UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de un circuito integrado con tecnología de 180nm usando la librería de diseño de TSMC. Verificación avanzada de la síntesis lógica y simulación avanzada del archivo final que incluye resistencias y capacitancias parásitas

Trabajo de graduación presentado por Gerardo Enrique Cardoza Marroquín para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2021

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de un circuito integrado con tecnología de 180nm usando la librería de diseño de TSMC. Verificación avanzada de la síntesis lógica y simulación avanzada del archivo final que incluye resistencias y capacitancias parásitas

Trabajo de graduación presentado por Gerardo Enrique Cardoza Marroquín para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2021

Vo.Bo.:

(f) _____

Ing. Carlos Esquit

Tribunal Examinador:

(f) _____

Ing. Carlos Esquit

(f) _____

MSc. Carlos Esquit

(f) _____

Ing. Jonathan de los Santos

Fecha de aprobación: Guatemala, 5 de diciembre de 2021.

Prefacio

El presente proyecto se realizó durante la pandemia por el COVID-19 en el 2021, en donde se llegaron a tener algunos contratiempos por la conexión al servidor, en este era en donde se realizaban las pruebas necesarias para poder justificar el comportamiento del circuito. Al haber ciertos contratiempos la investigación se estuvo realizando de forma efectiva, gracias a los trabajos previos que se realizaron en el 2020 y en el 2019. En estos trabajos se impulsó mucho la investigación de los recursos que brindaba Synopsys y como incorporarlo al flujo de diseño para la realización del chip.

Aunque gran parte de la investigación fue mi responsabilidad hubo personas que me ayudaron a concretar ciertos objetivos y con apoyo a distintos problemas que podían llegar a surgir con la conexión a mi computadora desde mi casa. Me gustaría agradecer al Ingeniero Jonathan de los Santos por su orientación a la investigación y correcciones a este trabajo. También me gustaría agradecer a Steven Rubio que, a pesar de sus propios intereses, estuvo revisando mis avances y apoyándome en descifrar ciertos problemas que iban surgiendo. Por último, me gustaría agradecer a MSc. Carlos Esquit por proponer las bases de la investigación en los cursos programados en la carrera de ingeniería electrónica y por el apoyo como mi asesor.

Índice

Lista de figuras

Lista de cuadros

Resumen

Este proyecto consta de la simulación y verificación de archivos obtenidos de la etapa de diseño lógico y la extracción de parásitos de los componentes de un circuito. La fase de simulación del diseño lógico se enfoca en la depuración y optimización. Esto se realiza con una tecnología de 180nm, la misma se llevará a cabo usando las librerías de TSMC, con el fin de realizar un chip a escala nanométrica. Esta parte busca concentrar su atención en optimizar el circuito descrito en Verilog, en simular archivos HDL esquemáticos, que pueden llegar a generar las herramientas de Synopsys, y para corroborar la síntesis lógica se utilizará Verdi y formality.

La parte de la simulación sobre la extracción parásitos se realizará en HSPICE, esto permite simular el deck que contiene informacion generada por StartRC, el cual nos ayuda a caracterizar el circuito , para luego proceder a la fabricación el chip. Con este proceso se pretende comprender lo que HSPICE ofrece, las librerías y los comandos que se utilizan en este proyecto, para que las personas en un futuro puedan utilizar el flujo de diseño y realizar sus propios proyectos desde un lenguaje descriptor de hardware.

Resumiendo, el trabajo muestra las simulaciones y verificaciones para garantizar que la síntesis lógica se realice de la forma correcta y poder obtener resultados congruentes con lo planteado en Verilog. Además, se busca poder corroborar por medio de una simulación en HSPICE que el circuito funciona y poder conocer las características técnicas que describen al circuito como: Potencia, Frecuencia, Análisis de tiempos, etc..

Abstract

This project consists of the simulation and verification of files obtained from the logical design stage and the extraction of parasites from the components of a circuit. The simulation phase of the logic design focuses on debugging and optimization. This is done with a 180nm technology, it will be carried out using the TSMC libraries, in order to make a chip at the nano-scale. This part seeks to focus its attention on optimizing the circuit described in Verilog, in simulated schematic HDL files, which can be generated by Synopsys tools, and Verdi and formality will be used to corroborate the logical synthesis.

The part of the simulation about the extraction of parasites will be executed in HSPICE, This allows us to simulate the deck that contains information generated by StartRC, which helps us to characterize the circuit, and then proceed to the manufacture of the chip. The aim of this process is to understand what HSPICE offers, the libraries and the commands used in this project, so that in the future people can use the design flow and carry out their own projects from a hardware descriptor language.

In summary, the work shows the simulations and verifications to guarantee that the logical synthesis is carried out in the correct way and to be able to obtain results consistent with what is proposed in Verilog. In addition, it seeks to be able to corroborate through a simulation in HSPICE that the circuit works and to be able to know the technical characteristics that describe the circuit such as: Power, Frequency, Time analysis, etc.

CAPÍTULO 1

Introducción

La nanoelectrónica es un tema que actualmente en Guatemala no se maneja, ya que se carece de conocimientos y equipo para generar chips a nano escala. Al diseñar un chip es proceso muy complejo sin las debidas herramientas, ya que cada parte del diseño debe cumplir con las reglas preestablecidas por el fabricante, además se debe tener en cuenta que funcionamiento del diseño para que la fabricación salga exitosa.

Como todo diseño requiere de pruebas y simulaciones, en este caso en las fases iniciales del diseño se creo un archivo con lenguaje descriptor de hardware y el archivo tuvo que pasar por una síntesis lógica, esta síntesis paso por varias pruebas, para poder transmitirla a otras fases de diseño (La síntesis física). Con estas pruebas se puede descartar cualquier problema en la síntesis lógica, haciendo un proceso más cómodo para depurar procesos. Estas simulaciones se llevaron a cabo por las librerías de TSMC.

La investigación presente muestra la forma de utilizar Verdi desde la interfaz gráfica y ver la solución avanzada para la depuración de los diseños, que logran aumentar la productividad del diseño. Lo que nos enseña Verdi una forma más simple de comprender el comportamiento del diseño, optimizar procesos tediosos de depuración y resolver errores. En formality se realizó proceso simple que detecta las diferencias inesperadas que se puede haber introducido en un diseño durante el desarrollo. En este proceso se logra ver las respuestas que nos da formality en cada uno de los flujos con diferentes circuitos y el uso de la herramienta por medio de la interfaz gráfica y por comandos, para la detección de errores. Por último, en este documento se muestra la fase final del flujo, la cual es la simulación de parásitos por medio de la herramienta de HSpice y WaveView, la cual me va a indicar la caracterización del circuito y el funcionamiento de este con parásitos. El fin de esto es poder visualizar si el flujo realizado fue exitoso, ya que nos mostrara el comportamiento que tiene el circuito con los parásitos obtenidos.

CAPÍTULO 2

Antecedentes

El siguiente proyecto se llevó a cabo gracias al Ing.Carlos Esquit, quien en 2009 fue nombrado director de carrera del departamento de Ingeniería Electrónica y Mecatrónica en la Universidad Del Valle de Guatemala. En 2013 Impartió el curso VLSI posteriormente nanoelectrónica 1 y gracias a este se aprendieron los conceptos básicos para comenzar a realizar este proyecto.

El diseño ha ido evolucionando conforme a los cursos de nanoelectrónica 1 y 2, que dieron lugar en la reforma curricular realizada en el 2015, en la Universidad del Valle por Carlos Esquit. Con esto se pudo ir evaluando conceptos claves que nos servirían en un futuro, para completar el flujo de diseño.

En el 2019 un grupo de estudiantes de la Universidad inició un proyecto de un chip a escala nanométrica, y posteriormente otro grupo de estudiantes en el 2020 realizaron mejoras sobre el mismo, conservando siempre la nano escala.

Los pioneros anteriormente mencionados son los Luis Nájera y Steven Rubio, quienes iniciaron con una estructura del flujo de diseño, La cual consta de proceso que se divide en 2 categorías: En diseño lógico y el diseño físico. Este esquema planteado, tenía el fin elaborar ambas partes, dejando los instrumentos necesarios para la ejecución del flujo utilizando siempre herramientas de Synopsys, ya que sobre esta se plantearon las guías de instalación. Aunque al final el uso de las herramientas resultó ambiguo.

Durante los años 2020 y 2021 se continuo con el proceso de investigación sobre cómo realizar el flujo de diseño para realizar un chip a nano escala. Luego fue gracias a Synopsys el poder instalar el software Splashtop, el cual hizo posible el ingreso a las computadoras de la Universidad y de manera remota poder completar el diseño.

En el año 2020 varios alumnos fueron encargados de las distintas etapas de diseño, pues esto tenía como fin dejar una plantilla casi terminada, para poder correr cualquier programa y que el flujo de diseño pasara sin ningún inconveniente, lastimosamente por el COVID 19 (2020) se completó solamente un circuito pequeño, y no fue comprobado en su totalidad el

diseño físico. Incluso no se tenía el conocimiento completo de cada parte del flujo del diseño. Sin embargo, se llegó a tener una gran parte del proyecto completada, y fue posible definir que herramientas de software son las necesarias para realizar cualquier proceso del flujo de diseño.

[1] [2] [3]

CAPÍTULO 3

Justificación

La fabricación de un chip a nano escala representa un gran avance para la Universidad del Valle de Guatemala y para el país mismo, ya que esta será la primera vez que una Universidad de Guatemala pueda ser capaz de realizar el diseño de un chip con una tecnología de 180nm. Con el flujo de diseño se pretende diseñar y posteriormente enviar a fabricar chips que sean diseñados por medio de un programa de descriptor de hardware, además que puedan ser utilizados en proyectos de mayor escala. Esto no solo representa una revolución en la universidad, sino que incluso representa un incentivo para el desarrollo en Guatemala sobre la investigación en nanoelectrónica.

