UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería


# Diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC: Implementación de un alternativo flujo de diseño proporcionado por Synopsys con las herramientas *PrimeTime* y *TetraMAX*.

Trabajo de graduación presentado por José David Ponce Del Cid para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería


# Diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC: Implementación de un alternativo flujo de diseño proporcionado por Synopsys con las herramientas *PrimeTime* y *TetraMAX*.

Trabajo de graduación presentado por José David Ponce Del Cid para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2023

Vo.Bo.:

(f) _____

Ing. Jonathan De Los Santos

Tribunal Examinador:

(f) _____

Ing. Jonathan De Los Santos

(f) _____

MSc. Carlos Esquit

(f) _____

Ing. Luis Nájera

Fecha de aprobación: Guatemala, 5 de enero de 2023.

# Prefacio

La realización de este trabajo de graduación no habría sido posible sin mi familia, a quienes agradezco por el apoyo incondicional y los ánimos para mantener mi excelencia en los estudios. A mis amigos y compañeros de Ingeniería Electrónica en UVG, quienes me acompañaron a lo largo de 5 años en esta maravillosa carrera llena de retos y descubrimientos. Por último, pero no menos importante a los profesores que me guiaron en el proceso, principalmente a mi asesor de trabajo de graduación, el Ing. Jonathan De Los Santos, quien con mucha vocación e interés, me ayudó a trazar el camino para finalizar el presente documento, y al MSc. Carlos Esquit, quien me enseñó desde su noble pasión por la enseñanza lo esencial sobre nanotecnología y a tener una intuición con los circuitos en general, siendo el tema principal de mi trabajo de graduación.

# Índice

# Lista de figuras

# Resumen

En este estudio, el principal objetivo es identificar y detallar una alternativa al flujo de diseño para la creación de un nanochip utilizando tecnología de 180nm. Destacan particularmente las herramientas *TetraMAX* y *PrimeTime*, abordando desde su instalación hasta su aplicación práctica. Este trabajo se basa en el procedimiento de la alternativa propuesta, respaldado por laboratorios descriptivos proporcionados por *Synopsys* en el sitio web de *SolvNet*.

Nuestra intención es ofrecer una guía estructurada y fácilmente replicable, sirviendo como manual para diseñar cualquier nanochip, partiendo de un archivo verilog y finalizando con los archivos requeridos por *TSMC* para su manufactura. Aunque se presenta un panorama completo del diseño, se pone un énfasis particular en las etapas de análisis de tiempos (*PrimeTime*) y en el análisis de rutas críticas (*TetraMAX*).

# Abstract

In this study, the primary goal is to identify and detail an alternative design flow for creating a nanochip using 180nm technology. It particularly highlights the tools *TetraMAX* and *PrimeTime*, addressing everything from their installation to their practical application. This work is based on the procedure of the proposed alternative, supported by descriptive labs provided by *Synopsys* on the *SolvNet* website.

The intention is to provide a structured and easily replicable guide, serving as a manual for designing any nanochip, starting from a verilog file and concluding with the files required by *TSMC* for its manufacturing. While a comprehensive overview of the design is presented, particular emphasis is placed on the time analysis stages (*PrimeTime*) and critical path analysis (*TetraMAX*).

# CAPÍTULO 1

# Introducción

Hoy en día, existe un proceso muy complicado para diseñar nanochips, especialmente de tecnologías con features muy pequeños, puesto que la capacidad instalada para fabricarlos se concentra en Taiwán, específicamente en las fábricas de *TSMC*. Con el paso del tiempo, los nanochips de uso cotidiano en todo el mundo se fabrican con más frecuencia en las instalaciones de *TSMC* pues han especializado mucho el proceso de fabricación.

En este trabajo se explora un flujo de diseño capaz de cumplir con los requisitos de fabricación de *TSMC*, presentado de manera ordenada y clara para que el lector y futuros investigadores puedan replicar el proceso de un nanochip de 180nm.

# CAPÍTULO 2

# Antecedentes

Antes de entender la necesidad de realizar un flujo de diseño alternativo al principal, se tuvo que replicar el proceso del flujo de diseño de un nanochip de 180 nm que se ha realizado en iteraciones pasadas del proyecto. Se documentó el proceso de diseño, que se replicó también para este trabajo de investigación. Los investigadores a cargo de iteraciones anteriores fueron estudiantes de la UVG de último año, quienes aportaron en el proceso de diseño de un nanochip de 180 nm nombrado "El Gran Jaguar". El proceso hasta este punto ha sido el siguiente:

## 2.1. Creación del archivo *verilog*

Para la elaboración del nanochip, es necesario partir de un diseño de un circuito funcional, en este caso se optó realizarlo en *verilog* como se muestra en la Figura 1.

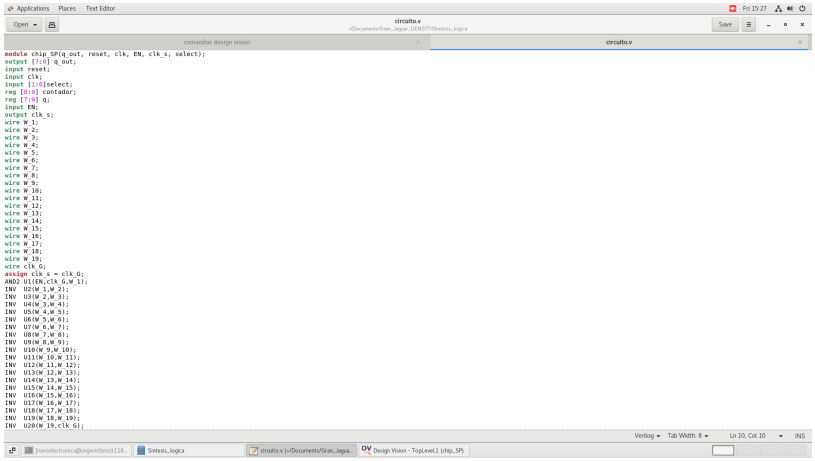



Figura 1: Creación del archivo *verilog*.

Al ser un poco tediosa la conexión en texto de tantas compuertas que logren el objetivo del circuito, se programó en *python* un código cuyo objetivo es generar el *verilog*. La funcionalidad de este *verilog* resultante, consiste en una nube combinacional con un contador de entrada y un bus de 8 bits de salida. Este contador se conecta a su vez a un reloj, que lo hace sumarle 1 a su valor en cada ciclo. El trabajo de la nube combinacional, es recibir un valor del contador y que se propaguen las señales de tal manera que la salida en el bus de 8 bits sea un carácter representado en binario. Gracias a esta visualización en binario, es posible leer los caracteres y formar un texto, en este proyecto, el Gran Jaguar será capaz de imprimir 2 textos diferentes.

## 2.2. Síntesis lógica

En el proceso de síntesis lógica, se pretende tomar el archivo *verilog* y obtener 3 archivos resultantes: .gds, .sdc y .v . El proceso de síntesis lógica se conforma de 2 etapas principales:

Primera etapa: El nuevo *verilog* resultante, ya no estará descrito en términos de condicionales ïfçomo lo estaba en el generado por el *python*, sino que el programa *Design Vision* busca la manera de conectar compuertas, flip flops y multiplexores que logren el trabajo deseado. Por esto al observar el archivo *verilog* resultante se identifican muchas compuertas conectadas de una manera aparentemente desordenada, como se puede observar en la Figura 2.

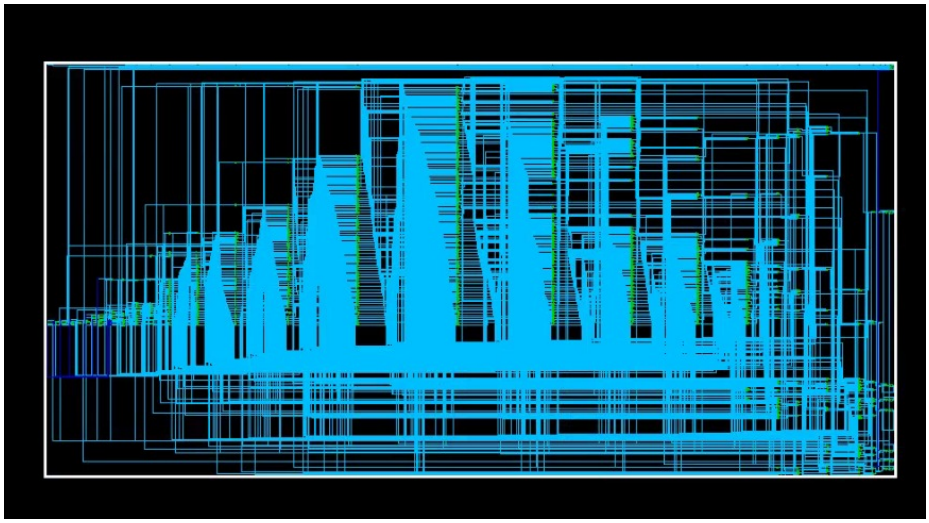


Figura 2: Compuertas sintetizadas lógicamente

Segunda etapa: En la segunda etapa, se toman las mismas compuertas obtenidas como resultado en la primera etapa, pero se agregan líneas en el *verilog* de tal manera que se definan las entradas y salidas de los bloques principales, entre ellas la señal de reloj, el contador y la salida de 8 bits que representa el carácter en cada ciclo de reloj. De esta manera tenemos como resultado un nuevo *verilog* "modular"pues está separado por bloques más grandes (Figuras 3 y 4) y se puede pasar a la etapa de convertirlo en un layout físico.

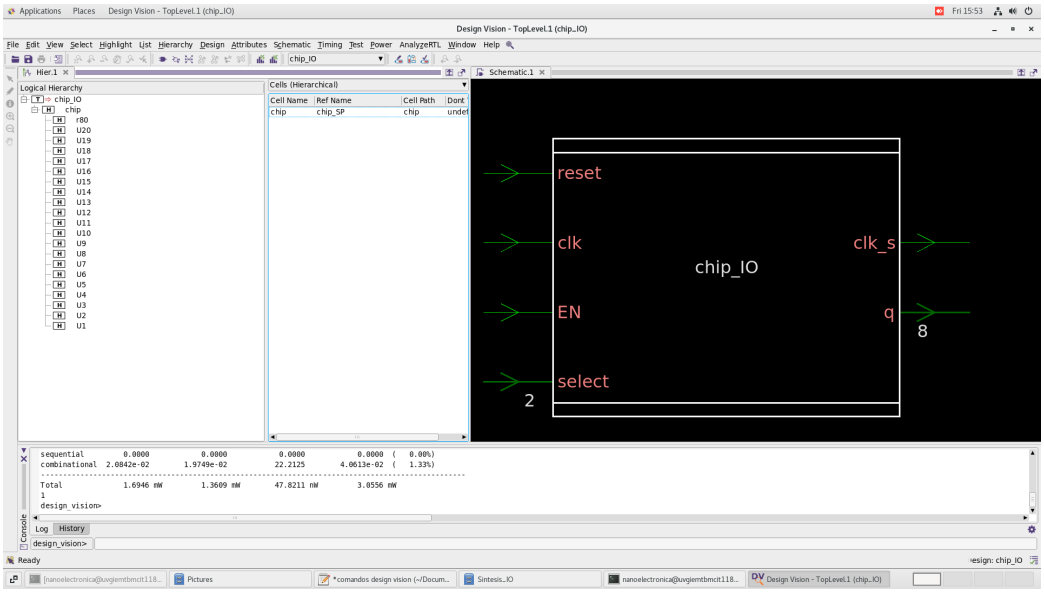



Figura 3: Bloque con entradas y salidas

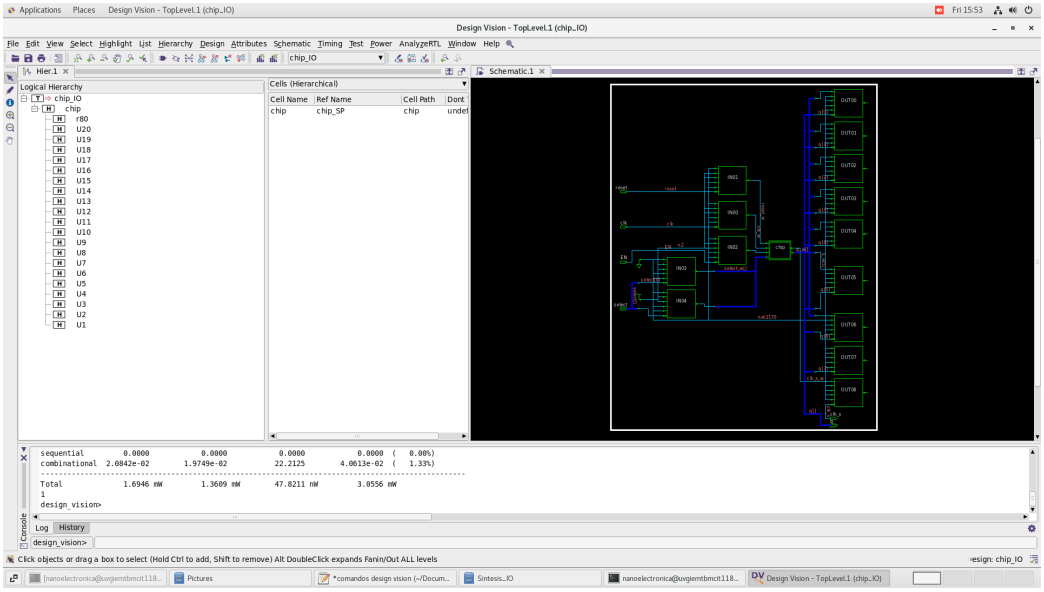


Figura 4: Bloques dentro de Chip IO

## 2.3. Síntesis física

Esta etapa es más complicada que la síntesis lógica, esto porque su objetivo es usar el esquemático representado por el *verilog* y representarlo físicamente en forma de dopajes, polisilicio y metales sobre la superficie de un wafer.

Se usaron los comandos recomendados por la tesis del año pasado de Antonio Altuna, y como resultado se obtuvieron los archivos de extensiones gds, y un nuevo *verilog*. Aquí

mismo en *ICC2*, se realiza un proceso de verificación DRC

En este proceso existe una etapa de verificaciones que se realizan para verificar que el diseño cumpla con las reglas establecidas por el fabricante, tanto de aspectos físicos como eléctricos. Durante la verificación DRC se encontraron 6 errores de densidad en las capas de Metal 1 hasta el Metal 6.

Lo que se haría normalmente para corregir estos errores de densidad, es utilizar un script brindado por el fabricante que se encarga de rellenar automáticamente espacios en las capas de metales para que alcancen la densidad deseada y no se mantengan por debajo de 0.3 en este caso, pero *Synopsys* descontinuó el uso y soporte del programa que corría el script, llamado *Hercules*.

Ante la descontinuación de *Hercules*, los errores de densidad deben ser corregidos por *ICC2*. El uso de otro programa para el proceso implica una nueva interfaz y nuevas herramientas, para las cuales no existe un equivalente al script anterior que corría en *Hercules*.

Con los errores de densidad es que se nota la necesidad de cuestionar el flujo de diseño usado tradicionalmente por anteriores estudiantes, puesto que representa un obstáculo significativo en el proceso de diseño y se considera valioso explorar simultáneamente otras alternativas como el flujo de diseño propuesto por *Synopsys*

# CAPÍTULO 3

## Justificación

La corrección de los errores de densidad en el flujo de diseño tradicional, representa un obstáculo en el diseño del nanochip que pone un alto al proceso de diseño, es decir que sin la corrección de los errores de densidad, el fabricante (en este caso *TSMC*) no podrá procesar la fabricación. No solamente es fundamental para la continuación del proceso, sino que muy probablemente requerirá de la corrección de pasos anteriores a la etapa de verificaciones a lo largo de todo el flujo de diseño.

A partir de este obstáculo, existen dos principales soluciones: buscar la metodología específica que solucione los problemas de densidad, o atribuir los errores de densidad al mismo flujo de diseño tradicional y en vez de ese usar uno alternativo. Es en esta investigación que se opta por la segunda opción, y por lo tanto se explora un flujo alternativo propuesto por *Synopsys*

Parte de la importancia de realizar esta investigación radica en adaptarse a un flujo de diseño ya documentado por *Synopsys* y que sea más fácil aprender a realizarlo y como es más utilizado, también se facilitaría el acceso a solución de errores tanto por la comunidad como por el mismo proveedor.

# CAPÍTULO 4

# Objetivos

## 4.1. Objetivo general

- Permitir la continuación del proyecto de diseño de un nanochip con tecnología de 180 nm por medio de la propuesta de un flujo de diseño alternativo sugerido por *Synopsys*, Utilizando las librerías educativas de *Synopsys* y replicando el proceso con las librerías de *TSMC*.

## 4.2. Objetivos específicos

- Utilizar la herramienta *PrimeTime* para realizar el análisis de rutas críticas exitosamente.
- Utiliza la herramienta *TetraMAX* para realizar el diagnóstico de fallas exitosamente.
- Documentar el proceso paso a paso del flujo de diseño alternativo, incluyendo detalles de las herramientas utilizadas, en especial *PrimeTime* y *TetraMAX*.
- Mantener un estado de diálogo constructivo con el resto del grupo investigador.
- Realizar el proceso de diseño completo con las librerías educativas de *Synopsys* y replicar el proceso con las librerías de *TSMC*.

# CAPÍTULO 5

# Alcance

Este trabajo busca ser una guía a seguir para un proceso ordenado de diseño de un nanochip, especialmente si se necesita saber más acerca de la instalación de 2 programas en específico: *TetraMAX* y *PrimeTime*. También cuenta con explicaciones pertinentes respecto a los formatos de entrada y salida de los programas utilizados y las explicaciones de los comandos utilizados.

Como guía, el presente trabajo de graduación explicará paso por paso el proceso de flujo de diseño recomendado por *Synopsys* para que se realice de la manera más ordenada posible.

Independientemente de los resultados, más adelante también se cubre un ejemplo del flujo de diseño completo usando como circuito de prueba el diseño del "Gran Jaguar", diseño de circuito creado por estudiantes que ya se han graduado de Ingeniería Electrónica en UVG.

# CAPÍTULO 6

# Marco teórico

Se mencionaba que el diseño de un nanochip es un proceso muy complejo que requiere del conocimiento de teoría nanoelectrónica y también competencias técnicas para usar los programas para el diseño. Estos programas son brindados por una empresa llamada *Synopsys*, que le otorgó una licencia especial a la Universidad del Valle mediante un trato educativo. Es necesario conocer las nociones básicas del flujo de diseño y de los programas que se utilizarán para realizarlo, es por esto que se explicarán a continuación:

## 6.1. Errores de densidad:

Para entender el obstáculo principal del flujo de diseño tradicional, el cual hace necesario este trabajo de investigación, es imprescindible entender qué es un error de densidad, las implicaciones que tienen y la forma de resolverlos. Cuando se habla de densidad, se está refiriendo a las capas individuales que conforman el nanochip. Éstas van desde el wafer hasta el metal más alto. La importancia del concepto de densidad entra a discusión cuando se procede a hacer el ruteo del nanochip en las capas de metal.

En el proceso de ruteo, la herramienta correspondiente hace las conexiones utilizando algoritmos de optimización en los que se basa para que exista conectividad en todo el nanochip y su funcionalidad no se vea comprometida. Después de este proceso de ruteo, se obtienen como resultado varias capas de metal (que funciona como conductor) y cada capa tiene una densidad diferente. La densidad se refiere a la relación entre cantidad de metal y cantidad de espacio vacío en la misma capa.

Es estándar en la industria de muchos fabricantes que se le pida al cliente que cumpla con algunas reglas de diseño, una de éstas es una regla de densidad, que indica un rango dentro del cuál la densidad es permitida o aceptable para lograr un proceso de fabricación sin problemas.

Los errores de densidad se muestran cuando no se cumple con estas reglas de diseño, y en

esta investigación se exploran diferentes métodos para resolverlos antes de enviar el diseño del nanochip a la fábrica de *TSMC* y poder tener un proyecto exitoso.

## 6.2. Flujo de diseño tradicional:

Como se explicó en la sección de errores de densidad, el nanochip pasa por varias etapas de diseño, en esta sección se explicarán las etapas desde el inicio, hasta el momento en el que se encuentran los errores de densidad.

- **Diseño del *verilog*** El primer paso para diseñar un nanochip, es contar con un archivo *verilog*, que puede ser tanto behavioral como structural. La diferencia entre estos dos términos es que el primero describe el comportamiento deseado mientras el segundo describe la estructura de las conexiones que se desean. Un código VHDL estructural sí es sintetizable, mientras que uno behavioral no lo es. Una vez que se obtenga el archivo *verilog*, se puede proceder a la síntesis lógica.
- **Síntesis lógica:** La síntesis lógica se encarga de convertir el archivo *verilog* en otro del mismo formato, que tenga la misma funcionalidad, pero sus conexiones estén sintetizadas, es decir que se pueden agrupar varios subcircuitos del *verilog* original en uno más grande que solamente indique las entradas y salidas. Este proceso se puede hacer varias veces dependiendo de qué tan sintetizado se desea el resultado. Es necesario pasar por este proceso antes de la síntesis física, para que las conexiones del nanochip final sean más ordenadas.
- **Síntesis física:** La síntesis física es el proceso más largo del flujo de diseño, debido a su complejidad de comprensión tanto como sus capacidades. Es en esta etapa en donde se convierte el *verilog* anterior en un diseño físico. Este diseño físico cuenta con varias capas de interconnects y conductores que terminan de conectar los subcircuitos y hacen que el nanochip cumpla su funcionalidad. Es importante que luego de ruteado el nanochip, se realizan varias etapas de verificación para conocer los errores de diseño, layout y otros como antena o energía.
- **Design Rule Check:** Existen varias verificaciones post-síntesis física, pero la más importante para explicar este reporte es la verificación DRC. Esta verificación usa un runset proveído por el fabricante para que la herramienta busque todos aquellos aspectos del diseño que no cumplan con las reglas para proceder según el caso: resolver el error, o indicarlo al usuario para que lo resuelva. La mayoría de errores en esta etapa se corrigen automáticamente, pero existen algunos más problemáticos que necesitan intervención del usuario.

## 6.3. Flujo de diseño alternativo propuesto por *Synopsys*:

En vista de los obstáculos que presenta el flujo de diseño usado con anterioridad, es necesario explorar otras alternativas. A lo largo de este trabajo, se explorará un flujo de diseño alternativo al mencionado, propuesto por *Synopsys*. Cabe destacar que al igual que

el flujo anterior referido como tradicional, inicia con un diseño de *verilog*. Ésta alternativa consiste en las siguientes etapas:

- **Simulación Lógica (*VCS*):** Es recomendado como primer paso correr simulaciones del diseño del circuito realizado en VHDL, para verificar su funcionamiento y corregir errores en una etapa temprana del flujo general. Éstas pruebas se pueden realizar con cualquier software de simulación de VHDL, en nuestro caso se optó por *VCS*, software proporcionado por *Synopsys* utilizado como simulador de *Verilog* de alto rendimiento y alta capacidad [1].

