UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Definición del flujo en la herramienta VCS para la simulación de HDLs en la Fabricación de un Chip con Tecnología Nanométrica CMOS

Trabajo de graduación presentado por Jefferson Noel Ruano Orellana
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2020

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Definición del flujo en la herramienta VCS para la simulación de HDLs en la Fabricación de un Chip con Tecnología Nanométrica CMOS

Trabajo de graduación presentado por Jefferson Noel Ruano Orellana
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

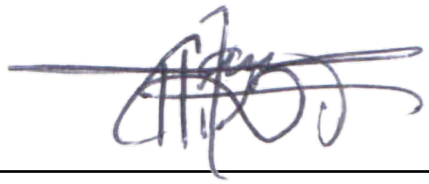
2020

Vo.Bo.:

(f) _____

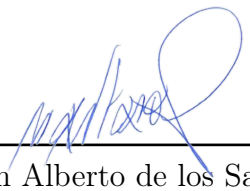
MSc. Carlos Alberto Esquit Hernández

Tribunal Examinador:



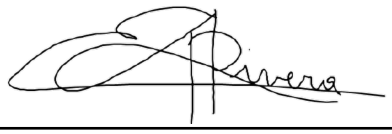
(f) _____

MSc. Carlos Alberto Esquit Hernández



(f) _____

Ing. Jonathan Alberto de los Santos Chonay



(f) _____

PhD. Luis Alberto Rivera Estrada

Fecha de aprobación: Guatemala, 26 de enero de 2021.

Prefacio

Este proyecto de investigación fue llevado a cabo durante la pandemia causada por el COVID-19. Se presentaron muchas dificultades debido a las restricciones impuestas a los centros educativos lo cual redujo el tiempo en el que esta investigación se realizó. A pesar de todos estos contratiempos la investigación pudo ser teminada de forma efectiva.

A pesar que esta investigación es de mi responsabilidad en gran parte, no se pudo haber concretado sin el apoyo de varias personas. En primer lugar agradecer al Dr. Luis Rivera por su apoyo en la corrección y orientación de esta investigación. De igual forma, mi agradecimiento a MSc. Carlos Esquit por su apoyo técnico y por sentar las bases de la investigación de nanoelectrónica en la Universidad del Valle de Guatemala.

De forma personal quisiera agradecer a mi familia por su apoyo incondicional en mis deseos educativos y a la fundación Juan Bautista Gutiérrez por apoyarme en mis estudios universitarios.

Índice

Lista de figuras

Lista de cuadros

Resumen

El proyecto general, bajo el cual se encuentra el siguiente trabajo, se enfoca en la depuración y optimización de un flujo de diseño para la elaboración de un chip nanométrico con tecnología CMOS. El flujo de diseño general propuesto consta de 10 pasos generales con los que se busca tener un circuito listo para la fabricación al final del proceso.

El presente trabajo se enfoca en el paso no. 4 del flujo de diseño general propuesto, con el objetivo principal de optimizar y depurar el flujo de diseño específico para la prueba y simulación de archivos HDL y esquemáticos generados en la herramienta Design Vision. Para alcanzar dicho objetivo se utilizaron las herramientas VCS y Verdi.

Se realizaron distintas pruebas utilizando varios circuitos con el fin de poder validar cada uno de los flujos de diseños propuestos para poder alcanzar el objetivo general de este trabajo. Cada uno de los comandos, librerías y *software* utilizado en este trabajo se describen a detalle con el fin de poder ser utilizado como una guía para estudiantes interesados en trabajar en el diseño y desarrollo de chips nanométricos.

Abstract

This work is part of a larger project which focuses on the debugging and optimization of a design flow for the development of a nanometric chip with VLSI CMOS technology. The project proposes a ten-step design flow, and the goal is to have a ready-to manufacture circuit at the end of the design process.

The focus of this work is the fourth step of the design flow. The main goal of this step is to debug and optimize specific aspects for the simulation and testing of HDL and schematic files generated in the Design Vision tool. To achieve this objective, VCS and Verdi tools will be used.

A few alternatives were proposed for this step in the flow. They were validated using serveral circuits to contribute to the fulfillment of the general investigation line main objective. Each of the commands, libraries and textit software used in this document, so it can be used as a guide for students interested in working on the design and development of nanometric chips.

CAPÍTULO 1

Introducción

El diseño y fabricación de un chip con tecnología VLSI CMOS es un proceso bastante complejo ligado a muchas reglas de diseño de las cuales depende su correcto funcionamiento. En la fase inicial de este flujo, partiendo de un diseño en un descriptor de hardware, es recomendable realizar pruebas y simulaciones que aseguren que el circuito del cual parte el diseño presente el mismo comportamiento que el circuito sintetizado bajo las librerías propias del fabricante. Estas simulaciones permitirán prevenir errores que puedan presentarse en el funcionamiento del chip a lo largo del flujo de diseño establecido o bien una vez fabricado.

La presente investigación trata sobre el diseño, validación y documentación de varios flujos para la herramienta VCS en simulación de lenguajes descriptores de hardware. Para la ejecución de esta investigación se probaron cada uno de los flujos con sus distintas funcionalidades utilizando el mismo circuito para luego poder ser comparados. Cada uno de los flujos de simulación se validó a través de circuito de circuito de baja y mediana complejidad.

El presente trabajo está organizado en cuatro capítulos principales. En el capítulo 8 se brindan recomendaciones de diseño para evitar condiciones de carrera que puedan perjudicar las simulaciones de los diseños elaborados. En el capítulo 9 se presentan cada uno de las propuestas de flujos y funcionalidades propuestas para la evaluación de los diseños creados. Por último, en los capítulos 11 y 12 se presentan pruebas sobre circuitos de complejidad baja y media respectivamente, con el fin de validar cada uno de los flujos diseñados en los capítulos anteriores.

CAPÍTULO 2

Antecedentes

El campo de la nanoelectrónica ha presentado avances importantes en las últimas décadas. Empresas y centros de investigación y estudio dedicados al desarrollo, fabricación o estudio de este campo elaboran sus propios flujos de diseño con el fin de fabricar módulos eficientes con tecnología VLSI CMOS que cumplan con requerimientos específicos.

El curso de Nanoelectrónica, impartido por Msc. Carlos Esquit en la Universidad del Valle de Guatemala, presenta el primer acercamiento en el estudio y diseño de circuitos de tecnología nanoelectrónica. El acuerdo dentro de la universidad con la empresa Synopsis permitió el uso de herramientas sofisticadas y de alto desempeño para elaboración y diseño de circuitos nanoelectrónicos. El uso de estas herramientas ha permitido entender de una forma más práctica todos los procesos de diseño de chips nanoelectrónicos.

El proyecto para la implementación y documentación de un flujo de diseño para la fabricación de chips con tecnología nanométrica fue presentado en el año 2019 y estuvo a cargo de estudiantes del departamento de ingeniería electrónica. Este proyecto presentó grandes avances en el estudio de la nanoelectrónica, siendo este el acercamiento más significativo al diseño VLSI dentro de la Universidad. Entre los avances más importantes de este proyecto se encuentran la base del flujo de diseño planteado así como la documentación y flujos específicos de cada uno de los pasos correspondientes y los video tutoriales acerca del funcionamiento y uso de cada una de las herramientas utilizadas a lo largo del flujo de diseño. Vale la pena mencionar la importancia de la documentación proporcionada por el grupo anterior ya que esta conforma la base para la realización del presente trabajo.

CAPÍTULO 3

Justificación

La implementación de un flujo funcional para el diseño y fabricación de chips con tecnología nanométrica representa avances y aplicaciones importantes tanto dentro como fuera de la Universidad del Valle de Guatemala. Con este flujo de diseño se puede pensar en diseñar y fabricar chips con funciones muy específicas, que brinden un desempeño superior a los chips de propósito general, para cualquier proyecto dentro de la universidad. Este proyecto sienta una base muy importante para la elaboración de chips de cualquier función y complejidad, lo que permitirá realizar proyectos de investigación y desarrollo más complejos y desafiantes. Por otra parte, este proyecto abre una brecha importante para la investigación de nanoelectrónica dentro de Guatemala, ya que el país cuenta con un pobre desarrollo de este campo.

Este trabajo, en específico, tiene una gran importancia dentro del flujo de diseño completo ya que todos los pasos posteriores dependen del correcto funcionamiento del circuito sintetizado proporcionado por la herramienta de Synopsys. Con la simulación y comparación de ambos diseños digitales (original y sintetizado) se puede corroborar que el circuito sintetizado efectivamente se comporta de la manera deseada. En la simulación del circuito sintetizado, por otra parte, se pueden detectar errores de diseño que podrían afectar posteriormente en el flujo de diseño.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Optimizar y depurar el flujo de diseño para la prueba y simulación de archivos HDL y esquemáticos generados en la herramienta Design Vision.

4.2. Objetivos específicos

- Estudiar la herramienta VCS que permite realizar simulaciones de código en verilog y otros lenguajes HDL.
- Estudiar la herramienta Verdi que permite depurar diseños digitales complejos ASIC, Soc o diseños basados en FPGA's.
- Comparar los códigos en Verilog tanto depurados en Design Vision como los creados originalmente con el fin de comprobar su correcto funcionamiento.
- Validar distintos flujos en VCS con el objetivo de encontrar el mas eficiente para esta aplicación.

CAPÍTULO 5

Alcance

En esta investigación se pretende documentar y validar distintos flujos dentro de la herramienta VCS en conjunto con Verdi para la simulación de archivos verilog utilizados en el diseño de cualquier chip con tecnología VLSI CMOS. El flujo utilizado permitirá tener certeza de que el proceso de diseño del chip, en sus fases iniciales, se está realizando de forma correcta. Esto evitará posibles errores en etapas posteriores de diseño o bien que el chip, una vez fabricado, presente un comportamiento distinto al diseño original.

La documentación de todo el proceso de diseño de estos flujos será importante para el seguimiento del proyecto principal al que esta investigación pertenece.

A pesar de que la pandemia a casua del COVID-19 limitó en gran parte el alcance de este proyecto, se validaron ciertos flujos y funcionalidades de la herramienta VCS y Verdi que serán de gran ayuda para trabajos futuros en el diseño y fabricación de circuitos con tecnología VLSI. Se ampliaron las bases en la simulación de diseños digitales basados en tecnología VLSI en la herramienta de VCS para que grupos futuros dedicados a este campo dispongan de mas herramientas para la simulación y depuración de los diseños propuestos.

CAPÍTULO 6

Marco teórico

6.1. Diseño VLSI

En 1963, Frank Wanlass describió la primera compuerta lógica utilizando MOSFETS. Estas compuertas utilizaban transistores pMOS y nMOS por lo que fueron llamadas compuertas CMOS. Esta técnica de diseño trajo notables ventajas como el bajo costo de producción y un menor consumo de energía. Procesos de diseño mas antiguos hacían uso únicamente de transistores nMOS, por ejemplo el procesador 4004 de 4 bits de Intel. El consumo de energía de volvió un problema a medida que se incluían cientos de miles de transistores en un solo circuito integrado. Por esta razón los procesos CMOS se volvieron más aceptados debido a su eficiencia en cuanto al consumo de energía.

En 1965, Gordon Moore observó cómo la cantidad de transistores en un chip se duplicaba cada 18 meses, a esta observación se le conoce como la Ley de Moore. El nivel de integración de un chip se ha clasificado como pequeña, mediada, grande y muy grande. A esta ultima clasificación se le conoce como *very large-scale integration* (VLSI) y es utilizada para describir circuitos desde 1980 en adelante. Así pues se le conoce como VLSI al proceso de integración a muy gran escala en el diseño de circuitos [1].

6.2. Flujo de diseño

Flujo de diseño hace referencia a una serie de pasos ordenados para el diseño y fabricación de un circuito integrado. Debido a que la cantidad de transistores utilizados en este tipo de aplicaciones rondan los miles de millones, el diseño VLSI presenta un reto en cuanto al manejo de la complejidad de estos circuitos. Por esta razón, el proceso de diseño procede a través de múltiples niveles de abstracción. La práctica de diseño estructural, que es en la que se basa este proyecto, usa los principios de jerarquía, regularidad, modularidad y localidad para poder manejar la complejidad del diseño [1].

Abstracción del diseño

El diseño VLSI comúnmente se divide en 5 niveles de abstracción: diseño de arquitectura, diseño de micro arquitectura, diseño lógico, diseño de circuito y diseño físico [1].

- Diseño de arquitectura: describe las funciones del sistema. Por ejemplo, la arquitectura de un procesador especifica el grupo de instrucciones, el modelo de memoria y el grupo de registros.
- Diseño de micro arquitectura: describe como la arquitectura es particionada en registros y unidades funcionales mas específicas.
- Diseño lógico: describe como las unidades funcionales son construidas. En este apartado se pueden mencionar los distintos tipos de diseños lógicos por los que se puede realizar un sumador, por ejemplo un ripple carry, carry lookahead y carry select.
- Diseño del circuito: describe como los transistores son específicamente utilizados para implementar el diseño lógico propuesto.
- Diseño físico: describe el layout del chip [1].

6.3. Lenguaje descriptor de *Hardware*

Al utilizar dispositivos lógicos programables para la elaboración de circuitos se utilizan lenguajes descriptores de *hardware* en donde por medio de texto se describen conexiones, funciones y módulos de nuestro diseño físico. A estos lenguajes se les conoce como HDLs por sus siglas en inglés. El software dedicado se encarga de convertir cada una de las expresiones especificadas creando un archivo que posteriormente es utilizado para un circuito físico en un módulo dedicado para esta función [2]. Entre los lenguajes descriptores de hardware mas conocidos y utilizados se encuentran Verilog y VHDL.

6.4. Verilog

Desde su comienzo en 1984 en Gateway Design Automation, Verilog se ha convertido en un estándar industrial producto de su uso en el diseño de circuitos integrados y sistemas digitales [3]. Verilog es un lenguaje de descripción de hardware que permite especificar un sistema digital en una amplia gama de niveles de abstracción. Este lenguaje incluye construcciones jerárquicas que permiten al diseñador controlar la complejidad de un bloque descriptivo. Estos bloques pueden ser simulados para poder determinar la funcionalidad del circuito descrito. Verilog fue diseñado con el objetivo de ser usada en todas las fases de la creación de un sistema electrónico. Debido a que es un lenguaje legible tanto para tanto para humanos como para máquinas, soporta el desarrollo, verificación síntesis y simulaciones de diseños de *hardware* [4].