Este trabajo, es un parte importante dentro el flujo de diseño, ya que es base para partes posteriores. Esto porque plantea que se sintetizará por medio de las herramientas de Synopsys y con la simulación y la comparación de los diseños digitales que serán: el original y el sintetizado. Estos jugarán un papel importante pues dentro del esquema ayudan a resolver la complejidad que pueda representar el circuito en las otras fases. Teniendo en cuenta esto se espera que, con el esquema claro a la hora de sintetizar un circuito este logre comportar de la manera deseada y de ser así este pueda ayudar a la parte del diseño lógico para ver si se encuentra un error en el circuito.

En la fase de diseño enfocada a la simulación en el 'deck' generado de la extracción de parásitos, es una parte que depende de todas las fases anteriores, ya que esta enseñará las simulaciones, en donde podremos ver las respuestas más reales gracias a las herramientas de Synopsys del circuito que se pretende realizar, por lo tanto, esta etapa será la que indique el correcto funcionamiento del chip antes de ser enviado a fabricar.

CAPÍTULO 4

Objetivos

Objetivos Generales

- Realizar las verificaciones necesarias de la síntesis lógica.
- Realizar las simulaciones del deck generado por la extracción de parásitos. Con el fin de garantizar los resultados esperados y la caracterización del circuito.

Objetivos Específicos

- Establecer un proceso para validar los archivos generados en la síntesis lógica.
- Realizar una extracción de los parásitos, con el fin de analizar como va a afectar al circuito.
- Simular el circuito tomando en cuenta los parásitos y mostrar la caracterización final del chip, utilizando las herramientas de Synopsys.
- Usar el deck generado en el proceso de la extracción de parásitos para luego simularlo con HSpice y corroborar el correcto funcionamiento del circuito.
- Brindar archivos y la información necesaria, con el fin de automatizar el proceso de verificación de síntesis lógica y la simulación del deck generado en el proceso de la extracción de parásitos.
- Tener una comunicación efectiva con los demás grupos de trabajo para la solución rápida de errores.

CAPÍTULO 5

Alcance

En la presente investigación se pretende mostrar el proceso requerido para la realización de distintos flujos utilizando la herramienta VCS, Verdi y Formality. En otro caso, se pretende mostrar el final del flujo de diseño comprobando que realmente funciona el circuito propuesto utilizando la tecnología VLSI CMOS para la realización del chip. Con la utilización correcta de estos flujos se podrán evitar errores en fases posteriores y con la certeza de que el flujo se está realizando de la forma correcta. Además de mostrar una documentación exhaustiva para tener un flujo adecuado, con el fin de mandar a fabricar el chip y tener un flujo para proyectos futuros.

CAPÍTULO 6

Marco teórico

Diseño VLSI

En este proceso se detalla el diseño y la fabricación de los CMOS, con el fin de poder fabricar un chip. Este proceso requiere de los transistores de canal n (nMOS) y canal p (pMOS). Estos sistemas basados en VLSI contienen grandes ventajas en el mundo de la electrónica, ya que permiten hacer un chip con muchas funcionalidades a un tamaño muy pequeño, La velocidad también se considera un factor importante, ya que las capacitancias parásitas se logran reducir a un nivel considerable y por último la potencia se logra reducir, y se puede ver respecto al costo un beneficio en la alimentación más pequeña, enfriamiento, etc. Desde que William Schockley, John Bardeen y Walter Brattain logran inventar el transistor de puntas de contacto, comenzaron a revolucionar este mundo de los transistores y al proponer el transistor bipolar en 1948, se convierte en una de las bases que se utiliza en la actualidad, para realizar el proceso de un chip a nano escala. Luego de los años se descubrió la estructura de los MOSFET'S que llegaron a reemplazar a los JFET'S, esto fue gracias a Ian Munro Ross. La idea de los MOSFET'S era más antigua, pero él logró que se hiciera viable los efectos de campo. El VLSI se conoce como *Very Large Scale Integration*, en donde el número de componentes es de 10,000 a 100,000 y el número de compuertas es de 1000 a 10,000. El ingeniero Gordon Moore, logró observar en 1965 que la cantidad de transistores iba aumentando según los años que pasaban por un factor de 2, es decir que por cada año la cantidad de transistores en un microprocesador aumentaba cada año. En el 2007, el mismo Moore produjo que su ley dejara de existir dentro de 10 o 15 años, dependiendo de lo que se realice en estos años. [1] [2]

Flujo de diseño

El flujo de diseño nos muestra el esquema requerido para poder realizar una serie de pasos para el diseño y la fabricación un circuito integrado. Este flujo de diseño se logra dividir en 2 partes (Front End y Back End). Lo que se busca con este flujo es encontrar la

manera de fabricar un chip no tenga errores y que sea funcional respecto a las necesidades requeridas.

Front End Este se encarga de resolver un problema pasándolo a un lenguaje descriptivo, en este caso se usa los lenguajes de Verilog y VHDL. En esta etapa se encuentran los pasos necesarios para la arquitectura de diseño. A continuación, se logran ver los pasos necesarios para que se pueda elaborar el diseño front end.

Primero: Se debe realizar el circuito, por medio del lenguaje descriptivo de hardware como Verilog, el cual presenta la solución del problema establecido.

Segundo: Se denomina la síntesis lógica, este es un proceso en donde se toma el diseño RTL, que fue descrito en Verilog o VHDL. En esta también se producen comprobaciones y simulaciones, con el fin de comprobar el modelo RTL propuesto.

Tercero: El netlist, este es un circuito generado, después de la síntesis lógica en donde se logra observar por medio de librerías.

Cuarto: Se utilizan varios softwares con el fin de verificar la funcionalidad de la síntesis del circuito.

[4]

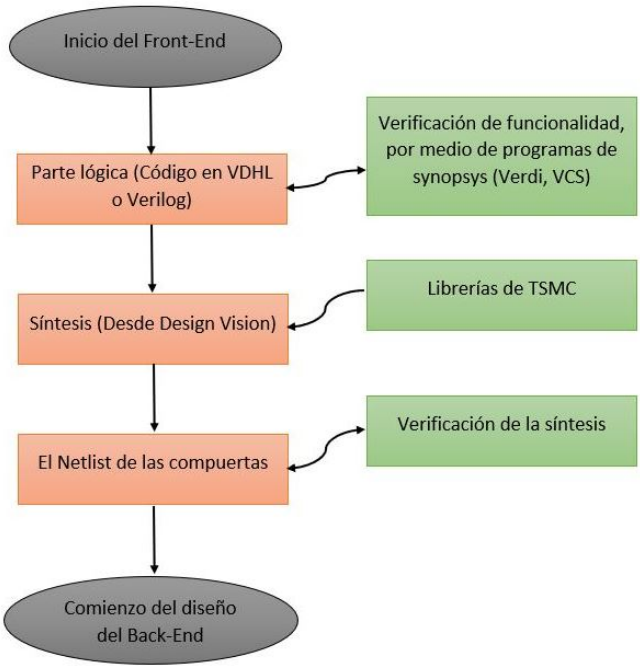

Figura 1: Diagrama del Front-End

Back End Esta parte va más inclinada a la implementación del circuito en físico. Esta toma la parte del lenguaje descriptivo, en forma de un diseño en físico, el cual se convierte

en un layout, en donde podemos ver a más detalle como está compuesto el circuito.

Primero: En este se encuentra la síntesis física, el cual se basa en el netlist. Al pasar a esta etapa se encuentra el layout, en donde tenemos más información acerca del circuito y además podemos manipularlo, con el fin que cumpla las reglas de diseño especificadas por la empresa que va a realizar el chip.

Segundo: Obtenemos lo que es el Placement, en este caso nosotros decidimos donde se van a poner las celdas con el fin de que llegue a ser lo más óptimo posible y sencillo de arreglar en un futuro.

Tercero: Al tener las celdas puestas en la mejor posición posible se procede a interconectar las salidas y las entradas de cada parte del circuito descrito, este parte se conoce como routing.

Cuarto: Este se denomina *Design Rule Check*, este consiste en verificar que el layout no contenga errores o reglas de diseño que son impuestas dependiendo de la tecnología.

Quinto: Este se denomina *layout versus schematic*, este consiste en verificar la integridad del diseño. Este compara el layout con el netlist (Síntesis lógica).

Sexto: La extracción de parásitos, aquí es donde se simula la parte del layout con la cantidad de parásitos que pueda tener el layout, y se logra ver que tanto llega a afectar al final del proceso.

[4]

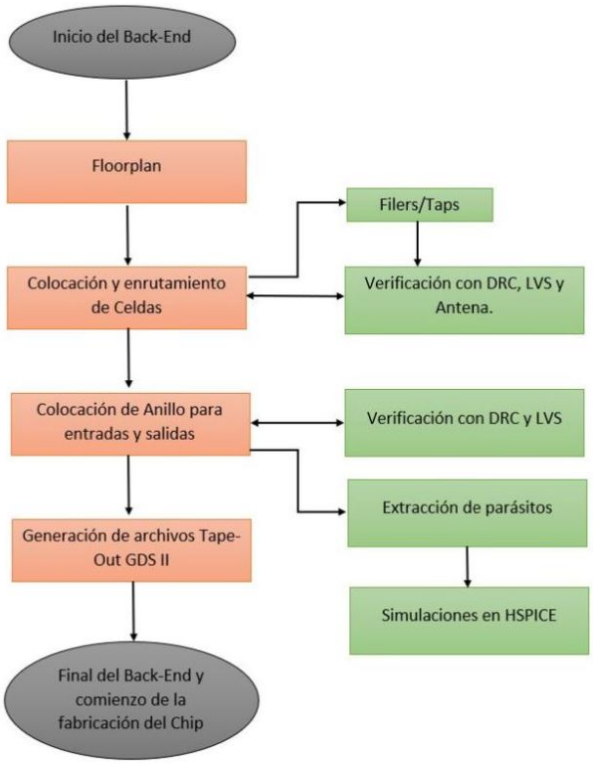

Figura 2: Diagrama del Back-End

Lenguaje Descriptor de Hardware

Estos lenguajes son utilizados con el fin de describir una arquitectura y lograr visualizar el comportamiento de un sistema electrónico. La forma más fácil de representar un circuito es mediante la utilización de diagramas o esquemas estos hacen una evidencia gráfica de lo que se pretende realizar. Ya que con el paso del tiempo comenzaron a salir circuitos más complejos que no pueden ser representados de una manera visual, nació la idea de integrar las herramientas de descripción de síntesis, simulación y realización. Al principio se logró utilizar un lenguaje de descripción que solo permitía secuencias simples, a este se le nombro Netlist. Luego de estos programas simples que ya eran reconocidos como lenguajes de descripción de hardware, lograron ver el gran impacto que podían tener a la hora de describir circuitos complejos.