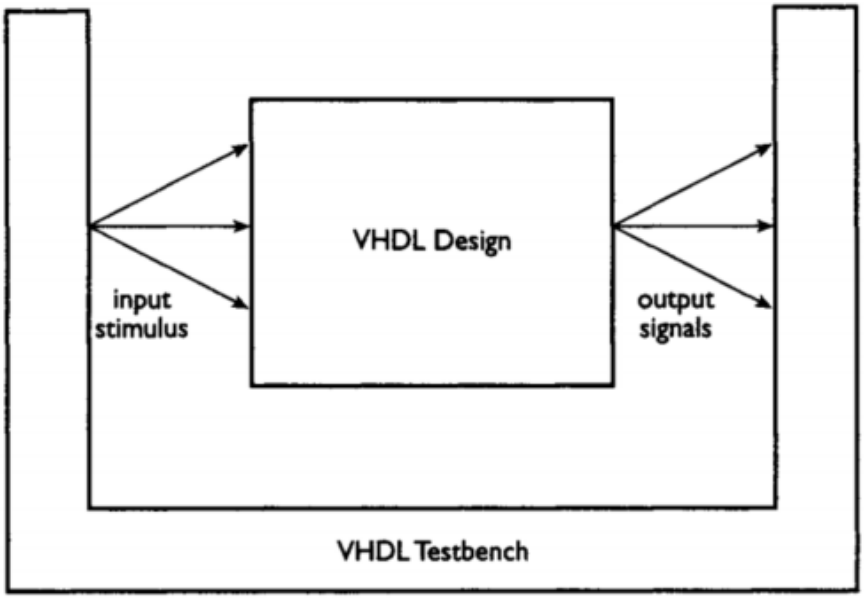



Figura 5: Diagrama de simulación VHDL

Como se puede observar en la Figura 5, para realizar una simulación de un diseño VHDL, es necesario un archivo de Testbench también en VHDL, ya que en éste se incluye la rutina de entradas junto a una verificación de salidas para correr pruebas específicas dentro de la simulación . Es en el archivo Testbench en el que el usuario tiene la capacidad de escoger vectores de entrada para verificar el funcionamiento lógico de su diseño. Más adelante en el flujo de diseño también se realizan pruebas y simulaciones, pero lo que diferencia la simulación lógica del resto de etapas, es que aquí solamente se comprueba el funcionamiento lógico del circuito, dejando de lado todo aspecto relacionado al hardware e implementación.

- **Síntesis lógica (*Design Compiler*):** Una vez que se tenga un diseño satisfactorio junto con una simulación exitosa, se procede a la etapa de síntesis lógica, en donde se sintetiza el circuito. Esto significa que el código VHDL diseñado que acaba de ser simulado, es mapeado hacia una combinación de compuertas lógicas de una tecnología de diseño en específico. En esta etapa, una de las entradas necesarias es un grupo de restricciones que el programa de síntesis (en este caso Design Compiler) debe cumplir, en donde los criterios más importantes son el desempeño y tiempos de retardo. [2]. El circuito puede sufrir varias iteraciones de implementación en esta etapa para cumplir

con las restriccionse colocadas, pero si no se logran cumplir con modificaciones de la síntesis, será necesario analizar el diseño original para arreglar errores de arquitectura.

La herramienta más utilizada para llevar a cabo la síntesis lógica, la provee *Synopsys* y se llama *Design compiler*. En este trabajo también se hará uso de la herramienta mencionada, junto a las librerías educativas aportadas también por *Synopsys*, que suelen ser usadas para que estudiantes aprendan a realizar el proceso de diseño VLSI.

- **Síntesis física (*IC Compiler II*):** La síntesis física es similar a la síntesis lógica, en cuanto a que intenta mejorar criterios como desempeño y área que utilizarán los componentes, pero la gran diferencia es que ésta brinda un flujo de diseño convergente que toma en consideración rutas más óptimas y por lo tanto requiere menos iteraciones [3].

  Esta etapa es muy importante porque considera aspectos de tiempos de retardo que se originan en el aspecto físico, ya que algunos tracks o cables largos inducen más retardo en el circuito ya que la corriente tarda más en pasar por ellos que el resto de conexiones. También usa librerías al igual que la ´sintesis lógica solamente que éstas incluyen el modelo físico del transistor a utilizar, y el formato del archivo resultante es .gds.

- **Análisis de sincronización estática (*PrimeTime*):**

  Luego de tener un diseño físico originado del VHDL diseñado como primer paso del flujo, se puede proceder a validar el desempeño de los tiempos de retardo comprobando todos los posibles caminos dentro del circuito por violaciones de restricciones. El análisis de sincronización estática solamente verifica tiempos y no funcionalidad.

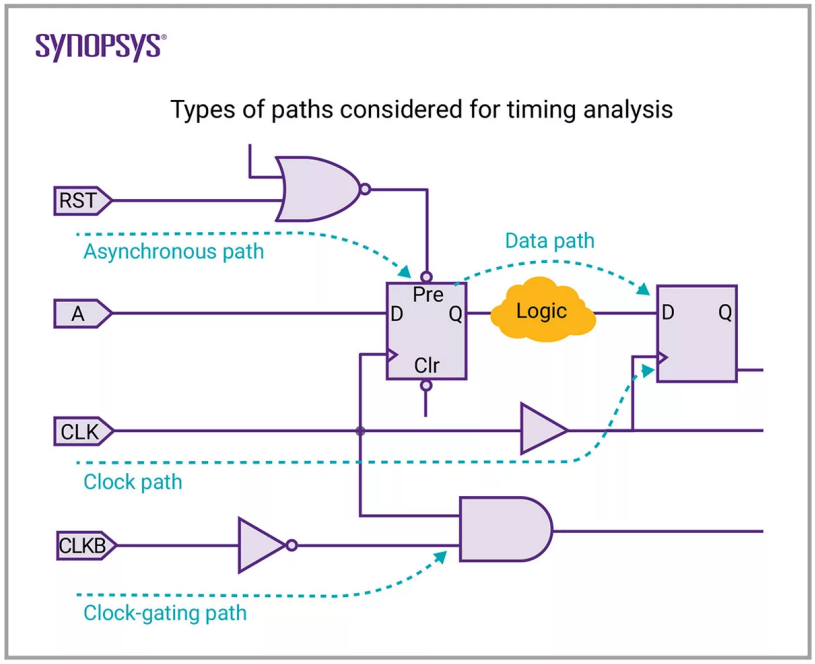



Figura 6: Tiempos de retardo importantes en el análisis de sincronización estática con *PrimeTime*

Como se puede observar en la Figura 6, los tiempos de retardo principales que se

miden en el análisis son de las siguientes 4 rutas (en orden de arriba para abajo): Ruta asíncrona, ruta de datos, ruta de reloj y ruta de compuerta de reloj [4].

- **Verificación formal (*Formality*):**

  La Verificación formal se refiere al proceso de establecer equivalencia funcional entre dos diseños escritos en algún lenguaje de descriptivo de hardware. La verificación formal se realiza entre el diseño original de VHDL y el circuito generado en la síntesis física. Esta verificación se realiza para que el modelo RTL sirva de referencia para comprobar la funcionalidad de la netlist .gds creada [5].

- **Generación de patrones automáticos de prueba (*TetraMAX*):**

  En esta sección se tiene como objetivo medir diferentes parámetros del circuito (desempeño, potencia, tiempos) en conjunto usando vectores en sus entradas. Los vectores óptimos para hacer estas mediciones no son aleatorios, sino que existen algoritmos que los generan de la manera más eficiente posible, es por eso que se usa software especializado como *TetraMAX* para generar los patrones de prueba.

## 6.4. Otras herramientas de *Synopsys*

*Synopsys* es la empresa encargada de diseñar los programas encargados de cumplir con todo el flujo de diseño de nanotecnología, y a continuación se explica la funcionalidad de los programas utilizados.

- ***Hercules*:** *Hercules* es el programa que antes se usaba para correr runsets de corrección de errores y validación de reglas de diseño, ahora el programa equivalente se llama *ICV*.

- ***IC Validator (ICV)*:** *ICV* es el programa que reemplaza a *Hercules* para la validación de errores de diseño y corrección de éstos a través de runsets.

- ***IC Compiler 2 (ICC 2)*:** *IC Compiler 2* es el más complicado, puesto que se encarga de visualizar el diseño del chip, y también desde su interfaz se pueden correr varias pruebas de verificación de diseño, y reglas de fabricación.

## 6.5. Formatos de archivos importantes

- ***Verilog (.v, .vhd)*:** La extensión de archivo .v se refiere a archivos de código descriptor de hardware (hdl), es decir que representa circuitos físicos pero a nivel de estructura y lógica. Este lenguaje ha crecido en popularidad por el mayor uso del software de Automatización de Diseño Electrónico (EDA) con el paso del tiempo. Existen 2 tipos principales de código: Behavioral, y structural. En el contexto de diseño VLSI, se utiliza para describir el código inicial que será diseñado y fabricado. A pesar de que el formato verilog sea capaz de describir a detalle los componentes a nivel de transistor, al inicio del proceso éste no es tan detallado e incluso puede solamente describir el circuito a nivel de lógica.

- ***Librerías de Celda de Estándar Digital (.db)*:** Los archivos de extensión .db, son usados por muchas aplicaciones para almacenar datos en formatos que varían ligeramente según el uso y software. Para el flujo de diseño nanoelectrónico, ésta extensión es usada para almacenar la información de las librerías a utilizar, es decir que contiene los atributos de las celdas y distintos elementos de cada tecnología y las relaciones necesarias entre la información para describir satisfactoriamente un circuito.

- ***Restricciones de Diseño (.sdc)*:** El archivo con extensión .sdc indica las restricciones de diseño que se tendrán a la hora de concretar el diseño en un circuito físico en la síntesis. Usualmente estos archivos son proveídos por el fabricante pues es quien conoce las restricciones y capacidades de su maquinaria.

- ***Reportes y registros (.rpt, .log)*:** Todos los programas utilizados a lo largo de este trabajo de investigación, realizan reportes y registros, los cuales se guardan en archivos con extensión .rpt y .log respectivamente. En los archivos de reporte se pueden encontrar resúmenes de información de la acción recién realizada por el programa y parámetros de los circuitos, como potencia, delay, corriente, etc. En cuanto a los registros, almacenan las respuestas de cada comando, es decir, si se realizaron exitosamente, códigos de error, retroalimentación, etc.

- ***Netlist (.ddc, .v, .sp)*:** Una netlist en diseño electrónico es una descripción de cómo se conecta un circuito electrónico. Es una lista de los componentes electrónicos de un circuito y una lista de los nodos a los que están vinculados, en su forma más básica. Un grupo de dos o más componentes conectados se denomina red (red), y un grupo de redes se entiende como una lista de redes (netlist).

- ***TluPlus (.map)*:** Un archivo TLUPlus contiene información acerca del modelo parasítico de los componentes para la etapa de extracción de parásitos. También contiene información de resistencias y restricciones sobre todas las capas, incluidas las capas activas, polivinílicas y metálicas. Principalmente es usado por herramientas de extracción de parásitos, como StarRC.

- ***Archivo de Tecnología (.tf, .lef)*:** Los archivos de extensión .tf y .lef juegan un papel importante en el diseño de circuitos integrados, ya que contienen información esencial que los programas de diseño utilizan para simular y optimizar el rendimiento de los dispositivos electrónicos. Los archivos .tf proporcionan información sobre las características de transferencia de un dispositivo, como su impedancia y su ganancia en función de la frecuencia, mientras que los archivos .lef contienen información sobre la geometría y la ubicación de los componentes en un diseño de circuito integrado. Esta información es utilizada por los programas de diseño para ayudar a garantizar que el rendimiento del dispositivo cumpla con los requisitos especificados y para identificar posibles problemas en el diseño.

- ***Sistema de Base de Datos Gráfica (.gds)*:** Los archivos .gds son archivos de diseño de circuito que contienen información sobre la geometría de un diseño de circuito integrado. Estos archivos se utilizan en el campo de la electrónica y el diseño de circuitos integrados para almacenar y transferir información sobre la ubicación y las dimensiones de los componentes en un diseño de circuito. Los archivos .gds son utilizados por programas de diseño de circuitos integrados para ayudar a simular y optimizar el rendimiento de los dispositivos electrónicos.

- ***Standard Parasitic Exchange Format (.spef)*:** Los archivos .spef son archivos que contienen información sobre la impedancia y la capacitancia de un diseño de circuito integrado. Estos archivos se utilizan en el campo de la electrónica y el diseño de circuitos integrados para ayudar a simular y optimizar el rendimiento de los dispositivos electrónicos. Los archivos .spef contienen información sobre la impedancia y la capacitancia de los diferentes componentes de un diseño de circuito en función de la frecuencia, lo que permite a los programas de diseño simular el comportamiento del circuito en diferentes condiciones. Esta información es utilizada para ayudar a garantizar que el diseño cumpla con los requisitos especificados y para identificar posibles problemas en el diseño.

- ***Tool Command Language (.tcl)*:** Los archivos de extensión .tcl son archivos de script que utilizan el lenguaje TCL (Tool Command Language). TCL es un lenguaje de programación interpretado que se utiliza en aplicaciones de diversos campos, como la automatización de pruebas, la ingeniería de software y el desarrollo web. Los archivos .tcl contienen instrucciones escritas en el lenguaje TCL que pueden ser ejecutadas por un intérprete para realizar tareas específicas. Por ejemplo, un archivo .tcl puede contener instrucciones para realizar pruebas en una aplicación o para generar un informe de salida en un formato determinado. Los archivos .tcl se utilizan comúnmente en el campo de la electrónica y el diseño de circuitos integrados para automatizar tareas de diseño y simulación.

## 6.6. Archivos requeridos por el fabricante *TSMC*

- GDSII (Formato para el Intercambio de Datos de Diseño de Circuitos Integrados): Este es un formato de archivo que se utiliza para describir el diseño de un chip en términos geométricos. Incluye información sobre las formas y tamaños de las diferentes capas y características del chip, así como sus posiciones en el chip.

- LEF (Formato de Intercambio de Bibliotecas): Este es un formato de archivo que se utiliza para describir el diseño de un chip en términos de celdas y sus ubicaciones. Incluye información sobre las celdas que se utilizan en el diseño, así como sus tamaños y formas.

- LIB: Este es un formato de archivo que se utiliza para describir las características eléctricas de las celdas del chip. Incluye información sobre los pines de entrada y salida de las celdas, así como sus características de tiempo y energía.

- Netlist de SPICE: Este es un archivo que describe la conectividad del circuito y el comportamiento del chip. Incluye información sobre los componentes que conforman el circuito, así como las conexiones entre ellos.

- Archivo de restricciones de tiempo: Este es un archivo que especifica los requisitos de tiempo para el chip. Incluye información sobre el rendimiento deseado del chip, como la frecuencia máxima del reloj y los tiempos de configuración y mantenimiento requeridos para las entradas y salidas.

CAPÍTULO 7

# Instalación de software necesario

## 7.1. Instalación de *PrimeTime*

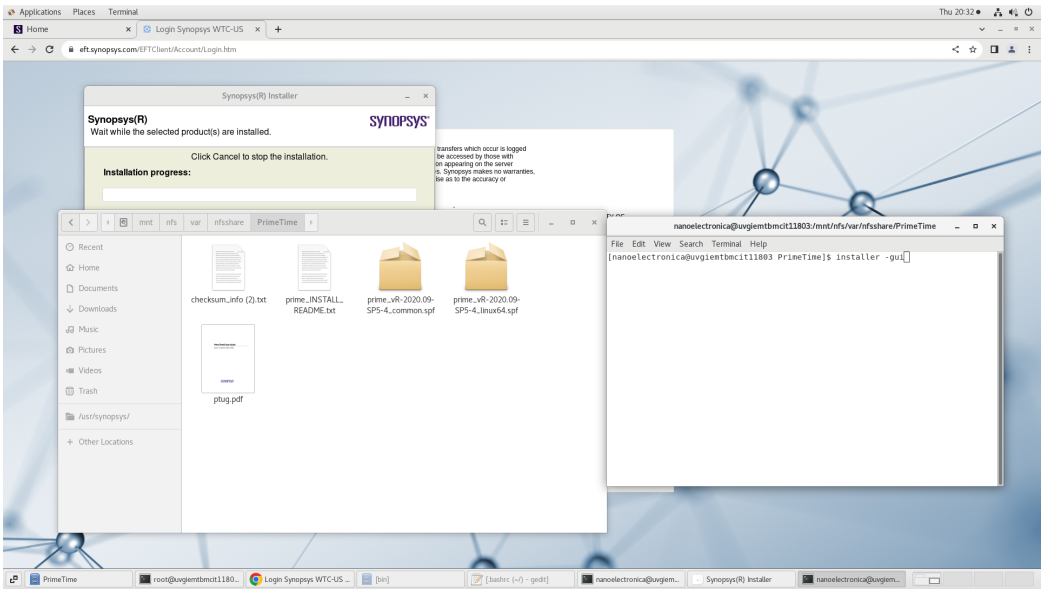



Figura 7: Ejecución del Installer de Synopsys en CMD

El primer paso para la instalación de un programa de Synopsys es invocar la interfaz gráfica con el comando que se muestra en la Figura 7:

```
% installer -gui
```

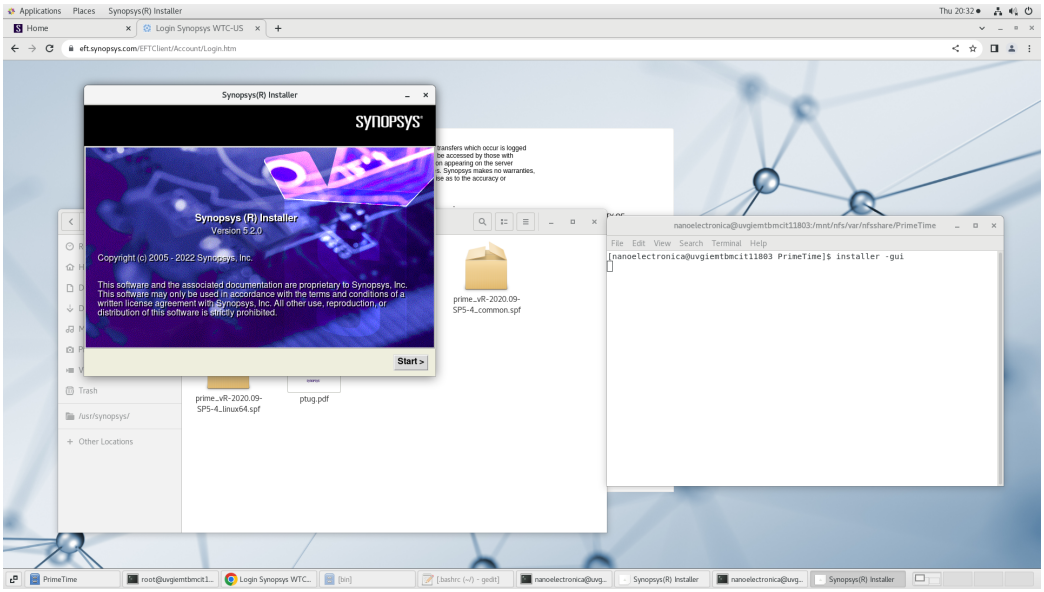



Figura 8: Ventana del Synopsys Installer

Aparecerá una ventana que guiará a través de la instalación de los programas (Figura 8). Hacer click en ”“Start”.

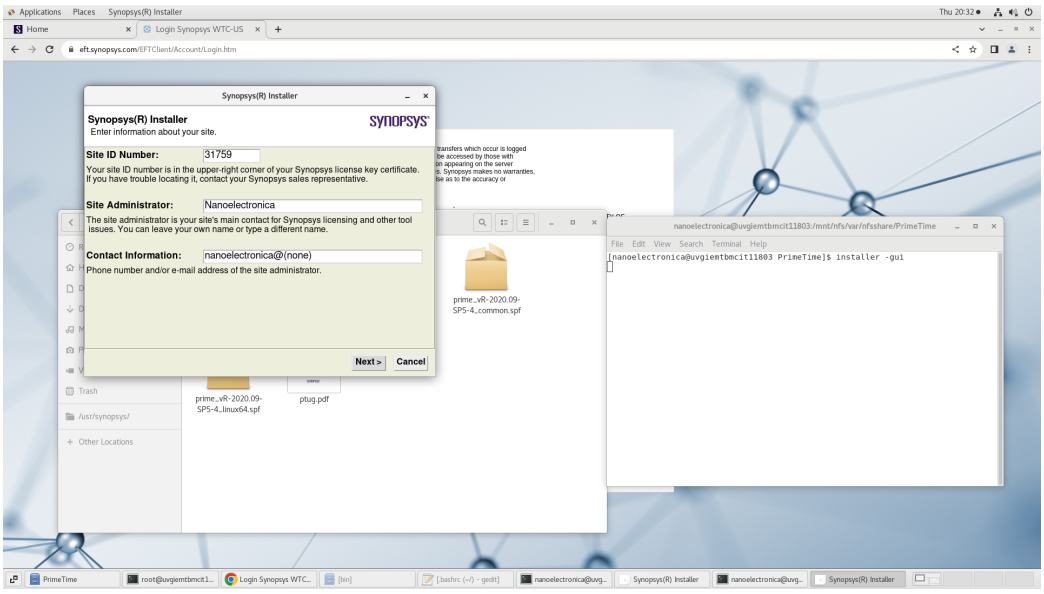



Figura 9: Información del administrador del usuario de Synopsys

Verificar la información y hacer click en “Next” en las siguientes ventanas (9 - 13).