6.5. VCS

VCS es un simulador de alta capacidad y desempeño que incorpora tecnologías avanzadas de alto nivel de abstracción y verificación en una sola plataforma. Permite analizar, compilar y simular descripciones de diseño en Verilog, VHDL, mixedHDL, SystemVerilog, OpenVera y SystemC. También, proporciona un conjunto de herramientas de simulación y depuración para la validación de diseños HDL. La herramienta cuenta con dos tipos de flujos de diseño (dos y tres pasos) que nos permiten evaluar distintos funcionamientos de nuestro circuito [5].

Flujo de dos pasos

Este tipo de flujo soporta únicamente diseños en Verilog HDL y SystemVerilog. Consta de los siguientes pasos:

1. Compilar el diseño
2. Simular el diseño

Compilar el diseño: es el primer paso en el flujo de dos pasos para la simulación del diseño evaluado. En este paso, VCS construye una instancia de jerarquía y genera un archivo .simv que luego es usado para la simulación.

Simular el diseño: en este paso VCS toma el archivo generado en el paso anterior y lo usa para correr la simulación [5].

Flujo de dos pasos

Este tipo de flujo soporta diseños en Verilog, VHDL y mixedHDL. Consta de los siguientes pasos:

1. Analizar el diseño
2. Elaborar el diseño
3. Simular el diseño.

Analizar el diseño: VCS provee los ejecutables vhdlan y clogan para analizar el código de diseño. vhdlan/vlogan analiza el diseño y almacena archivos intermediarios en el diseño o librería de trabajo.

Elaborar el diseño: VCS provee el ejecutable .vcs para compilar y elaborar el diseño. El ejecutable compila/elabora el diseño utilizando los archivos intermediarios generados en el paso anterior para generar el código objetivo y de forma estadística lo une con el archivo .simv(ejecutable binario de simulación).

Simular el diseño: La simulación se realiza al ejecutar el ejecutable binario de simulación [5].

6.6. Verdi

Verdi es una plataforma avanzada para la depuración y simulación de diseños digitales. Esta plataforma provee de herramientas que ayudan a comprender diseños digitales complejos y comportamientos extraños. Sumado a las funcionalidades para debugging de esquemáticos, waveforms, esquemáticos de maquinas de estados y comparación de diseños digitales, Verdi permite el rastreo de la actividad de una señal así como el análisis de transacción y mensajes de dados del diseño [6].

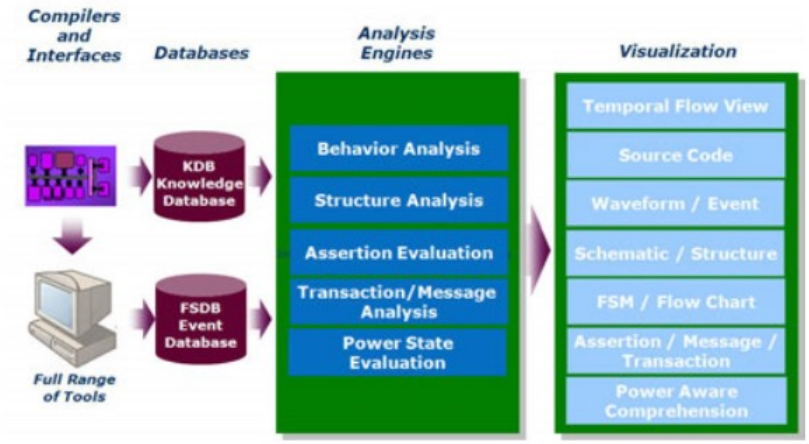


Figura 1: Esquema de módulos utilizados por Verdi [6].

La figura no. muestra un resumen sobre la tecnología que implementa Verdi para poder proporcionar cada una de sus funciones de alto desempeño.

Compiladores, interfaces e interoperabilidad

- Compiladores: Verdi cuenta con los compiladores para los lenguajes utilizados en la mayoría de entornos de diseño y verificación como Verilog, VHDL y System Verilog.
- Interfaz: Los lectores de plataforma implementados por Verdi siguen los estándares industriales de datos VCD y SDF. Los resultados leídos por la herramienta de detección son guardados en la Fast Signal Database (FSDB). El direccionamiento directo del simulador hacia esta base de datos reduce la cantidad de archivos generados y permite un acceso mas flexible a los archivos generados por el simulador.
- Interoperabilidad: Verdi permite la interoperabilidad con todos los simuladores lógicos y herramientas de verificación y análisis de tiempo. También, permite la habilidad de integrar aplicaciones extras de verificación por medio de API's [6].

Bases de datos

- Base de datos de conocimiento (KDB): Esta base de datos es utilizada para almacenar información lógica, estructural y funcional sobre el diseño mientras se compila.
- Base de datos de señal rápida(FSDB): En esta base de datos se almacenan los resultados de la simulación incluyendo la información de transición y mensajes de salida [6].

Motores de análisis

Utilizando la información de las bases de datos, Verdi posee un grupo de herramientas de análisis para diferentes aplicaciones.

- Análisis de estructura
- Análisis de comportamiento
- Evaluación de afirmación
- Análisis de transacciones/mensajes [6]

Interfaz gráfica de usuario

La interfaz gráfica que posee Verdi permite al usuario realizar los proceso de una forma mas amigable. Posee funciones como historial de sesiones y asistencia en el ingreso de comandos [6].

Visualización

Verdi cuenta con una visualización muy completa y flexible de los resultados de simulación encontrados. Permite funciones de visualización avanzadas en donde partes del diseño lógico podrían ser seleccionadas para su análisis [6].

6.7. *Formality*

Formality es una herramienta utilizada para la detección de diferencias inesperadas que puedan ser introducidas durante el desarrollo de un diseño lógico. Utiliza un motor comparativo de verificación para validar o descartar la equivalencia de dos diseños proporcionados y brindar un reporte del análisis de comparación con sus diferencias. [7]

6.7.1. Proceso general de verificación

Formality en su proceso general de verificación toma dos diseños para compararlos antes y después de ejecutar un proceso metodológico sobre estos. El objetivo de este proceso es asegurarse que la integridad de dos diseños, luego de algun proceso, son logicamente equivalentes. La Figura 2 muestra el flujo a gran escala de este proceso. [7]

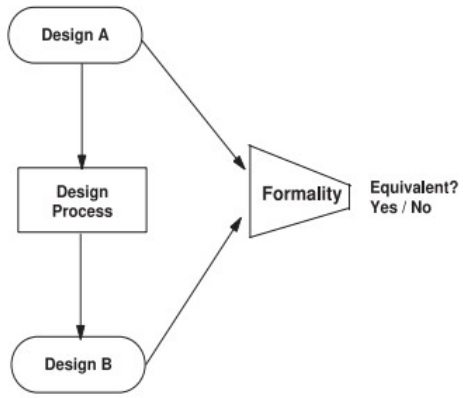


Figura 2: *Ripple carry adder* de dos bits [7].

En este caso, el proceso de diseño mostrado en la Figura 2 se refiere al proceso de síntesis realizado sobre el diseño A con librearías de TSMC 180nm.

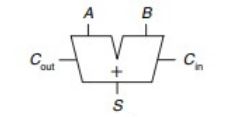
6.8. Condiciones de carrera

Una condición de carrera puede definirse como un comportamiento inesperado debido a una dependencia del tiempo de ocurrencia de los eventos. En la mayoría de los casos las condiciones de carrera ocurren debido a que uno o mas procesos compiten por acceder a un recurso compartido sin ningún tipo de control o orden. Esto puede recaer en un funcionamiento impredecible de un circuito o diseño. Cualquiera de los procesos puede ser interrumpido debido a una solicitud de recursos por parte de otro proceso o señal [8]. Las condiciones de carrera en datos particularmente, son difíciles de observar debido a que violan las estructuras de datos en lugar de causar fallos puntuales inmediatos. Los efectos de estas violaciones únicamente manifiestan millones de ciclos después de que el error ocurra haciendo muy difícil poder rastrear el error hasta su origen [9].

6.9. Circuito *Full Adder*

Un *full adder* es un circuito combinacional que cumple con la funcionalidad de un sumador de dos entradas, de un bit cada una en este caso. El circuito acepta una señal de *carry in* (C_in) y dos entradas A y B como entrada y una salida compuesta de una señal S y una

señal de *carry out* ($C_o ut$) como se muestra en la Figura 3. Esta figura también muestra las ecuaciones de salida para la salida S y la salida *carry out* [10].



C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Figura 3: Circuito, ecuaciones y tabla de verdad de full adder de un bit [10].

6.10. Circuito ripple carry adder

La forma mas sencilla de contruir un sumador de n-bits es usando n *full adders* en cadena. El $C_o ut$ de un bloque actua como el $C_i n$ del siguiente bloque como se muestra en la Figura 4 [10].Debido a que de transporte se propaga a traves de los *full adders* estos circuitos se les conoce como *ripple carry adders*. Utilizando esta configuración podemos construir sumadores de n-bits utilizando n *full adders* de un bit como se muestra en la Figura 5 [11].

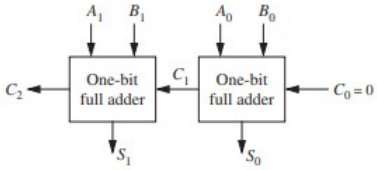


Figura 4: *Ripple carry adder* de dos bits [10].

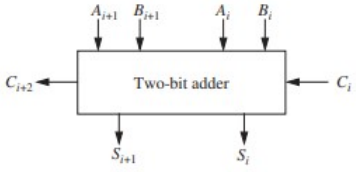


Figura 5: Módulo *Ripple carry adder* de dos bits [11].

CAPÍTULO 7

Metodología

Para la metodología se fijaron 4 pasos generales en donde los últimos 3 se vuelven recurrentes. En resumen, la metodología consta del diseño de varios flujos en la herramienta de VCS para luego ser testeados y validados.

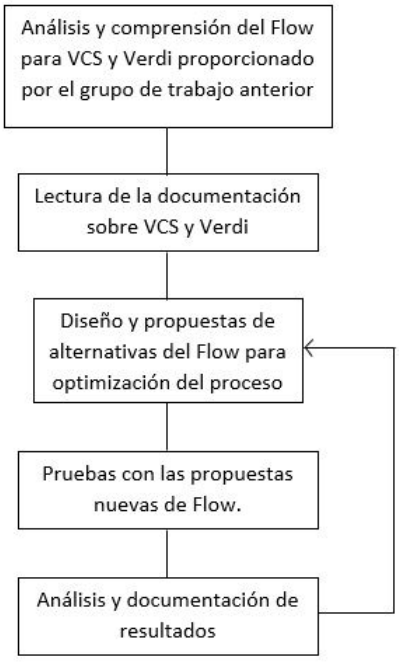


Figura 6: Metodología para diseño de flow en VCS.

Análisis y comprensión de flujo disponible

La documentación y flujo para VCS proporcionado por el equipo anterior presenta un gran avance para la realización de este trabajo. El primer paso en el plan de trabajo para este proyecto consistió en el análisis y comprensión de dicha documentación. El flujo propuesto para VCS por el grupo anterior es la base de este trabajo y todas las nuevas propuestas de flujo tomarán como base dicho flujo.

Lectura de la documentación de VCS y Verdi

Synopsys, en su entorno SolvNet, proporciona documentación extensa y detallada sobre cada una de las herramientas utilizadas en este proyecto (VCS y Verdi). Dicha documentación es la fuente principal de información para las nuevas propuestas de flujo que se mencionaron en este proyecto. Como segundo paso en el plan de trabajo se tuvo la lectura de la documentación puntal de cada una de las herramientas, la cual fue de gran ayuda no solo para familiarizarse en lo posible con el entorno de trabajo de VCS y Verdi, si no también para diseñar propuestas de flujo mas óptimas.

Diseño de propuestas alternativas del flujo

Como se mencionó en el paso anterior, se buscó optimizar el flujo presentado por el equipo previo. A lo largo de de este trabajo se diseñaron propuestas alternativas del flujo para intentar optimizarlo. VCS y Verdi son herramientas muy completas que poseen características que no fueron incluidas en el flujo anterior por lo que fueron incluidas y documentadas correspondientemente.

Pruebas con las propuestas nuevas de flujo

Con cada uno de los flows diseñados se realizaran pruebas y se documentaran debidamente, esto con el fin de comparar los resultados entre cada uno de los flujos y depurar el circuito lo mejor posible el circuito. El diseño de un flujo mas completo para la simulación de diseños lógicos será útil para futuras aplicaciones donde se desee fabricar circuitos de alta complejidad.

Análisis y documentación de resultados

Para cada una de las nuevas propuestas de flujo se realizaron pruebas teniendo la misma entrada para todas. Posterior a las pruebas se realizó un análisis de los resultados en donde se hicieron observaciones para ser documentadas. Cada uno de los resultados fueron comparados con el fin de depurar de forma efectiva el circuito evaluado. Como se ve en la Figura 6, estos 3 últimos pasos se repitieron de forma constante. La mayor parte de este trabajo giró entorno a estos tres últimos pasos.

CAPÍTULO 8

Modelado del diseño HDL

El estilo de codificación en Verilog es el factor mas importante que afecta la simulación de un diseño en VCS. La forma en la que se describe un diseño representa una gran diferencia entre una simulación libre de errores y una que presente problemas como condición de carrera o un mal desempeño.

8.1. Evitar condiciones de carrera

Una condición de carrera se define como un estilo de codificación para el cual hay más de un resultado correcto. Debido a que el resultado de la salida es impredecible pueden existir problemas durante la simulación en VCS. Muchas veces se crean condiciones de carreras por el estilo de diseño utilizado. A continuación se presentan situaciones comunes en las que se pueden presentar condiciones de carrera en el diseño de un circuito.

8.1.1. Ajustar y usar un valor al mismo tiempo

Este problema puede ocurrir al usar bloques paralelos sin un orden garantizado. El ejemplo en la Figura 7 muestra un módulo en donde se evalúa una condicional sobre un registro y al mismo tiempo se toma una decisión sobre esta condición.[5]

```
module ejemplo_rc1;
    reg reg_eval;
    initial
    begin
        reg_eval = 0;
        #10 reg_eval = 1;
    end
    initial
    begin
        #10 if (reg_eval) $display("Puede que esto no se imprima");
    end
endmodule


%Recomendación:
    .
    .
    .
    initial
    begin
        #10 if (reg_eval)
            #0 $display("Puede que esto no se imprima")
    end
endmodule
```

Figura 7: Ejemplo de condición de carrera creado al usar y definir valores al mismo tiempo

En este ejemplo los bloques paralelos no tienen un orden específico por lo cual es incierto si el *display* se va a ejecutar o no. De igual forma en la Figura 3 se muestra una posible solución para evitar este tipo de condiciones de carrera.