Mientras que todo iba evolucionando, la posibilidad de desarrollar circuitos digitales mediante dispositivos programables se hacía más viable, ya que los circuitos, ya representaban una cantidad considerable de componentes. Este tipo de lenguajes estaban orientados para la realización de simulaciones, por lo que no importaba mucho que el nivel de abstracción fuera tan alto. Al momento de la aparición de técnicas para la síntesis de circuitos a partir de lenguajes que se consideraban de alto nivel de abstracción, se comenzaron a utilizar también para la síntesis de circuitos.

[5]

Verilog y VHDL

Verilog es un lenguaje que se utiliza para describir hardware, en el cual podemos diseñar y simular circuitos electrónicos (Digitales). Este lenguaje es una derivación de la programación en C, con el fin que los ingenieros puedan entenderlo. VHDL también se encuentra englobado dentro de los lenguajes de descripción de Hardware. En sí, un HDL es un tipo de lenguaje especializado que define la estructura, diseño, operación de circuitos electrónicos y circuitos digitales. Este tipo de lenguaje es una forma de representación formal de un circuito electrónico, con la posibilidad de hacer un proceso más simple.

EL lenguaje VHDL se considera un estándar, este no depende de ningún fabricante o dispositivo, es independiente. El fin de este lenguaje es que se puedan reutilizar los diseños y tiene la ventaja de ser un diseño jerárquico. Verilog este puede llegar a soportar lo que son circuitos analógicos, pruebas y señales mixtas a distintos niveles de abstracción.

[5]

VCS

Es una de las soluciones de la verificación funcional que se utiliza en la mayoría de las empresas de semiconductores en el mundo. Este es un programa que proporciona características innovadoras para lograr un mayor rendimiento y permite los flujos de verificación de desplazamiento al principio del ciclo de diseño. Este ofrece como solución con Native Testbench, compatibilidad con Verilog, análisis y cobertura e integración con Verdi. VCS satisface las necesidades de los diseñadores para poder verificar los resultados obtenidos de la prueba de un circuito elaborado en un programa compatible de descripción de hardware.

Esta herramienta cuenta con un flujo de dos pasos para evaluar el funcionamiento del circuito descrito.

[6]

Flujos de VCS

Este es un flujo que se utiliza para poder comprobar si el diseño en Verilog HDL y SystemVerilog, son funcionales para la fabricación del Chip.

Primero: El primer paso es lograr compilar el diseño, en esta etapa VCS logra construir una instancia de jerarquía y logra generar un archivo .simv que se utiliza para poder realizar la simulación.

Segundo:En esta etapa se simula el diseño con el archivo generado en la primera parte (.simv). En esta etapa se logra ver si se logró correr la simulación.

Existe otro flujo de dos pasos, este permite diseños en Verilog, VHDL y mixedHDL. Este consta de 3 pasos.

Primero: Analizar el diseño, en esta etapa VCS logra dar los ejecutables, los cuales son vhdlan y clogan, con el fin de analizar el diseño y lograr almacenar archivos en la carpeta

de trabajo.

Segundo: Elaborar el diseño, en esta parte se adquiere un archivo .vcs, el cual se compila y logra elaborar el diseño utilizando los archivos generados en la librería de trabajo. Luego da un archivo .simv.

Tercero: Simular el diseño, este se logra simular con el archivo, el cual es un ejecutable binario que se describió en el paso anterior.

[7]

Verdi

Esta herramienta de Synopsys se considera un sistema de depuración automatizado y permite la depuración integral de todos los flujos de diseño y verificación. Este sistema tiene una tecnología poderosa que le ayuda a comprender el comportamiento del diseño definido, y lo más relevante es que ayuda a optimizar los procesos difíciles y tediosos. En esta herramienta pretende reducir el tiempo de depuración en más del 50 por ciento, esto es posible porque automatiza el comportamiento, por medio de un seguimiento con una tecnología única de análisis. También logra extraer, aislar y mostrar la lógica pertinente en los diseños. Logra revelar el funcionamiento y la interacción con el diseño propuesto. El sistema de depuración por completo que tiene Verdi, utiliza la tecnología y las capacidades de un sistema de depuración, cabe resaltar que este utiliza funciones avanzadas de depuración con soporte para una gran cantidad de lenguajes y metodologías.[8]

Funciones principales

- Seguimiento rápido de la actividad de muchos ciclos de reloj con una potente tecnología de análisis.
- Vistas de flujo temporal que proporciona una visualización de tiempo y estructura para comprender las relaciones de causa y efecto.
- Analizar diseños en niveles más altos de abstracción.
- Depuración basada en aserciones con soporte integrado.

Depuración de SystemVerilog Testbench

- Soporte de código fuente para SystemVerilog Testbench, incluyendo la metodología de verificación universal (UVM).
- Vistas especializadas que ayudan a comprender el código, viendo la navegación y exploración de jerarquías basadas en declaraciones.
- Capacidad de registro de transacciones automatizadas, que brindan una imagen completa de la actividad del banco de pruebas en el entorno de verificación después de la simulación.

- Control de simulación interactivo que permite correr el código completo.
- Las vistas de depuración compatibles con UVM permite explorar los resultados de aspectos específicos.
- Vistas de depuración a nivel de transacción y logran admitir el registro de datos.

[7]

Mas acerca de Verdi

Este sistema de depuración automatizado tiene compiladores e interfaces. Los compiladores que son utilizados en la parte de diseño mayoritariamente son Verilog, VHDL Y System Verilog. La interfaz que tiene Verdi implementa los estándares industriales de datos VCD y SDF. Normalmente los resultados por la herramienta son guardados en la Fast Signal Database. Verdi también permite la interoperabilidad con lo que se conocen los simuladores lógicos, las herramientas de verificación y los análisis de tiempo. Las bases de datos que Verdi maneja son las de conocimiento que se pueden llamar KDB y la base de datos de señal rápida conocida como FSDB. En KDB normalmente se utiliza para almacenar información lógica y funcional para el diseño implementado. La base de datos FSDB, logra almacenar los resultados de la simulación. Al utilizar la información que proporcionan estas bases de datos, Verdi tiene unas herramientas de análisis que resultan ser muy útiles dependiendo de la aplicación que se quiera dar, estas son: Análisis de estructura, comportamiento, evaluación y Mensajes.

Formality

Esta herramienta utiliza la comparación y la verificación para la equivalencia de dos circuitos dados, en este caso sería un archivo. v con el archivo de Verilog sintetizado y con estos archivos se muestra la diferencia que pueda a ver en el circuito y da seguimiento a un análisis detallado.

Esta herramienta se usa más como una alternativa para la simulación de la síntesis lógica, en si la simulación logra aplicar una gran cantidad de vectores de salida que resultan de los valores esperados y cuando los circuitos se llegan a volver más complejos es mayor la cantidad de vectores que se utilizan y además logra evitar cuellos de botella en el flujo de diseño.

Los cuellos de botella se dan por: Gran cantidad de vectores, Los simuladores deben de procesar más eventos por cada vector, lo que indica mayor tamaño y complejidad del diseño y por último mientras existan más vectores van a provocar un mayor intercambio de memoria, y esto va a ralentizar el rendimiento.

Este utiliza para realizar la verificación formal un diseño de referencia, este diseño no requiere vectores de entrada. Esta verificación solo requiere de funciones lógicas durante la comparación y es independiente de las propiedades físicas del diseño. La mayor fortaleza de esta herramienta es la capacidad de mostrar diferencias inesperadas sin depender de conjuntos de vectores y verificando diseños grandes con el fin de mostrar una cobertura del diseño del 100 % y realizando una simulación más eficaz.[9]

Esta herramienta consta de 2 funciones básicas verificar la equivalencia y verificación del modelo.

- La verificación de equivalencias prueba si una representación del diseño es lógicamente equivalente a otro, es decir, que los 2 circuitos muestran el mismo comportamiento en cualquier condición a pesar de las diferentes representaciones, el circuito sintetiza y el circuito descrito en hardware.
- La verificación del modelo prueba si un diseño se adhiere a un conjunto especifico de propiedades lógicas.

[9] Para la verificación individual se puede ver representada por 2 diseños el de referencia y el implementado, por medio de "matches"va emparejándolos según corresponda y ya realiza la verificación formal. Para una representación visual, se muestra a continuación un flujo de verificación de formality:

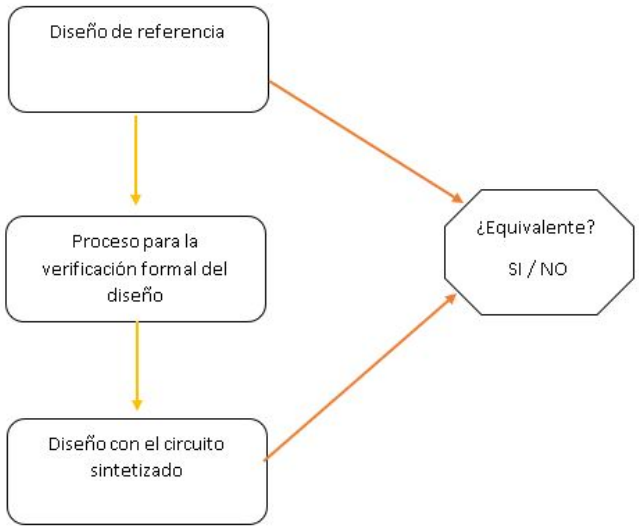

Figura 3: Diagrama del flujo de verificación utilizando formality

[9] Formality contiene conceptos interesantes que nos ayuda a entender el proceso de la herramienta de una forma más detallada y va muy de la mano a la interacción que tenemos con la interfaz gráfica y se enfoca en estas definiciones para mostrar los pasos necesarios para realizar buenas comparaciones entre ambos circuitos.