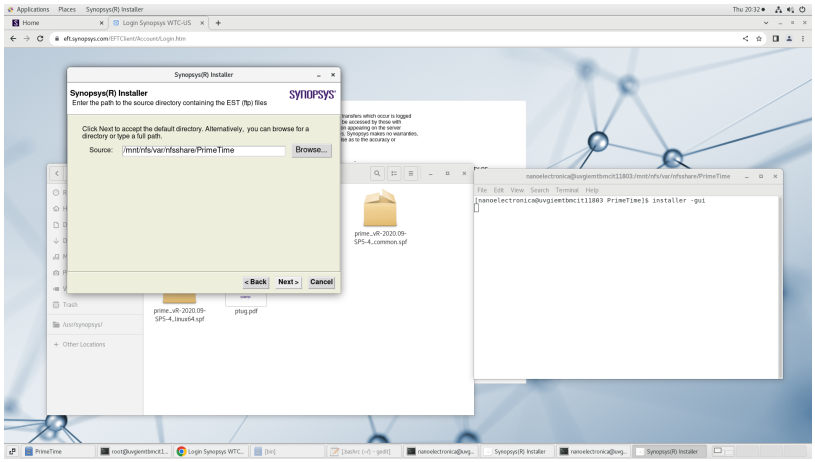



Figura 10: Directorio de ubicación de archivos EST de *PrimeTime*

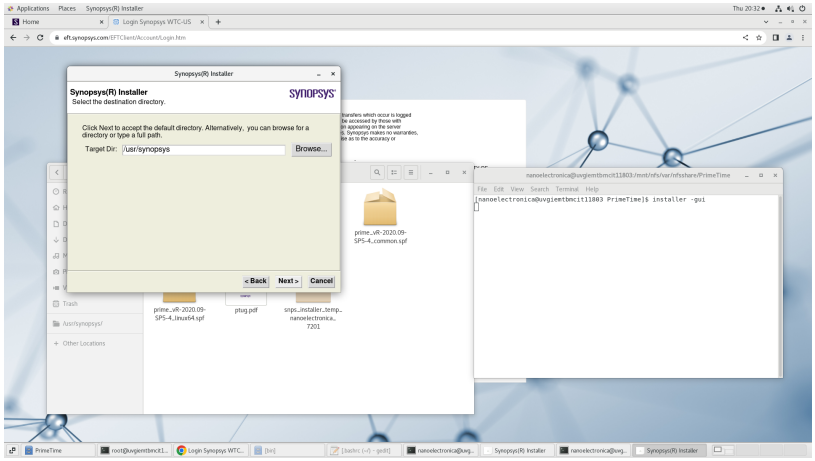



Figura 11: Directorio de instalación de *PrimeTime*

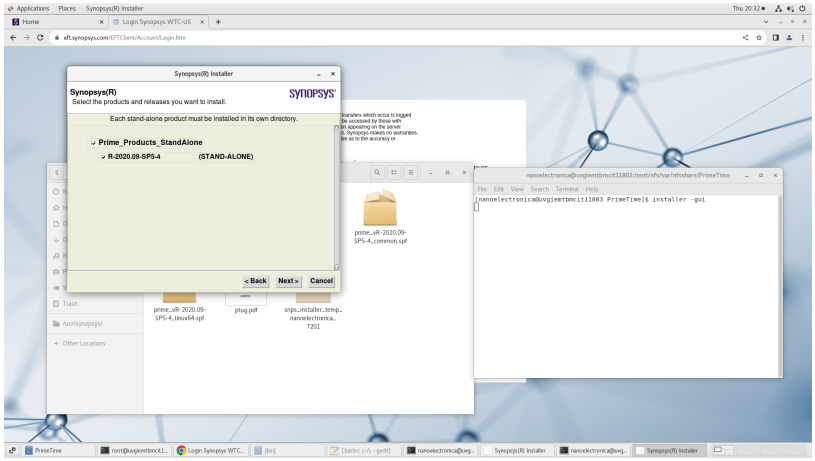



Figura 12: Selección de productos relacionados a *PrimeTime*

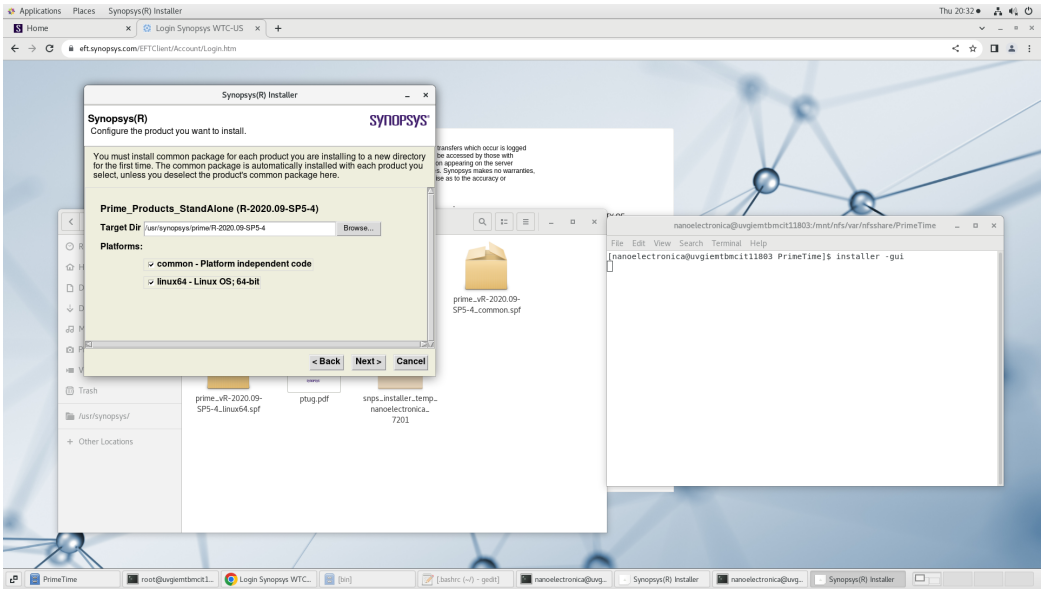



Figura 13: Configuración del producto a instalar

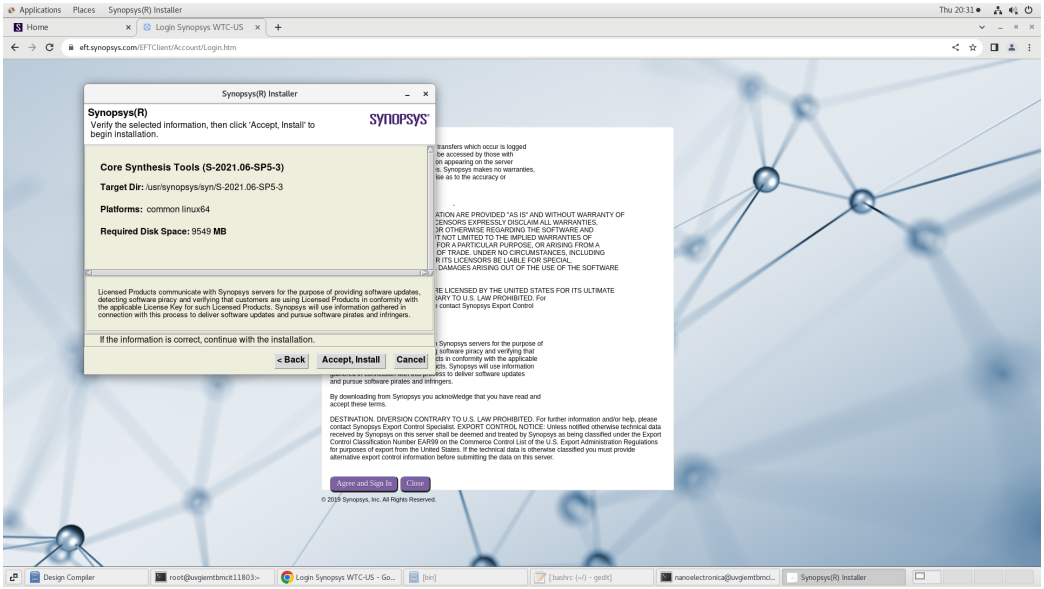



Figura 14: Verificación antes de instalación

En la verificación, debe aparecer lo mismo a la Figura 14, y se da click a "Accept, Install". Inmediatamente aparece una ventana de progreso de instalación como se muestra en la Figura 15.

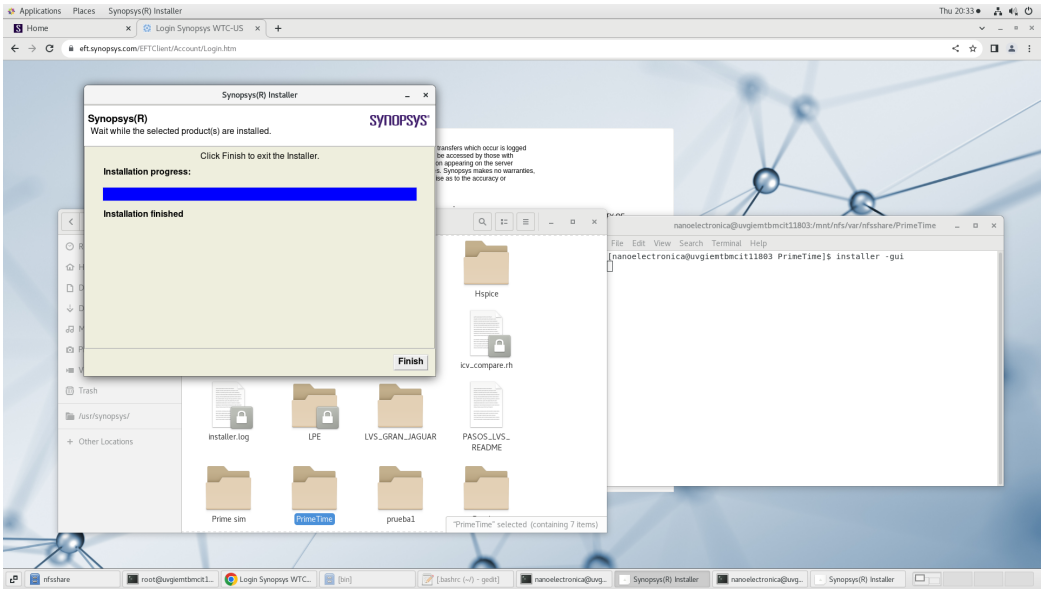



Figura 15: Barra de progreso de instalación de *PrimeTime*

Una vez terminado el progreso de la instalación, se mostrarán algunas notas acerca de nuevas versiones, hacer click en "Dismiss" (Figura 16).

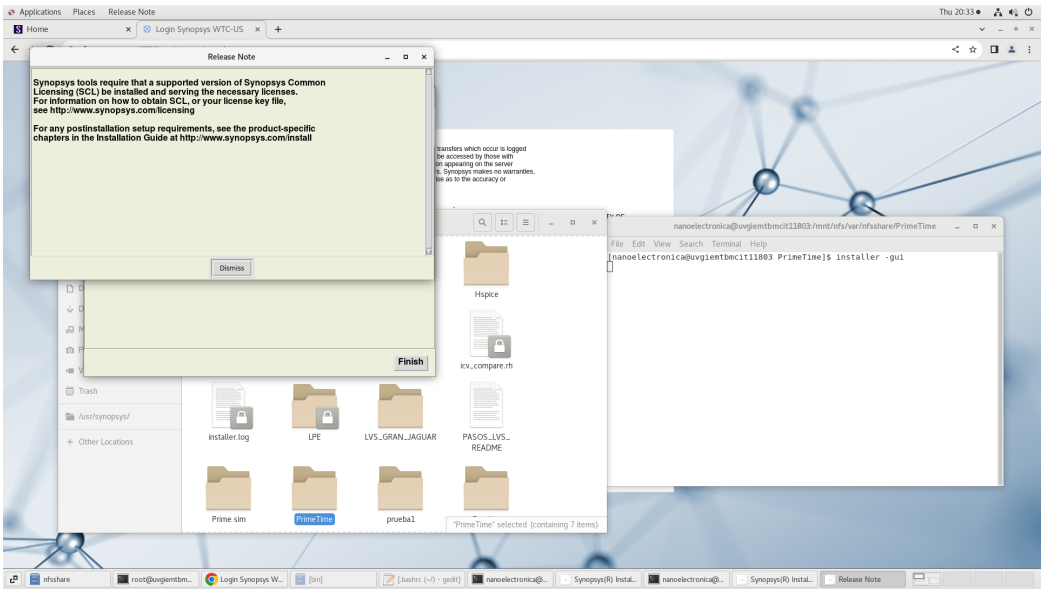



Figura 16: Notas de actualización de *PrimeTime*

## 7.2. Instalación de *TetraMAX*

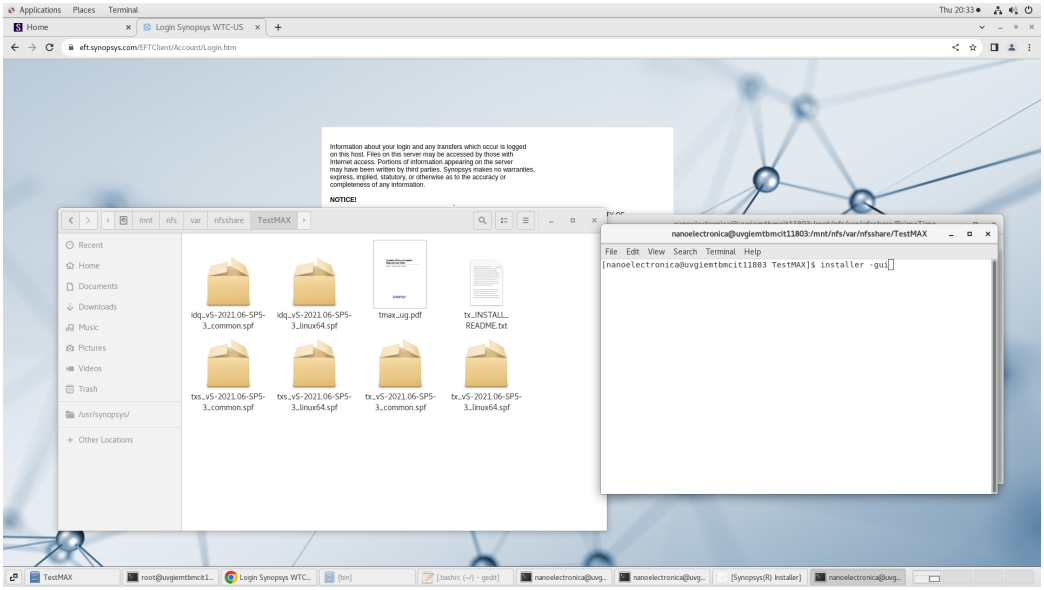



Figura 17: Ejecución del Installer de Synopsys en CMD

Al igual que con *PrimeTime*, el primer paso para la instalación de un programa de Synopsys es invocar la interfaz gráfica con el comando que se muestra en la Figura 17:

```
% installer -gui
```

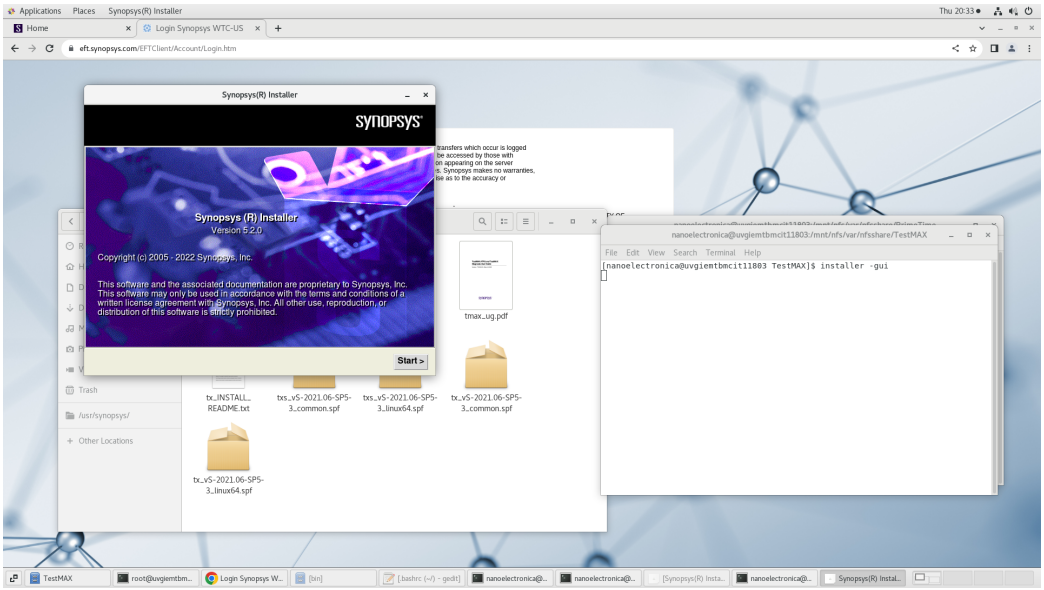



Figura 18: Ventana del Synopsys Installer

Aparecerá una ventana que guiará a través de la instalación de los programas (Figura 18). Hacer click en “Start”.

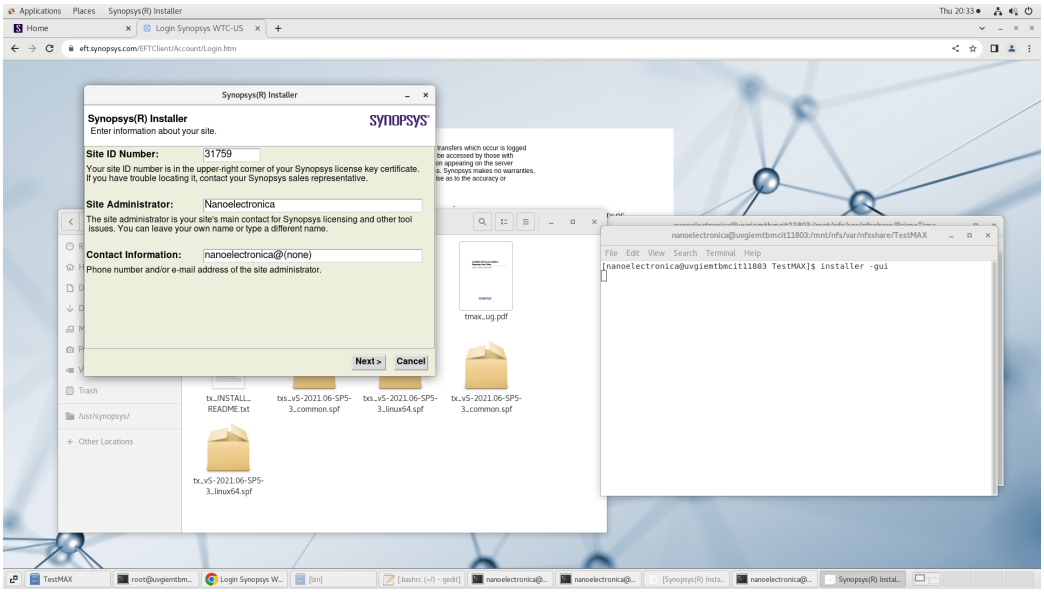



Figura 19: Información del administrador del usuario de Synopsys

Verificar la información y hacer click en “Next” a lo largo de las siguientes Figuras (19 - 23).

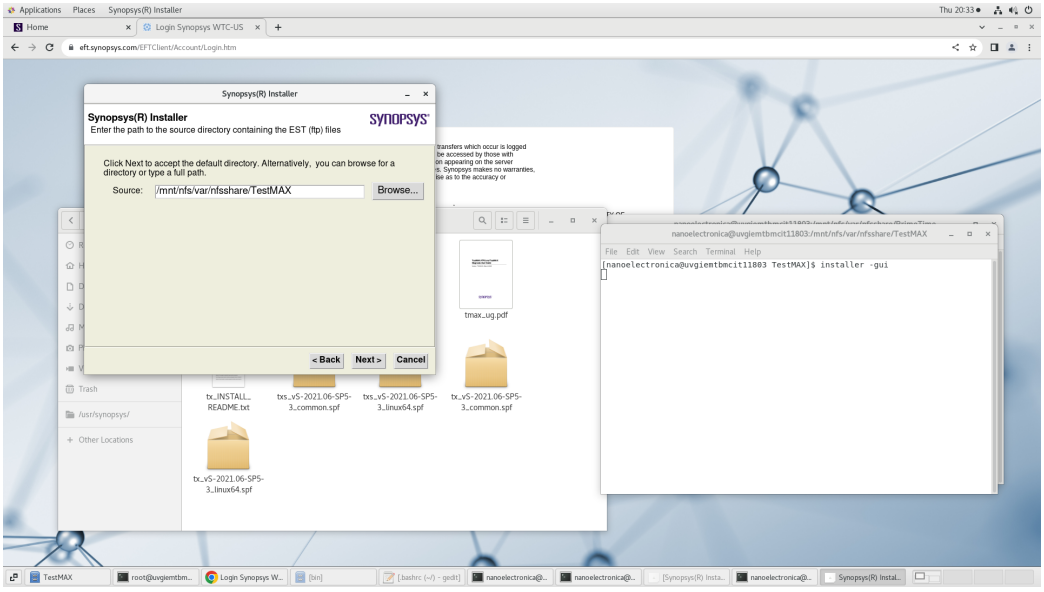



Figura 20: Directorio de ubicación de archivos EST de *TetraMAX*

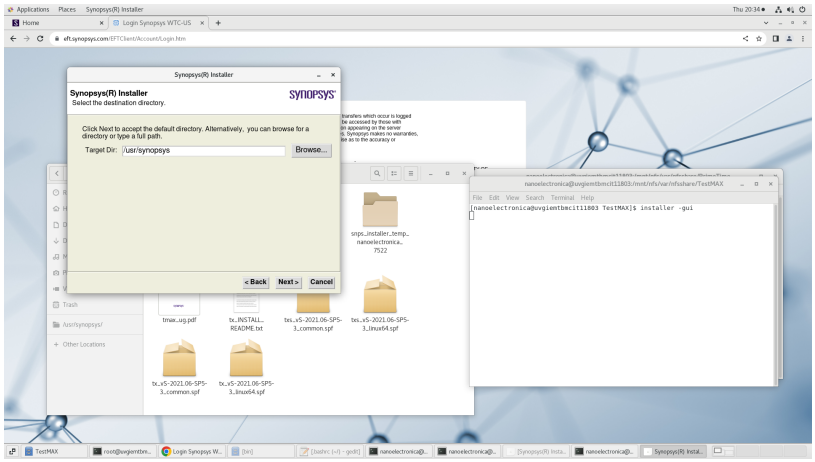



Figura 21: Directorio de instalación de *TetraMAX*

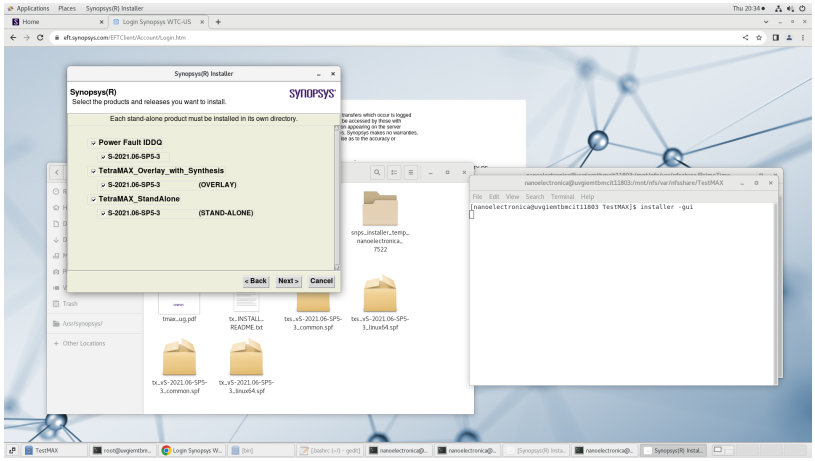



Figura 22: Selección de productos relacionados a *TetraMAX*

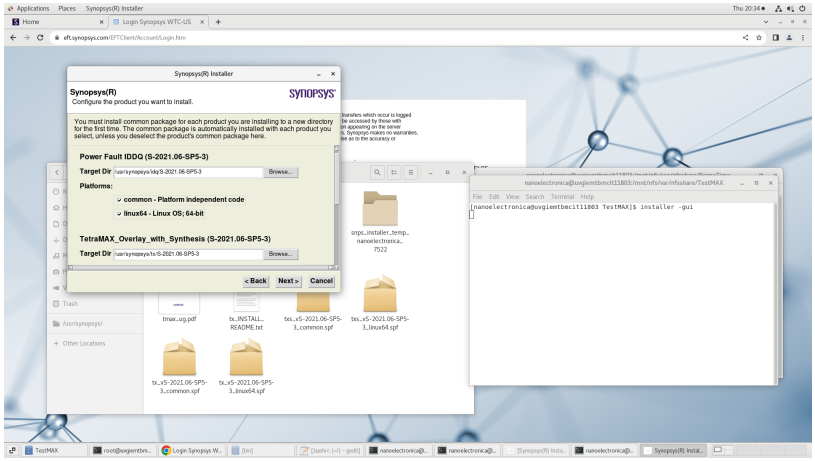



Figura 23: Configuración del producto a instalar

En la verificación, debe aparecer lo mismo a la Figura 14, y se da click a “Accept, Install”. Inmediatamente aparece una ventana de progreso de instalación como se muestra en la Figura 15, solamente que esta vez está haciendo referencia a *TetraMAX*.