8.1.2. Ajustar un valor dos veces al mismo tiempo

Esta condición de carrera ocurre cuando no tenemos un orden garantizado al ajustar dos veces el mismo valor en bloques paralelos. En el ejemplo de la Figura 4 se muestra un ejemplo de este tipo de condición de carrera. La condición de carrera ocurre en el tiempo 5 ya que estos bloques son paralelos y se ejecutan al mismo tiempo. El valor del registro "condicion" es incierto en ese tiempo. La recomendación para evitar esta condición de carrera es dejar un pequeño retardo entre cada ajuste que realicemos al registro.

8.1.3. Condición de carrera en *flip-flops*

Debido al estilo de codificación utilizado, al usar *flip-flops* es muy común tener condiciones de carrera. En el siguiente ejemplo se muestra un caso muy frecuente en el cual el nodo *samp* se ajusta y se muestrea al mismo tiempo. El valor de la salida del segundo *flip-flop* es incierto debido a que hay una carrera entre el valor de *samp* como salida en el primero *flip-flop* y el valor del mismo al ser muestreado en el segundo *flip-flop* como entrada.

```
module dff(q,d,clk);
    output q;
    input d, clk;
    reg q;
    always @(posedge clk)
    q = d;
endmodule

module fftest(out,in,clk);
    input in, clk;
    output out;
    wire samp;
    dff dffa(samp,in,clk);
    dff dffb(out,samp,clk);
endmodule

// Solución 1
always @(posedge clk)
    q <= d;

// solución 2
always @(posedge clk)
    q = #1 d;

//solución 3
always @(posedge clk)
    q <= #1 d;
```

Figura 8: Ejemplo de condición de carrera creado en flip-flops

Para esta condición de carrera tenemos varias posibles soluciones.Como primera solución podemos colocar un non-blocking *assignment* en el *flip-flop* para garantizar el orden de asignación en en la instancia del *flip-flop*. La segunda solución es darle un retardo a la salida de *flip-flop*. La tercera solución es darle un retardo adicional al uso de un *non-blocking assignment*. Cada una de estas soluciones se puede ver en Figura 8.

8.1.4. Evaluación de asignaciones continuas

En VCS, las asignaciones continuas a menudo se propagan mas rápido que en Verilog-XL. Este comportamiento se puede tomar como correcto pero puede causar condiciones de carrera.

```
assign a = b;
initial begin
    b = 2'b11;
    #1
    y = 2'b00;
    $display(a);
end

// Solución
a <= b
```

Figura 9: Ejemplo de condición de carrera por evaluación de asignaciones continuas

El ejemplo mostrado en al Figura 9 muestra este caso. En VCS puede que se tengan resultados distintos que al usar otro simulador como Verilog-XL. Esto es debido a que la asignación a al registro compite con el uso del *display* para usar ese valor. La solución nuevamente es utilizar un *non-blocking assignment* para garantizar un orden en el uso de los recursos.

8.1.5. Condición de carrera en el tiempo cero

Esta condición de carrera es muy común a la hora de describir un circuito y al memento de hacer pruebas sobre el mismo. La Figura 7 muestra un ejemplo de esta condición de carrera, la cual se presenta debido a que la asignación de 1 al *clk* ocurra antes o después de la evaluación de flanco del mismo *(always@(posedge clk))*. La solución para este problema es garantizar que ningún cambio de flanco se haga en el reloj en el tiempo 0. Podemos lograr esto asignándole un valor de contención al reloj y en el tiempo cero y luego de cierto tiempo darle un valor como se muestra en la solución de la Figura 10.

```
always @(posedge clk)
    $display("Puede que esto se imprima o no")
initial begin
    clk = 1;
    forever #50 clk = ~clk;
end

//solución
    .
    .
    .
initial begin
    clk = 1'bx;
    #50 clk = 1'b0;
    forever #50 clk = ~clk;
end
```

Figura 10: Ejemplo de condición de carrera en el tiempo cero de simulación

CAPÍTULO 9

Flujos y funcionalidades propuestas

Con fines de ajustar diferentes tipos de flujos según la complejidad o el detalle de la simulación requerida, se diseñaron diferentes flujos sobre la herramienta VCS. Debido a que el objetivo de las simulaciones es verificar el comportamiento del circuito sintetizado bajo las librerías del fabricante, para cada uno de los flujos propuestos se incluyen dos lineas de comandos. Una para las simulaciones del circuito original y otra para el circuito ya sintetizado. En este último se generan dos módulos, un módulo corresponde al circuito sintetizado con las librerías de TSMC de 180 nm y otro módulo de entradas y salidas que permite a nuestro circuito sintetizado poder interactuar con el mundo exterior. El módulo que contiene las entradas y salidas será el de interés. Cada una de las pruebas realizadas y documentadas en este trabajo serán, como se mencionó anteriormente, comparando el circuito diseñado originalmente con el módulo de entradas y salidas dentro del circuito sintetizado en Design Vision. De igual forma se pueden realizar simulaciones sobre el módulo del circuito sintetizado con la única diferencia de que las librerías utilizadas para la simulación son distintas a las del modulo de entradas y salidas. Las librerías necesarias para la simulación de ambos módulos se encuentran en la Sección 9.1.

9.1. Librerías y archivos utilizadas para la simulación

En el proceso de simulación de un diseño de Verilog se necesitan de archivos adicionales para llevar a cabo el proceso. Si la simulación es sobre el diseño original del circuito se necesita del archivo Verilog con el diseño y de un *testbench* para poder probarlo. Por otro lado, si la simulación es sobre el circuito sintetizado necesitamos de dos archivos adicionales al diseño y el *testbench* para poder realizar las simulaciones. Al momento de ejecutar nuestro flujo de simulación debemos agregar las librerías que hacen referencia a los módulos sintetizados.

9.1.1. Librería para módulo E/S

Al realizar la síntesis en Design Vision se referencia cada uno de nuestros componentes a módulos de las librerías que se van a utilizar (en este caso librerías de TSMC de 180 nm). En este proceso se crea un módulo en entradas y salidas que permite a nuestro chip interactuar con el mundo exterior. Como se mencionó en 9 este módulo será el se simulará para los resultados de este trabajo. El módulo de I/O hace referencia a otra librería por lo que es necesario incluir este archivo en nuestro flujo de simulación. Esta librería se encuentra en la ruta: */usr/synopsys/TSMC/180/CMOS/G/1O3.3V/iolib/LINEAR/tpd018nv_280a_FE/TSMCHOME/digital/Front_End/verilog/"bajo el nombre de "tpd018nv_260a* [12].

9.1.2. Librería para módulo original

Para simular el módulo independiente (sin entradas y salidas) debemos utilizar distintas librerías. Para este caso debemos incluir en los flujos de simulación la librería de TSMC 180nm correspondiente. Este archivo se encuentra en la siguiente ruta:*/TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/tcb018gbwp7t_270a/frame_only/tcb018gbwp7t/LM/* bajo el nombre de tcb018gbwp7t.v [12].

9.2. Flujo de simulación básico

El flujo descrito en esta sección corresponde al creado por el equipo de trabajo antecesor a este. Este flujo se utilizó para la simulación en Verdi del circuito original y el sintetizado con el objetivo de compararlos.

VCS_HOME=/usr/synopsys/vcs-mx
export VCS_HOME
PATH=$VCS HOME/bin:$PATH
export PATH
PATH=usr/synopsys/verdi/bin:$PATH
export PATH

vcs -Mupdate -RPP -v /yourPath/chip.v /yourPath/testBench.v -o demo -full64 -debug_all

vcs -V -R /yourPath/saed90nm.v /yourPath/chip_syn.v /yourPath/testBench_s.v -o yourDesignName -full64 -debug_all

./yourDesignName -gui [12]

9.3. Flujo de simulación con detección de condiciones de carrera

Como se menciona en 8 las condiciones de carrera pueden presentar inconsistencias en las simulaciones ejecutadas. VCS proporciona dos funciones que permiten evaluar las condiciones de carrera que puedan presentarse al momento de describir nuestro circuito en Verilog: herramienta de detección de carrera dinámica y estática. Al habilitar estas funciones se generan archivos como parte del reporte de VCS acerca de las condiciones de carrera encontradas. En estos archivos se detallan tanto el tipo de condición de carrera encontrado como el tiempo en el que se dio, linea del archivo Verilog en donde se encuentra la condición, entre otros.

9.3.1. Herramienta de detección de carrera dinámica

Esta herramienta permite encontrar dos tipos básicos de condiciones de carreras al momento de simular un diseño: Condiciones de lectura-escritura y condiciones de escritura-escritura. Las condiciones de carrera lectura-escritura se dan cuando tanto la escritura como la lectura de una señal ocurre en el mismo tiempo de simulación. La condición de carrera escritura-escritura ocurre cuando múltiples escrituras se realizan sobre una señal en el mismo tiempo de simulación. Para habilitar esta función se coloca la opción de compilación *-race*. Para evaluar las condiciones de carreras dinámicas solo en una porción del diseño se usa la opcion de compilación *-racecd* encerrando la porción del diseño que se evaluará con las directivas *'race* y *'endrace*. Al terminar el proceso de simulación VCS genera dos archivos *race.out* y *race.unique.out*. El archivo *race.out* contiene una linea para todas las condiciones encontradas durante la simulación. El archivo *race.unique.out* contiene solamente las lineas para las condiciones de carrera que son únicas y que no han sido reportadas en las lineas anteriores [5].

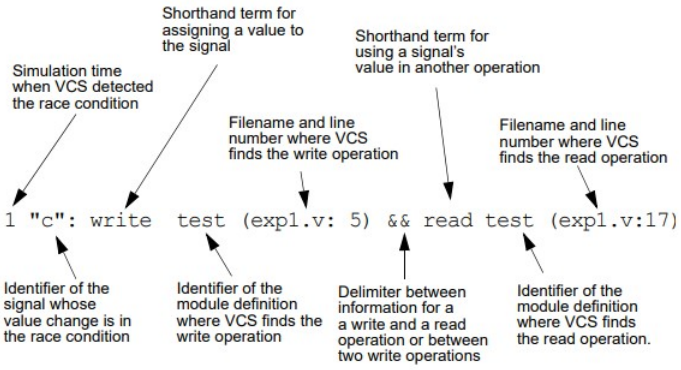


Figura 11: Explicación de una linea en el reporte de condiciones de carrera [5].

En la Figura 11 se muestra un ejemplo de una de las lineas encontradas en un reporte de condiciones de carrera detallando el significado de cada una de las secciones que la componen. Los dos tipos de condiciones de carrera que detecta esta herramienta se muestran en la Figura 8.1.2 y 8.1.1. A continuación se muestra el flujo propuesto el cual incluye

incluye la simulación del modulo y la evaluación de condiciones de carrera de este. VCS_-HOME=/usr/synopsys/vcs-mx
export VCS_HOME
PATH=$VCS HOME/bin:$PATH
export PATH
PATH=usr/synopsys/verdi/bin:$PATH
export PATH
% vcs -V -R tcb018gbwp7t.v tpd018nv.v yourDesign.v yourTestbench.v -o racesim -race -full64 -debug_all
./racesim -gui

Al tener el comando run-time *-gui* este nos abrirá la interfaz gráfica de Verdi lo cual nos permitirá ver las señales de cada uno de los registros en nuestro diseño. El flujo mostrada evaluará las condiciones de carrera en todo nuestro circuito como se mencionó ene este capitulo. Si se desea evaluar solo una porción del diseño cambiar solamente la opción de compilación *-race* por *-racecd* y encerrar la porción entre las directivas *race* y *endrace*. Los reportes *race.out* y *race.unique.out* se generaran al terminar la tarea de simulación gráfica en Verdi.

9.3.2. Herramienta de detección de carrera estática

En nuestro diseño es posible que se formen loops al combinar grupos de declaraciones. Es posible que los ciclos de estos loops varíen según el simulador utilizado. Estos loops también son considerados como condiciones de carrera. Esto ocurre cuando eventos de control que se encuentran justo después de un *always* (y que no contienen un *posedge* o *negedge*) y declaraciones de asignación de procedimiento en un bloque *always* se combinan con otras declaraciones como instancias de módulos o asignaciones continuas. Dichas condiciones de carrera pueden evitarse al colocar un delays dentro de las asignaciones o usando *non-blocking assignment*[5].En esta etapa tenemos varias funcionalidades que podemos aplicar para la evaluación de este tipo de condiciones de carrera. Para esta parte del flujo se proponen dos pruebas sobre el diseño evaluado, una para evaluar las condiciones de carrera mencionadas en esta sección y otra para evaluar las condiciones de carrera que puedan existir entre el reloj y los datos. El flujo de dos pasos propuesto es el siguiente:

VCS_HOME=/usr/synopsys/vcs-mx
export VCS_HOME
PATH=$VCS HOME/bin:$PATH
export PATH
PATH=usr/synopsys/verdi/bin:$PATH
export PATH
% vcs yorDesign.v yourLibrary.v -hsopt=racedetect
% ./simv
% cat hsRaceInfo.db

El comando *+race=all* corresponde a la herramienta para el análisis de condiciones de carreras estáticas. Cuando la compilación de diseño termine VCS generará un archivo con el nombre *race.out.statics*. En 12 se muestra un ejemplo del reporte generado sobre un módulo simple diseñado en [5]. En la segunda parte de este flujo se utilizará el comando *-hsopt=racedetect*, este corresponde a la herramienta de detección de condiciones de carrera entre el reloj y los datos. Los resultados de esta herramienta se guardan en el archivo de salida *hsRaceInfo.db*. Para leer este tipo de archivos utilizaremos el comando *cat* como se muestra en el flujo propuesto. La estructura del reporte generado en este archivo se muestra en 13. En 2 se describe a detalle cada parte del reporte generado por la herramienta de detección de carreras entre reloj y datos[5].

Module:

```
35   always @( A or C ) begin
36     D = C;
37     B = A;
38   end
39
40   assign C = B;
```

Report:

```
       "source.v", 35: The trigger 'C' of the always block
can cause
        the following sequence of event(s) which can again
trigger
         the always block.
        "source.v", 37: B = A;
        which triggers 'B'.
        "source.v", 40: assign C = B;
        which triggers 'C'.
```

Figura 12: Módulo y reporte de condiciones carreras estáticas creado en Manual de VCS [5].

```
Clock Data Race detected @ <n> Module: <Module_name>
Instance: <Instance_name> Line: <Line_no> File: <File_name>
RTL_MOD_NAME: <Module_name> Object: <Data_signal>
```

Figura 13: Estructura generada en la evaluación de condiciones de carrera de reloj y datos [5].