Guidance: Como primer concepto se encuentra el de guía, que es el primer paso a realizar en el flujo, este consiste en entender el análisis de equivalencias y ver los procesos que cambian en los diseños causados por otra herramienta de procesos anteriores.

Black Boxes: Una caja negra es considerada una instancia del diseño que no se conoce realmente. Este aparece en componentes no sintetizados como: RAMs, ROMs y analog circuits.

Limitaciones: Al usar restricciones ayuda a limitar el número de combinaciones que

se pueden dar en los valores de entrada, también logra reducir el tiempo de verificación y elimina posibles fallas sobre falsas verificaciones que son consideradas no utilizadas de valores de entrada.

Matching: En este caso, formality busca los puntos de comparación, en donde cada uno de esos puntos debe tener una correspondencia uno a uno entro los objetos de diseño en la referencia. Esto se realiza antes de la verificación del diseño y tiene que estar coincidiendo cada salida, elemento secuencial, pin de entrada de una "black box". ´Cada punto de comparación es un objeto del diseño que se utiliza como un punto final de la lógica combinacional durante la verificación. Formality lo que verifica es un punto de comparación que llega a comparar el cono lógico de un punto en el diseño de implementación contra otro cono lógico en el diseño de referencia.

Verificación: Esta se considera la función principal de equivalencias, en donde la herramienta muestra si realmente es consistente con el diseño o la librería a implementada.[9]

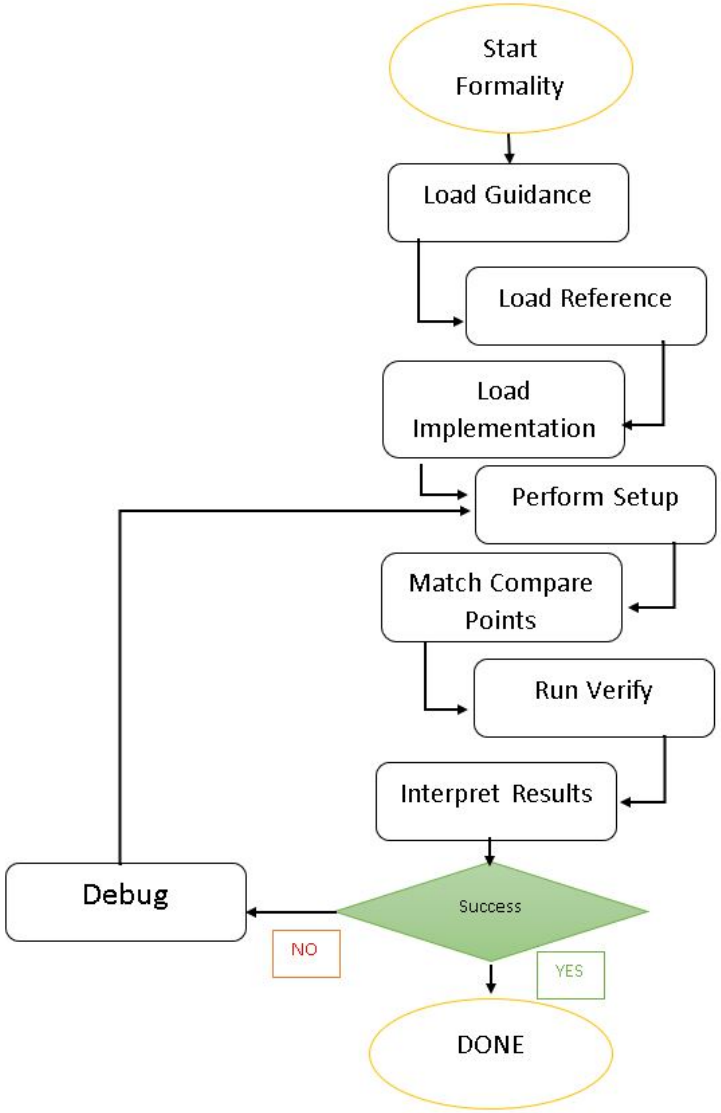

Figura 4: Pasos para la verificación del diseño

Extracción de parásitos

La extracción de parásitos se puede realizar después de haber hecho las pruebas pertinentes en el flujo del diseño, este pertenece a la parte del Back-End, después con HSPICE se busca el archivo generado con parásitos para ver cómo se comporta la simulación, en donde también se utiliza el StarRC. Todo esto es con el fin de garantizar que el diseño VLSI es factible y que las capacitancias parásitas no van a afectar el circuito.

Esto sucede al jugar con los transistores que generan lo que se llaman capacitancias parásitas que generan un efecto adicional en los conductores que funcionan como placas entre el dieléctrico, y este tipo de parásitos va a ir aumentando conforme las frecuencias, más que todo en este caso al meter muchos transistores las capacitancias parásitas van a ir aumentando. En la implementación de circuitos nanométricas las interconexiones entre cables pueden llegar a representar problemas por los parásitos que se pueden generar, al igual que puede pasar con las difusiones de los componentes que pueda proveer las librerías de TSMC.

El fin de la extracción de parásitos es la representación de los efectos electromagnéticos que los cables y las difusiones puedan generar al momento de correr la simulación, en componentes como la capacitancia, resistencia e inductancia. [4]

StarRC

Esta herramienta se considera la solución para la extracción de parásitos. Esta herramienta amplia los beneficios de rendimiento, en donde se basa en mejorar la arquitectura y hace que el proceso sea más efectivo. Este programa logra eliminar la necesidad de lo que es la escritura parásita en un netlist y este mismo logra ahorrar el espacio del disco. Más que todo se especializa en optimizar el proceso reduciendo las capacitancias parásitas que puedan llegar a afectar el comportamiento deseado de un circuito.

[10]

HSPICE

Hspice es una herramienta de synopsys que permite realizar las simulaciones de un programa generado (.spf), en donde viene de la mano con Custom Wave View. El fin de este programa es ver una simulación y lograr ver el comportamiento del circuito. Este se considera uno de los simuladores más potentes, ya que muestra gran precisión en los datos y la mayoría de las empresas de semiconductores dependen de esta herramienta. Además, se pueden realizar análisis de los circuitos eléctricos en estado estacionario, en el dominio de frecuencia y el transitorio.

[4]

Trabajo realizado

En los años anteriores se realizaron trabajos de investigación para poder utilizar y lograr entender los softwares como Verdi, Formality, VCS y HSPICE.

Se realizaron diferentes tipos de flujos según la complejidad requerida, en donde se lograron diseñar distintos flujos en la herramienta VCS, con el objetivo de verificar el comportamiento del circuito sintetizado. En este se realizan 2 simulaciones una para el circuito original y uno para el circuito sintetizado que ya hace referencia a las librerías de TSMC de 180 nm. Entonces el trabajo realizado por Jefferson Ruano, ya nos dice las librerías utilizadas para la simulación. En el trabajo del año 2020, también se logra ver el flujo de simulación con detección de condiciones de carrera, las cuales puede mostrar inconsistencias en las simulaciones ejecutadas. También logra mostrar el flujo para la evaluación de desempeño, en donde VCS cuenta con una herramienta conocida como generación de perfiles de simulación unificado, esta herramienta nos permite ver la cantidad de tiempo de CPU y memoria usada por nuestro diseño.

Con VCS en el trabajo realizado en el 2019 se realizo un flujo de simulación básico, el cual se utilizo para la simulación de Verdi del circuito original y el sintetizado. En este flujo básico se utilizaron ciertos comandos para llegar a realizar esto:

vcs-Muddate -RPP -v /yourPath/chip.v/yourPath/testBench.v -o demo -full 64 -debug_-all

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

./yourDesignName -gui [7] [3]

Luego con el tiempo se fue desarrollando métodos mas complejos para el desarrollo de circuitos mas complejos, en donde ya se realizaron flujos con detecciones de carrera, en esta se utilizaron detecciones de carrera dinámica y detecciones de carrera estática.

Para la dinámica se llega a generar lo que es un archivo *race.out* y *race.unique.out*, en donde una contiene una linea para las condiciones encontradas en la simulación y el otro contiene las lineas para las condiciones de carrera que son únicas respectivamente. Se mostrará a continuación el flujo propuesto por Jefferson con detección de carrera dinámica:

VCS_-HOME=/usr/synopsys/vcs-mx

export VCS_HOME

PATH=$VCSHOME/bin$: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

Para la herramienta de detección estática, se utilizan los *loops*, ya que estos son considerados como condiciones de carrera y esto suele ocurrir cuando eventos de control se encuentran después de un *always*. En esta parte del flujo se llega a proponer dos pruebas del diseño evaluado, una para evaluar los *loops* y otra para evaluar las condiciones de carrera que puedan existir en el reloj y los datos. El flujo que fue 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

Como ultima parte se habla del flujo para la evaluación de desempeño. VCS cuenta con una función que se le conoce como generador de perfiles de simulación unificado y con esta herramienta podemos ver la caracterización del circuito. Al utilizar esta herramienta requiere del 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

El trabajo realizado con Formality, se logró aprender que es una alternativa a la validación en las herramientas de VCS y Verdi. Este llega a generar un reporte más practico, ya que logra generar un reporte más simple con las coincidencias e inconsistencias al ver los circuitos.

En el trabajo realizado por Charlie Ayenci, en donde logro simular el diseño con parásitos por medio HSPICE, hubo ciertos errores al correr la simulación en donde, la empresa TSMC solamente logra proveer un kit académico, por lo que solo se encontraran las *black boxes*, que solo contienen el diseño de silicio del componente y con esto el netlist no logra poseer

los puertos de entrada, salida y alimentación.