Una vez terminado el progreso de la instalación, se mostrarán algunas notas acerca de nuevas versiones, hacer click en ’Dismiss’. No se debe olvidar añadir los programas instalados al PATH del bash en Linux.

# CAPÍTULO 8

## Problemáticas con el flujo tradicional de diseño usado por estudiantes UVG

En años interiores, los estudiantes de UVG han realizado flujos de diseño *VLSI* con el objetivo de fabricar un circuito integrado a través del apoyo de alguna empresa fabricante de semiconductores. En el caso de UVG, se convino con *TSMC* para usar sus librerías, generar los archivos de salida y trasladarlos a *TSMC* para que el chip sea fabricado.

Esta metodología, a pesar de haber iniciado como proyecto del departamento en el 2019, actualmente está cumpliendo su 4to año de implementación y aún no se han logrado resultados finales significativos. Si bien se han logrado resultados paraciales o de investigación, es decir que se han obtenido mejoras que permiten el avance del proyecto y nuevas ideas para su implementación, éstos no necesariamente garantizan que se esté cumpliendo con los requisitos del fabricante.

En este trabajo se le llama 'flujo tradicional' al conjunto de pasos que han tomado los estudiantes de UVG para trabajar el proyecto del nanochip. En contraste con los flujos propuestos por *Synopsys*, el flujo tradicional presenta cuatro principales problemáticas: No se tiene una métrica de referencia, no existe buena planificación detrás del archivo RTL, la linealidad del flujo y por último, que requiere un aprendizaje a "prueba y error"de parte de los estudiantes en vez de un aprendizaje organizado.

La primera problemática se refiere a la métrica del proceso, es decir, ¿De qué manera se miden los avances del flujo? ¿Qué información indica que se está en un punto específico del proceso?. En el flujo tradicional, la respuesta a estas preguntas se ha encontrado en cada herramienta utilizada para cada paso del proceso, por ejemplo, Design Compiler es síntesis lógica, ICC2 es síntesis física, etc. El problema es que los flujos utilizados en la industria no son tan simples, ya que cada herramienta no solamente es capaz de realizar un paso en el proceso, sino que puede ir más allá incluso más de una vez. Por lo que entra a la conversación la posibilidad de flujos iterativos y que no solamente se ejecutan una vez.

Como segundo problema del flujo tradicional, se encuentra la mala planificación del diseño RTL, puesto que ha sido elaborado como una nube combinacional no optimizada, que no toma en consideración cosas como: diseño modular, jerarquía entre bloques, lenguaje específico para síntesis, y el despliegue de los caracteres para ser leídos fácilmente en una interfaz. Es buena práctica considerar éstos factores en un diseño RTL desde el inicio, ya que evita errores que puedan surgir más adelante en el flujo de diseño y no sean tan evidentes inicialmente.

Entre las bases fundamentales del flujo tradicional se asume una idea errónea, ésta es: "El flujo de diseño es un proceso lineal". Ésta suposición nace de los procesos improvisados que se han llevado a cabo de parte de los estudiantes, ya que al usar cierta herramienta y obtener archivos resultantes, asumen que en ese punto finalizó el uso de esa herramienta y se debe pasar a trabajar a la siguiente, lo cual raramente ocurre en los flujos utilizados en la industria. Es muy frecuente en los flujos de diseño que una herramienta se use más de una vez luego de que haya exportado los archivos, porque muchos de los procesos son iterativos y las herramientas de Synopsys no solamente trabajan en un mismo entorno sino que incluso pueden trabajar simultáneamente.

Por último, la última de las problemáticas encontradas con el flujo tradicional, es que se basa en conocimiento no necesariamente bien fundamentado de parte de los estudiantes, puesto que éste se obtiene de manera experimental y se traslada de promoción a promoción sin que sea requisito leer la documentación y recursos de las herramientas. Es posible que esto se deba a la poca accesibilidad de los recursos y documentación.

Una vez desplegadas las problemáticas, es posible observar cómo se relacionan entre ellas, por ejemplo, el traslado de promoción a promoción de la información, hace que sea más difícil desprenderse de las suposiciones erróneas que se han plantado en un inicio.

El presente trabajo de investigación junto a su propuesta principal, plantea solucionar o por lo menos traer a luz posibles respuestas a las problemáticas indicadas anteriormente. Éstas soluciones se verán de la siguiente manera:

Se leyó la documentación y estudiaron los cursos en la plataforma de SolvNet, los cuales brindan información mucho más completa de la que se tenía empíricamente entre promociones acerca del uso de las herramientas de *Synopsys*. Con esta información, se puede iniciar el proceso de flujo de diseño con bases mejor fundamentadas y sin suposiciones que más adelante del proceso provocarán errores.

Algunos de los errores que se espera corregir con la alternativa propuesta, son los errores de densidad de las capas de metales, errores de ruteo en la síntesis física y errores de librerías que se provocan por la falta de comprensión de la funcionalidad de cada archivo de librería brindado tanto por el fabricante como por *Synopsys*

# CAPÍTULO 9

# Propuesta del flujo de diseño de *Synopsys*

## 9.1. *RTL*: Creación de un buen diseño

### 9.1.1. Definición de requisitos y especificaciones

Se comienza con una comprensión clara de los requisitos del sistema, incluyendo sus funcionalidades, rendimiento, consumo de energía y restricciones de área, que son documentadas detalladamente.

### 9.1.2. Selección de la arquitectura

La arquitectura del sistema es seleccionada, incluyendo la CPU, configuración de memoria, interfaces, y otros componentes principales. Un estudio de factibilidad es realizado para asegurar la viabilidad de la arquitectura elegida.

### 9.1.3. Diseño de alto nivel

Modelos de alto nivel del sistema son creados usando herramientas de modelado y simulación, y se validan mediante simulaciones para verificar la arquitectura y los algoritmos propuestos.

### 9.1.4. Desarrollo de RTL

El código RTL es escrito en un lenguaje de descripción de hardware como Verilog o VHDL, asegurando que sea legible, bien organizado y adecuadamente comentado.

### 9.1.5. Validación funcional y simulación

La funcionalidad del diseño RTL es validada a través de extensas simulaciones, empleando casos de prueba y bancos de pruebas para cubrir todos los aspectos del diseño.

### 9.1.6. Optimización

El diseño es optimizado en términos de rendimiento, área y consumo de energía, realizándose ajustes para mejorar la eficiencia sin comprometer la funcionalidad.

### 9.1.7. Verificación y linting

Herramientas de verificación formal son utilizadas para asegurar que el diseño cumple con las especificaciones, y se realiza linting de RTL para identificar y corregir problemas en el estilo de codificación y posibles errores.

### 9.1.8. Síntesis y análisis de tiempos estáticos

El diseño RTL es sintetizado a un netlist, y se lleva a cabo un análisis de tiempos estáticos para garantizar el cumplimiento de los requisitos de temporización.

### 9.1.9. Iteraciones de diseño

Basado en los resultados de la síntesis y el análisis de tiempos, se realizan ajustes iterativos en el diseño RTL, repitiéndose la simulación y verificación tras cada cambio significativo.

### 9.1.10. Preparación para la implementación

El diseño se prepara para las etapas subsiguientes, como el diseño de la disposición y la fabricación, asegurando la compatibilidad con las herramientas y procesos de fabricación.

## 9.2. *VCS*: Simulación lógica

### Objetivo

El objetivo de esta etapa de diseño, es comprobar el correcto funcionamiento lógico de un circuito descrito en un archivo Verilog por medio de la simulación y observación de sus entradas y salidas (definidas en el archivo Testbench).

### Archivos de entrada

- circuito.v
- circuito_testbench.v

### Archivos de salida

- ejecutable de simulación

### Procedimiento

Como primer paso, es necesario ubicarse en el directorio de trabajo que contenga los archivos de entrada. Una vez en el directorio, deb ejecutarse el siguiente comando:

```
% vcs circuito_testbench.v circuito.v -gui -debug_acc+all -full64
```

La función del comando anterior, es compilar una simulación del circuito descrito en "circuito.v" utilizando como entradas y salidas aquellas colocadas en "circuito_testbench.v", obteniendo como salida un archivo ejecutable de simulación. La opción -gui habilita el dve, y el resto de opciones son necesarias para que el programa funcione con un procesador de 64 bits. Luego de la compilación, se puede abrir DVE con el siguiente comando:

```
% dve -full64
```

Una vez abierto dve, se debe abrir la ventana de configuración de la simulación desde "Simulator > Setup". Esta se muestra en la Figura 24. Desde aquí se puede abrir el ejecutable de simulación en "Simulator executable".

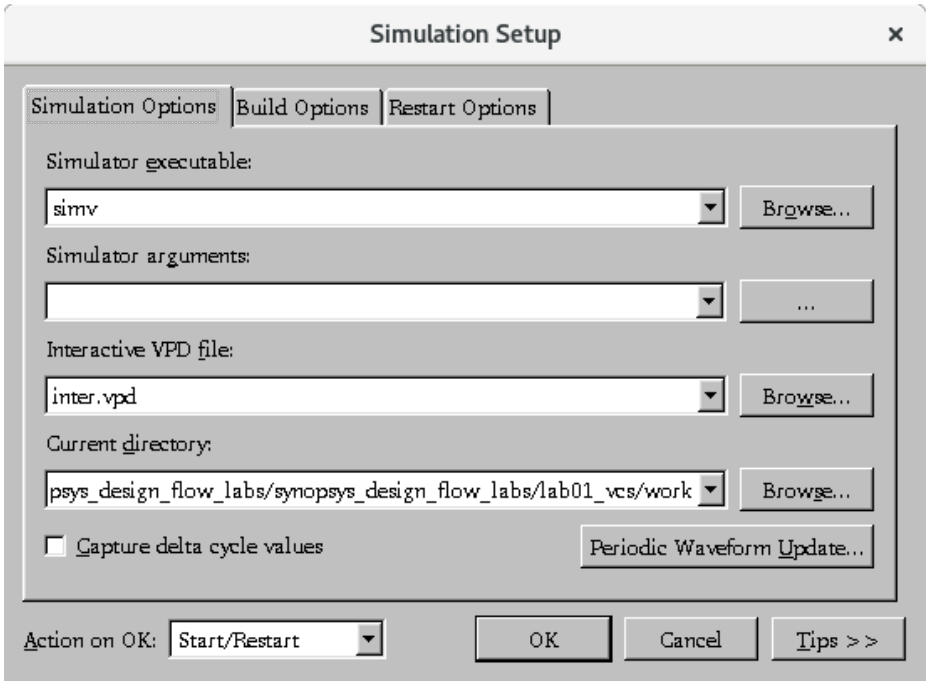



Figura 24: Ventana de configuración de simulación

Luego de hacer click en "OK", aparecerán disponibles varias vistas de la simulación, desde el código hasta la vista de señales. Es importante notar que las características del circuito y la simulación están enlazadas entre todas las vistas disponibles, por lo que se facilita el proceso de identificación de errores o simplemente de simulación.

## 9.3. *Design Compiler*:

### Archivos de entrada

- Diseño RTL en formato VHDL
- Librerías lógicas y genéricas (.db)
- Librerías físicas
- Datos de tecnología (.tf)
- Pendiente

### Archivos de salida

- reportes (.rpt)
- registro.log
- Base de datos de diseño (formato interno de Synopsys)

**Procedimiento**

Design Compiler cuenta con dos modos principales, NDM y Milkyway. Para la realización sin errores de algunos comandos referentes a librerías físicas, es necesario activar el modo NDM, ya que por defecto cuando se ejecuta el programa con el siguiente comando se inicia en el modo Milkyway:

```
>dc_shell -topo -gui
```

En este modo, se mostrará el error (CMD-005) con algunos comandos que necesitan invocar a ICC2 para crear y manejar librerías físicas

Antes de ejecutar el programa, es necesario ubicarse en la carpeta de trabajo deseada, y en el caso que se deseen configuraciones iniciales, éstas deben estar indicadas en el archivo ".synopsys_dc.setup" ubicado en el directorio donde será ejecutado el programa. Se inicia una ventana de *Design Compiler* con los siguientes comandos:

```
% dc_shell
dc_shell> start_gui
```

Dentro del archivo ".synopsys_dc.setup" se deben indicar las librerías a utilizar de la siguiente manera:

```

set target_library  {../ref/db_nldm/saed14rvt_tt0p8v25c.db   }
set link_library    {* ../ref/db_nldm/saed14rvt_tt0p8v25c.db }
```

El procedimiento a seguir para compilar un diseño para síntesis lógica, se puede escribir a través de comandos que se ejecutan secuencialmente en un script, a continuación se presenta un script de ejemplo y posteriormente se explicará detalladamente para tener un mejor entendimiento del proceso de síntseis lógica (se cambiarán los nombres de Johnson por Circuito.v y se adaptará a la tesis):

```

# these can be also set in .synopsys_dc.setup file in working directory
# always keep the asterisk in link_library
set target_library  {../ref/db_nldm/saed14rvt_tt0p8v25c.db   }
set link_library    {* ../ref/db_nldm/saed14rvt_tt0p8v25c.db }



read_verilog  ../source/johnson.v

## read command can be replaced with:
#analyze -library WORK -format verilog {../source/johnson.v}
#elaborate Johnson_count -architecture verilog -library DEFAULT

link
check_design

read_sdc ../source/johnson.sdc
compile -exact_map

report_area  > ../results/rpt.area.report
report_constraint > ../results/rpt.constraints.report
```

```
23 report_timing > ../results/rpt.timing.report
24 report_qor > ../results/rpt.qor.report
25
26 change_names -rule verilog
27 write -hierarchy -format verilog -output ../results/johnson_compiled.v
28 write -hierarchy -format ddc  -output ../results/johnson_compiled.ddc
29
30 exit
```

El primer paso a realizar por el programa, es leer el diseño del circuito descrito en un archivo verilog, usando el comando "read_verilog" como se muestra en la línea 9 del script. Una vez leído el verilog, se realiza el enlace en la línea 15 e inmediatamente se revisa usando "check_design"

## 9.4. *IC Compiler II*: Diseño físico

### Archivos de entrada

- Netlist: Formato SPICE o Verilog.
- Archivo de diseño RTL: Lenguajes de descripción de hardware como Verilog o VHDL.
- Archivos de restricciones de diseño:
  - Restricciones de tiempo: Formato SDC (Synopsys Design Constraints).
  - Restricciones físicas: Formatos UPF (Unified Power Format) o CPF (Common Power Format).
- Archivos de tecnología y bibliotecas:
  - Información de proceso de fabricación.
  - Características eléctricas de los componentes.
- Archivos de datos de celdas estándar y bibliotecas de IP: Información sobre celdas estándar y bloques de propiedad intelectual.
- Archivos de diseño de referencia: Diseños anteriores o plantillas.
- Archivos de máscaras de capas: Definición de capas y características geométricas para fabricación.

### Archivos de salida

- GDSII o OASIS: Formatos de archivo de máscaras utilizados para la fabricación de semiconductores.
- LEF (Library Exchange Format): Archivo que contiene información sobre las celdas utilizadas en el diseño.

- DEF (Design Exchange Format): Archivo que describe la ubicación y conexión de las celdas y rutas en el diseño.
- Reportes de síntesis: Incluyen detalles del rendimiento, área y consumo de energía del diseño.
- Reportes de análisis de tiempos: Proporcionan información detallada sobre los tiempos de propagación y las violaciones de temporización.
- Archivos de verificación DRC/LVS: Contienen resultados de verificaciones de Reglas de Diseño (DRC) y Verificación de Layout frente a Esquemático (LVS).
- Archivos de extractores de parásitos: Ofrecen información sobre los efectos parásitos en el diseño.

#### Procedimiento

La síntesis física en *IC Compiler II* es un proceso que convierte un netlist lógico en una representación física del circuito en un chip de silicio. Este proceso incluye varios pasos y herramientas, y puede variar según las necesidades del diseño y las opciones disponibles en el software.

1. Importar el netlist lógico generado en la síntesis lógica del diseño.
2. Especificar las restricciones de diseño, como el tamaño del chip, las necesidades de enfriamiento y la densidad de componentes.
3. Ejecutar la síntesis física, que optimiza el diseño y lo divide en bloques que pueden ser colocados en el chip.
4. Ejecutar el proceso de colocación, que asigna posiciones a los bloques en el chip de manera que quepan todos los bloques (cells) en el floorplan establecido.
5. Ejecutar el proceso de ruteado, que conecta los bloques entre sí mediante líneas de señal que se ajustan al diseño y a las restricciones especificadas.
6. Generar el archivo GDSII que contiene la representación física del diseño en formato de diseño de integración de circuitos (CAD).

## 9.5. *PrimeTime*: Análisis de tiempos estáticos

#### Archivos de entrada

- Archivo de diseño RTL: Lenguajes de descripción de hardware como Verilog o VHDL.
- Netlist: Formato SPICE o Verilog, representando la interconexión de los componentes.
- Archivos de restricciones de tiempo: Formato SDC (Synopsys Design Constraints), especificando restricciones como frecuencias de reloj y márgenes de tiempo.

- Archivos de tecnología: Conteniendo información sobre el proceso de fabricación y las características eléctricas de los componentes.

#### Archivos de salida

- Informes de análisis de tiempo: Detallando los resultados del análisis, incluyendo áreas donde el diseño no cumple con las restricciones de tiempo.
- Gráficos y tablas: Representaciones visuales de los datos de temporización, como histogramas de retardo y gráficas de caminos críticos.
- Archivos de resumen: Proporcionando una visión general del rendimiento del diseño en términos de temporización.

#### Procedimiento

Para realizar un análisis de tiempos estáticos en PrimeTime:

1. Importar el diseño RTL en PrimeTime.
2. Configurar las opciones de análisis, como la frecuencia de reloj y los criterios de fallo de tiempo.
3. EjecutarEjecutar el análisis de tiempo estático utilizando la opción Run Static Timing Analysis en el menú de PrimeTime.
4. Revisar y analizar los resultados del análisis de tiempo estático, que se muestran en forma de informes y gráficos.

## 9.6. *Formality*: Verificación formal

#### Archivos de entrada

- Archivo de diseño RTL: En formatos como Verilog o VHDL, que representan la descripción de alto nivel del circuito o sistema.
- Archivo de síntesis: Representando la implementación física del diseño RTL, comúnmente en un formato como Netlist.
- Archivos de restricciones y condiciones: Incluyendo restricciones de diseño y condiciones ambientales o de operación.

#### Archivos de salida

- Informes de verificación formal: Resumen de los resultados de la verificación, indicando la conformidad del diseño RTL con los criterios de verificación establecidos.

- Gráficos y tablas de cobertura: Muestran qué partes del diseño han sido verificadas completamente y cuáles requieren atención adicional.
- Archivos de resumen de verificación: Proporcionan una visión general detallada del proceso de verificación y sus resultados.

**Procedimiento**

1. Importar el diseño RTL y el archivo de síntesis en Formality.
2. Configurar las opciones de verificación, como la estrategia de verificación y los criterios de cobertura.
3. Ejecutar el análisis de verificación formal utilizando la opción Run Formal Verification.[en] el menú de Formality.
4. Revisar y analizar los resultados del análisis de verificación formal, que se muestran en forma de informes y gráficos.

# CAPÍTULO 10

## Conclusiones

- El flujo de diseño propuesto por *Synopsys* es una alternativa sólida capaz de lograr el proceso de manera ordenada, y que evita los errores provocados por los flujos de diseño anteriores llevados a cabo por otros estudiantes.
- El archivo del circuito diseñado en la primera etapa en VHDL debe ser sintetizable, es decir que debe estar escrito con código estructural en lugar de comportamiento.
- La herramienta ideal para realizar un análisis de tiempos de retardo y de sincronización estática es *PrimeTime*
- La herramienta ideal para realizar un análisis de tiempos de retardo y de sincronización estática es *PrimeTime*

# CAPÍTULO 11

## Recomendaciones

- Seguir el flujo de diseño propuesto por *Synopsys* en vez de otros flujos de diseño para lograr un trabajo más ordenado y evitar errores.
- Realizar un análisis de tiempos de retardo utilizando *PrimeTime* para obtener una etapa exitosa.
- Utilizar *TetraMAX* para el análisis con patrones de prueba generados automáticamente (ATPG) para generar vectores de entrada aleatorios y óptimos.

# CAPÍTULO 12

# Bibliografía


[1] *VCS/VCSi User Guide*, ver. X-2005.06, Synopsys, Inc, ago. de 2005.

[2] W. Fook Lee, *VHDL Coding and Logic Synthesis with Synopsys*. San Diego: Academic Press, 2000.

[3] Synopsys. «What is Physical Synthesis?» (), dirección: https://www.synopsys.com/glossary/what-is-physical-synthesis.html.

[4] Synopsys. «What is Static Timing Analysis (STA)?» (), dirección: https://www.synopsys.com/glossary/what-is-static-timing-analysis.html.

[5] A. Mittal. «VLSI IP.» (), dirección: http://www.vlsiip.com/asic_dictionary/F/formal_verification.html.

[6] A. Altuna Hernández, «Diseño de un Circuito Integrado con Tecnología de 180nm usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificación de Reglas de Diseño y Corrección de Errores Obtenidos.,» Universidad del Valle de Guatemala, 2021, págs. 8-97.

[7] J. Williams, *Digital VLSI Design with Verilog: A Textbook from Silicon Valley Technical Institute*. California: Springer, 2008.

[8] H. Kaeslin, *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. New York: Cambridge University Press, 2008.

[9] L. Scheffer, L. Lavagno y G. Martin, *EDA for IC Implementation, Circuit Desgin, and Process Technology*. San Diego: CRC Press, 2006.

CAPÍTULO 13

# Anexos

## 13.1. Laboratorio 1: Simulación lógica en VCS

**Tutorial de Flujo de Diseño de Synopsys**

**Lab 1: Simulación lógica. VCS**

**Objetivo**

Aprenda a simular RTL y diseño de nivel de puerta con VCS.

**Introducción**

VCS se utiliza para compilar archivos de entrada y simular el diseño. Para la depuración se utiliza el entorno visual de detección (DVE). VCS es una herramienta de línea de comandos que compila fuentes de entrada. Usando DVE es posible arrastrar y soltar señales en varias vistas o usar las opciones del menú para ver la fuente de la señal, rastrear controladores, comparar formas de onda y ver esquemas. Utilice DVE para encontrar rápidamente errores en RTL o gate, aserciones, banco de pruebas y cobertura.