Item	Descripción
<n>	Tiempo en el cual la condición de carrera ocurrió
$< Module\_name >$	Módulo en cual se detectó la condición de carrera
$< Instance\_name >$	Marcador jerárquico hacia una instancia específica del módulo
$< Line\_no >$	Número de linea del archivo de origen
$< File\_name >$	Nombre del archivo de origen
$< Module\_name >$	Nombre del módulo
$< Darta\_signal >$	Señal de datos detectada en la condición de carrera reloj-data

Cuadro 1: Descripción de estructura de reporte de condiciones de carrera reloj-data

Item	Descripción
Mupdate	Habilita la creación de una carpeta para almacenar todo los archivos generados en la compilación del diseño lógico
v	Habilita el modo "verbose"que da mas detalle del proceso de compilación y simulación
RPP	Habilita el modo post procesamiento
o	permite colocar un nombre al ejecutable de salida
$debug\_access + all$	permite utilizar la interfaz gráfica para la simulación
full64	Se especifica que el proceso se realizará sobre una computadora de 64 bits

Cuadro 2: Descripción de estructura de reporte de condiciones de carrera reloj-data

9.4. Flujo para la evaluación de desempeño

VCS cuenta con una herramienta conocida como generador de perfiles de simulación unificado. Con esta herramienta podemos saber la cantidad de tiempo de CPU y memoria usada por nuestro diseño. Esta herramienta crea un reporte creado por el generador de reportes *profrpt*. El reporte se puede generar en formato de texto o html[5]. Para el uso de esta herramienta se propone el siguiente flujo:

VCS_HOME=/usr/synopsys/vcs-mx
export VCS_HOME
PATH=$VCS HOME/bin:$PATH
export PATH
PATH=usr/synopsys/verdi/bin:$PATH
export PATH
% vcs tcb018gbwp7t.v tpd018nv.v yourDesign.v yourTestbench.v -simprofile=time
./simv -simprofile mem (or time)
profrpt simprofile_dir -view time_summary -format text -outout NOTperformance -filter 0.0001

Este flujo genera una salida como reporte en formato de texto o html (en este caso texto) en donde se describe la información relacionada con la opción de compilación seleccionada para la herramienta.

Para la evaluación del desempeño VCS genera varios archivos que son necesarios al momento de compilar el diseño bajo esta función. De estos archivos se utiliza el directorio *simprofile_ dir*(este sera el nombre por defecto que el directorio adquiere) para poder correr

las pruebas de desempeño. Synopsys recomienda evaluar cada una de las funciones(tiempo y memoria) por separado. Los archivos generados en el tiempo de compilación serán diferentes para cada una de las pruebas.

En la Tabla 3 se muestran cada uno de los comandos utilizados para la generación del reporte con su descripción correspondiente.

Comando	Descripción
$-view$	Especifica las vistas que se quieran ver en el reporte
$-format$	especifica si el formato del reporte es texto, html o ambos formatos
$-output$	Especifica el nombre del directorio de salida que contiene el reporte generado
$-filter$	especifica el porcentaje mínimo de memoria o tiempo de CPU que un módulo, instancia o estructura necesita usar antes de que la herramienta profrpt habilite la reportería. Para simulaciones mas precisa colocar el porcentaje en 0.0001

Cuadro 3: Descripción de los comandos para la generación de reporte de performance

Nota importante: si no es primera vez que se compila el diseño, borrar las los directorios csrc y simv.daidir y el ejecutable simv antes de compilar el diseño con esta herramienta.

CAPÍTULO 10

Comparación de diseños en *Formality*

Como alternativa a la validación en las herramientas de VCS y Verdi de los diseños elaborados, se utiliza *Formality* para la validación de estos. Como se mencionó en 6.7 esta herramienta es utilizada para comparar dos circuitos (pre y post síntesis en este caso) para validar que el proceso de síntesis se realizó de forma correcta. *Formality* presenta una ventaja ante VCS cuando el diseño evaluado posee una cantidad alta de entradas y salidas [12]. *Formality* brinda un reporte mas práctico al tener como salida un reporte simple de coincidencias o inconsistencias en la comparación de dos circuitos. La herramienta proporciona una serie de pasos divididos en pestañas para realizar la comparación de los diseños. Los pasos generales son:

- *0. Guid*: Se agregan los archivos (.svf) asociados a incidencias de otras herramientas en el diseño.
- *1. Ref*: Se carga el primer diseño a comparar según su formato al igual que las librerías utilizadas para la elaboración de este.
- *2. Impl*: Al igual que *1. Ref*, en este paso se carga el segundo diseño a comparar con las librerías asociadas a este.
- *3. Setup*: En este paso se coloca restricciones que queremos que la verificación tome en cuenta. Por ejemplo excluir cierta parte del diseño del proceso.
- *4. Match*: En este paso se realizará la primera verificación de igualdad por parte de *Formality* en los diseños.
- *5. Verify*: En este paso se realizará la segunda verificación de igualdad por parte de *Formality* en los diseños

El flujo en esta herramienta, al ser en su totalidad sobre una interfaz gráfica, se muestra en los vídeo tutoriales adjuntos a esta investigación. Cada una de las pruebas realizadas bajo esta herramienta será utilizando el diseño original(no sintetizado) y el diseño original sintetizado (sin módulo de I/Os) debido a dos razones, la primera debido a que el módulo agrega ciertos puertos adicionales de entradas y salidas causando inconsistencias en la comparación de ambos diseños; segundo debido a que en los flujos presentados en 9 ya se evalúa dicho módulo.

CAPÍTULO 11

Flujos de simulación sobre circuitos de complejidad baja

Para este capítulo se utilizaran cada una de las herramientas y flujos descritos en el Capítulo 9. Como se menciona en el Capítulo 4.2 uno de los objetivos de las simulaciones es comparar el funcionamiento de nuestro diseño original con el diseño sintetizado bajo las librerías de TSMC. Para estas pruebas y las siguientes se usara el flujo descrito en 9.2 para esta comparación. Los resultados con el uso de este flujo serán los esquemáticos y las señales de los registros de cada módulo de la simulación generados por la herramienta Verdi en su interfaz gráfica. Esto permitirá validar el comportamiento esperado (que los resultados del diseño original y el sitetizado sean iguales) por parte de los módulos evaluados.

11.1. Flujos sobre una compuerta NOT

Como parte del proceso para la realización del proyecto general al que este trabajo pertenece, se utilizó una compuerta NOT para validar el flujo completo en el diseño de un chip con tecnología VLSI CMOS. Esto con el objetivo de completar el flujo de diseño general con un diseño con complicidad menor (fácil debuggeo) para luego ir aumentando la complejidad del circuito diseñado. A continuación se muestran los resultados sobre un diseño de una compuerta NOT en Verilog.

11.1.1. Flujo básico

Como se mencionó en 11 este flujo se utilizará principalmente para comparar el correcto funcionamiento del circuito sintetizado al compararlo con el diseño original codificado en Verilog. Para esto se corrió el flujo en ambos diseños. Para todas las prubas realizadas bajo este flujo se utilizaran las librerías correspondientes, el diseño a simular y un *testbench*.

El *testbench* utilizado para las simulaciones se muestra en la Figura 14. Tanto el *testbench* utilizado para el circuito sintetizado como el utilizado para la circuito original tendrán las mismas entradas en el mismo tiempo de simulación para poder validar ambos circuitos al momento de compararlos.

```
`timescale 1ns/1ns
module stimulus(  );
  reg A;
  wire Y, dummy;

  Not_IO Not_IO(.A(A),.Y(Y));


  initial
  begin
  A = 1'b1;
  #10 A = 1'b1;
  #10 A = 1'b0;
  #10 A = 1'b1;
  #10 A = 1'b0;
  #10 A = 1'b1;
  #10 A = 1'b0;
  #10 A = 1'b1;
  #10 A = 1'b0;
  #10 A = 1'b1;
  #110 $finish;
  end


endmodule
```

Figura 14: Modelo de testbench para la simulación de compuerta NOT en Verilog

Al utilizar Verdi para la simulación de los circuitos, este nos permite obtener los esquemáticos de nuestros diseños en base a los módulos utilizados. En las figuras 15 y 16 se muestran los esquemáticos para la compuerta NOT sintetizada y para la compuerta NOT original respectivamente. Al correr el módulo creado por Verdi para nuestro diseño podemos generar una vista con el comportamiento de las señales de nuestro diseño en base al testbench asignado. En las figuras 17 y 18 podemos ver el comportamiento de las entradas y salidas del diseño original y sintetizado respectivamente. Como primer punto podemos notar que el comportamiento del circuito sintetizado es el esperado, ya que este cumple con la función de una compuerta NOT. Al comparar las señales de las figuras 17 y 18 podemos notar que estas son idénticas por lo que podemos asegurar que la síntesis del circuito se realizó de forma correcta.

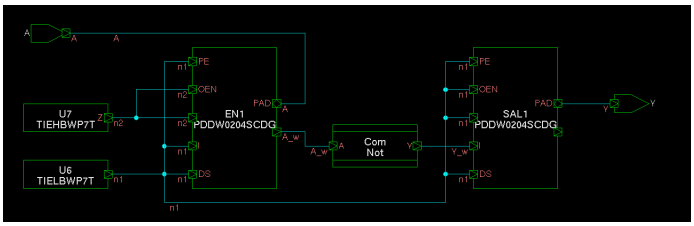


Figura 15: Esquemático de compuerta NOT sintetizada con módulo de I/Os

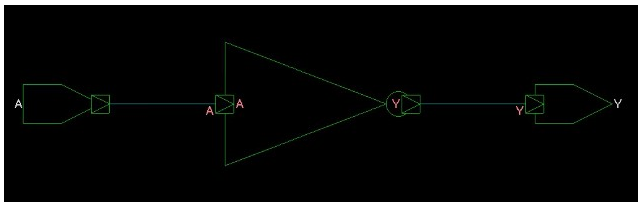


Figura 16: Esquemático de compuerta NOT

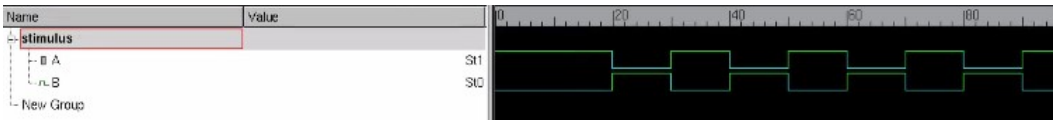


Figura 17: Señales de entrada y salida de compuerta NOT

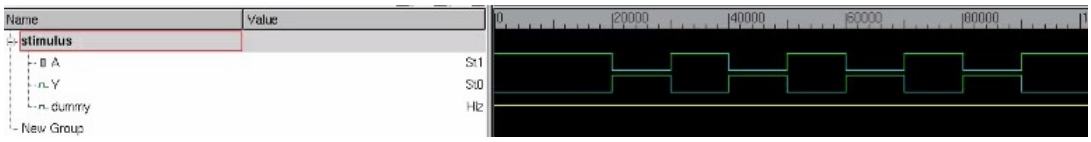


Figura 18: Señales de entrada y salida de compuerta NOT sintetizada

11.1.2. Flujo de simulación con detección de condiciones de carrera

Al ser un circuito de una única entrada y salida, los resultados encontrados bajo este flujo no son significativos. De igual forma, cada uno de los flujos se corrieron sobre este diseño. En las figuras 19 y 20 se muestras los resultados de los reportes en la evaluación de condiciones de carrera dinámicos y estáticos respectivamente. Como era de esperarse estos se encuentran vacíos.

Figura 19: Reporte de condiciones de carreras dinámicas en compuerta NOT

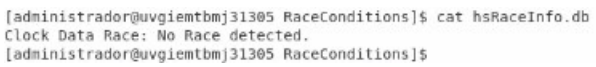


Figura 20: Reporte de condiciones de carreras estáticas en compuerta NOT

11.1.3. Evaluación de desempeño

Utilizando el flujo en la sección 9.4 se pueden obtener los resultados para memoria y tiempo de CPU que el equipo utilizará al correr la simulación sobre VCS. Los resultados de desempeño para la compuerta NOT se muestran en las figuras 21 y 22

```
##################################################################################################################
     VCS build date:                                  Feb 28 2019 22:34:30
   Compiler version:                                  O-2018.09-SP2_Full64
    Runtime version:                                  O-2018.09-SP2_Full64
       Machine Name:                                  uvgiemtbmj31305
     Profile runner:                                  administrador
       Profile data:                                  /home/administrador/Escritorio/Chip/VCS/NOTJEFF/RaceCon
                                                      ditions/simprofile_dir

      Creation date:                                  Wed Sep 16 20:59:47 2020
      Profile start:                                  0
       Profile stop:                                  200000
     Total CPU Time:                                  0.23
    Compile Command:                                  /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                                      8nv.v NOT_syn_tsmc.v testBench_sio.v -o demo -simprofil
                                                      e -full64 -debug_all
 Simulation Command:                                  ./demo -simprofile time
##################################################################################################################
==================================================================================================================
                                                Time Summary View
==================================================================================================================
   Component                                                                                    Percentage
------------------------------------------------------------------------------------------------------------------
  KERNEL                                                                                         80.95%
  HSIM                                                                                            9.52%
  PLI/DPI/DirectC                                                                                 9.52%
------------------------------------------------------------------------------------------------------------------
  TOTAL                                                                                         100.00%
```

Figura 21: Reporte de tiempo de CPU en simulación de compuerta NOT

```
##################################################################################################################
     VCS build date:                                  Feb 28 2019 22:34:30
   Compiler version:                                  O-2018.09-SP2_Full64
    Runtime version:                                  O-2018.09-SP2_Full64
       Machine Name:                                  uvgiemtbmj31305
     Profile runner:                                  administrador
       Profile data:                                  /home/administrador/Escritorio/Chip/VCS/NOTJEFF/RaceCon
                                                      ditions/simprofile_dir
      Creation date:                                  Wed Sep 16 21:17:15 2020
      Profile start:                                  0
       Profile stop:                                  200000
     Total CPU Time:                                  0.40
    Compile Command:                                  /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                                      8nv.v NOT_syn_tsmc.v testBench_sio.v -o demo -simprofil
                                                      e -full64 -debug_all
 Simulation Command:                                  ./demo -simprofile mem
##################################################################################################################
==================================================================================================================
                                              Peak Memory Summary View
==================================================================================================================
   Component                                          Memory                                    Percentage
------------------------------------------------------------------------------------------------------------------
  KERNEL                                             6.90 M                                      3.07%
  PLI/DPI/DirectC                                    6.10 M                                      2.71%
  HSIM                                               1.40 M                                      0.62%
  VERILOG                                          541.85 K                                      0.24%
    Module                                         541.85 K                                      0.24%
    Function Coverage Kernel                             584                                             0.00%
  Anonymous                                         38.07 M                                     16.92%
  Library/Executable                               172.00 M                                     76.44%
    VCS                                            131.00 M                                     58.22%
    Third-party                                     41.00 M                                     18.22%
      libc-2.17                                      3.79 M                                      1.68%
      libm-2.17                                      3.01 M                                      1.34%
      libstdc++                                      2.95 M                                      1.31%
      libdw-0.176                                    2.32 M                                      1.03%
      liblzma                                        2.15 M                                      0.95%
      libpthread-2.17                                2.09 M                                      0.93%
      libresolv-2.17                                 2.09 M                                      0.93%
      libelf-0.176                                   2.09 M                                      0.93%
      libgcc_s-4.8.5-20150702                          2.09 M                                            0.93%
      libz                                           2.09 M                                      0.93%
      libnss_myhostname                               2.08 M                                      0.93%
      libnss_files-2.17                               2.05 M                                      0.91%
      librt-2.17                                     2.03 M                                      0.90%
      libnss_dns-2.17                                2.03 M                                      0.90%
      libcap                                         2.02 M                                      0.90%
      libattr                                        2.02 M                                      0.90%
      libdl-2.17                                     2.02 M                                      0.90%
      ld-2.17                                      144.00 K                                      0.06%
------------------------------------------------------------------------------------------------------------------
  TOTAL                                            225.00 M                                    100.00%
```

