REGFILE_DATE );

endinterface: DispatchIf

//Dispatch Interface for Internal Signals
interface DispatchIntIf();

  logic           state_r;
  logic     [5:0] inst_opcode;
     modport IFQ ( input state_r,
                   input inst_opcode );
  modport DUV ( output state_r,
                output inst_opcode );

endinterface: DispatchIntIf




module Dispatch_DUV ( DispatchIf.DUV DisIf, DispatchIntIf.DUV
DisIntIf);
```

```
   dispatch dispatch( .clk(DisIf.Clk),
                      .reset(DisIf.Reset),
                      .ifq_pcout_plus4(DisIf.IFQ_PCOUT_PLUS4),
                      .ifq_inst(DisIf.IFQ_INST),
                      .ifq_empty(DisIf.IFQ_EMPTY),
                      .cdb_tag(DisIf.CDB_TAG),
                      .cdb_valid(DisIf.CDB_VALID),
                      .cdb_data(DisIf.CDB_DATA),
                      .cdb_branch(DisIf.CDB_BRANCH),
                      .cdb_branch_taken(DisIf.CDB_BRANCH_TAKEN),
                      .equeuels_ready(DisIf.EQUEUELS_READY),
                      .equeueint_ready(DisIf.EQUEUEINT_READY),
                      .equeuemult_ready(DisIf.EQUEUEMULT_READY),
                      .equeuediv_ready(DisIf.EQUEUEDIV_READY),

.debug_regfile_addr(DisIf.DEBUG_REGFILE_ADDR),
                      .ifq_ren(DisIf.IFQ_REN),
                      .ifq_branch_addr(DisIf.IFQ_BRANCH_ADDR),
                      .ifq_branch_valid(DisIf.IFQ_BRANCH_VALID),
                      .equeue_imm(DisIf. EQUEUE_IMM),
                      .equeue_rdtag(DisIf.EQUEUE_RDTAG),
                      .equeue_rstag(DisIf.EQUEUE_RSTAG),
                      .equeue_rttag(DisIf.EQUEUE_RTTAG),
                      .equeue_rsdata(DisIf.EQUEUE_RSDATA),
                      .equeue_rtdata(DisIf.EQUEUE_RTDATA),
                      .equeue_rsvalid(DisIf.EQUEUE_RSVALID),
                      .equeue_rtvalid(DisIf.EQUEUE_RTVALID),
                      .equeuels_opcode(DisIf.EQUEUELS_OPCODE),
                      .equeuels_en(DisIf.EQUEUELS_EN),
                      .equeueint_opcode(DisIf.EQUEUEINT_OPCODE),
                      .equeueint_en(DisIf.EQUEUEINT_EN),
                      .equeuemult_en(DisIf.EQUEUEMULT_EN),
                      .equeuediv_en(DisIf.EQUEUEDIV_EN),

.debug_regfile_data(DisIf.DEBUG_REGFILE_DATE) );

   assign DisIntIf.state_r = dispatch.state_r;
   assign DisIntIf.inst_opcode = dispatch.inst_opcode;



endmodule: Dispatch_DUV
```

Verification.sv

```
// ====================================
// Dispatch Package
// This package contains the class definitions for verfication
// ====================================
package DispatchPkg;

  // Transaction class: Simple packet of bytes (instructions of
32 bits)
     class BasePacket;
       typedef struct packed{
       bit[5:0] OpCode;
       bit[4:0] Rs;
       bit[4:0] Rt;
       bit[4:0] Rd;
       bit[4:0] Shamt;
       bit[5:0] Func;
    } TRType_Inst;
    rand TRType_Inst RType_Inst = 'h0;

    typedef struct packed{
       bit[5:0]  OpCode;
       bit[4:0]  Rs;
       bit[4:0]  Rt;
       bit[15:0] Immediate;
    } TIType_Inst;
    rand TIType_Inst IType_Inst = 'h0;

    typedef struct packed{
       bit[5:0]   OpCode;
       bit[25:0]  Address;
    } TJType_Inst;
    rand TJType_Inst JType_Inst = 'h0;

    enum{RType, IType, JType} Enum_Type_Inst;
    enum{LS, Int, Mult, Div} Enum_Queue;

    rand bit[1:0] Type_Inst;
    bit           EmptyPkt[$];  //Empty
    bit[1:0]      ReadyQueue;
    byte Data[$];        // List on bytes

          constraint c0 { (1==1) -> Type_Inst  < 3; };
          constraint c1 { (1==1) -> RType_Inst < 3; };
          constraint c2 { (1==1) -> IType_Inst < 3; };
          constraint c3 { (1==1) -> JType_Inst < 3; };
```