**Tareas de laboratorio**

1. Primero compile el código fuente johnson.v RTL y el testbench Johnson_test.v usando VCS con el siguiente comando (Debe ejecutar la terminal desde la carpeta 'work') (Puede encontrar los comandos en el archivo 'RUN.txt').

```
% vcs ../source/johnson_test.v ../source/johnson.v -gui -debug_acc+all -full64
```

-gui Habilita DVE
-debug_acc+all Habilita la depuración de la línea de comandos, incluido el seguimiento de línea

Si la compilación se realiza correctamente, abra DVE desde la línea de comandos:

```
% dve -full64
```

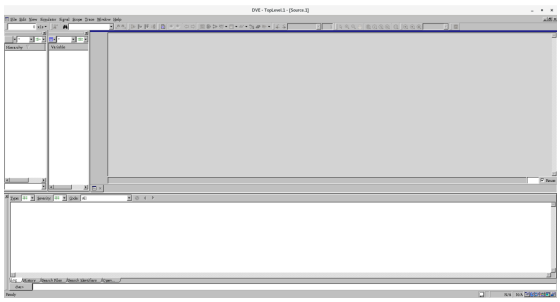


Fig.1. Ventana de GUI de nivel superior de DVE

1. Configurar DVE
   - En la barra de menús, seleccione *Simulator* > *Setup*,
   - En “Simulator executable” presione “Browse…” y seleccione el archive “simv” en el directorio actual

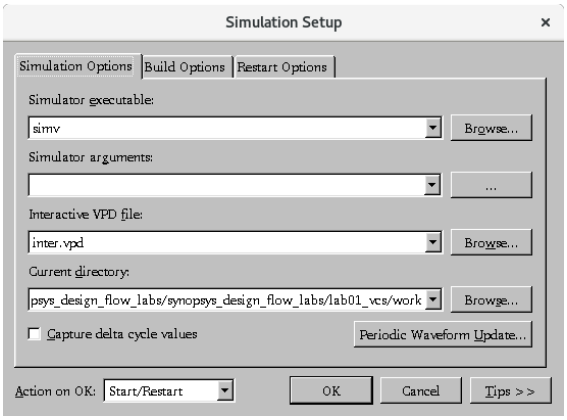



Fig.2. Configuración de la simulación

La configuración de simulación tiene las siguientes opciones (Fig.2.) :

**Simulator Executable -** especifica el nombre de un ejecutable de simulador.

**Simulator arguments** - identifica los argumentos del simulador.

**Interactive VPD file** - especifica el nombre del archivo VPD. Los archivos VPD (archivos de base de datos de diseño) son archivos versionados independientes de la plataforma en los que es posible volcar las señales seleccionadas durante la simulación. DVE obtiene información de jerarquía, cambio de valor y cierta información de aserción de estos archivos.

**Current directory** - especifica la ruta de acceso completa del ejecutable del simulador.

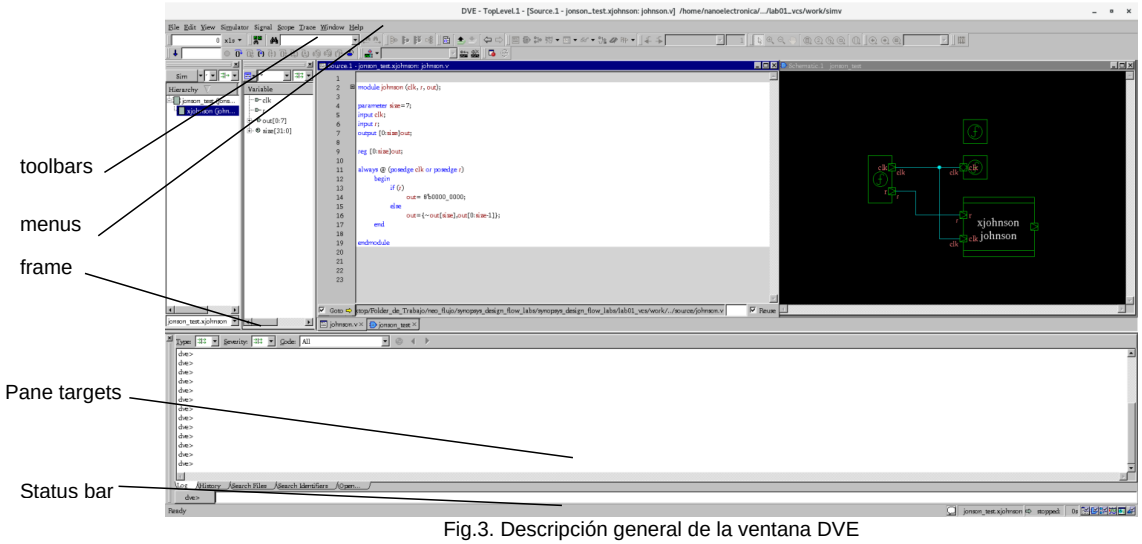



Fig.3. Descripción general de la ventana DVE

Una ventana TopLevel contiene:
1. Marco
2. Menús
3. Barras de herramientas
4. Barra de estado
5. Selección de panel.

**Visualización de una señal**

**Panel de jerarquía**: muestra la jerarquía de ámbito del diseño.

**Panel de datos**: muestra las variables de los ámbitos seleccionados del panel Jerarquía.

**Vista de código fuente**: muestra el código fuente y admite características relativas al código fuente, como el seguimiento del driver o , y la configuración de puntos de interrupción de línea.

**Vista de lista**: proporciona una vista de tabla para mostrar los valores de las señales a lo largo del tiempo.

**Vista esquemática**: proporciona un esquema basado en módulos para mostrar la conectividad del objeto.

**Vista de aserciones:** muestra el resumen de los resultados de las aserciones de la simulación, incluidos los éxitos, los errores y los incompletos.

Para ver información de onda en señales en la vista de onda (Fig.4).

1. Seleccione un objeto del panel Jerarquía, el panel Datos, la vista Origen, la vista Lista, la vista Esquema o la vista Aserción.
2. Haga click en el icono 'Add Waves' de la barra de herramientas.  (También se puede hacer click derecho en las señales y hacer click en Add to Waves)

Usando el botón para correr la simulación, la ejecuta hasta que se llega a un breakpoint, en ese momento

finaliza la simulación. También puede acabar cuando acabe el tiempo especificado.

Cuando la simulación se está ejecutando, este icono se activa. Haga clic para detener la simulación.

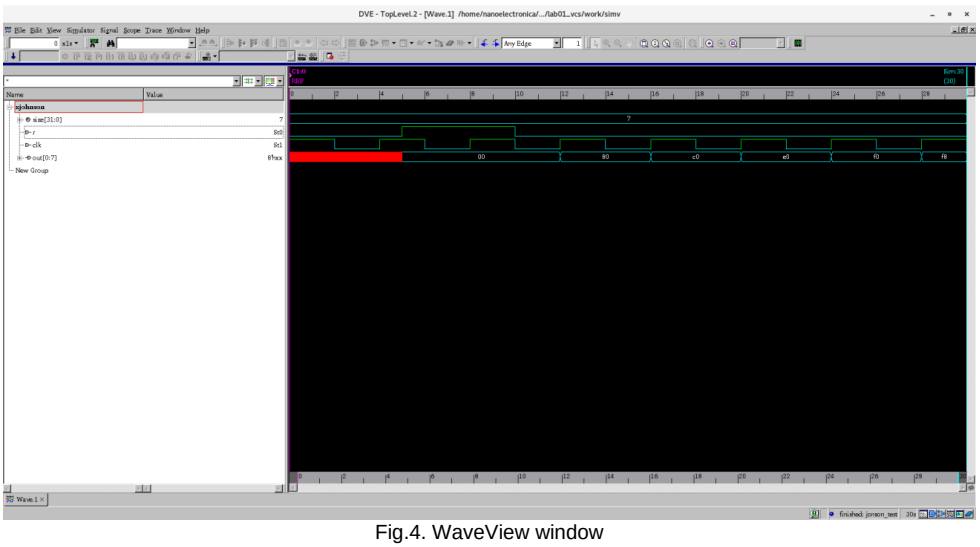


Fig.4. WaveView window

**Localizando Drivers y Cargas**

1. Localice Drivers y Cargas de una señal en cualquier momento para ver los drivers/cargas que causaron un cambio de valor y también los que posiblemente contribuyan al valor de una señal.
2. Seleccione una señal en una vista o panel. Por ejemplo, panel de datos, vista de código fuente, vista de lista, etc.
3. Haga click derecho y seleccione “Trace Drivers” o “Trace Loads”. Cuando un driver es localizado, un nuevo panel de Drivers será creado si ninguno existe en el cuadro de top level actual. Si ya existe un panel de driver, la info del driver será agregada al inicio de la lista.

Adicionalmente, el primer driver será resaltado en la vista de código fuente y anotado con un nodo azul en otros paneles. En la vista de señal, haga doble click sobre una señal para ver sus drivers. (Fig. 5)

Vincule los paneles del driver a la vista de código fuente en el mismo marco de top level y en la vista de esquemático. Los botones de opción Vincular a, en la parte superior derecha del panel, muestran las ventanas vinculadas actuales. Al vincular una vista de código fuente y esquemático, cuando se selecciona el objeto en el panel de driver, el objeto también se seleccionará en las vistas vinculadas.

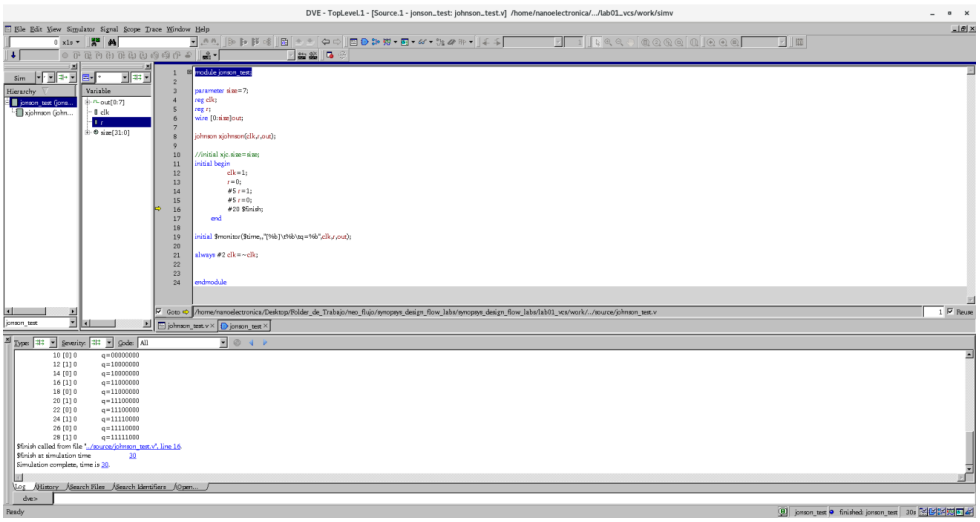


Fig.5. Ventana de top level con rastreo de drivers y cargas

**Comparación de señales, ámbitos y grupos**

Para comparar señales individuales con los mismos números de bits, ámbitos (para comparar hijos variables), buses o grupos de señales de uno o dos diseños.

Para ver una comparación

1. Seleccione una o dos señales, grupos de señales, ámbitos o buses en el panel señal de la vista de onda.
2. Haga clic con el botón derecho y seleccione Comparar.

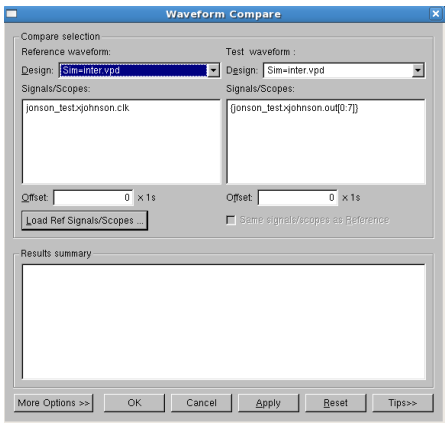



Fig.6.Cuadro de diálogo Comparación de forma de onda

3. Haga clic en Cargar señales/ámbitos de referencia y seleccione el archivo de texto con las señales y los ámbitos a los que hacer referencia.
   Nota: Si se comparan dos diseños desde la raíz, la región de forma de onda de referencia y la región de forma de onda de prueba pueden estar vacías.
4. Haga clic en el botón de Más opciones (Fig. 6).
   El cuadro de diálogo se expande y se muestran opciones adicionales.
5. En la sección Tipos de señal e ignorar opciones, seleccione los tipos de señal que desea comparar y seleccione las opciones de ignorar. Por ejemplo, si se selecciona Ignorar X y si el valor de la señal de referencia es X, siempre hay una coincidencia, cualesquiera que sean los valores de la señal de prueba..
6. Introduzca una tolerancia de tiempo para filtrar los valores de desajuste que tienen intervalos de tiempo más pequeños que el intervalo de tolerancia.
7. En la sección General, seleccione comparar recursivamente o crear solo señales con discrepancias.
8. Introduzca la configuración de desajustes para el máximo de desajustes por señal y el total máximo de desajustes para informar.
9. Haga clic en Aplicar para iniciar la comparación y mantener abierto el cuadro de diálogo. O haga clic en Aceptar para iniciar la comparación y cerrar el cuadro de diálogo (para abrirlo en cualquier momento desde el panel Señal CSM). Los resultados se muestran en la vista de señal actual.
10. Seleccione un resultado en la vista de señal, haga clic con el botón derecho y seleccione Mostrar información de comparación.



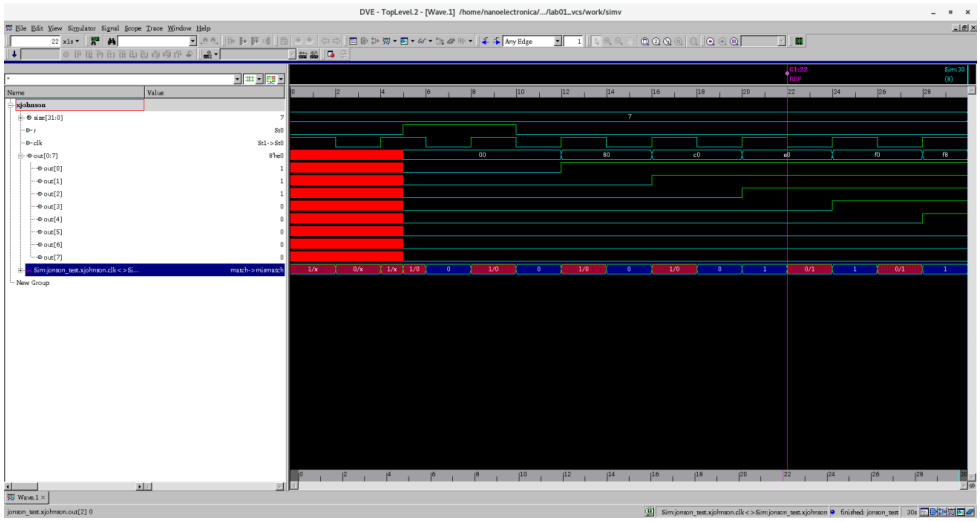


Fig.7. Grupos de señal comparados en la vista de señal.

**Selección de señal en la vista esquemática**

Para seleccionar una señal en la vista de esquemático:

1. Escriba el nombre de la señal en el cuadro “Buscar” en la barra de herramientas de la vista de esquemático y, a continuación, haga clic en el botón “Buscar siguiente”. La señal se resalta en el esquemático.
2. Con la señal seleccionada, haga clic en el botón de la barra de herramientas rastreo de Drivers o seleccione el elemento de menú rastreo de Drivers. La señal está resaltada en púrpura (Fig. 8).

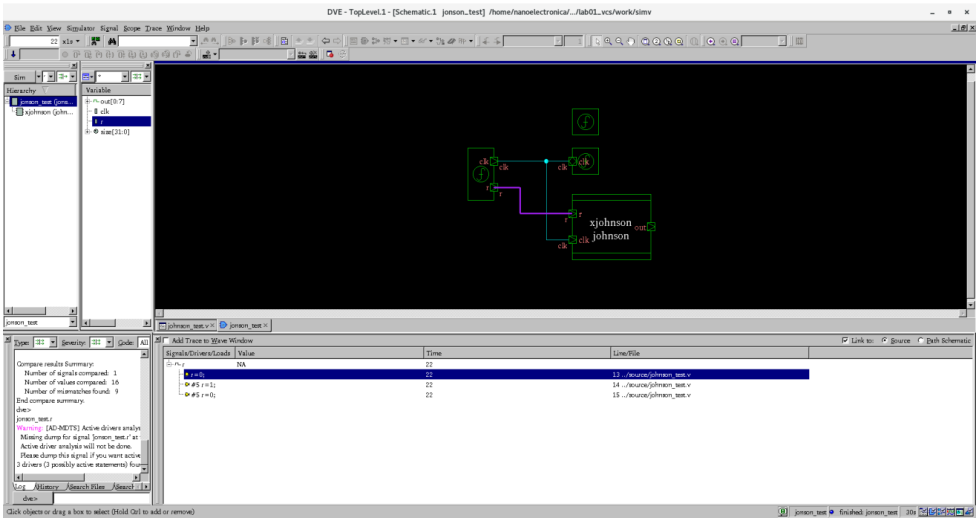



Fig.8. Grupos de señales resaltados en vista de Señal

**Uso de radixes definidos por el usuario**

En esta sección se describe cómo crear, editar, importar y exportar radiixes definidos por el usuario. Puede definir una asignación mnemotécnica personalizada de valores a cadenas para mostrarla en la vista de señal .

Para crear, eliminar, importar y exportar un radix definido por el usuario

1. Seleccione Señal > Establecer radix > definido por el usuario > editar.
2. Haga clic en Nuevo, escriba un nombre radix y, a continuación, pulse la tecla Intro del teclado. Todos los botones de Edit User-defined Radix se habilitan.
3. Haga clic en Agregar fila para activar una fila para el radio definido por el usuario y realizar los pasos siguientes:
   - Seleccione el texto y los colores de fondo para cada entrada de fila.
   - Seleccione el radio, haga clic en una celda de la columna Valor y visualización y, a continuación, introduzca los valores.
   Se edita el radix.
4. Seleccione una fila y, a continuación, haga clic en Eliminar fila. Se elimina la fila.
5. Seleccione un radix en el menú desplegable Nombre de tabla de Radix y haga clic en el botón Eliminar. Se elimina el radix.
6. Haga clic en Importar y, a continuación, busque y seleccione el radio deseado. El radix se importa.
7. Haga clic en Exportar, seleccione el radio y, a continuación, escriba un nombre de radio. El radix se exporta.
8. Seleccione la casilla de verificación Aplicar radio definido por el usuario a las señales seleccionadas.(Fig. 9). El radio definido por el usuario se aplica a la señal seleccionada en la vista de señal.
9. Haga clic en Aceptar o en Aplicar para guardar el radio definido por el usuario.

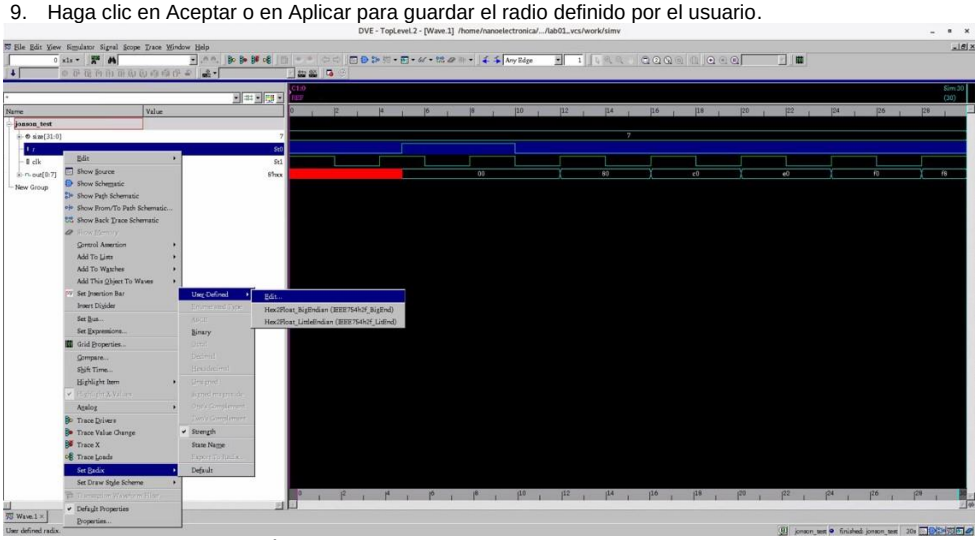


Fig.9. Cuadro de diálogo Editar radix definido por el usuario con ventana vista de señal

Para salir de DVE, escriba exit en la línea de comandos de la consola.

```
DVE> exit
```

## 13.2. Laboratorio 2: Síntesis lógica en Design Compiler

**Tutorial de Flujo de Diseño de Synopsys**

**Lab 2: Síntesis lógica en Design Compiler**

---

**Objetivo**

Aprenda a compilar el diseño RTL mediante la GUI del compilador de diseño y las interfaces de línea de comandos.

**Introducción**

Usando Design Compiler, es posible:

- Producir diseños ASIC rápidos y eficientes en el área mediante el empleo de bibliotecas de celdas estándar o especificadas por el usuario
- Traducir diseños de una tecnología a otra
- Explorar las compensaciones de diseño que involucran restricciones de diseño, como el tiempo, el área y la potencia en diversas condiciones de carga, temperatura y voltaje.
- Sintetizar y optimizar máquinas de estado finito
- Integrar las entradas de netlist y las salidas de netlist o esquemáticas en entornos de terceros sin dejar de admitir información de retardo y restricciones de lugar y ruta
- Crear y dividir esquemas jerárquicos automáticamente.

**Flujo de síntesis básico**

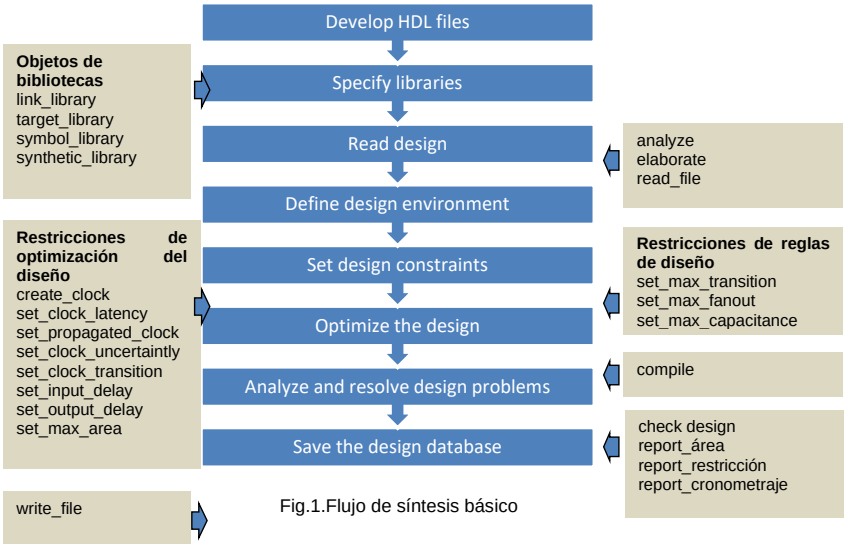
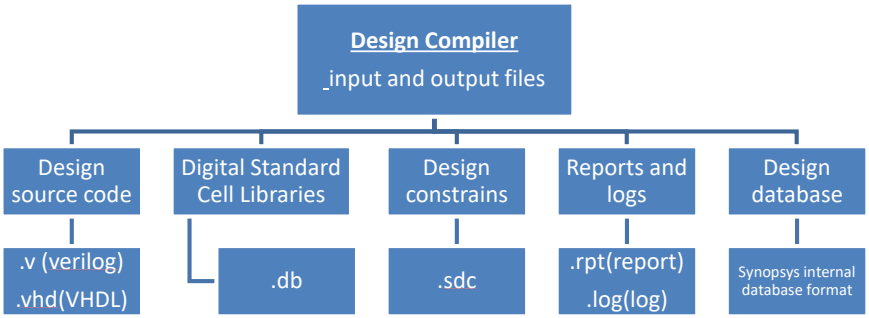



Fig.1.Flujo de síntesis básico

**Archivos de entrada y salida de Design Compiler**




1. Inicie la interfaz gráfica de usuario (GUI) de Design Compiler desde el directorio **work**. Para iniciarlo utilice el siguiente comando:

```
% dc_shell
dc_shell> start_gui
```

Esto abre la ventana GUI de nivel superior del compilador de diseño (Design Vision). (Figura 1)

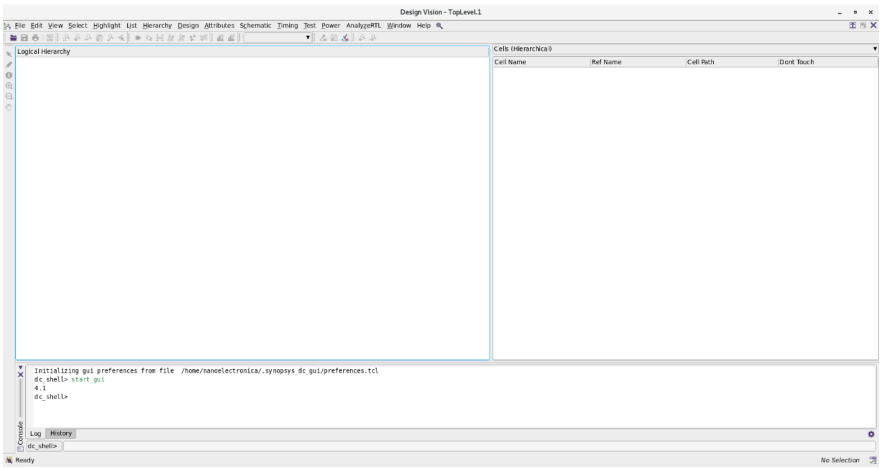


Fig.1. Ventana gui de nivel superior de Design Compiler

2. Cuando se inicia Design Compiler, lee automáticamente el archivo ".synopsys_dc.setup" del directorio actual. En este laboratorio tsu archivo se encuentra en el directorio **work**. Inicie Design Compiler desde ese directorio para leer automáticamente el archivo y

configurar las bibliotecas. Para comprobar si las bibliotecas se establecieron, abra la ventana en **File > Setup** (Fig. 2)..