Figura 22: Reporte de desempeño de memoria en simulación de compuerta NOT

11.1.4. Verificación en *Formality*

Se corrió el flujo mencionado en 6.7 para la verificación de los diseños pre o post síntesis obteniendo los resultados de la Figura 23. En el reporte se puede observar las pruebas de *Match* y *Verification* mostrando una igualdad completa entre ambos diseños.

```
Formality (match)> verify
Reference design is 'r:/WORK/Not'
Implementation design is 'i:/WORK/Not'

********************************** Matching Results **********************************
 1 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 1 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
**************************************************************************************

Status:  Verifying...


******************************** Verification Results ********************************
Verification SUCCEEDED
---------------------
 Reference design: r:/WORK/Not
 Implementation design: i:/WORK/Not
 1 Passing compare points
--------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       1       0       0       1
Failing (not equivalent)       0       0       0       0       0       0       0       0
**************************************************************************************
```

Figura 23: Reporte en herramienta *Formality* para diseño de compuerta NOT

11.2. Flujos sobre compuerta NOT con error

En esta sección se muestra una compuerta NOT con errores presentados durante el proceso de síntesis. Dichos errores fueron detectados utilizando el flujo descrito en 9.2.

11.2.1. Flujo básico

Al correr este flujo sobre el circuito sintetizado se pudieron detectar inconsistencias en las señales de salida del circuito. La Figura 24 muestra el esquemático del circuito sintetizado en donde se presenta una contención en la señal de salida. La Figura 25 muestra las señales de salida de la compuerta. Como se puede observar en la señal de salida, esta muestra un estado de contención ante cada cambio de flanco de la señal de entrada por lo que se puede comprobar el mal funcionamiento del circuito. Con esta información se procedió a descartar este diseño sintetizado y regresar al paso anterior del flujo general de diseño para corregir dicho error.

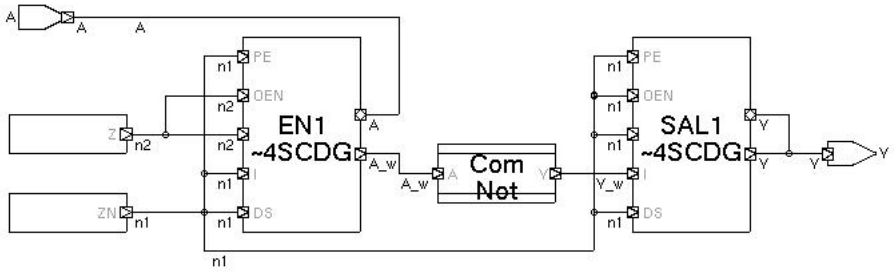


Figura 24: Señales de entrada y salida de compuerta NOT con error

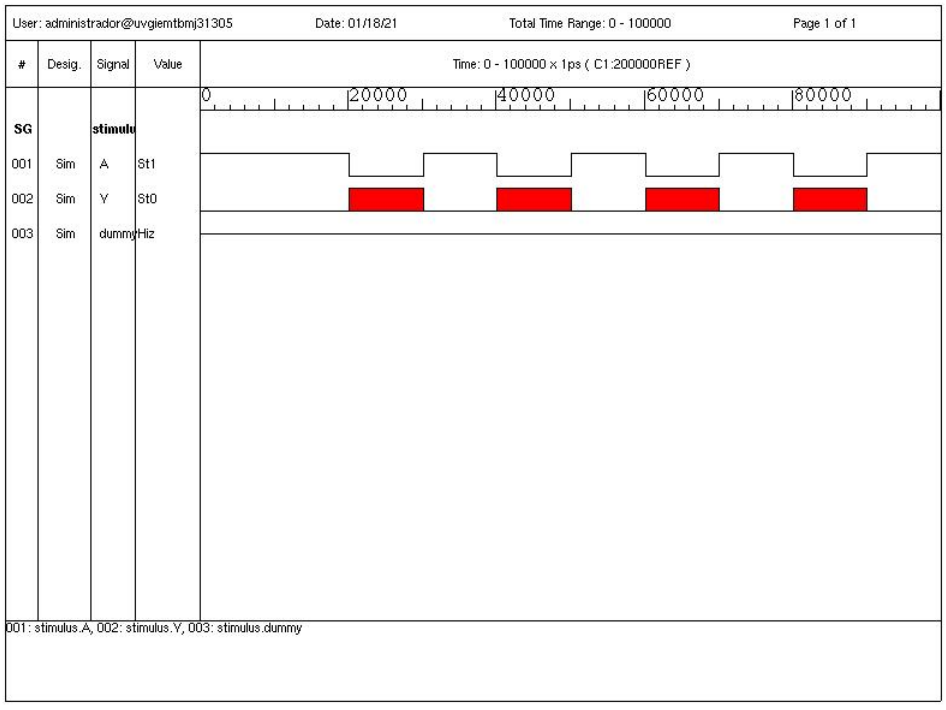


Figura 25: Señales de entrada y salida de compuerta NOT con error

11.3. Flujos sobre un Full-Adder

Como alternativa a la validación de los flujos diseñados se probaron cada uno de ellos sobre otro circuito de complejidad baja conocido como full-adder. Las pruebas se realizaron en la misma forma que en la sección 11.1. El funcionamiento y comportamiento esperado de este circuito se muestra en 3

11.3.1. Flujo básico

En las figuras 27 y 28 se muestran el esquemático y la respuesta de este a las entradas mostradas en 26 para el circuito de un full-adder. En las figuras 29 y 30 se muestran el esquemático y la respuesta de este a las entradas mostradas en 26 para el circuito de un full-adder sintetizado con un módulo de I/Os. Como primer punto podemos notar el correcto funcionamiento de ambos módulos ya que ambos responden de forma esperada ante las entradas impuestas. Por otra parte, al comparar la figura 28 con la 30 notamos que las respuestas son exactamente iguales por lo que podemos decir que para este circuito la síntesis se realizó de forma correcta.

```
`timescale 1ns/1ns
module stimulus(  );
  reg X1,X2,Cin;
  wire S, Cout;

  FA_IO FA_IO1(.X1(X1), .X2(X2), .Cin(Cin), .S(S), .Cout(Cout));


  initial
  begin
  X1 = 1'B1;
  X2 = 1'B0;
  Cin = 1'B1;
  #2 X1 = 1'B0;
  X2 = 1'B0;
  Cin = 1'B1;
  #4 X1 = 1'B1;
  X2 = 1'B1;
  Cin = 1'B0;
  #10 $finish;
  end


endmodule
```

Figura 26: Modelo de *testbench* para la simulación de circuito *full-adder*

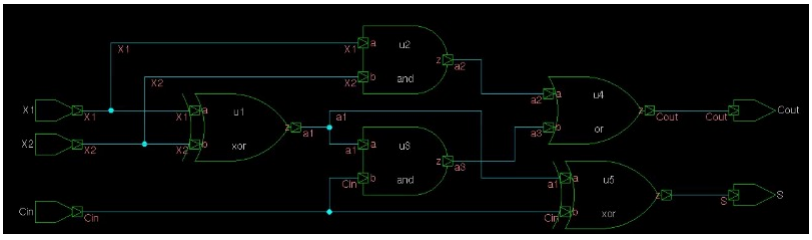


Figura 27: Esquemático de circuito full-adder

Figura 28: Señales de entrada y salida de full-adder

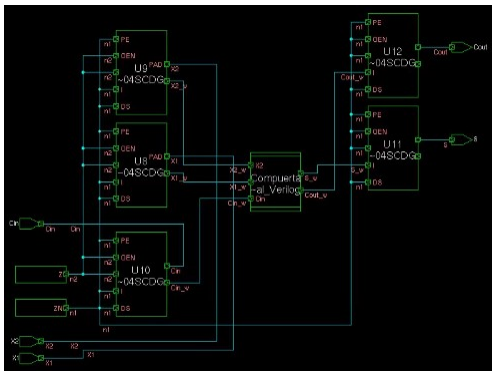


Figura 29: Esquemático de full-adder sintetizado con módulo de I/Os

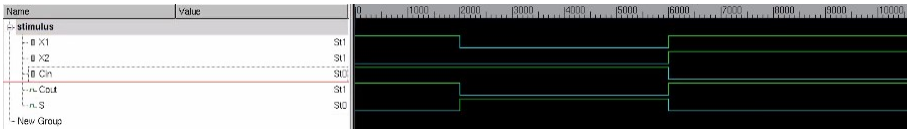


Figura 30: Señales de entrada y salida de full-adder sintetizado

11.3.2. Flujo de simulación con detección de condiciones de carrera

Al igual que en 11.1.1, este es un circuito bastante simple que no esta expuesto en gran parte a la mayoría de condiciones de carrera mencionadas en 8. No se encontraron condiciones de carrera, como se observa en los resultados mostrados en las Figuras 31 y 32.

Figura 31: Reporte de condiciones de carrera dinámicas para full-adder

Figura 32: Reporte de condiciones de estáticas para full-adder

11.3.3. Evaluación de desempeño

Utilizando el flujo en la sección 9.4 se pueden obtener los resultados para memoria y tiempo de CPU que el equipo utilizará al correr la simulación sobre VCS. Los resultados de

desempeño para un full-adder se muestran en la figura 33 y 39

```
##################################################################################################
    VCS build date:                          Feb 28 2019 22:34:30
  Compiler version:                          O-2018.09-SP2_Full64
   Runtime version:                          O-2018.09-SP2_Full64
      Machine Name:                          uvgiemtbmj31305
    Profile runner:                          administrador
      Profile data:                          /home/administrador/Escritorio/Chip/VCS/NOTJEFF/Full_ad
                                             der/simprofile_dir
     Creation date:                          Thu Sep 17 22:35:01 2020
     Profile start:                          0
      Profile stop:                          10000
    Total CPU Time:                          0.21
   Compile Command:                          /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                             8nv.v FA_syn.v testBench_sio.v -o demo -simprofile -ful
                                             l64 -debug_all
Simulation Command:                          ./demo -simprofile time
##################################################################################################
==================================================================================================
                                        Time Summary View
==================================================================================================
  Component                                                                        Percentage
--------------------------------------------------------------------------------------------------
 KERNEL                                                                             75.00%
 HSIM                                                                               10.00%
 PLI/DPI/DirectC                                                                    10.00%
 VERILOG                                                                             5.00%
   Module                                                                            5.00%
--------------------------------------------------------------------------------------------------
  TOTAL                                                                            100.00%
```

Figura 33: Reporte de tiempo de CPU en simulación de full-adder

```
##################################################################################################
    VCS build date:                          Feb 28 2019 22:34:30
  Compiler version:                          O-2018.09-SP2_Full64
   Runtime version:                          O-2018.09-SP2_Full64
      Machine Name:                          uvgiemtbmj31305
    Profile runner:                          administrador
      Profile data:                          /home/administrador/Escritorio/Chip/VCS/NOTJEFF/Full_ad
                                             der/simprofile_dir
     Creation date:                          Thu Sep 17 22:42:45 2020
     Profile start:                          0
      Profile stop:                          10000
    Total CPU Time:                          0.38
   Compile Command:                          /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                             8nv.v FA_syn.v testBench_sio.v -o demo -simprofile -ful
                                             l64 -debug_all
Simulation Command:                          ./demo -simprofile mem
##################################################################################################
==================================================================================================
                                   Peak Memory Summary View
==================================================================================================
  Component                                Memory                                 Percentage
--------------------------------------------------------------------------------------------------
 KERNEL                                    6.90 M                                   3.07%
 PLI/DPI/DirectC                           6.13 M                                   2.73%
 HSIM                                      1.40 M                                   0.62%
 VERILOG                                 547.29 K                                   0.24%
   Module                                547.29 K                                   0.24%
   Function Coverage Kernel                  584                                    0.00%
 Anonymous                                38.03 M                                  16.90%
 Library/Executable                      172.00 M                                  76.44%
   VCS                                   131.00 M                                  58.22%
   Third-party                            41.00 M                                  18.22%
     libc-2.17                             3.79 M                                   1.68%
     libm-2.17                             3.01 M                                   1.34%
     libstdc++                             2.95 M                                   1.31%
     libdw-0.176                           2.32 M                                   1.03%
     liblzma                               2.15 M                                   0.95%
     libpthread-2.17                       2.09 M                                   0.93%
     libresolv-2.17                        2.09 M                                   0.93%
     libelf-0.176                          2.09 M                                   0.93%
     libgcc_s-4.8.5-20150702               2.09 M                                   0.93%
     libz                                  2.09 M                                   0.93%
     libnss_myhostname                     2.08 M                                   0.93%
     libnss_files-2.17                     2.05 M                                   0.91%
     librt-2.17                            2.03 M                                   0.90%
     libnss_dns-2.17                       2.03 M                                   0.90%
     libcap                                2.02 M                                   0.90%
     libattr                               2.02 M                                   0.90%
     libdl-2.17                            2.02 M                                   0.90%
     ld-2.17                             144.00 K                                   0.06%
--------------------------------------------------------------------------------------------------
  TOTAL                                  225.00 M                                 100.00%
```

Figura 34: Reporte de desempeño de memoria en simulación de full-adder

11.3.4. Verificación en *Formality*

Se corrió el flujo mencionado en 6.7 para la verificación de los diseños pre o post síntesis para el diseño de un *full adder* obteniendo los resultados de la figura 55. En el reporte se pueden observar las pruebas de *Match* y *Verification* mostrando una igualdad completa entre ambos diseños.

```
Formality (match)> verify
Reference design is 'r:/WORK/Full_Add'
Implementation design is 'i:/WORK/Full_Adder_Structural_Verilog'

********************************** Matching Results **********************************
 2 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 3 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
******************************************************************************************

Status:  Verifying...


******************************** Verification Results ********************************
Verification SUCCEEDED
---------------------
 Reference design: r:/WORK/Full_Add
 Implementation design: i:/WORK/Full_Adder_Structural_Verilog
 2 Passing compare points
------------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
------------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       2       0       0       2
Failing (not equivalent)       0       0       0       0       0       0       0       0
******************************************************************************************
```

Figura 35: Reporte en herramienta *Formality* para diseño de *full adder*

11.4. Flujos sobre compuerta NAND

Como alternativa a la validación de los flujos diseñados se probaron cada uno de ellos sobre otro circuito de complejidad baja conocido como compuerta NAND. Las pruebas se realizaron en la misma forma que en la sección 11.1.