```
     endclass: BasePacket


// ====================================
// Responder Class
// ====================================

     class Responder;
        virtual interface DispatchIf.DUV DisIf;         // For
Interface connection
        virtual interface DispatchIntIf.DUV DisIntIf;   // For
Interface connection
        event eReadInstruction;
        event eQueueIntRdy;
        event eQueueMultRdy;
        event eQueueDivRdy;
        event eQueueLSRdy;
        event eQueueRdy;
        event eJumpBranch;

        // Constructor: Connects the inferface
        function new ( virtual interface DispatchIf DisIfDUV,
virtual interface DispatchIntIf DisIntIfDUV);
           DisIf    = DisIfDUV;
           DisIntIf = DisIntIfDUV;
        endfunction

        //Monitor for ifq_ren signal
        task Run_Monitor();

         forever @(posedge DisIf.Clk) begin

             if(DisIf.IFQ_REN)          -> eReadInstruction;
             if(DisIf.EQUEUELS_READY)   -> eQueueLSRdy;
             if(DisIf.EQUEUEINT_READY)  -> eQueueIntRdy;
             if(DisIf.EQUEUEMULT_READY) -> eQueueMultRdy;
             if(DisIf.EQUEUEDIV_READY)  -> eQueueDivRdy;
             if(DisIf.IFQ_BRANCH_VALID) -> eJumpBranch;
             if(DisIf.EQUEUELS_READY ||  DisIf.EQUEUEINT_READY ||
DisIf.EQUEUEMULT_READY || DisIf.EQUEUEDIV_READY ) -> eQueueRdy;

          end

        endtask: Run_Monitor

        task Run_Checker_IFQStop();
```

```
            forever@(eJumpBranch) begin

                if(!DisIf.IFQ_REN) $display("Ok. IFQ was
stopped");
                else $display("Error, IFQ was not stopped when a
branch instrucion was received");

            end

            forever@(eQueueRdy) begin

                if(!DisIf.IFQ_REN) $display("Ok. IFQ was
stopped");
                else $display("Error, IFQ was not stopped when a
queue is full");

            end

        endtask: Run_Checker_IFQStop



        task Run_Checker_Queues(/* const ref BasePacket Pkt,*/
const ref RM Rm );

            forever@(eReadInstruction)  begin

               #10ns; //Delay for instruction decoding

               //case(Pkt.ReadyQueue)
               case(Rm.DecodedQueue)

                 Rm.LS: begin
                   if(DisIf.EQUEUELS_EN)  $display("Ok.
Instruction Dispatched");
                   else $display( "Error, [%0tns][State Checker]
Dispatched Instruction=%p", $time, DisIf.IFQ_INST);
                 end

                 Rm.Int: begin
                   if(DisIf.EQUEUEINT_EN)  $display("Ok.
Instruction Dispatched");
                   else $display( "Error, [%0tns][State Checker]
Dispatched Instruction=%p", $time, DisIf.IFQ_INST);
                 end
```

```
                Rm.Mult: begin
                  if(DisIf.EQUEUEMULT_EN)  $display("Ok.
Instruction Dispatched");
                  else $display( "Error, [%0tns][State Checker]
Dispatched Instruction=%p", $time, DisIf.IFQ_INST);
                end

                Rm.Div: begin
                  if(DisIf.EQUEUEDIV_EN)  $display("Ok.
Instruction Dispatched");
                  else $display( "Error, [%0tns][State Checker]
Dispatched Instruction=%p", $time, DisIf.IFQ_INST);
                end


              endcase

            end

        endtask: Run_Checker_Queues

        task Run_Checker_Address_Calculation( /*, const ref RM
Rm*/ );

              forever@(eJumpBranch) begin

                if(DisIf.IFQ_BRANCH_ADDR /*== Rm.AddrCalc*/)
$display("Ok. Branch or Jump Address is Correct");
                else $display("Error. Branch or Jump Address is
Incorrect");

            end


        endtask: Run_Checker_Address_Calculation

        //Check for Decoding Signals for 2nd Functional Area
        task Run_Checker_Decode(  /*, const ref RM Rm*/ );
          forever@(eReadInstruction)  begin

              if(DisIntIf.state_r == XXXXXXXXXXX) $display("Ok.
Instruction Decoded")
              else $display( "Error, [%0tns][State Checker]
Decoding Instruction=%p", $time, DisIf.IFQ_INST );
              if(DisIntIf.inst_opcode == XXXXXXX) $display("Ok.
Instruction Decoded")
```

```
            else $display( "Error, [%0tns][Opcode Checker]
Decoding Instruction=%p", $time, DisIf.IFQ_INST );

          end

        endtask: Run_Checker_Decode


        task Enable_Queues( const ref BasePacket Pkt );