[3] [4] [7]

CAPÍTULO 7

Verdi

Al utilizar la plataforma de Verdi nos podemos dar cuenta que es una herramienta para resolver y entender los errores en un pequeño fragmento de tiempo. Con esta herramienta estamos mejorando la eficiencia y la productividad de recursos que pueden llegar a ser costosos.

Esta plataforma necesita información de “knowledge DataBase” y “Fast Signal Database”, en donde en esta ocasión se utilizó el archivo .fsdb para poder tener una simulación representativa del Verilog.

En esta parte se va a mostrar el Verilog con el análisis que utiliza, al correr el punto Verilog, para ver si detecta algún error o incluso "warnings", que pueden llegar a afectar el circuito en un futuro.

La generación de la simulación se tuvo que señalar las señales que se muestran en las figuras 7, 10, 13, 16 y 19, para demostrar el funcionamiento del circuito tomando en cuenta el archivo .fsdb y se uso las señales de entrada y salida en su totalidad.

7.1. Flujos de circuitos con una complejidad baja

7.1.1. Flujo de una compuerta NOT

Figura 5: Resultados del análisis de una compuerta NOT (VERDI)

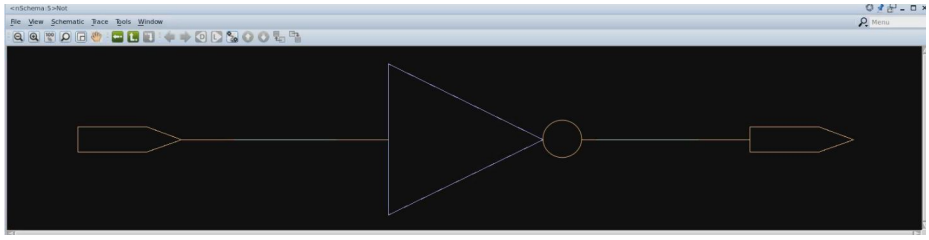

Figura 6: Esquemático de la compuerta NOT (VERDI)

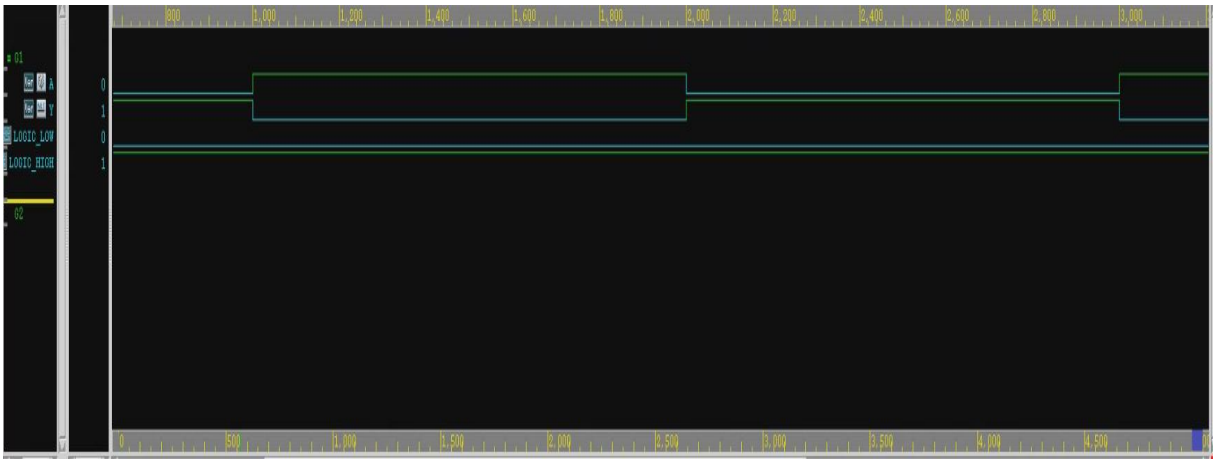

Figura 7: Simulación generada con el archivo .fsdb mostrando señales de entrada y salida de una not (VERDI)

7.1.2. Flujo de una compuerta XOR

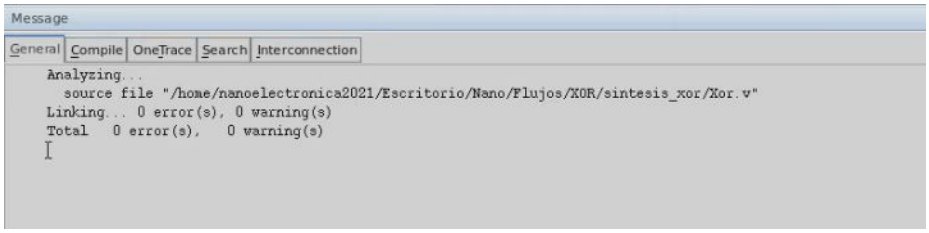

Figura 8: Resultados del análisis de una compuerta XOR (VERDI)

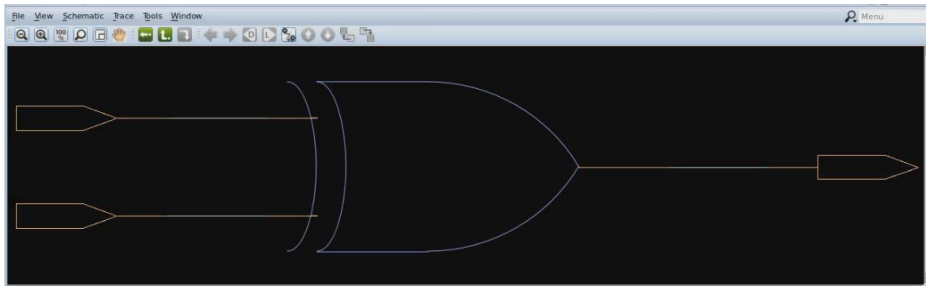

Figura 9: Esquemático de la compuerta XOR (VERDI)

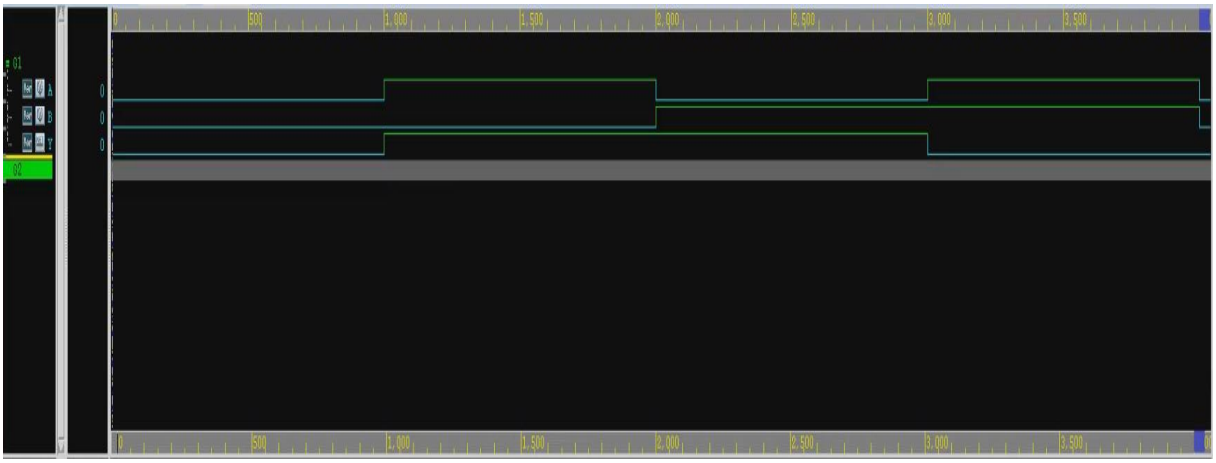

Figura 10: Simulación generada con el archivo .fsdb mostrando señales de entrada y salida de una XOR (VERDI)

7.2. Flujos de circuitos con una complejidad media

7.2.1. Flujo de un Full Adder

Figura 11: Resultados del análisis de un full adder (VERDI)

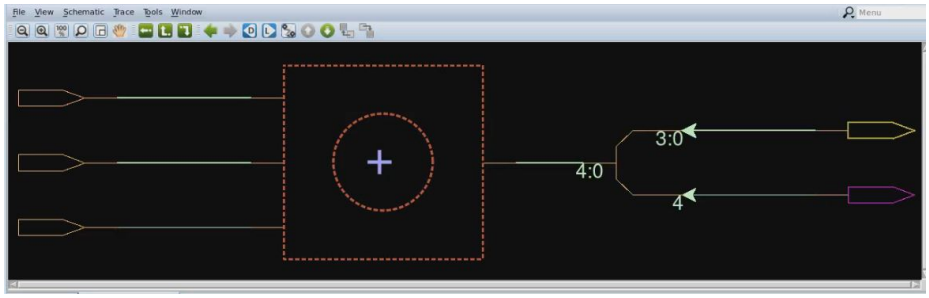

Figura 12: Esquemático de un full adder (VERDI)

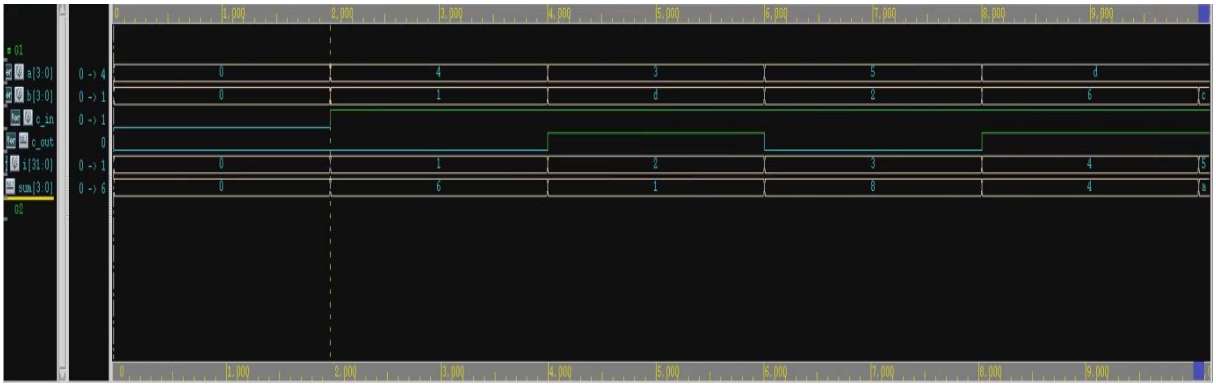

Figura 13: Simulación generada con el archivo .fsdb mostrando señales de entrada y salida de un full adder (VERDI)