11.4.1. Flujo básico

En las figuras 36 y 37 se muestran el esquemático y la respuesta del circuito respectivamente en la herramienta Verdi. En los resultados obtenidos, basados en el conocido funcionamiento de una compuerta NAND de dos entradas, se notó el correcto funcionamiento del circuito sintetizado. Basados en estos resultados se concluyó en que la síntesis se realizó de forma correcta.

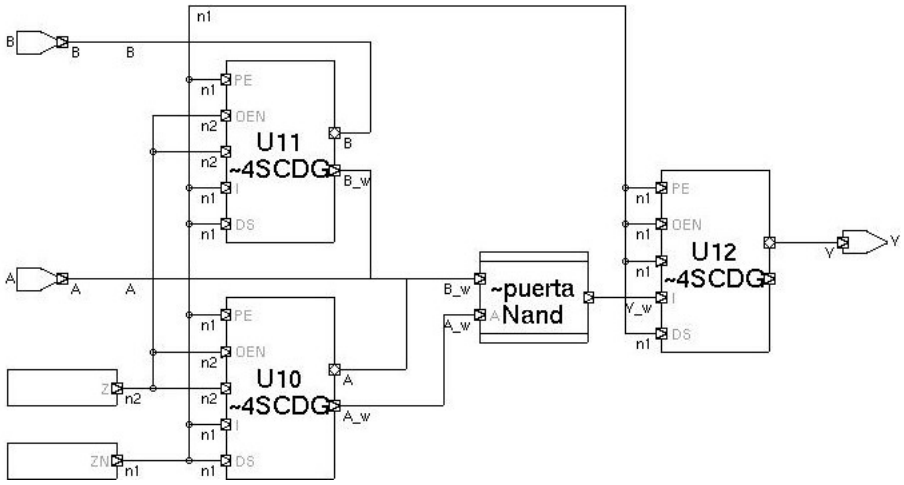


Figura 36: Esquemático de compuerta NAND

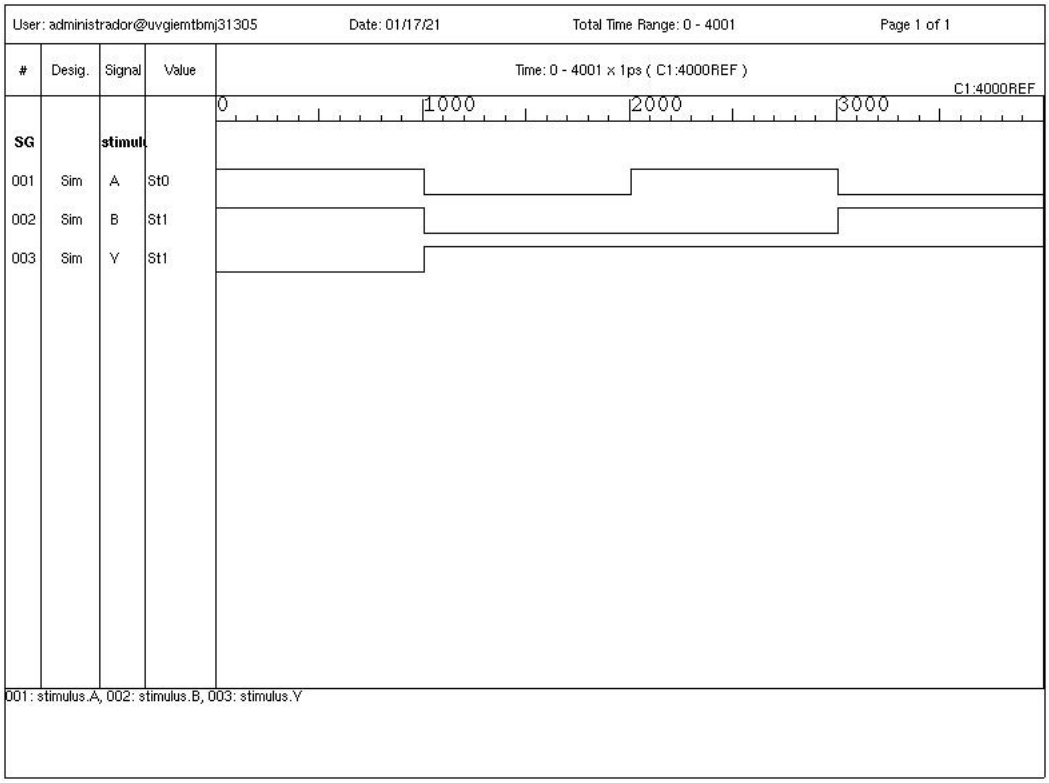


Figura 37: Señales de entrada y salida de compuerta NAND

11.4.2. Evaluación de desempeño

Utilizando el flujo en la sección 9.4 se pueden obtener los resultados para memoria y tiempo de CPU que el equipo utilizará al correr la simulación sobre VCS. Los resultados desempeño para la compuerta NAND se muestran en la Figura 38

```
################################################################################################################
     VCS build date:                                 Feb 28 2019 22:34:30
   Compiler version:                                 O-2018.09-SP2_Full64
    Runtime version:                                 O-2018.09-SP2_Full64
       Machine Name:                                 uvgiemtbmj31305
     Profile runner:                                 administrador
       Profile data:                                 /home/administrador/Escritorio/Chip/VCS/NOTJEFF/NAND/si
                                                     mprofile_dir

      Creation date:                                 Tue Jan 19 23:01:36 2021
      Profile start:                                 0
       Profile stop:                                 4000
     Total CPU Time:                                 0.21
    Compile Command:                                 /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                                     8nv.v NAND_syn.v testBench_sio.v -o cput -simprofile -f
                                                     ull64 -debug_all
 Simulation Command:                                 ./cput -simprofile time
################################################################################################################
================================================================================================================
                                                Time Summary View
================================================================================================================
   Component                                                                              Percentage
----------------------------------------------------------------------------------------------------------------
  KERNEL                                                                                   78.95%
  HSIM                                                                                     10.53%
  PLI/DPI/DirectC                                                                          10.53%
----------------------------------------------------------------------------------------------------------------
   TOTAL                                                                                  100.00%
----------------------------------------------------------------------------------------------------------------
```

Figura 38: Reporte de tiempo de CPU en simulación de compuerta NAND

```
################################################################################################################
     VCS build date:                                 Feb 28 2019 22:34:30
   Compiler version:                                 O-2018.09-SP2_Full64
    Runtime version:                                 O-2018.09-SP2_Full64
       Machine Name:                                 uvgiemtbmj31305
     Profile runner:                                 administrador
       Profile data:                                 /home/administrador/Escritorio/Chip/VCS/NOTJEFF/NAND/si
                                                     mprofile_dir

      Creation date:                                 Tue Jan 19 22:53:06 2021
      Profile start:                                 0
       Profile stop:                                 4000
     Total CPU Time:                                 0.42
    Compile Command:                                 /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                                     8nv.v NAND_syn.v testBench_sio.v -o memory -simprofile
                                                     -full64 -debug_all
 Simulation Command:                                 ./memory -simprofile mem
################################################################################################################
================================================================================================================
                                              Peak Memory Summary View
================================================================================================================
   Component                                          Memory                                  Percentage
----------------------------------------------------------------------------------------------------------------
  KERNEL                                             6.90 M                                    3.07%
  PLI/DPI/DirectC                                    6.10 M                                    2.71%
  HSIM                                               1.40 M                                    0.62%
  VERILOG                                          543.42 K                                    0.24%
    Module                                         543.42 K                                    0.24%
    Function Coverage Kernel                             584                                          0.00%
  Anonymous                                         38.07 M                                   16.92%
  Library/Executable                               172.00 M                                   76.44%
    VCS                                            131.00 M                                   58.22%
    Third-party                                     41.00 M                                   18.22%
      libc-2.17                                      3.79 M                                    1.68%
      libm-2.17                                      3.01 M                                    1.34%
      libstdc++                                      2.95 M                                    1.31%
      libdw-0.176                                    2.32 M                                    1.03%
      liblzma                                        2.15 M                                    0.95%
      libpthread-2.17                                2.09 M                                    0.93%
      libresolv-2.17                                 2.09 M                                    0.93%
      libelf-0.176                                   2.09 M                                    0.93%
      libgcc_s-4.8.5-20150702                        2.09 M                                          0.93%
      libz                                           2.09 M                                    0.93%
      libnss_myhostname                              2.08 M                                     0.93%
      libnss_files-2.17                              2.05 M                                     0.91%
      librt-2.17                                     2.03 M                                    0.90%
      libnss_dns-2.17                                2.03 M                                    0.90%
      libcap                                         2.02 M                                    0.90%
      libattr                                        2.02 M                                    0.90%
      libdl-2.17                                     2.02 M                                    0.90%
      ld-2.17                                      144.00 K                                    0.06%
----------------------------------------------------------------------------------------------------------------
   TOTAL                                           225.00 M                                  100.00%
----------------------------------------------------------------------------------------------------------------
```

Figura 39: Reporte de desempeño de memoria en simulación de compuerta NAND

11.4.3. Verificación en *Formality*

Se corrió el flujo mencionado en 6.7 para la verificación de los diseños pre o post síntesis para el diseño de una compuerta NAND obteniendo los resultados de la Figura 40. En el reporte se pueden observar las pruebas de *Match* y *Verification* mostrando una igualdad completa entre ambos diseños.

```
Formality (match)> verify
Reference design is 'r:/WORK/Nand'
Implementation design is 'i:/TECH_WORK/Nand_IO'

********************************** Matching Results **********************************
 1 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 2 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
**************************************************************************************

Status:  Verifying...


******************************** Verification Results ********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/Nand
 Implementation design: i:/TECH_WORK/Nand_IO
 1 Passing compare points
--------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       1       0       0       1
Failing (not equivalent)       0       0       0       0       0       0       0       0
**************************************************************************************
1
```

Figura 40: Reporte en herramienta *Formality* para diseño de compuerta NAND

11.5. Flujos sobre compuerta NOR

Como alternativa a la validación de los flujos diseñados se probaron cada uno de ellos sobre otro circuito de complejidad baja conocido como compuerta NOR. Las pruebas se realizaron en la misma forma que en la sección 11.1

11.5.1. Flujo básico

En las figuras 41 y 42 se muestran el esquemático y la respuesta del circuito respectivamente en la herramienta Verdi. En los resultados obtenidos, basados en el conocido funcionamiento de una compuerta NOR de dos entradas, se notó el correcto funciona-miento del circuito sintetizado. Basados en estos resultados se concluyó en que la síntesis se realizó de forma correcta.

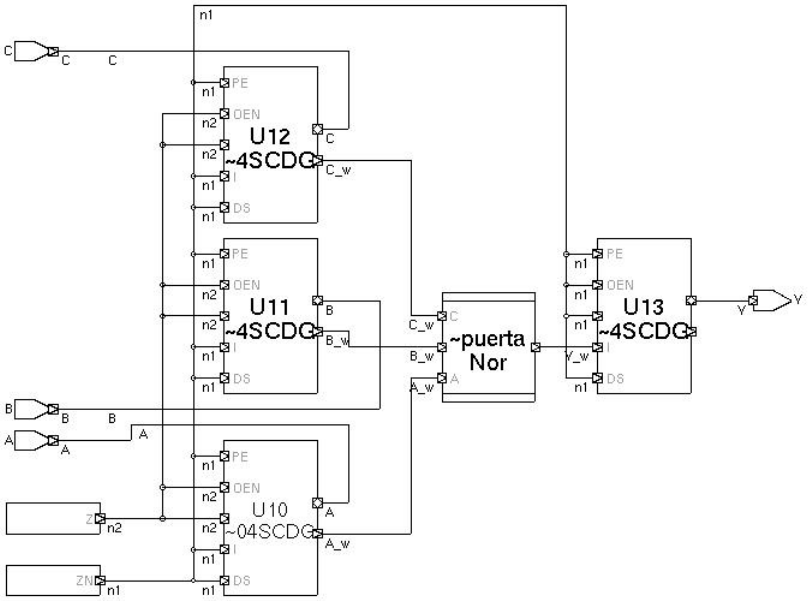


Figura 41: Esquemático de compuerta NOR

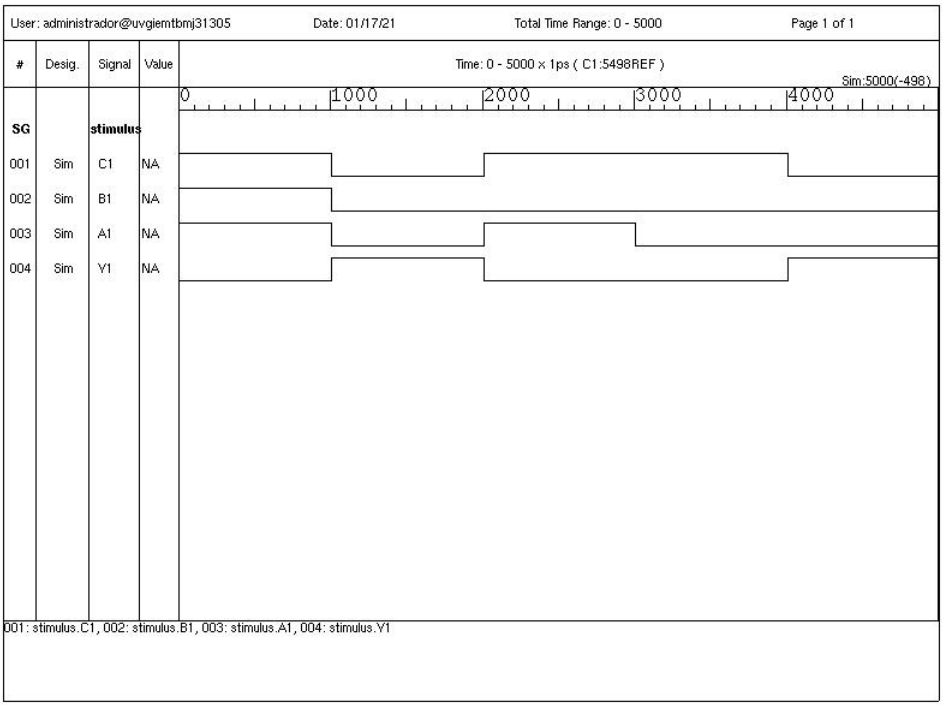


Figura 42: Señales de entrada y salida de compuerta NOR

11.5.2. Evaluación de desempeño

Utilizando el flujo en la sección 9.4 se pueden obtener los resultados para memoria y tiempo de CPU que el equipo utilizará al correr la simulación sobre VCS. Los resultados desempeño de CPU Y memoria para la compuerta NOR se muestran en las figuras 43 y 44

```
##############################################################################################################
     VCS build date:                          Feb 28 2019 22:34:30
   Compiler version:                          O-2018.09-SP2_Full64
    Runtime version:                          O-2018.09-SP2_Full64
       Machine Name:                          uvgiemtbmj31305
     Profile runner:                          administrador
       Profile data:                          /home/administrador/Escritorio/Chip/VCS/NOTJEFF/NOR/sim
                                              profile_dir
      Creation date:                          Tue Jan 19 23:18:56 2021
      Profile start:                          0
       Profile stop:                          5000
    Total CPU Time:                           0.23
    Compile Command:                          /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                              8nv.v NOR_syn.v testBench_sio.v -o cput -simprofile -fu
                                              ll64 -debug_all
 Simulation Command:                          ./cput -simprofile time
##############################################################################################################
==============================================================================================================
                                          Time Summary View
==============================================================================================================
   Component                                                                          Percentage
--------------------------------------------------------------------------------------------------------------
  KERNEL                                                                               81.82%
  HSIM                                                                                  9.09%
  PLI/DPI/DirectC                                                                       9.09%
--------------------------------------------------------------------------------------------------------------
   TOTAL                                                                              100.00%
--------------------------------------------------------------------------------------------------------------
```