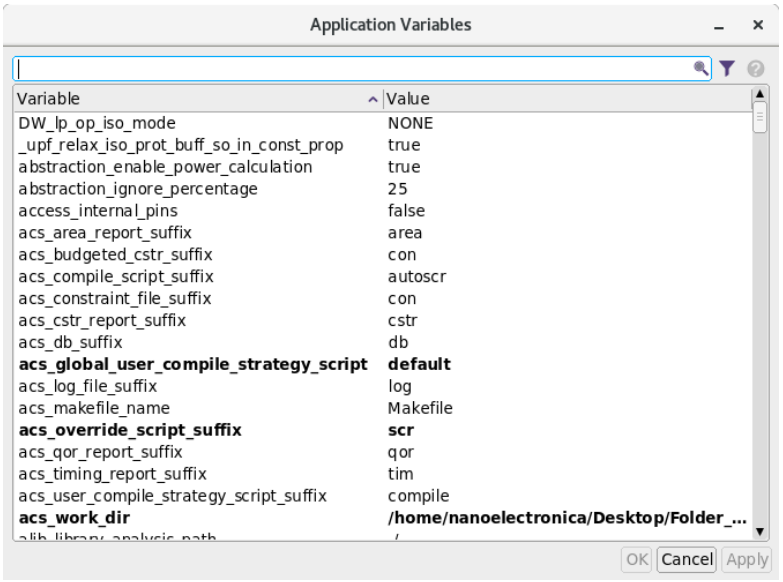



Figura 2. Ventana de configuración de la aplicación

3. Utilice el script creado para llevar a cabo comandos complejos. El script se encuentra en el directorio **scripts**. El nombre de este script: **compile.tcl.** Para obtener el script, escriba:

```
dc_shell> .. /scripts/compile.tcl
```

Ahora muestre todos los pasos del script **compile.tcl**.

3.1. Los comandos de diseño de lectura se derivan de los comandos read_file -format VHDL o read_file -format Verilog (Fig.3). En el diseño se utilizó el formato Verilog (Fig.4).

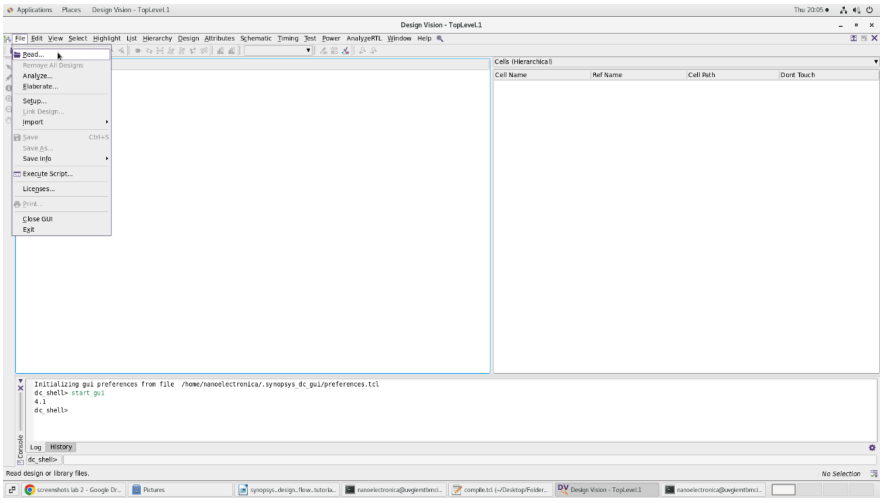


Figura 3. Leer diseño

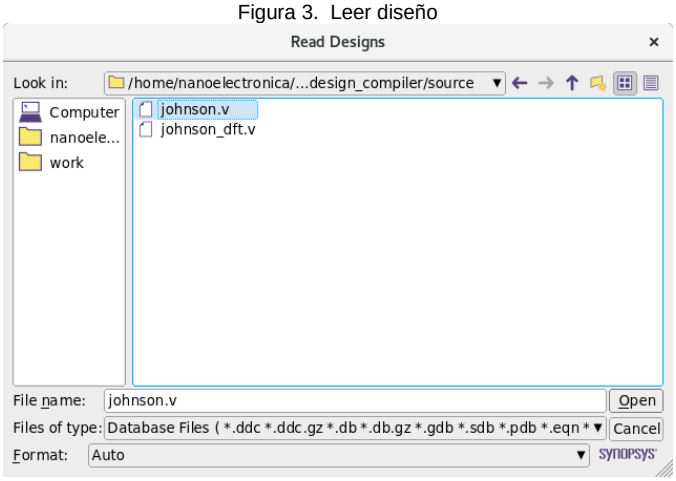



Figura 4. Ventana para leer diseños

3.2. A continuación, utilice los comandos **analyze** y **elaborate**.
El comando analyze hace lo siguiente:

- Lee un archivo fuente HDL
- Comprueba si hay errores (sin construir una lógica genérica para el diseño)
- Crea objetos de biblioteca HDL en un formato intermedio independiente de HDL
- Almacena los archivos intermedios en una ubicación definida

Si el comando analyze informa de errores, corríjalos en el archivo de origen HDL y ejecute analyze de nuevo. Después de analizar un diseño, vuelva a analizarlo solo cuando cambie (Fig.5 y Fig.6).

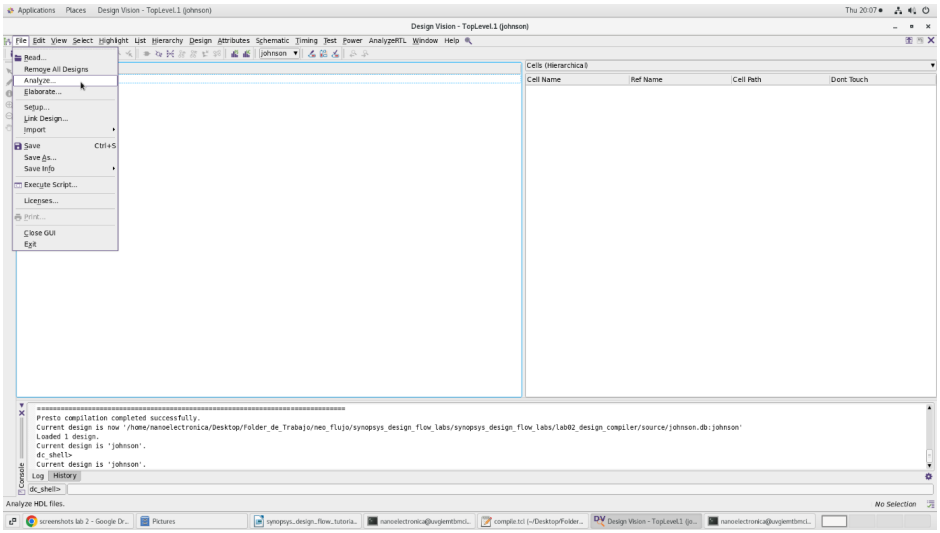



Figura 5. Analizar diseños

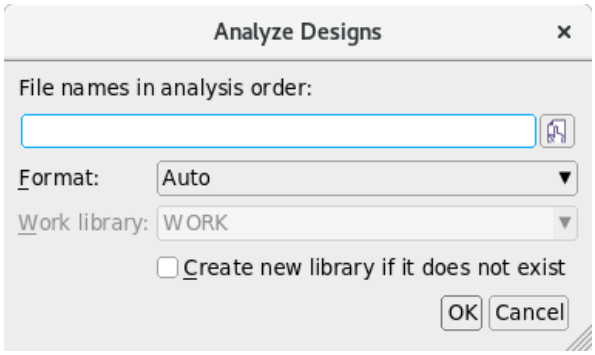



Figura 6. Cuadro Analizar diseños

3.3. El comando **elaborate** hace lo siguiente:

- Traduce el diseño en un diseño independiente de la tecnología (GTECH) a partir de los archivos intermedios producidos durante el análisis
- Permite cambiar los valores de los parámetros definidos en el código fuente
- Permite la selección de la arquitectura VHDL
- Reemplaza los operadores aritméticos HDL en el código con componentes de DesignWare
- Ejecuta automáticamente el comando link , que resuelve las referencias de diseño
- Utilice las opciones del comando elaborado de la siguiente manera (Fig.7 y Fig.8)

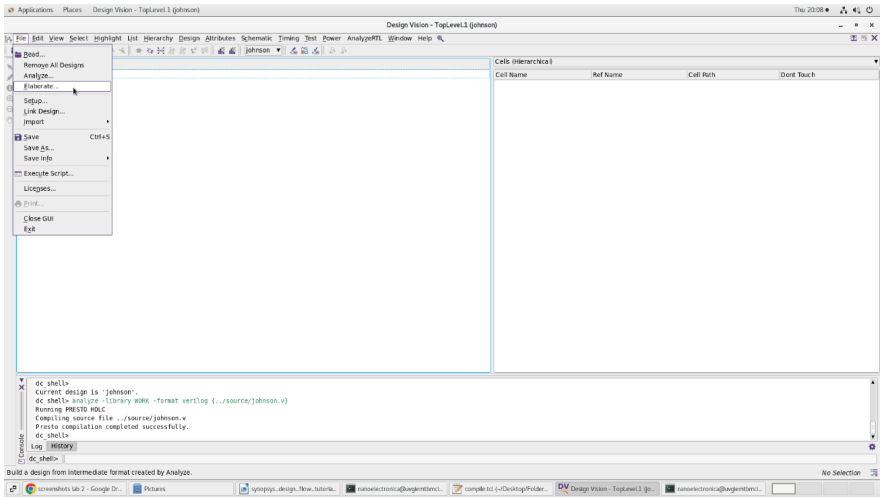


Figura 7. Diseños elaborados

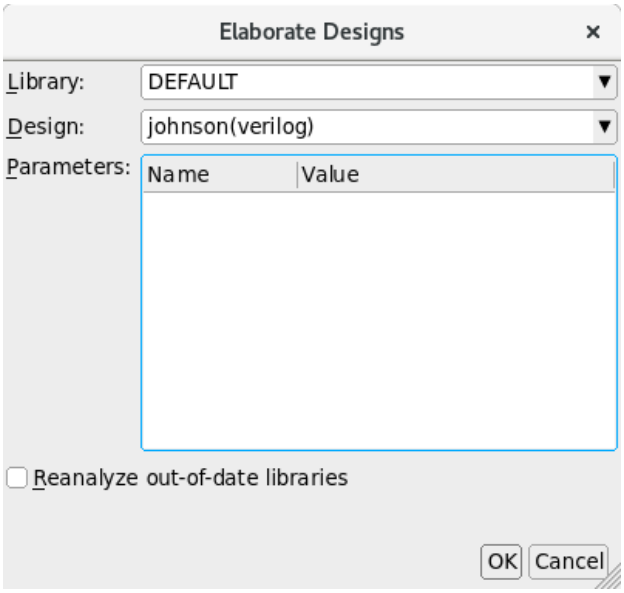



Figura 8. Caja de diseños elaborados

3.4. Para que un diseño esté completo, debe estar conectado a todos los componentes de la biblioteca y a los diseños a los que hace referencia. Por lo tanto, para realizar una resolución basada en nombres de referencias de diseño para el diseño actual, use el comando **link**.

```
dc_shell> link
```

3.5. El comando **check_design** comprueba la coherencia de la representación interna de las restricciones de diseño de Synopsys actuales y emite mensajes de error y advertencia según corresponda.

```
dc_shell> check_design
```

3.6. Establezca restricciones de optimización del diseño, lea johnson.sdc.
Las restricciones de optimización comprenden:

- Limitaciones de tiempo (rendimiento y velocidad)
- Retrasos de entrada y salida (rutas síncronas)
- Retardo mínimo y máximo (rutas asincrónicas)
- Superficie máxima (número de puertas)

Para leer constraints con formato sdc (Synopsys Design Constraints):

```
dc_shell> read_sdc .. /source/johnson.sdc
```

Después de todos los pasos mencionados anteriormente, la vista de diseño de Design Compiler se muestra en la Fig.9.

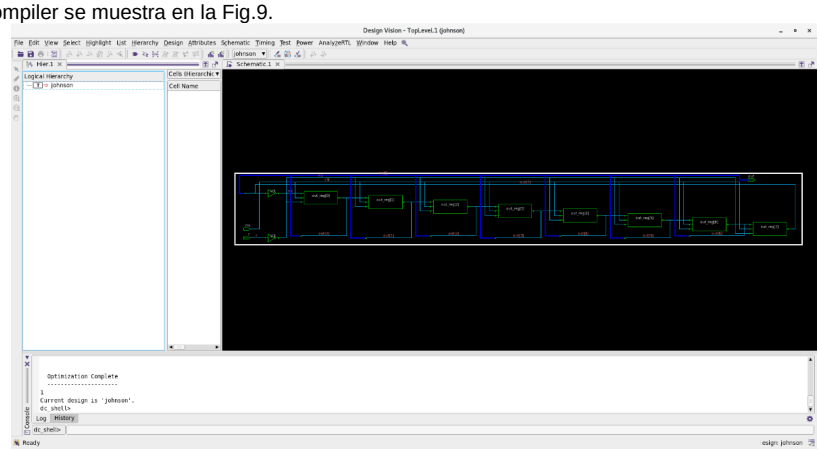


Figura 9. Vista Diseño desde Design Compiler

3.7. El comando **compile** realiza la síntesis y optimización lógica a nivel de compuerta en el diseño actual. La optimización es controlada por las restricciones especificadas por el usuario en el diseño. Para realizar esto, utilice el siguiente comando:

```
dc_shell> compile
```

Esto especifica que los elementos secuenciales en el diseño final deben coincidir exactamente con las descripciones especificadas en el HDL.

Compilar el diseño para producir una descripción a nivel de puerta (Fig.10).

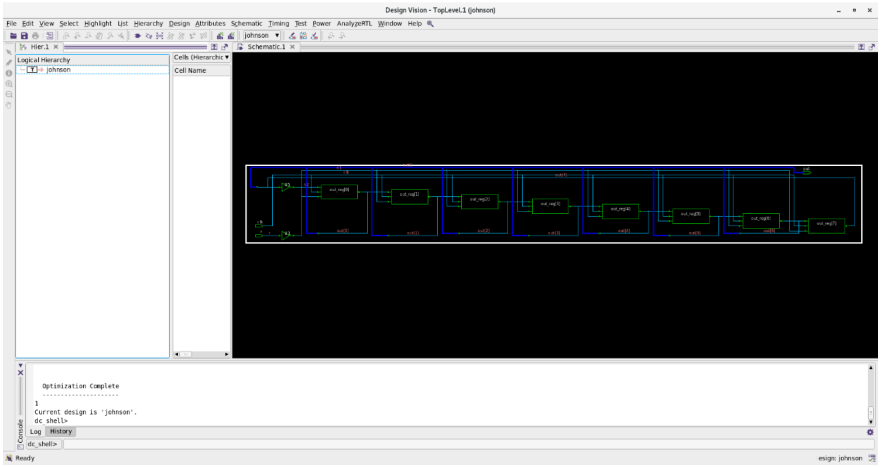



Figura 10. Vista Diseño de Design Compiler después de compilar

3.8. Design Compiler puede generar numerosos informes sobre los resultados de la síntesis y optimización del diseño. Los informes se utilizan para analizar y resolver cualquier problema de diseño o para mejorar los resultados de síntesis.

Obtener informes de Design Compiler

```
****************************************
Report : area
Design : johnson
Version: L-2016.03-SP5-5
Date   : Mon Jul 23 06:21:10 2018
****************************************


Library(s) Used:


    saed14rvt_tt0p8v25c (File:
/VIM/Courses/synopsys_df/lab02_design_compiler/ref/db_nldm/saed
14rvt_tt0p8v25c.db)


Number of ports:                           10
Number of nets:                            12
Number of cells:                           10
Number of combinational cells:              2
Number of sequential cells:                 8
Number of macros/black boxes:               0
Number of buf/inv:                          2
Number of references:                       3


Combinational area:                 0.355200
Buf/Inv area:                       0.355200
```

```
Noncombinational area:                8.524800
Macro/Black Box area:                 0.000000
Net Interconnect area:                3.550286

Total cell area:                      8.880000
Total area:                          12.430287

****************************************
Report : constraint
Design : johnson
Version: L-2016.03-SP5-5
Date   : Mon Jul 23 06:23:23 2018
****************************************
                                                  Weighted
    Group (max_delay/setup)      Cost     Weight     Cost
    --------------------------------------------------
    clk                          0.00      1.00      0.00
    default                      0.00      1.00      0.00
    --------------------------------------------------
    max_delay/setup                                 0.00

                             Total Neg  Critical
    Group (critical_range)      Slack    Endpoints   Cost
    --------------------------------------------------
    clk                          0.00         0      0.00
    default                      0.00         0      0.00
    --------------------------------------------------
    critical_range                                  0.00

                                                  Weighted
    Group (min_delay/hold)       Cost     Weight     Cost
    --------------------------------------------------
    clk (no fix_hold)            0.00      1.00      0.00
    default                      0.00      1.00      0.00
    --------------------------------------------------
    min_delay/hold                                  0.00


    Constraint                                      Cost
    --------------------------------------------------
    min_capacitance                                 0.00 (MET)
```



```
    max_capacitance                                 0.00 (MET)
    max_delay/setup                                 0.00 (MET)
    sequential_clock_pulse_width                    0.00 (MET)
    critical_range                                  0.00 (MET)


****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : johnson
Version: L-2016.03-SP5-5
Date   : Mon Jul 23 06:23:41 2018
****************************************

Operating Conditions: tt0p8v25c   Library: saed14rvt_tt0p8v25c
Wire Load Model Mode: top

  Startpoint: out_reg_7_ (rising edge-triggered flip-flop
clocked by clk)
  Endpoint: out[7] (output port clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  johnson            ForQA                 saed14rvt_tt0p8v25c

  Point                                        Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                        0.00       0.00
  clock network delay (ideal)                  0.00       0.00
  out_reg_7_/CK (SAEDRVT14_FDPRBQ_V2_0P5)      0.00       0.00 r
  out_reg_7_/Q (SAEDRVT14_FDPRBQ_V2_0P5)       0.04       0.04 r
  out[7] (out)                                 0.00       0.04 r
  data arrival time                                       0.04

  clock clk (rise edge)                        2.00       2.00
  clock network delay (ideal)                  0.00       2.00
  clock uncertainty                           -0.30       1.70
```



```
output external delay                          -1.20       0.50
data required time                                         0.50
-------------------------------------------------------------
data required time                                         0.50
data arrival time                                         -0.04
-------------------------------------------------------------
slack (MET)                                                0.46
```

3.9 Escribe un diseño de memoria a formato .ddc (synopsys internal database format), así como .v y .vhdl.

```
dc_shell> write -format verilog ../results/johnson1.v
```

4. Para iniciar DFT Compiler (Design for Test Compiler), restablezca el diseño mediante el siguiente comando:

```
dc_shell> reset_design
```



## DFTC (Diseño para el compilador de pruebas)

**Objetivo**

Aprenda los comandos y cómo compilar el diseño con el compilador DFT.

**Introducción**

Utilice el compilador DFT para comprobar RTL y diseños asignados en busca de infracciones de DFT, insertar cadenas de análisis en los diseños y exportar todos los archivos necesarios para las herramientas posteriores.

DFT Compiler requiere que el usuario especifique un estilo de escaneo para realizar la síntesis de escaneo. Un estilo de exploración dicta las celdas de exploración adecuadas para insertar durante la optimización. El estilo de escaneo, que se selecciona, se utiliza en todos los módulos de diseño. Hay tres tipos de estilos de escaneo disponibles en DFT Compiler:

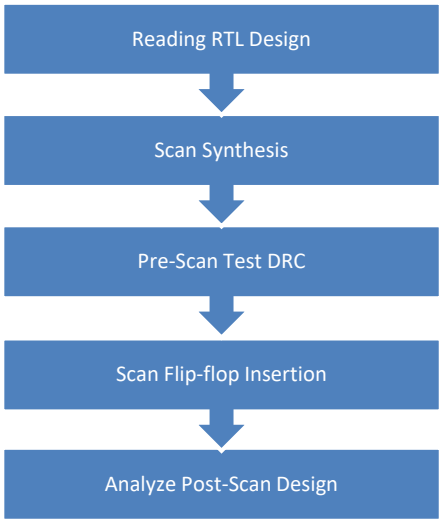
- Flip-flop multiplexado
- Escaneo cronometrado
- Diseño de escaneo sensible al nivel

Para especificar el estilo de escaneo de este diseño se utiliza el siguiente comando:

```
dc_shell> set test_default_scan_style
```

Usando este comando con no marcar el estilo de escaneo, se elige de forma predeterminada el estilo de escaneo **multiplexed_flip_flop**.

**Flujo básico de síntesis de escaneo**



## Tareas de laboratorio

Para ejecutar DFT, invoque el script “script.tcl”, que se encuentra en el directorio **scripts**. Para invocar script.tcl escriba.

```
dc_shell> source ../scripts/script.tcl
```

Incluye todos los comandos necesarios para DFT. Muestra el contenido de esta paso a paso.

1. Para especificar el estilo de análisis de este diseño, utilice el siguiente comando:

```
dc_shell> set test_default_scan_style
```

Usando este comando sin marcar el estilo de escaneo predeterminado, elija el estilo de escaneo **multiplexed_flip_flop** .