7.2.2. Flujo de una ALU de 4 bits

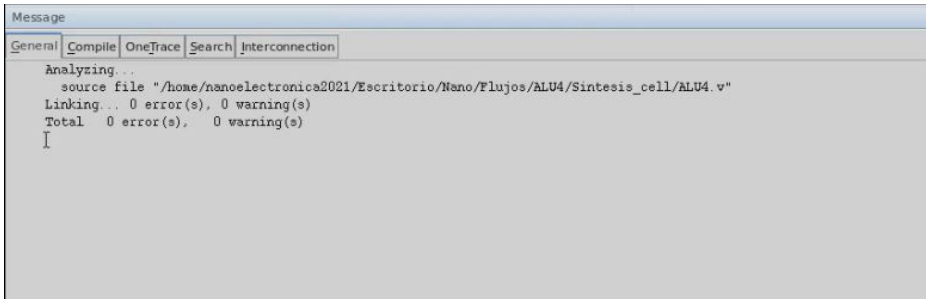

Figura 14: Resultados del análisis de una ALU de 4 bits (VERDI)

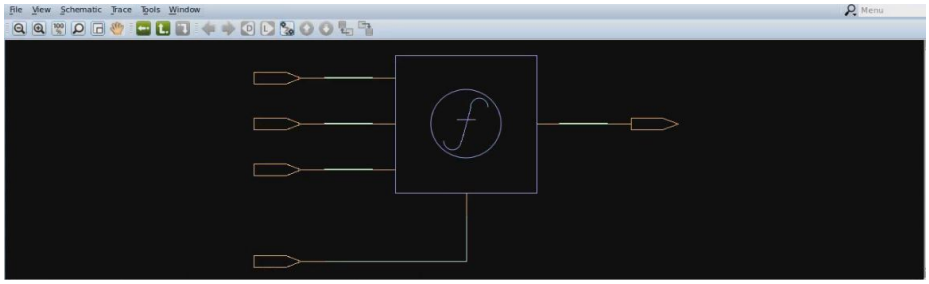

Figura 15: Esquemático de una ALU de 4 bits (VERDI)

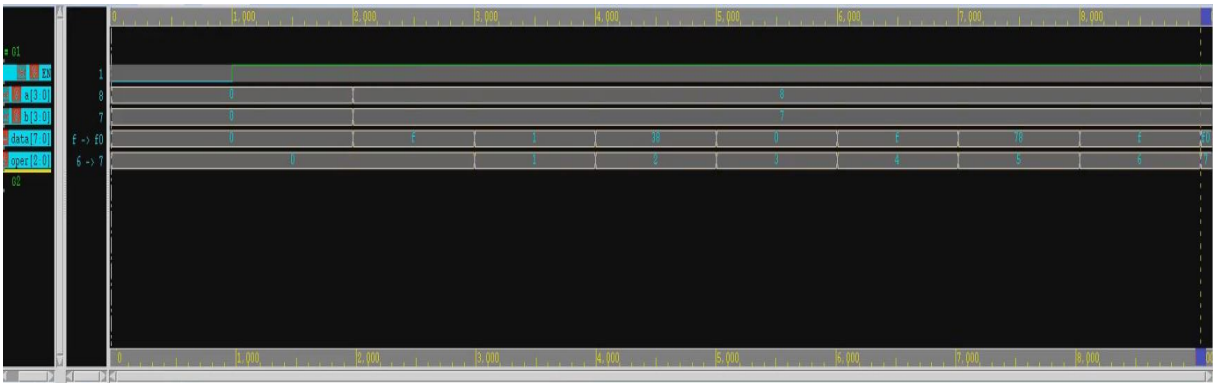

Figura 16: Simulación generada con el archivo .fsdb mostrando señales de entrada y salida de una ALU de 4 bits (VERDI)

7.3. Flujos de circuitos con una complejidad alta

7.3.1. Flujo de un contador de 4 bits

Figura 17: Resultados del análisis de un contador de 4 bits (VERDI)

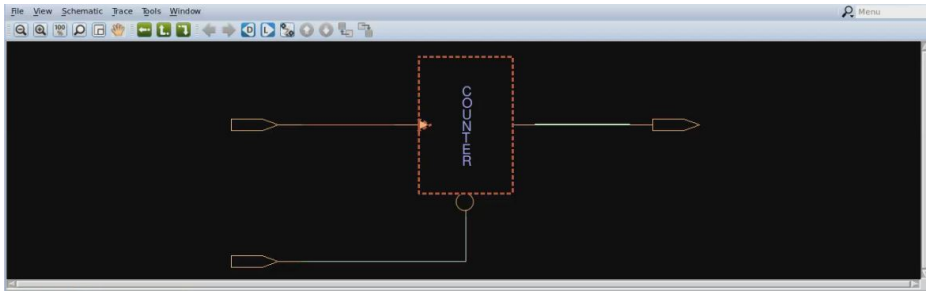

Figura 18: Esquemático de un contador de 4 bits (VERDI)

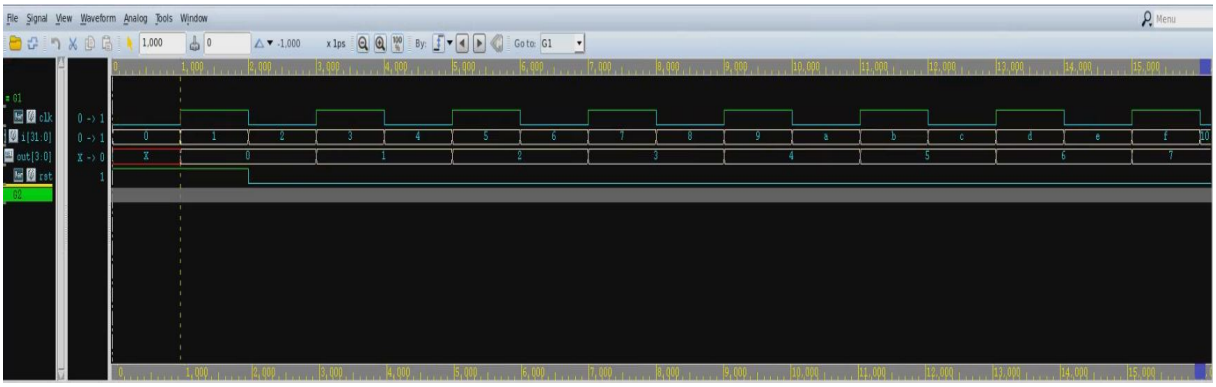

Figura 19: Simulación generada con el archivo .fsdb mostrando señales de entrada y salida de un contador de 4 bits (VERDI)

CAPÍTULO 8

Formality

8.1. Flujos de circuitos con una complejidad baja

8.1.1. Flujo de una compuerta NOT

Este flujo propuesto por una simple NOT, fue con el fin de poder validar el flujo con una complejidad baja y después realizar flujos mas complejos aprendiendo poco a poco de las herramientas que fueron propuestas. Los resultado se logran mostrar en las figuras 3 y 4, en este tipo de archivos al realizar la verificación se puede llegar a obtener inconsistencias, una verificación falsa de resultados y una verificación exitosa. Al comparar los puntos del archivo de verilog normal y la sintetizada muestra resultados correctos que no llego a haber un resultado incongruente.

```
********************************** 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...
```

Figura 20: Resultados de los "Matching Points"de una compuerta not

Al ser un circuito tan simple realmente no mostró muchos problemas, ya que no tiene muchos puntos para comparar .

```
********************************* 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
****************************************************************************************
1
```

Figura 21: Resultados de la verificación de una compuerta not

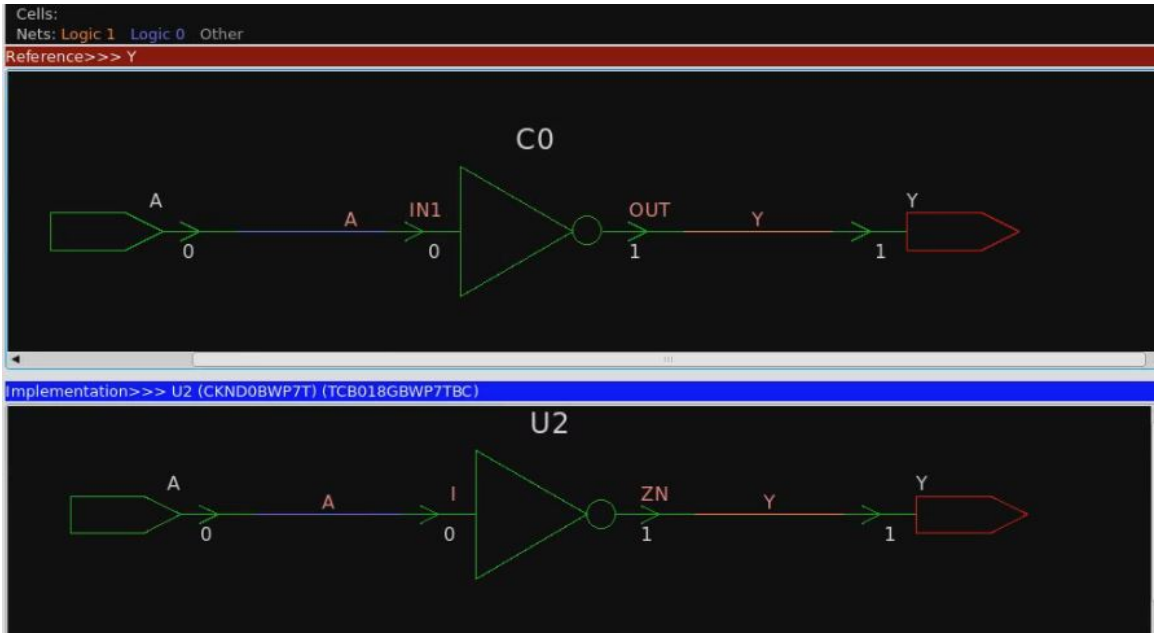

Figura 22: Esquemático del circuito de referencia vs el circuito sintetizado de una compuerta not

En el esquemático que se muestra en la figura.7 se puede ver que se están comparando 2 distintos circuitos, en este caso se muestra el circuito normal descripto en hardware sin pasar por ningún proceso y el otro circuito mostrando la Not sintetizada.