Figura 43: Reporte de tiempo de CPU en simulación de compuerta NOR

```
##################################################################################################################
     VCS build date:                                Feb 28 2019 22:34:30
   Compiler version:                                O-2018.09-SP2_Full64
    Runtime version:                                O-2018.09-SP2_Full64
       Machine Name:                                uvgiemtbmj31305
     Profile runner:                                administrador
       Profile data:                                /home/administrador/Escritorio/Chip/VCS/NOTJEFF/NOR/sim
                                                    profile_dir
      Creation date:                                Tue Jan 19 23:14:16 2021
      Profile start:                                0
       Profile stop:                                5000
     Total CPU Time:                                0.40
    Compile Command:                                /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                                    8nv.v NOR_syn.v testBench_sio.v -o memory -simprofile -
                                                    full64 -debug_all
 Simulation Command:                                ./memory -simprofile mem
##################################################################################################################
==================================================================================================================
                                              Peak Memory Summary View
==================================================================================================================
   Component                                      Memory                                  Percentage
------------------------------------------------------------------------------------------------------------------
  KERNEL                                          6.90 M                                    3.07%
  PLI/DPI/DirectC                                 6.13 M                                    2.73%
  HSIM                                            1.40 M                                    0.62%
  VERILOG                                       545.38 K                                    0.24%
    Module                                      545.38 K                                    0.24%
    Function Coverage Kernel                         584                                    0.00%
  Anonymous                                       38.04 M                                  16.90%
  Library/Executable                             172.00 M                                  76.44%
    VCS                                          131.00 M                                  58.22%
    Third-party                                   41.00 M                                  18.22%
      libc-2.17                                    3.79 M                                   1.68%
      libm-2.17                                    3.01 M                                   1.34%
      libstdc++                                    2.95 M                                   1.31%
      libdw-0.176                                  2.32 M                                   1.03%
      liblzma                                      2.15 M                                   0.95%
      libpthread-2.17                              2.09 M                                   0.93%
      libresolv-2.17                               2.09 M                                   0.93%
      libelf-0.176                                 2.09 M                                   0.93%
      libgcc_s-4.8.5-20150702                      2.09 M                                   0.93%
      libz                                         2.09 M                                   0.93%
      libnss_myhostname                            2.08 M                                   0.93%
      libnss_files-2.17                            2.05 M                                   0.91%
      librt-2.17                                   2.03 M                                   0.90%
      libnss_dns-2.17                              2.03 M                                   0.90%
      libcap                                       2.02 M                                   0.90%
      libattr                                      2.02 M                                   0.90%
      libdl-2.17                                   2.02 M                                   0.90%
      ld-2.17                                    144.00 K                                   0.06%
------------------------------------------------------------------------------------------------------------------
   TOTAL                                         225.00 M                                 100.00%
------------------------------------------------------------------------------------------------------------------
```

Figura 44: Reporte de desempeño de memoria en simulación de compuerta NOR

11.5.3. Verificación en *Formality*

Se corrió el flujo mencionado en 6.7 para la verificación de los diseños pre o post síntesis para el diseño de una compuerta NOR obteniendo los resultados de la Figura 58. En el reporte se puede observar las pruebas de *Match* y *Verification* mostrando una igualdad completa entre ambos diseños.

```
********************************** Matching Results **********************************
 1 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 3 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
******************************************************************************************

Status:  Verifying...


******************************** Verification Results ********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/Nor
 Implementation design: i:/TECH_WORK/Nor_IO
 1 Passing compare points
------------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
------------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       1       0       0       1
Failing (not equivalent)       0       0       0       0       0       0       0       0
******************************************************************************************
```

Figura 45: Reporte en herramienta *Formality* para diseño de compuerta NOR

CAPÍTULO 12

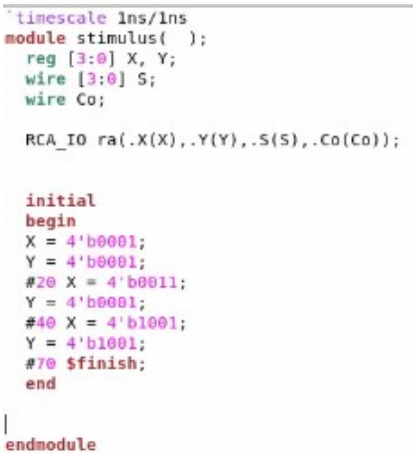
Flujos de simulación sobre circuito de una mayor complejidad

Con el fin de validar los flujos diseñados en este trabajo, se decidió correrlos sobre un circuito con una mayor complejidad de los evaluados en el capítulo 11. Para estas pruebas de utilizó un Ripple Carry Adder. El funcionamiento y decripción de este circuito se encuentra en 6.10. Los diseños de los circuitos evaluados se obtuvieron del paso previo a este trabajo bajo el flujo general de diseño al que ambos trabajos pertenecen.

12.1. Flujos sobre circuito Ripple Carry Adder

12.1.1. Flujo básico

Como se mencionó en 10 este flujo se utilizará principalmente para comparar el correcto funcionamiento del circuito sintetizado al compararlo con el diseño original codificado en-Verilog. Para esto se corrió el flujo en ambos diseños. Para todas las prubas realizadas bajo este flujo se utilizaran las librerías correspondientes, el diseño a simular y un testbench.El testbench utilizado para las simulaciones se muestra en la Figura 46. Tanto el testbench utilizado para el circuito sintetizado como el utilizado para la circuito original tendrán las mismas entradas para poder validar ambos circuitos al momento de compararlos.



```
`timescale 1ns/1ns
module stimulus(  );
  reg [3:0] X, Y;
  wire [3:0] S;
  wire Co;

  RCA_IO ra(.X(X),.Y(Y),.S(S),.Co(Co));


  initial
  begin
  X = 4'b0001;
  Y = 4'b0001;
  #20 X = 4'b0011;
  Y = 4'b0001;
  #40 X = 4'b1001;
  Y = 4'b1001;
  #70 $finish;
  end

endmodule
```

Figura 46: Modelo de testbench para la simulación de circuito Ripple carry adder en verilog

En las figuras 47 y 48 podemos ver el esquemáticos y las señales de entrada y salida respectivamente. De la simulación se puede apreciar que el diseño de este sumador funciona de forma correcta o esperada, las salidas del módulo son correctas en relación a las entras entradas impuestas. Podemos decir que el circuito fue diseñado correctamente. Por otra parte, en las figuras 49 y 50 podemos ver el esquemático y las señales de entrada y salida respectivamente. Al comparar ambas señales del modulo original y el sintetizado vemos que existen discrepancias en los resultados del circuito sintetizado. Efectivamente el módulo no esta teniendo el comportamiento esperado por lo que podemos decir que el proceso de síntesis sobre este diseño no se realizó de forma correcta. Debido a que el objetivo de este trabajo no es la corrección ni la elaboración de la síntesis lógica, el paso siguiente será escalar dicha falla al proceso previo a esta parte del flujo general para la solución de esta antes de continuar el proceso.

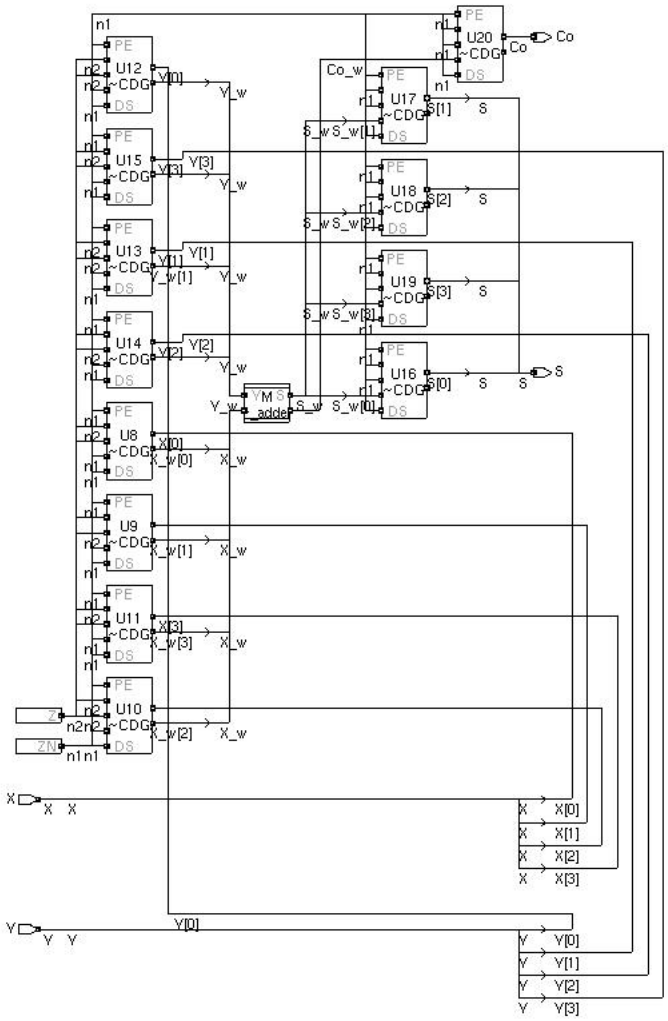


Figura 47: Esquemático de circuito ripple carry adder

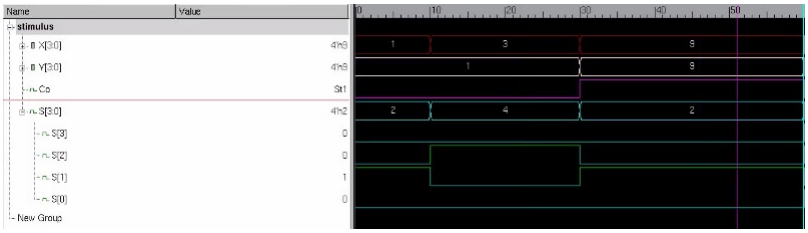


Figura 48: Señales de entrada y salida de circuito ripple carry adder

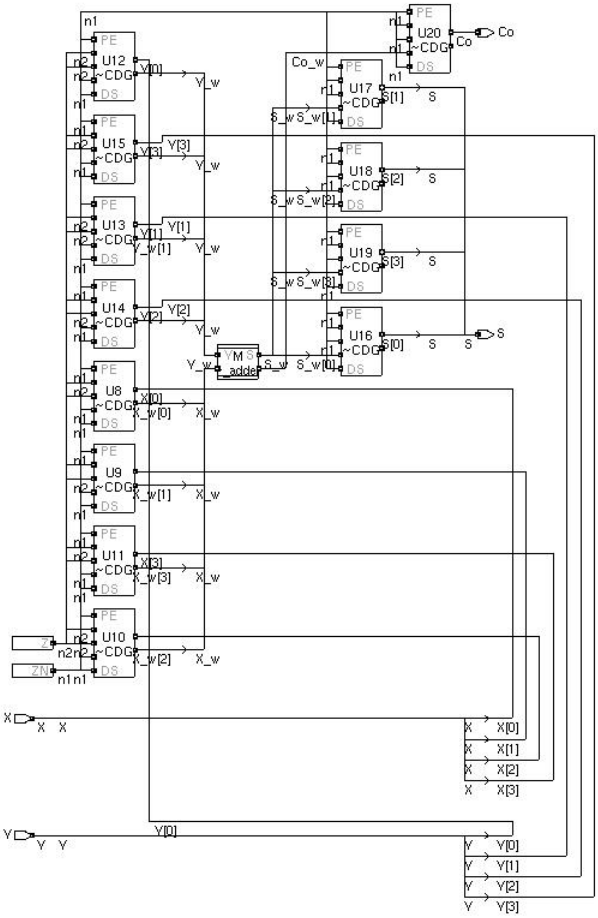


Figura 49: Esquemático de circuito ripple carry adder sintetizado

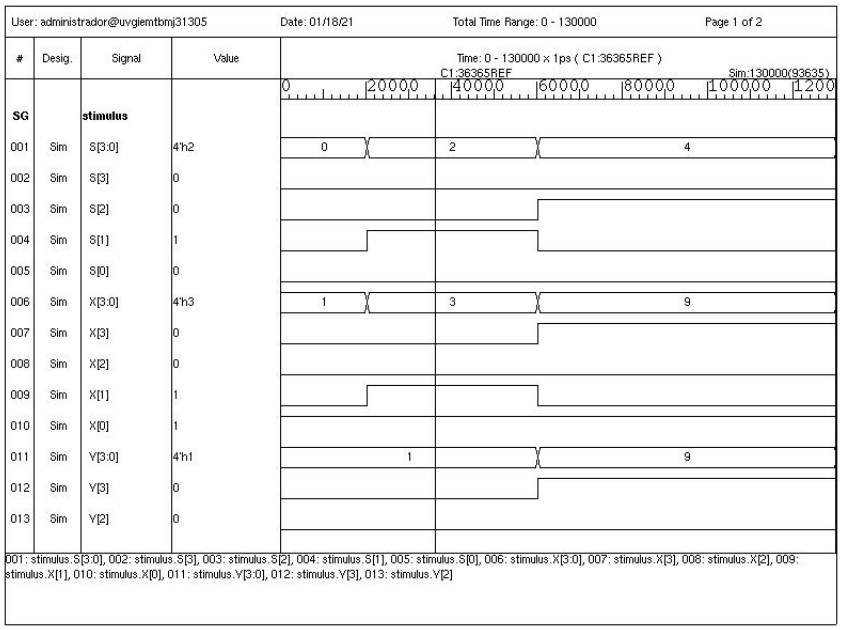


Figura 50: Señales de entrada y salida de circuito ripple carry adder sintetizado