           @(posedge DisIf.Clk ) begin

              {DisIf.EQUEUEDIV_READY, DisIf.EQUEUEMULT_READY,
DisIf.EQUEUEINT_READY, DisIf.EQUEUELS_READY}  <= Pkt.ReadyQueue;

           end

        endtask: Enable_Queues


        task Send_Instruction_Responder ( const ref BasePacket
Pkt );

         @(eReadInstruction) begin

                  case(Pkt.Type_Inst)

                    Pkt.RType: begin
                       $display( "[%0tns][Driver] Sending
Pkt=%p", $time, Pkt.RType_Inst );           // Informs a new
packet to send
                       @(posedge DisIf.Clk ) begin

                          DisIf.IFQ_INST <= Pkt.RType_Inst;
     // Bit activity conversion
                          DisIf.IFQ_EMPTY <= 1'b0;

                       end
              end

              Pkt.IType: begin
                       $display( "[%0tns][Driver] Sending
Pkt=%p", $time, Pkt.IType_Inst );           // Informs a new
packet to send
                       @(posedge DisIf.Clk ) begin
```

```
                            DisIf.IFQ_INST <= Pkt.IType_Inst;
     // Bit activity conversion
                            DisIf.IFQ_EMPTY <= 1'b0;

                         end

              end

              Pkt.JType: begin
                        $display( "[%0tns][Driver] Sending
Pkt=%p", $time, Pkt.JType_Inst );           // Informs a new
packet to send
                         @(posedge DisIf.Clk ) begin

                             DisIf.IFQ_INST <= Pkt.JType_Inst;
     // Bit activity conversion
                             DisIf.IFQ_EMPTY <= 1'b0;

                  end
              end

              default: $display( "[%0tns][Driver] Invalid", $time
);             // Informs a new packet to send

           endcase

                end

        endtask: Send_Instruction_Responder

     endclass: Responder


// ====================================
// Coverage Class
// ====================================

//   class Coverage;
//        Responder pRx;
//        bit[3:0] a;
//        integer x,y;
//         x = 50; y = 100;
//        covergroup cgState0 @(pRx.eReadInstruction);
//             CP0_State: coverpoint pRx.DisIf.IFQ_REN;
//             CP1_State: coverpoint pRx.DisIf.IFQ_EMPTY;
//        endgroup: cgState
//
```

```
//
//     /*covergroup cgState1 @(pRx.eReadInstruction);
//             CP0_State: coverpoint pRx.DisIntIf.state_r;
//             CP1_State: coverpoint pRx.DisIntIf.inst_opcode;
//        endgroup*/
//
//
//     /*covergroup cgState3 @(pRx.eReadInstruction);
//             CP0_State: coverpoint pRx.DisIf.EQUEUELS_EN;
//             CP1_State: coverpoint pRx.DisIf.EQUEUEINT_EN;
//             CP2_State: coverpoint pRx.DisIf.EQUEUEMULT_EN;
//             CP3_State: coverpoint pRx.DisIf.EQUEUEDIV_EN;

//        endgroup*/
//
//      /*covergroup cgState4 @(pRx.eJumpBranch);
//             CP0_State: coverpoint pRx.DisIf.IFQ_BRANCH_ADDR {
//                  bins a1 = {0:x};
//                  bins a2 = {x+1:y};
                  }
//        endgroup*/


//     /*covergroup cgState5 @(pRx.eJumpBranch);
//             CP0_State: coverpoint pRx.DisIf.IFQ_REN ;
//        endgroup*/


//   endclass: Coverage



// ====================================
// Scenario Class
// ====================================
   class Scenario;

      rand BasePacket Pkt;
           Responder pRespBfm;
           //RM pRM;

      function new();
                 Pkt = new();
           endfunction

      task Run();
```

```
               #300ns;                                          //
Delay

                  Pkt.RType_Inst = {6'h00, 5'h01, 5'h02, 5'h03,
5'h00, 6'h20};
               Pkt.IType_Inst = {6'h08, 5'h01, 5'h04, 16'h55};
               Pkt.JType_Inst = {6'h02, 26'h4};
               Pkt.Type_Inst  = 0;


                  repeat (5) begin

                       Pkt.randomize();

                       fork

                      pRespBfm.Send_Instruction_Responder (  Pkt
);
                      pRespBfm.Enable_Queues(  Pkt );


                    join


                   end

                   #300ns;


            endtask: Run

   endclass: Scenario

endpackage: DispatchPkg

// ====================================
// Testbench
// ====================================
module tb_Dispatch;

    // Import Package
          import DispatchPkg::*;                                 //
Import Classes for verification

    // Clock and Reset
       bit clk=1, rst=1;                                      // Top
Clock and Reset signals
```

```
       always   #50ns clk = !clk;           // Clock period:
100ns
       initial  #200ns rst = 0;                  // Reset
duration: 200ns