8.1.2. Flujo de una compuerta XOR

Este flujo es muy parecido al de la Not, ya que prácticamente lo único que cambia es el numero de entradas de la compuerta. Lo interesante de realizar esta compuerta de baja complejidad es comprobar con otro ejemplo si el flujo que estamos realizando es el adecuado.

```
********************************** 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
*******************************************************************************
********
```

Figura 23: Resultados de los "Matching Points"de una compuerta xor

A la hora de comparar puntos se puede ver la pequeña diferencia que tiene con respecto al flujo de la compuerta NOT, la cual es los "Matched primary points", en la compuerta Not se puede visualizar en la figura.5 que solo muestra uno, en este caso muestra 2 por las entradas de la XOR.

```
******************************** Verification Results *********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/Xor_2
 Implementation design: i:/WORK/Xor_2
 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 24: Resultados de la verificación de una compuerta xor

En el esquemático que se encuentra en la figura.10 se muestra con un poco mas de complejidad y además la compuerta que se muestra es la de un XOR, la cual detecto formality basado en el archivo.v, el cual describe hardware. Además, se aprecia en la figura que uno de los circuitos lo toma como referencia y el otro es el que se encuentra sintetizado.

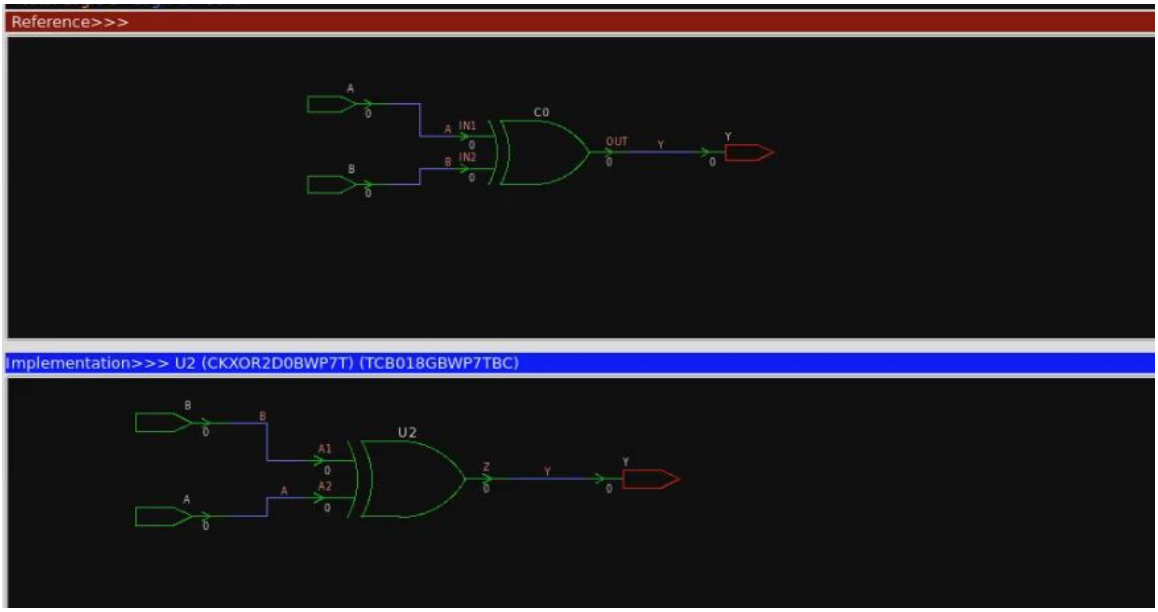

Figura 25: Esquemático del circuito de referencia vs el circuito sintetizado de una compuerta xor

Cabe resaltar que los circuitos de poca complejidad no son muy ilustrativos y tampoco representan una gran cantidad de resultados, los cuales analizar y mas en la herramienta de formality, ya que esta herramienta es mas utilizada para circuitos de mayor complejidad.

8.2. Flujos de circuitos con una complejidad media

8.2.1. Flujo de un Full adder

En el flujo propuesto de complejidad media se realizó un full-adder, en donde se realizaron equivalencias del circuito construido desde un verilog y del circuito sintetizado. Al ver el verilog sencillo no es un circuito que maneje una complejidad, pero al tener más entradas y salidas, al momento de sintetizar se van mostrando lo que va generando las librerías de TSMC y en este caso se puede ver que las librerías ya generan un montón de "pads", las cuales formality ya analiza.

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

1
```

Figura 26: Resultados de los "Matching Points"del full adder

En el momento de comparar puntos se logran ver más puntos respecto a los que se encuentran en los circuitos de complejidad baja y la cantidad de entradas y salidas que nos permiten ver un poco más el uso de esta herramienta.

```
********************************* Verification Results *********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/fulladd
 Implementation design: i:/WORK/fulladd
 5 Passing compare points
-------------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
-------------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       5       0       0       5
Failing (not equivalent)       0       0       0       0       0       0       0       0
*******************************************************************************************
1
```

Figura 27: Resultados de la verificación del full adder

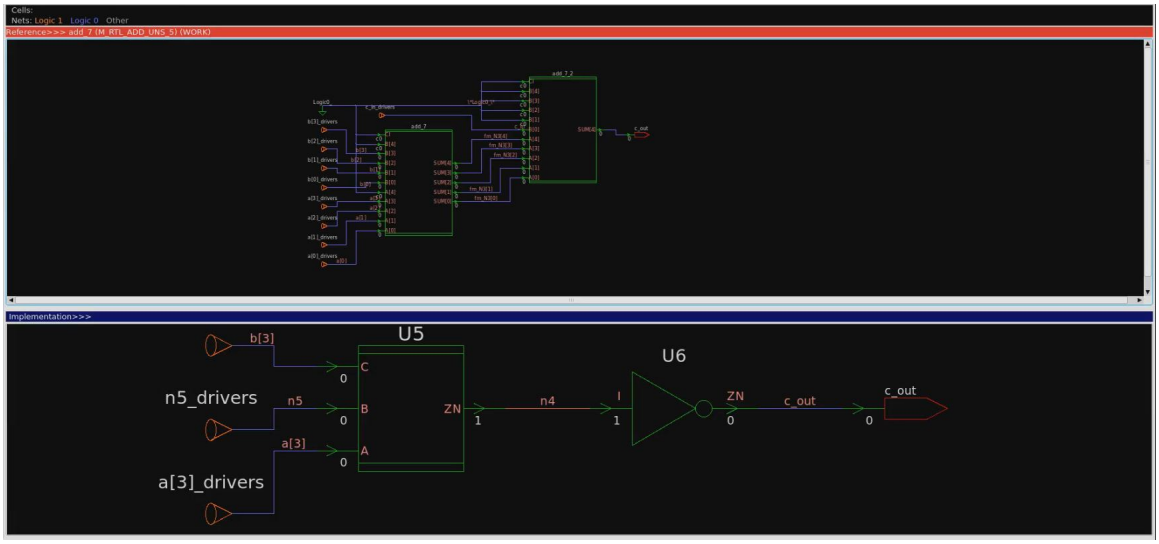

Figura 28: Esquemático del circuito de referencia vs el circuito sintetizado del full adder

En el circuito podemos ver la gran diferencia que se encuentra el circuito normal con el sintetizado y se puede ver que esta herramienta, procura ver los puntos del circuito para comprobar que el circuito sintetizado no tenga ninguna inconsistencia. En estos tipos de flujo, logramos adentrarnos un poco más al funcionamiento del circuito en la parte posterior de la verificación, la cual se llama "debug.[en] esta parte podemos ver los conos lógicos y el tamaño de estos, en los diferentes puertos que da el circuito.

8.2.2. Flujo de una ALU de 4 bits

Este es un circuito mas complejo que un full adder, con la verificación formal de este circuito podemos ver que la cantidad de "pads"que va generando el proceso de síntesis comienza a ser un problema, ya que todas estas de lineas generadas, a la hora de comparar pueden llegar a tener un inconsistencia que puede llegar a afectar en el proceso de síntesis física.

```
********************************** Matching Results **********************************
 8 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 12 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...
```

Figura 29: Resultados de los "Matching Points"de la ALU de 4 bits

```
******************************** Verification Results ********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/ALU_4bit
 Implementation design: i:/WORK/ALU_4bit
 8 Passing compare points
--------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       8       0       0       8
Failing (not equivalent)       0       0       0       0       0       0       0       0
**************************************************************************************
1
```

Figura 30: Resultados de la verificación de la ALU de 4 bits

	Type	Reference	r:Size	Implementation	i:Size	+/-
1	Port		data[0] 193		data[0] 155	
2	Port		data[1] 237		data[1] 149	
3	Port		data[2] 281		data[2] 192	
4	Port		data[3] 325		data[3] 243	
5	Port		data[4] 365		data[4] 255	
6	Port		data[5] 373		data[5] 275	
7	Port		data[6] 410		data[6] 258	
8	Port		data[7] 447		data[7] 128	

Figura 31: Listado de los puntos que pasaron la prueba

En el listado de puntos podemos ver que el proceso se comienza a complicar en la parte de "debug", aunque en este caso no se llego a encontrar ninguna inconsistencia, pero se logra ver el trabajo que esta realizando formality con las librerías de TSMC.