12.1.2. Flujo de simulación con detección de condiciones de carrera

Debido a las fallas encontradas en el circuito sintetizado, las pruebas de condiciones de carreras se realizaron sobre el diseño original, esto con le objetivo de realizar cierta depuración sobre el circuito original. En los reportes de las pruebas realizadas no se encontraron condiciones de carrera. Los reportes se muestran en las figuras 51 y 52

```
[administrador@uvgiemtbmj31305 RaceConditions]$ cat hsRaceInfo.db
Clock Data Race: No Race detected.
[administrador@uvgiemtbmj31305 RaceConditions]$
```

Figura 51: Reporte de condiciones de carrera estáticas en circuito ripple carry adder

```
race.out
        Synopsys VCS RACE REPORT
No RACES Found
        END RACE REPORT
```

Figura 52: Reporte de condiciones de carrera dinámicas en circuito ripple carry adder

12.1.3. Evaluación de desempeño

Las pruebas de desempeño, de igual forma que en la sección 12.1.2 se realizaron sobre el circuito original. Los resultados de los reportes obtenidos utilizando el flujo descrito en 9.4 se muestan en las figuras 53 54

```
##################################################################################################################
     VCS build date:                                  Feb 28 2019 22:34:30
   Compiler version:                                  O-2018.09-SP2_Full64
    Runtime version:                                  O-2018.09-SP2_Full64
       Machine Name:                                  uvgiemtbmj31305
     Profile runner:                                  administrador
       Profile data:                                  /home/administrador/Escritorio/Chip/VCS/NOTJEFF/RCA/sim
                                                      profile_dir

      Creation date:                                  Fri Sep 18 00:24:45 2020
      Profile start:                                  0
       Profile stop:                                  130000
     Total CPU Time:                                  0.22
    Compile Command:                                  /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                                      8nv.v RCA_syn.v testBench_sio.v -o demo -simprofile -fu
                                                      ll64 -debug_all
 Simulation Command:                                  ./demo -simprofile time
##################################################################################################################
==================================================================================================================
                                                Time Summary View
==================================================================================================================
   Component                                                                                    Percentage
------------------------------------------------------------------------------------------------------------------
  KERNEL                                                                                          78.95%
  HSIM                                                                                            10.53%
  PLI/DPI/DirectC                                                                                 10.53%
------------------------------------------------------------------------------------------------------------------
  TOTAL                                                                                          100.00%
```

Figura 53: Reporte de desempeño de tiempo de CPU en simulación de ripple carry adder

```
##################################################################################################################
     VCS build date:                                  Feb 28 2019 22:34:30
   Compiler version:                                  O-2018.09-SP2_Full64
    Runtime version:                                  O-2018.09-SP2_Full64
       Machine Name:                                  uvgiemtbmj31305
     Profile runner:                                  administrador
       Profile data:                                  /home/administrador/Escritorio/Chip/VCS/NOTJEFF/RCA/sim
                                                      profile_dir

      Creation date:                                  Fri Sep 18 00:29:15 2020
      Profile start:                                  0
       Profile stop:                                  130000
     Total CPU Time:                                  0.40
    Compile Command:                                  /usr/synopsys/vcs-mx/bin/vcs -V -R tcb018gbwp7t.v tpd01
                                                      8nv.v RCA_syn.v testBench_sio.v -o demo -simprofile -fu
                                                      ll64 -debug_all
 Simulation Command:                                  ./demo -simprofile mem
##################################################################################################################
==================================================================================================================
                                              Peak Memory Summary View
==================================================================================================================
   Component                                                Memory                                  Percentage
------------------------------------------------------------------------------------------------------------------
  KERNEL                                                   6.90 M                                    3.07%
  PLI/DPI/DirectC                                          6.16 M                                    2.74%
  HSIM                                                     1.41 M                                    0.62%
  VERILOG                                                570.37 K                                    0.25%
    Module                                               570.37 K                                    0.25%
    Function Coverage Kernel                                  584                                    0.00%
  Anonymous                                               37.97 M                                   16.88%
  Library/Executable                                     172.00 M                                   76.44%
    VCS                                                  131.00 M                                   58.22%
    Third-party                                           41.00 M                                   18.22%
      libc-2.17                                            3.79 M                                    1.68%
      libm-2.17                                            3.01 M                                    1.34%
      libstdc++                                            2.95 M                                    1.31%
      libdw-0.176                                          2.32 M                                    1.03%
      liblzma                                              2.15 M                                    0.95%
      libpthread-2.17                                      2.09 M                                    0.93%
      libresolv-2.17                                       2.09 M                                    0.93%
      libelf-0.176                                         2.09 M                                    0.93%
      libgcc_s-4.8.5-20150702                              2.09 M                                    0.93%
      libz                                                 2.09 M                                    0.93%
      libnss_myhostname                                    2.08 M                                    0.93%
      libnss_files-2.17                                    2.05 M                                    0.91%
      librt-2.17                                           2.03 M                                    0.90%
      libnss_dns-2.17                                      2.03 M                                    0.90%
      libcap                                               2.02 M                                    0.90%
      libattr                                              2.02 M                                    0.90%
      libdl-2.17                                           2.02 M                                    0.90%
      ld-2.17                                            144.00 K                                    0.06%
------------------------------------------------------------------------------------------------------------------
  TOTAL                                                  225.00 M                                  100.00%
```

Figura 54: Reporte de desempeño de memoria en simulación de ripple carry adder

12.1.4. Verificación en *Formality*

Se corrió el flujo mencionado en 6.7 para la verificación de los diseños pre o post síntesis para el diseño de un *ripple carry adder* obteniendo los resultados de la Figura 55. En el reporte se puede observar las pruebas de *Match* y *Verification* mostrando una igualdad completa entre ambos diseños.

```
Formality (match)> verify
Reference design is 'r:/WORK/Full_Add'
Implementation design is 'i:/WORK/Full_Adder_Structural_Verilog'

********************************** Matching Results **********************************
 2 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 3 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
**************************************************************************************

Status:  Verifying...


******************************** Verification Results ********************************
Verification SUCCEEDED
---------------------
 Reference design: r:/WORK/Full_Add
 Implementation design: i:/WORK/Full_Adder_Structural_Verilog
 2 Passing compare points
--------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       2       0       0       2
Failing (not equivalent)       0       0       0       0       0       0       0       0
**************************************************************************************
```

Figura 55: Reporte en herramienta *Formality* para diseño de *full adder*

Como se mencionó en 6.7 las pruebas de esta herramienta fueron realizadas sobre el módulo original sintetizado (sin E/Ss). En el reporte de *Formality* podemos ver que ambos módulos coinciden perfectamente. De las pruebas corridas en 12.1.1 (con módulo I/Os) notamos inconsistencias entre ambos diseños. Con estos resultados podemos notar que el problema de la síntesis se encuentra en el módulo sintetizado con E/Ss.

12.2. Flujos sobre contador de cuatro bits

Entre las pruebas realizadas sobre circuitos de mayor complejidad se corrieron los flujos sobre un contador de 4bits. En este caso se encontró que el proceso de síntesis no se realizó de forma correcta.

12.2.1. Flujo básico

En las figuras 56 y 57 se muestran el esquemático y la respuesta del circuito respectivamente en la herramienta Verdi. En los resultados obtenidos, se notó que la señal de salida se mantiene en contención a lo largo del tiempo por lo que pudimos concluir que el proceso de síntesis no se realizó de forma correcta.

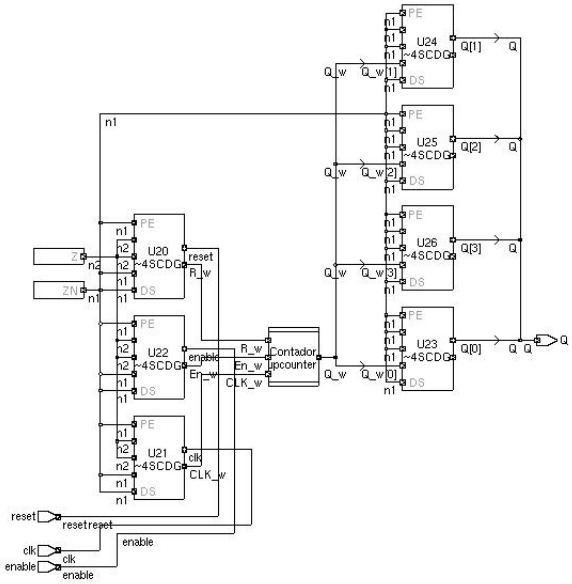


Figura 56: Esquemático de contador de 4 bits

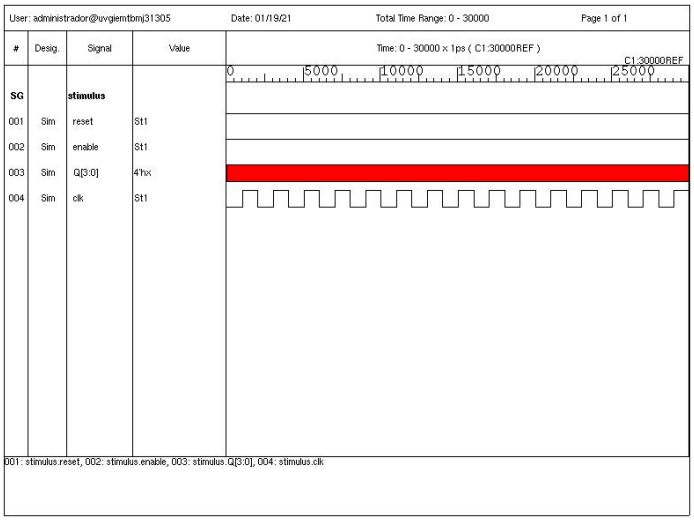


Figura 57: Señales de entrada y salida de contador de 4 bits

12.2.2. Verificación en *Formality*

Se corrió el flujo mencionado en 6.7 para la verificación de los diseños pre o post síntesis para el diseño del contador de 4 bits obteniendo los resultados de la Figura 58. En el reporte se puede observar las pruebas de *Match* y *Verification* mostrando diferencias en los 4 puntos de comparación evaluados por *Formality*.

```
********************************* Verification Results *********************************
Verification FAILED
------------------
 Reference design: r:/WORK/upcounter
 Implementation design: i:/TECH_WORK/CONT_IO
 4 Passing compare points
 4 Failing compare points (4 matched, 0 unmatched)
 0 Aborted compare points
 0 Unverified compare points
--------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       4       0       0       4
Failing (not equivalent)       0       0       0       0       0       4       0       4
**************************************************************************************
```

Figura 58: Reporte en herramienta *Formality* para diseño de contador de 4 bits

CAPÍTULO 13

Conclusiones

- Se comprendió el funcionamiento de la herramienta VCS al punto de poder diseñar un flujo que consta de tres partes para la evaluación del correcto funcionamiento de diseños Verilog y diseños Verilog sintetizados bajo las librerías de 180 nanómetros de TSMC.
- Se comprendió el funcionamiento de la herramienta Verdi para la simulación de diseños Verilog y diseños Verilog sintetizados bajo las librerías de 180 nanómetros de TSMC para la visualización de las señales de salida de ambos diseños.
- Se utilizó la herramienta de VCS en conjunto con Verdi para el diseño de un flujo que compara las señales de salida diseños Verilog y diseños Verilog sintetizado con el fin de validar su funcionamiento.
- Se diseñaron y validaron distintos flujos y funcionalidades de la herramienta VCS considerados por esta investigación como los mas eficientes y útiles para la depuración de diseños Verilog y diseños Verilog sintetizados bajo las librerías de 180 nanómetros de TSMC.

CAPÍTULO 14

Recomendaciones

Debido la limitación del alcance que esta investigación tuvo debido a la pandemia a causa del COVID-19, se recomienda en futuros trabajos ligados a la presente investigación a seguir indagando en las funcionalidades que la herramienta VCS en conjunto con Verdi poseen ya que en la presente investigación no se cubren todos los aspectos de estas. Por otra parte, se recomienda a futuros grupos de trabajo a continuar la linea de investigación general a la que este trabajo pertenece para poder culminar los objetivos principales que fueron impuestos en el año 2019, en le que inicia esta investigación.

Debido la complejidad que esta investigación presenta, siempre existen puntos de mejora y optimización en la misma. Por esta razón se recomienda a ir más allá de lo impuesto en este trabajo y en el proyecto general.

Los flujos propuestos en este trabajo fueron diseñados con manuales de los años 2019-2020 de las herramientas utilizadas. Se recomienda tomar en consideración las versiones de las herramientas en relación con el año del lanzamiento de los manuales ya que existen comandos que están próximos a descontinuares o adaptarse a otras funcionalidades.

Se recomienda en futuras investigaciones realizar, en conjunto con las pruebas mostradas en esta investigación, pruebas físicas sobre los diseños a fabricar con el uso de FPGAs y software de Altium.

CAPÍTULO 15

Bibliografía

[1] N. W. D. Harris, "CMOS VLSI design: a circuits and systems perspective", pág. 53, 2015.

[2] P. Horowitz y W. Hill, *The art of electronics*. Cambridge: Cambridge University, 2015.

[3] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*. Prentice Hall PTR, 2003.

[4] *Verilog Hardware Description Language. IEEE 1364*. Institute of Electrical Electronics Engineers, 2004.

[5] Synopsys, "VCS User Guide", Solvnet, 2019.

[6] ——, "Verdi User Guide and Tutorial", pág. 469, 2019.

[7] ——, "Formality User Guide", Solvnet, 2020.

[8] D. A. Wheeler, "Secure Programming HOWTO", pág. 200, 2015.

[9] D. Engler y K. Ashcraft, "Racer: Effective, Static Detection of Race Conditions and Deadlocks", 2017.

[10] D. Harris y S. Harris, *Digital design and computer architecture*. Morgan Kaufmann, 2015.

[11] A. Agarwal y J. Lang, *Foundations of analog and digital electronic circuits*. Elsevier, 2005.

[12] L. Najera, "Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado.", pág. 72, 2019.