     // Interface, DUV, and Object instances
       DispatchIf    DisIf(clk, rst);                    //
Interface instance using share signals clk and rst
       DispatchIntIf DisIntIf();                         //
Interface instance for internal signals
       Dispatch_DUV Dispatch_DUV( DisIf, DisIntIf );     // DUV
instance and interface connection
       Responder RespBfm = new ( DisIf, DisIntIf );      //
Responder instance and interface connection
       Coverage Cov;
       Scenario Test_Scenario;

       initial begin: Test
          Test_Scenario = new();
          Test_Scenario.pRespBfm = RespBfm;
          Test_Scenario.Run();
          #300ns;  // Delay
                $finish();
       end: Test

       initial begin: MonAndCheck
          Cov = new();
           Cov.pRx = RespBfm;


          fork
                       RespBfm.Run_Checker_Decode();
                      RespBfm.Run_Checker_Address_Calculation(
Rm);
                      pRespBfm.Run_Checker_Queues(  /* Pkt,*/  Rm
);
                      RespBfm.Run_Checker_IFQStop();
                      RespBfm.Run_Monitor();
          join


       end: MonAndCheck
```

```
endmodule: tb_Dispatch
```

Dispatch_RM.sv

```
`define OPCODE_RTYPE  (6'h00)
`define OPCODE_JTYPE  (6'h02)
`define OPCODE_BTYPE  (6'b0001XX)
`define OPCODE_LW     (6'h23)
`define OPCODE_SW     (6'h2B)

module Dispatch_RM(
          clk, rst,
          ifetch_pc_plus_four_RM,
          ifetch_instruction_RM,
          ifetch_empty_flag_RM,
          dispatch_jmp_br_addr_RM,
          dispatch_jmp_RM,
          dispatch_ren_RM,

          issueque_full_integer_RM,
          dispatch_opcode_RM,
          dispatch_en_integer_RM,
          dispatch_rd_tag_RM,
          dispatch_rs_data_RM,
          dispatch_rs_tag_RM,
          dispatch_rs_data_val_RM,
          dispatch_rt_data_RM,
          dispatch_rt_tag_RM,
          dispatch_rt_data_val_RM,

          dispatch_en_ld_st_RM,
          issueque_full_ld_st_RM
     );

     input rst, clk;
     input [31:0] ifetch_pc_plus_four_RM;
     input [31:0] ifetch_instruction_RM;
     input ifetch_empty_flag_RM;
     output reg [31:0] dispatch_jmp_br_addr_RM;
     output reg dispatch_jmp_RM;
     output reg dispatch_ren_RM;

     input issueque_full_integer_RM;
     output [2:0]   dispatch_opcode_RM;
     output    reg dispatch_en_integer_RM;
     output [5:0]   dispatch_rd_tag_RM;
     output [31:0]  dispatch_rs_data_RM,dispatch_rt_data_RM;
     output [5:0] dispatch_rs_tag_RM,dispatch_rt_tag_RM;
```

```
    output    dispatch_rs_data_val_RM,dispatch_rt_data_val_RM;

    input issueque_full_ld_st_RM;
    output reg dispatch_en_ld_st_RM;

    reg [31:0] instruction;
    wire [3:0] full;

    assign dispatch_opcode_RM = instruction[31:26];
    assign full = {issueque_full_integer_RM,
issueque_full_ld_st_RM};

    always@(posedge clk) begin
        if(~ifetch_empty_flag_RM && dispatch_ren_RM)
            instruction <= ifetch_instruction_RM;
    end

    always@(full) begin
        if (full) dispatch_ren_RM = 1'b0;
        else dispatch_ren_RM = 1'b1;
    end

    always@(instruction) begin
        dispatch_en_integer_RM = 1'b0;
        dispatch_en_ld_st_RM =1'b0;
        case(instruction[31:26])
            `OPCODE_RTYPE: begin
                dispatch_en_integer_RM = 1'b1;
            end

            `OPCODE_LW: begin
                dispatch_en_ld_st_RM =1'b1;
            end

            `OPCODE_SW: begin
                dispatch_en_ld_st_RM =1'b1;
            end

            `OPCODE_JTYPE: dispatch_jmp_RM = 1'b1;
            `OPCODE_BTYPE: dispatch_jmp_RM = 1'b1;
            default: dispatch_jmp_br_addr_RM = 32'h00000000;
        endcase
    end
    endmodule
```

REFERENCIAS:

Tipo de Referencia Bibliográfica	Ejemplo
Héctor Sucar, Alejandro Moreno “Verification Methodology”, *Curso Verificación de Sistemas Digitales.* ITESO, 2011.	[Sucar-Moreno-01]