2. Lea el diseño con el siguiente comando:

```
dc_shell> read_verilog ../source/Johnson_dft.v
```

Después de leer el diseño, vea la vista esquemática del recuento de Johnson utilizando la opción Schematic de la barra de herramientas como se muestra en la Fig. 1.

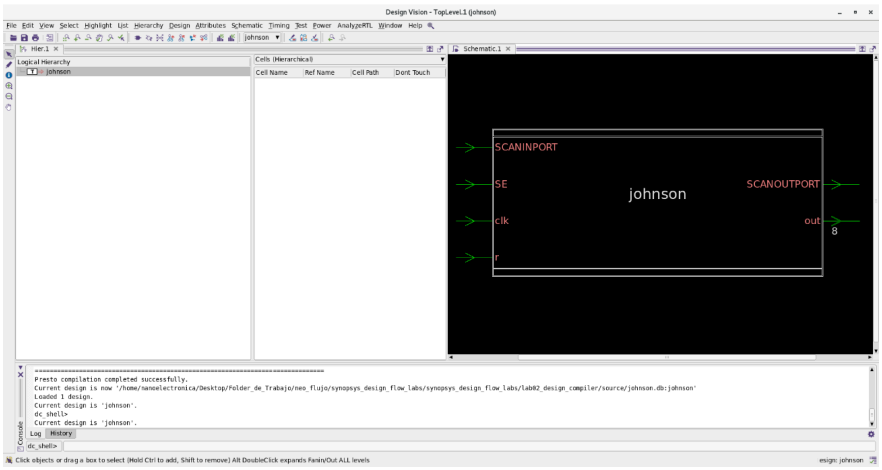



Fig.1 Vista esquemática de apertura de Johnson

La vista esquemática se muestra en la Fig. 2.

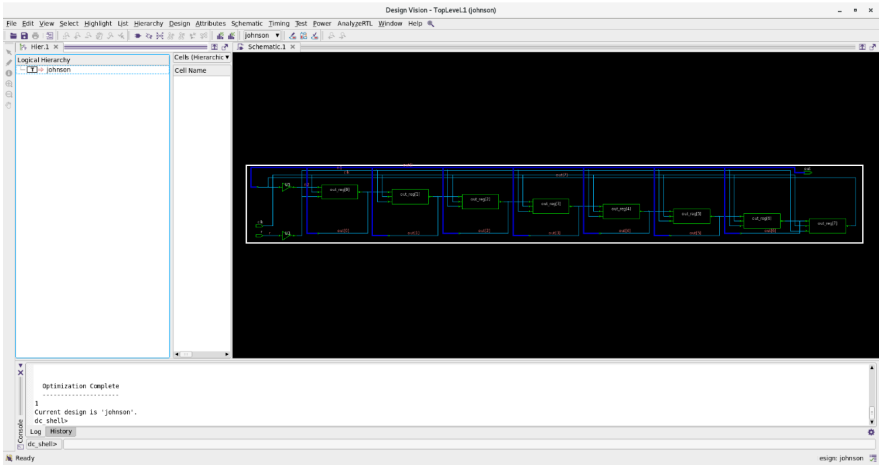



Figura 2. Vista esquemática de Johnson antes de DFT

3. Leer constraints con formato .sdc.

```
dc_shell> read_sdc ../source/johnson.sdc
```

4. Asigne los tipos de puerto a las señales que se utilizarán en DFT. Por ejemplo, ScanMasterClock, Reset y ScanEnable. Para dar tipos de puerto a las señales, utilice los siguientes comandos:

```
dc_shell> set_dft_signal -view exist -type
ScanMasterClock -timing {15 25} -port clk
dc_shell> set_dft_signal -view exist -type Reset -active
1 -port r
dc_shell> set_dft_signal -view exist -type ScanEnable -
active 1 -port SE
dc_shell> report_dft_signal -view exist
```

**set_dft_signal** especifica los tipos de señal DFT para la inserción de DRC y DFT.

**-view exist** option muestra los puertos que ya existen en el diseño y si el puerto debe usarse por primera vez, aplique la opción **-view spec**.

opción **-active** especifica el sentido activo del puerto (alto o bajo).

5. Cree unprotocolo t est para configurar el diseño para la prueba de análisis.

```
dc_shell> create_test_protocol
```

Esto crea un protocolo de prueba que DFT Compiler utiliza para la comprobación de reglas de diseño de pruebas, así como para la generación de patrones y los pasos de formato vectorial.

6. Compruebe el diseño correspondiente al estilo de escaneo **multiplexed-flip-flop**, que se define mediante **test_default_scan_style** establecidos. Ejecute la comprobación de reglas de diseño mediante el siguiente comando:

```
dc_shell> dft_drc
```

Este comando comprueba las infracciones y

- Si se denuncian infracciones, cambie el código RTL y repita los pasos 5 y 6.
- Si no se denuncian infracciones, proceda a analizar la síntesis.

7. Si no hay violaciones después **de dft_drc,** significa que el protocolo está funcionando, así que escriba el protocolo completo con formato .spf para su uso posterior en el flujo asignado.

```
dc_shell> write_test_protocol -out
../source/Johnson_test.spf
```

8. Cree un objeto de reloj y defina su forma de onda en el diseño actual. 1000 especifica el período de la forma de onda del reloj en las unidades de tiempo de la biblioteca. A continuación, establezca los retrasos de entrada en SCANINPORT y en TEST_SE en relación con una señal de reloj.

```
dc_shell> create_clock clk -period 100
dc_shell> set_input_delay 25 SCANINPORT -clock clk
dc_shell> set_input_delay 15 TEST_SE -clock clk
```

9. Scan Insertion cambiará el diseño para escapar de los efectos negativos (como la posible influencia de Scan Registers que puede aumentar el área de esquema y aumentar el fan-out en las redes), que deben tenerse en cuenta durante la síntesis. Es por eso que la síntesis utiliza disparadores de escaneo para evitar los cambios de características en esquemático. Para realizar esto, utilice el siguiente comando:

```
dc_shell> compile -scan
```

10. Para que un diseño esté completo, debe estar conectado a todos los componentes de la biblioteca y los diseños a los que hace referencia. Por lo tanto, para realizar una resolución basada en nombres de referencias de diseño para el diseño actual, use el comando **link**. Las referencias deben estar ubicadas y vinculadas al diseño actual para que el diseño sea funcional. El propósito de este comando es localizar todos los diseños y componentes de biblioteca a los que se hace referencia en el diseño actual y conectarlos (vincularlos) al diseño actual.

```
dc_shell> link
```

11. Lee un archivo de protocolo de prueba en la memoria.

```
dc_shell> read_test_protocol ../source/Johnson_test.spf
```




12. Utilice -view spec para definir la estructura de escaneo.

```
dc_shell> set_dft_signal -view spec -port SCANINPORT -
type ScanDataIn
dc_shell> set_dft_signal -view spec -port SCANOUTPORT -
type ScanDataOut
```

13. Establezca la configuración de inserción DFT para el diseño actual.

```
dc_shell> set_dft_insertion_configuration -
preserve_design_name true
```

**-preserve_design_name true** especifica si se debe conservar el nombre del diseño al escribir desde la herramienta a la base de datos durante la inserción de DFT. Los valores válidos son true para conservar el nombre del diseño y false para permitir el cambio de nombre del diseño. El valor predeterminado es false.

14. Especifique el diseño de la cadena de escaneo.

```
dc_shell> set_scan_configuration -chain_count 1
```

**-chain_count 1** specifica un entero positivo para el número de cadenas que insert_dft es construir. Si no se especifica, insert_dft crea el número mínimo de cadenas de análisis coherente con las restricciones de mezcla de reloj.

15. Antes de realizar la inserción del análisis, es posible obtener una vista previa del diseño del análisis especificando el comando **preview_dft**. Este comando genera una cadena de escaneo que satisface las especificaciones de escaneo en el diseño actual y muestra el diseño de la cadena de escaneo. Previsualiza, pero no implementa, los puntos de prueba, las cadenas de escaneo y la lógica de control de reloj en chip que se agregarán al diseño actual.

```
dc_shell> preview_dft
```

16. Después de obtener una vista previa del diseño, está listo para ensamblar las cadenas de escaneo mediante **insert_dft** comando. Es uncircuito de escaneo dds (ya sea escaneo interno o escaneo de límites) al diseño actual.

```
dc_shell> insert_dft
```

17. Establezca el atributo **fix_multiple_port_nets** en un valor especificado en el diseño actual.
Para almacenar constantes lógicas de búfer, utilice la opción **-buffer_constraints** con la opción **–all**. La opción
**-buffer_constraints** amortigua las constantes lógicas en lugar de duplicarlas.

```
dc_shell> set_fix_multiple_port_nets -all -
buffer_constraints
dc_shell> compile -scan -incremental
```

18. Ejecute de nuevo la comprobación de reglas de diseño y muestre la información del área para el diseño actual.

```
dc_shell> dft_drc -coverage_estimate
dc_shell> report_area
```

**-coverage_estimate** genera una estimación de la cobertura de la prueba al final de la verificación de la regla de diseño.

19. Escribir un diseño dememoria m en un archivo con formato . sdc (synopsys internal database format), también format .v y . vhdl.

```
dc_shell> write -format verilog -h -o
../results/johnson_compiled.v
```

20. Escriba un diseño con formato de retardo estándar (SDF).

```
dc_shell> write_sdf ../results/johnson.sdf
```

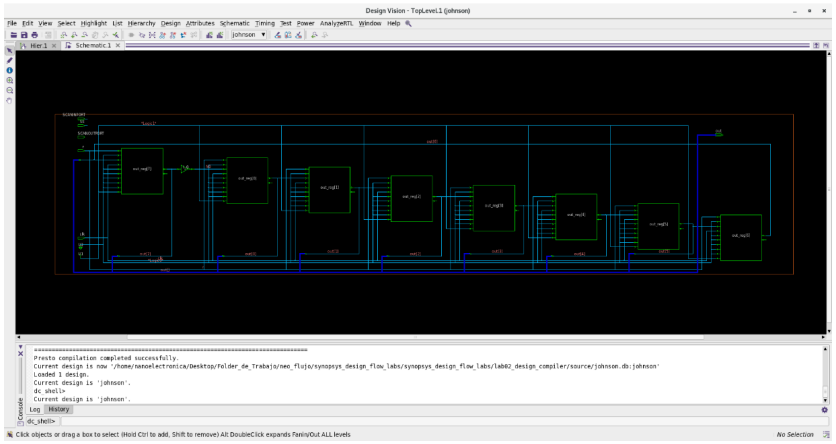


Figura 3. Vista esquemática de Johnson después de DFT

Los resultados del diseño de laboratorio se almacenan en el directorio **../results**/.

21. Para salir del compilador DFT, escriba exit en la línea de comandos.

```
dc_shell> exit
```

## 13.3. Laboratorio 4: Análisis en tiempo estático en PrimeTime

**Tutorial de Flujo de Diseño de Synopsys**

**Lab 4: Análisis de tiempo estático. PrimeTime**

---

**Objetivo**

Aprenda a usar PrimeTime para validar el rendimiento de temporización de un diseño comprobando todas las rutas posibles en busca de infracciones de temporización, sin usar simulación lógica o vectores de prueba.

**Introducción**

PrimeTime es una herramienta de análisis de tiempo estático de nivel de puerta de chip completo que es una parte esencial del flujo de diseño y análisis para los diseños de chips grandes de hoy en día.

**Tareas de laboratorio**

PrimeTime se puede utilizar después de Design Compiler o IC Compiler. La diferencia es que después de IC Compiler el netlist de parásitos en formato SPEF puede ser utilizado por PrimeTime para el cálculo de tiempo. Los pasos para ejecutar PrimeTime con los resultados de Design Compiler son muy similares.

1. Inicie la interfaz gráfica de usuario (GUI) de PrimeTime desde el directorio de **trabajo**, que se encuentra en post_lay directorio. Para iniciarlo ingrese:

```
% pt_shell
pt_shell> start_gui
```

Esto abre la ventana de GUI de nivel superior de PrimeTime. (Figura 1)

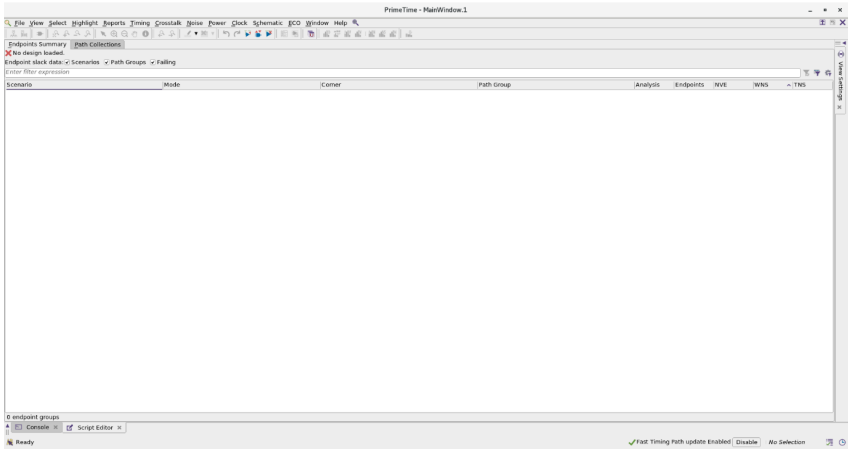



Figura 1. Ventana de GUI de nivel superior de PrimeTime

2. Las librerías se anexan a la ruta de búsqueda desde .synopsys_pt.setup ubicado en el directorio **de trabajo**. El archivo de instalación incluye lo siguiente:

```
set link_path  [list ../ref/lib/db_nldm/saed14rvt_tt0p8v25c.db ]
```

3. Utilice el script creado para llevar a cabo comandos complejos. Hay dos scripts en el directorio **de scripts**, que se encuentra en el directorio **post_lay**. Los nombres de estos scripts son: **sta_post_max.tcl** y **sta_post_min.scr**. El primero se crea para verificar el tiempo de configuración y el segundo para verificar el tiempo de espera. Para obtener el script, escriba:

```
pt_shell> source ../scripts/sta_post_max.tcl
```

Para comprobar el tiempo de espera, primero escriba el comando **reset_design** en la línea de comandos y, a continuación, obtenga el script **sta_post_min.tcl**.

Ahora muestra todos los pasos del  script **sta_post_max.tcl**.

3.1 Lea la netlist de diseño. (Figura 2)  PrimeTime acepta listas de redes de diseño a nivel de puerta en formatos Verilog y VHDL. Lea design netlist con el siguiente comando: **read_verilog**, **read_vhdl**.

```
pt_shell> read_verilog [list ../source/johnson_icc.v]
pt_shell> current_design johnson
```

3.2 Para que un diseño esté completo, debe estar conectado a todos los componentes de la biblioteca y los diseños a los que hace referencia.  Por lo tanto, para realizar una resolución basada en nombres de referencias de diseño para el diseño actual, use el comando **link**. Las referencias deben estar ubicadas y vinculadas al diseño actual para que el diseño  sea funcional. La función de este comando es localizar todos los diseños y componentes de biblioteca a los que se hace referencia en el diseño actual y conectarlos (vincularlos) al diseño actual.

```
pt_shell> link
```

3.3 Lea las restricciones de diseño y los parásitos. Lea las restricciones de diseño mediante el siguiente comando: **read_sdc**.

```
pt_shell> read_sdc ../source/johnson_icc.sdc
pt_shell> read_parasitics ../source/johnson.spef.max
```

3.4 Aplique un valor constante a los puertos de entrada r y SE con el **set_case_analysis** command. Especifica que un puerto o pin está en un valor lógico constante 1 o 0, o se considera con un tránsito ascendente o descendenten. Los puertos de este laboratorio no están activos con el valor 0, por lo que el puerto r y Se se da 0.

```
pt_shell> set_case_analysis 0 [get_port r]
pt_shell> set_case_analysis 0 [get_port SE]
```



3.5 Obtenga un informe detallado sobre todas las infracciones de restricciones en el diseño con **report_constraint -all_violators**.

```
pt_shell> report_constraint -all_violators -significant_digits
4 > ../results/johnson.min_constr.rpt
```

3.6 El comando **report_timing** es el comando de análisis PrimeTime más flexible y potente. La opción **-delay_type** especifica el tipo de comprobaciones de tiempo que se van a informar. Establezca el tipo de retardo en **máximo** para las comprobaciones de configuración, **mínimo** para las comprobaciones de retención.

```
pt_shell> report_timing -delay_type max 4 >
../results/johnson.min_timing.rpt
```

En los repositorios es posible tener infracciones, por ejemplo, infracción de retención o configuración. Para corregir la infracción de retención, agregue búferes al puerto respectivo. Y para corregir la violación de configuración, aumente el área de las celdas. Aquí hay un ejemplo de un informe que no tiene violaciones.

```
****************************************
Report : timing
   -path_type full
   -delay_type max
   -max_paths 1
   -sort_by slack
Design : johnson
Version: L-2016.06-SP2
Date   : Mon Jul 23 07:20:35 2018
****************************************
  Startpoint: out_reg[7] (rising edge-triggered flip-flop
clocked by clk)
  Endpoint: out_reg[0] (rising edge-triggered flip-flop clocked
by clk)
  Last common pin: clk
  Path Group: clk
  Path Type: max


  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.0000     0.0000
  clock network delay (propagated)         0.0000     0.0000
  out_reg[7]/CK (SAEDRVT14_FSDPRBQ_V2_1)   0.0000     0.0000 r
  out_reg[7]/Q (SAEDRVT14_FSDPRBQ_V2_1)    0.0247 &   0.0247 f
U14/X (SAEDRVT14_INV_1P5)                  0.0114 &   0.0361 r
out_reg[0]/D (SAEDRVT14_FSDPRBQ_V2_1)      0.0000 &   0.0361 r
```



```
data arrival time                                          0.0361
clock clk (rise edge)                          2.0000      2.0000
clock network delay (propagated)               0.0000      2.0000
clock reconvergence pessimism                  0.0000      2.0000
out_reg[0]/CK (SAEDRVT14_FSDPRBQ_V2_1)                     2.0000 r
library setup time                            -0.0167      1.9833
data required time                                         1.9833
-------------------------------------------------------------------
data required time                                         1.9833
data arrival time                                         -0.0361
-------------------------------------------------------------------
slack (MET)                                                1.9472
```

El informe también se puede ver en la consola del controlador de análisis de temporización al comentar una de las filas y hacer clic en el botón inspector. Ver Fig. 2 y Fig.3.

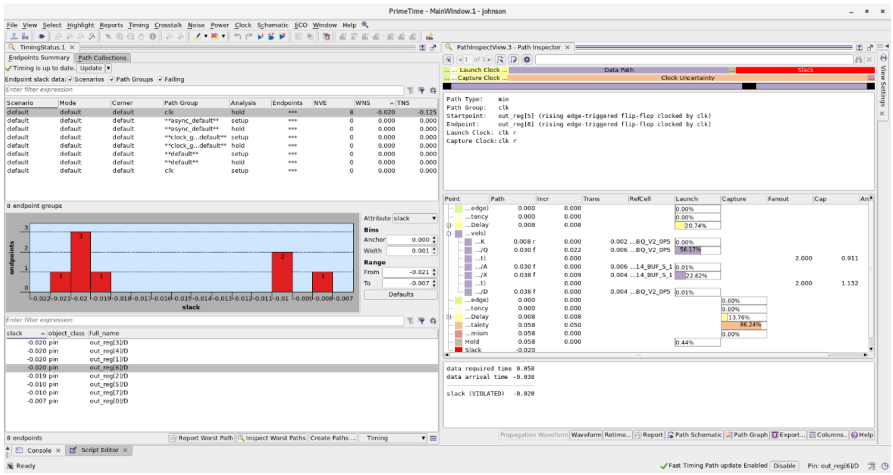



Figura 2. El controlador de análisis de temporización



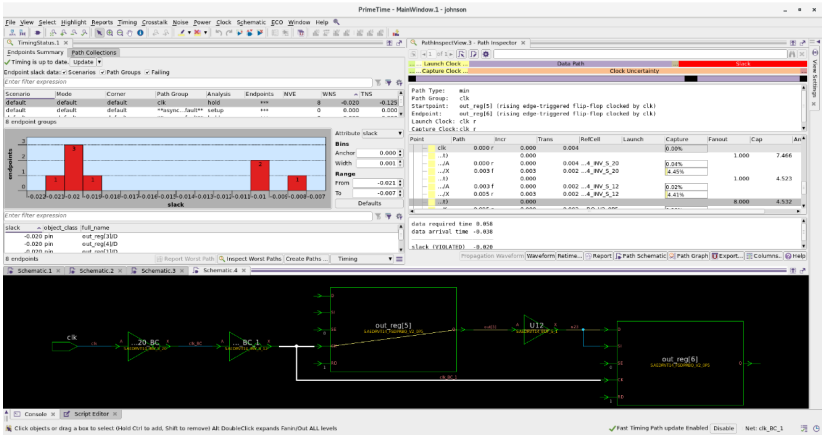


Figura 3. Consola del Inspector de rutas

3.7 Elarchivo out put de PrimeTime es el formato de retardo estándar (.sdf) e incluye información de retraso, como retrasos de celda de pin a pin y retrasos netos, y comprobaciones de tiempo, como tiempos de configuración, retención, recuperación y eliminación. En el script se hizo usando el siguiente comando:

```
pt_shell> write_sdf ../results/johnson.min.sdf
```

En este diseño, los resultados de salida se almacenan en el directorio ../**results**/.

4. Para salir de PrimeTime escribe:

```
pt_shell> exit
```

## 13.4. Laboratorio 5: Verificación formal en Formality

**Tutorial de Flujo de Diseño de Synopsys**

**Laboratorio 5: Verificación formal. Formality**

---

**Objetivo**

Aprenda a usar Formality para detectar diferencias inesperadas que pueden haberse introducido en un diseño durante el desarrollo.

**Introducción**

El propósito de Formality es detectar diferencias inesperadas que pueden haberse introducido en un diseño durante el desarrollo.

En Formality se utilizan los siguientes conceptos:

**Diseño de referencia:** Este diseño es el diseño dorado, el estándar contra el cual la Formality prueba la equivalencia.
**Diseño de implementación:** Este diseño es el diseño cambiado. Es el diseño cuya precisión quieres probar. Por ejemplo, un diseño recién sintetizado es una implementación del diseño RTL de origen.

Para iniciar Formality, escriba el siguiente comando en el terminal:

```
% fm_shell
fm_shell (setup)>
```

El comando fm_shell inicia el entorno de shell de Formality. Desde aquí, inicie la interfaz gráfica de usuario (GUI) de la siguiente manera:

```
fm_shell (setup)> start_gui
```

Cuando se invoque Formality, comience en el modo **de configuración**. El **programa de instalación** indica el modo en el que se encuentra actualmente cuando se utilizan comandos. Los modos son **configurar**, **coincidir** y **verificar**.

Este trabajo de laboratorio incluye las siguientes secciones:
0. Guía (Cargar archivo de configuración automatizada)
1. Referencia (especifique el diseño de referencia)
2. Implementación (Especifique el diseño de implementación)
3. Configurar (Configurar el diseño)
4. Match (Match Compare Points)
5. Verificar (Verificar los diseños)
6. Depurar.