8.3. Flujos de circuitos con una complejidad alta

8.3.1. Flujo de un contador de 4 bits

Lo que hace que este circuito sea considerado de complejidad alta, es porque es considerado un circuito secuencial, por lo que la salida no depende solo de sus entradas, sino que lleva un registro del historial de los valores que fue tomando de entradas anteriores.

```
********************************** Matching Results **********************************
 8 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...
```

Figura 32: Resultados de los "Matching Points"de un contador de 4 bits

```
******************************** Verification Results ********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/counter4
 Implementation design: i:/WORK/counter4
 8 Passing compare points
--------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       4       4       0       8
Failing (not equivalent)       0       0       0       0       0       0       0       0
**************************************************************************************
1
```

Figura 33: Resultados de la verificación de un contador de 4 bits

	Type	Reference	r:Size	Implementation	i:Size	+/-
1	DFF		out_reg[0] 25		out_reg[0] 16	
2	DFF		out_reg[1] 32		out_reg[1] 16	
3	DFF		out_reg[2] 39		out_reg[2] 22	
4	DFF		out_reg[3] 46		out_reg[3] 39	
5	Port		out[0] 1		out[0] 2	
6	Port		out[1] 1		out[1] 2	
7	Port		out[2] 1		out[2] 2	
8	Port		out[3] 1		out[3] 2	

Figura 34: Listado de los puntos que pasaron la prueba

En la figura 18 y 19, podemos ver que existe una diferencia respecto a los flujos anteriores y es que aquí muestra algo diferente a lo que son los puertos, lo que es DFF son los registros del contador de 4 bits.

CAPÍTULO 9

Conclusiones

- Se implementaron varios flujos de diseño para aprender mas sobre cada una de las herramientas como formality y verdi.
- Todos los flujos resultaron exitosos, por las herramientas de "debug"de Formality y Verdi.
- Se da a conocer una nueva herramienta que solo había sido mencionada hasta el momento, una herramienta que ayuda a localizar, entender y resolver los errores. Para maximizar la eficiencia y productividad de recursos costosos.

CAPÍTULO 10

Recomendaciones

- Al instalar futuras herramientas de synopsys procurar que tengan la misma versión, sino puede llegar a causar problemas y mas con las partes de flujo que van de la mano.
- Al utilizar formality, se necesita de circuitos de una complejidad media o alta para que la herramienta se pueda aprovechar, pero nunca esta de mas probar las compuertas de complejidad baja, para ver si se están realizando los pasos correctos.
- Realizar las tareas en conjunto con la persona encargada de la síntesis lógica, para la generación de archivos de VCS.
- Tener un conocimiento básico del uso de VCS, ya que Verdi utiliza los archivos generados desde VCS.
- Revisar el material generado y guiarse por las herramientas que vienen dentro del instalador de cada una de estas, en donde muestra el tutorial del uso de la herramienta aunque no sea en su totalidad y guías para familiarizarse mejor con la herramienta.

CAPÍTULO 11

Bibliografía

[1] J. F. M. Lenddech, "Circuitos integrados de pequeña, mediana y gran escala," en *Licenciatura en ingeniería en computación*, Universidad Autónoma del Estado de México, N/A, págs. 1-32.

[2] S. Kang e Y. Leblebici, "CMOS FABRICATION TECHNOLOGY AND DESIGN RULES," *Chapter 2 (Fabrication of MOSFETs) of the book CMOS Digital Integrated Circuit Design*, 2003.

[3] S. R. Vasquez, "Definición del flujo de diseño para fabricacion de un chip con tecnologia VLSI CMOS," *Facultad de Ingeniería Electrónica*, 2020.

[4] C. C. Girón, "Ejecución y utilización de un flujo de diseño para el desarrollo de un chip con tecnología nanométrica: Extracción de componentes parásitos y simulaciones en HSPICE," *Facultad de Ingeniería Electrónica*, 2020.

[5] F. torres del Valle, "LENGUAJES DE DESCRIPCIÓN DE HARDWARE," *xdoc.mx*, págs. 1-8, https://xdoc.mx/preview/lenguajes-de-descripcion-de-hardware-5c2d1aa29da5a.

[6] Synopsys, "VCS," *https://www.synopsys.com/verification/simulation/vcs.html*, N/A, 2021.

[7] J. R. Orellana, "Definición del flujo en la herramienta VCS para la simulación de HDLs en la Fabricación de un Chip con Tecnología Nanométrica CMOS," *Facultad de Ingeniería Electrónica*, 2020.

[8] Synopsys, "Verdi," *https://www.synopsys.com/verification/debug/verdi.html*, N/A, 2021.

[9] ——, "Formality User Guide, Version S-2021.06-SP2," *www.synpsys.com*, September, 2021.

[10] ——, "StarRC," *https://www.synopsys.com/implementation-and-signoff/signoff/starrc.html*, N/A, 2021.

CAPÍTULO 12

Anexos

12.1. Uso de comandos

12.1.1. Formality

Al iniciar formality en una terminal se utiliza el comando: "Formalityçon esto va a abrir 2 cosas, una que es la interfaz gráfica para poder realizar la verificación y va a seguir corriendo en paralelo los comandos necesarios dependiendo de las acciones que se hagan en la interfaz gráfica.

```
read_verilog -container r -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/
not/pruebas_not/Not.v }
set_top r:/WORK/Not
read_verilog -container i -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/
not/pruebas_not/salidas_not/out_not.v }
read_db { /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/not/pruebas_not/Formality/
tcb018gbwp7tbc.db /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/not/pruebas_not/Formality/
tpd018nvtc.db }
read_verilog -container i -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/
not/pruebas_not/salidas_not/out_not.v }
set_top i:/WORK/Not
match
verify
```

Figura 35: Comandos requeridos para la verificación de la síntesis lógica de una compuerta NOT

Para poder correr estos comandos que se presentan en la figura de arriba, Formality tiene que estar en fm_shell, en esta parte se puede ir variando en las diferentes etapas de la verificación, las cuales son: Guide, Setup, Match y Verification. Estas se pueden ir variando, poniendo en la terminal el nombre tal cual se muestra. Al utilizar fm_shell va a tirar a la etapa de Setup por defecto y corriendo línea por línea como se muestra en la figura, esta etapa debería ser realizada.

```
read_verilog -container r -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/Flujos/XOR/
sintesis_xor/Xor.v }
set_top r:/WORK/Xor_2
read_verilog -container i -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/Flujos/XOR/
sintesis_xor/salidas/out_xor.v }
read_db { /home/nanoelectronica2021/Escritorio/Nano/Flujos/XOR/sintesis_xor/tcb018gbwp7tbc.db /home/
nanoelectronica2021/Escritorio/Nano/Flujos/XOR/sintesis_xor/tpd018nvtc.db }
read_verilog -container i -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/Flujos/XOR/
sintesis_xor/salidas/out_xor.v }
set_top i:/WORK/Xor_2
match
verify
```

Figura 36: Comandos requeridos para la verificación de la síntesis lógica de una compuerta XOR

```
read_verilog -container r -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/
Flujos/Full_adder/Sintesis_cell/fulladder.v }
set_top r:/WORK/fulladd
read_verilog -container i -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/
Flujos/Full_adder/Sintesis_cell/salidas/Fulladd_out.v }
read_db { /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/not/pruebas_not/Formality/
tcb018gbwp7tbc.db /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/not/pruebas_not/
Formality/tpd018nvtc.db }
set_top i:/WORK/fulladd
match
verify
save_session -replace /home/nanoelectronica2021/Escritorio/Nano/Flujos/Full_adder/
Sintesis_cell/Formality/F_ADDER
```

Figura 37: Comandos requeridos para la verificación de la síntesis lógica de un Full Adder

```
read_verilog -container r -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/
Flujos/Full_adder/Sintesis_cell/fulladder.v }
set_top r:/WORK/fulladd
read_verilog -container i -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/
Flujos/Full_adder/Sintesis_cell/salidas/Fulladd_out.v }
read_db { /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/not/pruebas_not/Formality/
tcb018gbwp7tbc.db /home/nanoelectronica2021/Escritorio/Nano/Flujos/NOT/not/pruebas_not/
Formality/tpd018nvtc.db }
set_top i:/WORK/fulladd
match
verify
save_session -replace /home/nanoelectronica2021/Escritorio/Nano/Flujos/Full_adder/
Sintesis_cell/Formality/F_ADDER
```

Figura 38: Comandos requeridos para la verificación de la síntesis lógica de una ALU de 4 bits

```
read_verilog -container r -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/
Flujos/Counter4B/Sintesis_cell/counter4.v }
set_top r:/WORK/counter4
read_verilog -container i -libname WORK -05 { /home/nanoelectronica2021/Escritorio/Nano/
Flujos/Counter4B/Sintesis_cell/salidas/counter4_out.v }
read_db { /home/nanoelectronica2021/Escritorio/Nano/Flujos/Counter4B/Sintesis_cell/FORMALITY/
tcb018gbwp7tbc.db /home/nanoelectronica2021/Escritorio/Nano/Flujos/Counter4B/Sintesis_cell/
FORMALITY/tpd018nvtc.db }
set_top i:/WORK/counter4
match
verify
save_session -replace /home/nanoelectronica2021/Escritorio/Nano/Flujos/Counter4B/
Sintesis_cell/FORMALITY/Counter4
```

Figura 39: Comandos requeridos para la verificación de la síntesis lógica de un contador de 4 bits

Estos son los comandos generados desde la consola de formality, al realizar un verificación de un flujo en la interfaz gráfica se van generando una serie de comandos en la parte del "setupz esos comandos se guardaron en un .tcl en donde se sacaron las figuras anteriores.