**Tareas de laboratorio**

**0. Guidance (Cargar archivo de instalación automatizada)**

Antes de especificar los diseños de referencia e implementación, se puede cargar opcionalmente un archivo de instalación automatizada (.svf) en Formality. El archivo de configuración automatizado ayuda a Formality a procesar los cambios de diseño causados por otras herramientas utilizadas en el flujo de diseño. Formality utiliza este archivo para ayudar al proceso de verificación y coincidencia de puntos de comparación. Para cada archivo de instalación automatizada que se carga, Formality procesa el contenido y almacena la información para su uso durante el período de coincidencia de puntos de comparación basado en nombres.

Si a Formality se le da el resultado de DC, entonces trabaje en el directorio **pre_lay**, y si a Formality se le da el resultado de ICC, entonces trabaje en el directorio de **post_lay**. Este laboratorio muestra un ejemplo donde Formality trabaja con el resultado de ICC. Para ejecutar Formality, que funciona con el resultado de DC es similar a estos pasos de laboratorio.

Inicie la interfaz gráfica de usuario (GUI) de Formality desde el directorio de trabajo, que se encuentra en el directorio post_lay. Para iniciarlo, introduzca:

```
% fm_shell -gui
```

Esto abre la ventana gui de nivel superior de Formality (Fig. 1)..

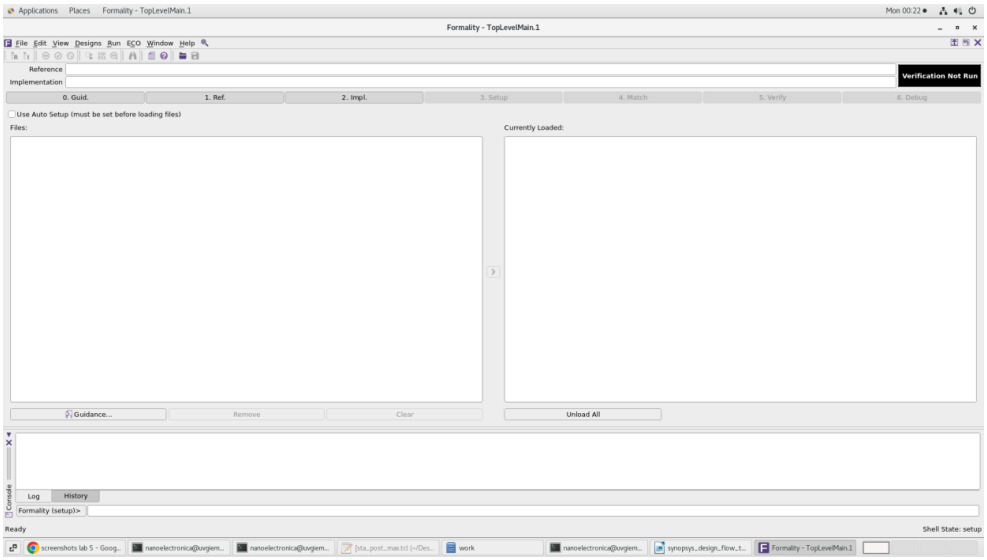



Figura 1. La ventana principal de la GUI de Formality

Utilice el script, creado para llevar a cabo comandos complejos. Hay un script en el directorio **de scripts**, que se encuentra en el directorio **post_lay**. El nombre de este script es **script.tcl,** que utiliza de source el siguiente comando:

```
Formality(verify)> source ../scripts/script.tcl
```

Ahora entienda las secciones generales de Formality.

1. **Referencia (Especifique el diseño de referencia)**

Especificar el diseño de referencia implica leer en archivos de diseño, opcionalmente leer en bibliotecas de tecnología y establecer el diseño de nivel superior.

El diseño de referencia es el diseño con el que se compara el diseño transformado (de implementación). El diseño de referencia es el archivo de origen RTL denominado archivo johnson.v. Haga click en abrir y cargar este archivo (Fig. 2).

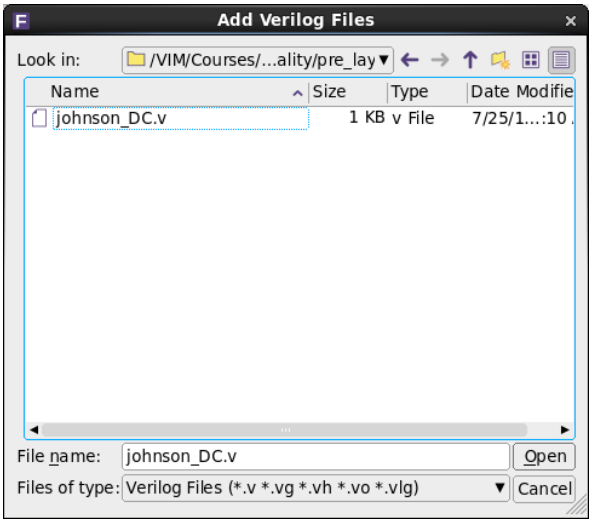



Figura 2. Configurar el nivel de compuerta de CC Fuente

Origen del archivo de base de datos. Seleccione ./ref/db_nldm/ saed14rvt_tt0p8v25c. db haga clic en Abrir y cargar este archivo (Fig. 3).

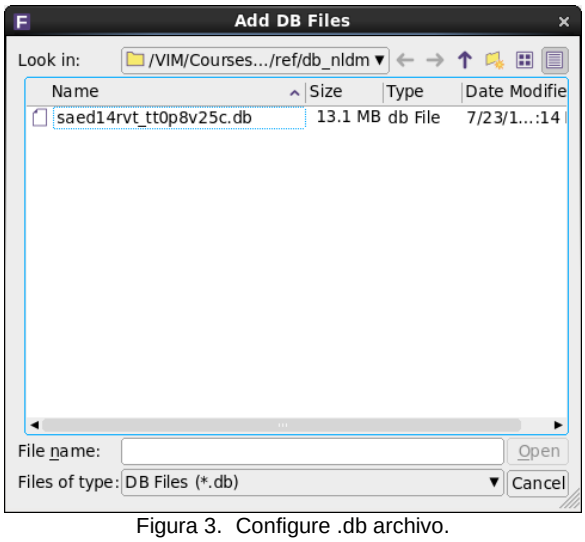



Figura 3. Configure .db archivo.

Set Top Diseño de referencia. Seleccione Library WORK, elija un diseño johnson y Set y haga clic en el botón Set Top (Fig. 4).

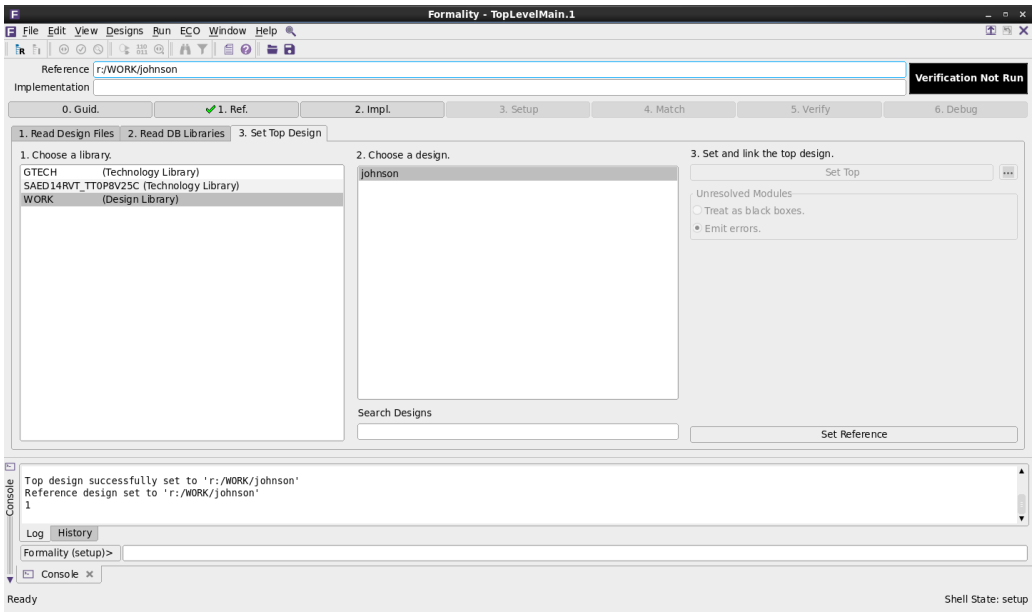



Figura 4. Decodificador

## 2. Implementación (especifique el diseño de implementación)

El procedimiento para especificar el diseño de implementación es idéntico al de especificar el diseño de referencia. El diseño de implementación es la netlist de nivel de compuerta después de que Design

Compiler  nombró a este archivo source file  johnson.v. Clamer Abrir y cargar este archivo (Fig. 5).

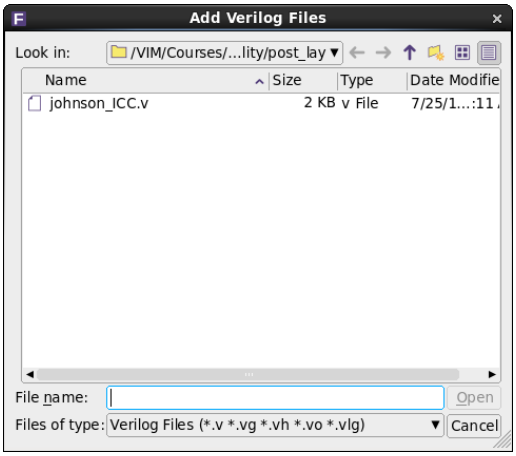



Figura 5.  Configurar la puerta ICC nivel Verilog

Origen del archivo de base de datos. Seleccione ./ref/db_nldm/ saed14rvt_tt0p8v25c. db. Haga click en abrir y cargar este archivo (Fig. 6).

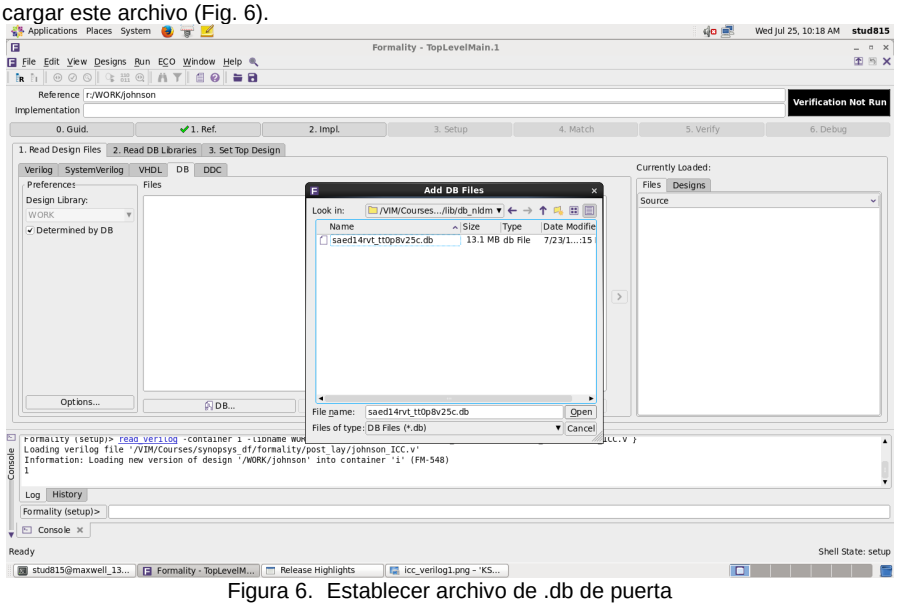



Figura 6.  Establecer archivo de .db de puerta

Establezca el Diseño superior de implementación. Seleccione Library WORK, elija un diseño johnson y Set y haga clic en el botón Set Top y cargue este archivo.

### 3. Configurar (Configurar el diseño)

Para configurar el diseño, haga clic en 3. configuración (Fig.7 y Fig.8)..

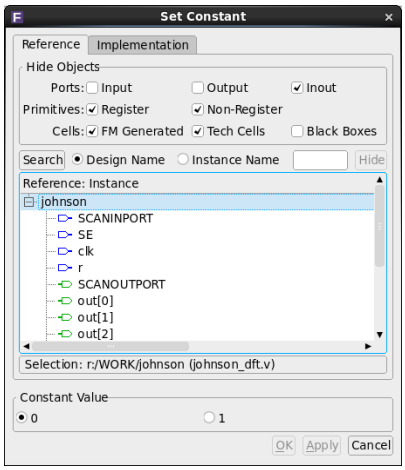



Figura 7. Configurar la ventana Diseño

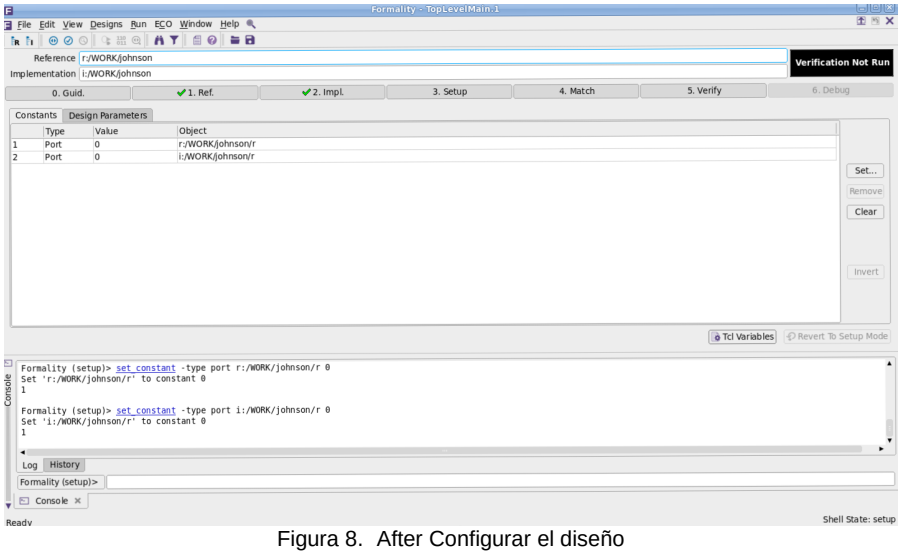



Figura 8. After Configurar el diseño

#### 4. Match (Match Compare Points)

Los puntos de comparación de coincidencias son el proceso mediante el cual Formality segmenta la referencia y
diseños de implementación en unidades lógicas, llamadas conos lógicos.

Para hacer coincidir los puntos de comparación entre johnson.v y johnson.v (después de ICC), haga lo siguiente:
En la ventana principal, haga clic en 4. Partido por partido compara puntos.

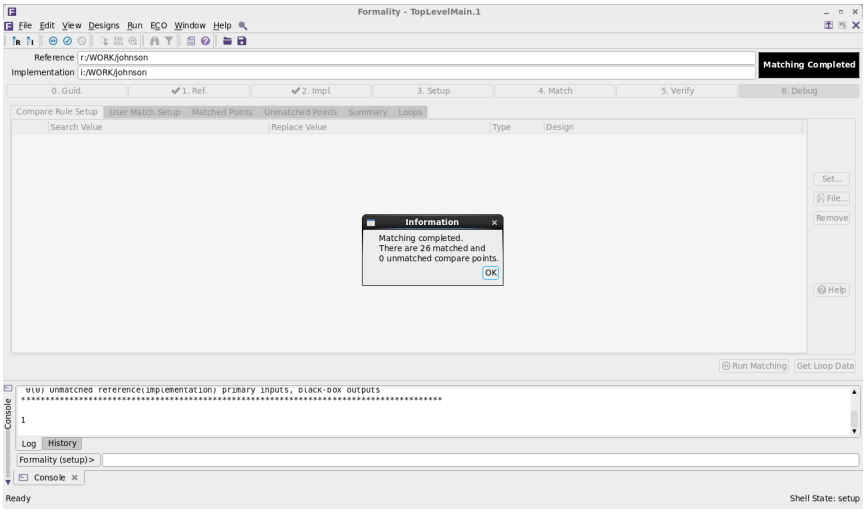



Figura 9. Puntos de comparación de partidos

Como se ve en el masaje que se muestra en la Fig.9, hay 0 puntos de comparación inigualables.

### 5. Verificar (Verificar los diseños)

Cuando se utiliza el comando verify, Formality intenta probar la equivalencia de diseño entre un diseño de implementación y un diseño de referencia. En esta sección se describe cómo Comprobar un diseño o un único punto de comparación, así como la forma de realizar la jerarquía tradicional
verificación.
En la barra de herramientas principal, haga clic en la pestaña Verificar y, a continuación, haga clic en Verificar.

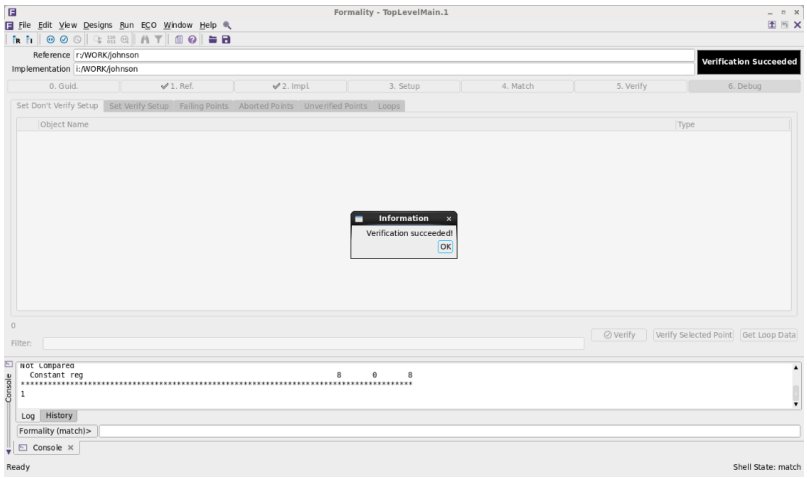



Figura 10. Verificar el diseño

Verificación del proceso completado correctamente.

## 6. Depurar

Durante la depuración, se deben encontrar los puntos exactos en los diseños que exhiben la diferencia en la funcionalidad y luego corregirlos (Fig. 11).

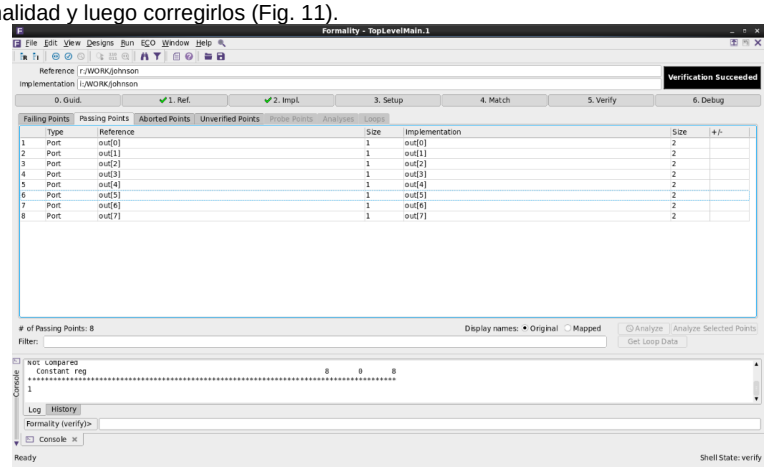



Figura 11. Depuración del diseño

Formality es capaz de mostrar simultáneamente vistas de referencia e implementación Verilog y marcar diferencias y/o similitudes (Fig. 12).

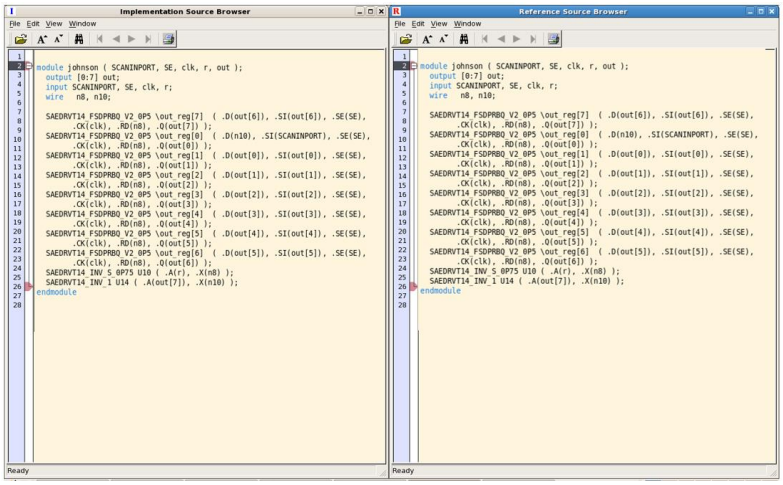



Fig.12 a). Implementación Verilog
Figura 12(b). Referencia Verilog

Además, Formality puede mostrar la vista esquemática y resaltar el objeto de referencia en ella (Fig. 13).

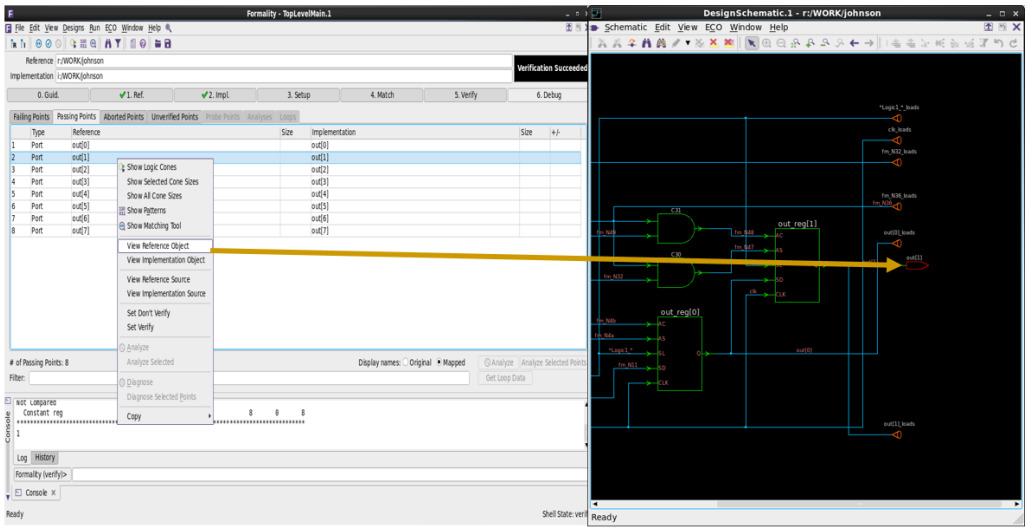



Figura 13. Ver objeto de referencia en esquema

Los resultados de Output del diseño se almacenan en el directorio ../.**location/results/** .

Para salir de Formality, escriba exit en la línea de comandos.

```
Formality (verify)> exit
```

# CAPÍTULO 14

## Glosario

**RTL:** "Register-Transfer-Level": es una abstracción en el diseño de circuitos que consiste en la representación de un circuito digital síncrono en términos del flujo de las señales digitales entre registros de hardware.

**Testbench:** Es una entidad en un entorno de pruebas que verifica la corrección funcional de un diseño de hardware. Es esencialmente un programa que se ejecuta en un simulador de hardware que genera señales de estímulo para el diseño bajo prueba (DUT, por sus siglas en inglés "Design Under Test") y analiza las salidas que produce para verificar su comportamiento correcto contra las especificaciones.

**TSMC:** es el mayor fabricante de semiconductores por contrato del mundo, especializado en la producción de chips y tecnologías de procesos avanzados para clientes que diseñan sus propios productos de silicio.

**VHDL:** Es un lenguaje de especificación definido por el IEEE utilizado para describir circuitos digitales y para la automatización de diseño electrónico.