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 de *Formality* y *Design Compiler*

Trabajo de graduación presentado por Helder Arnoldo Ovalle Barrios para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

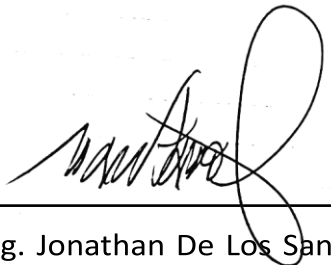
Diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC: Implementación de un alternativo flujo de diseño proporcionado por Synopsys con las herramientas de *Formality* y *Design Compiler*

Trabajo de graduación presentado por Helder Arnoldo Ovalle Barrios
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

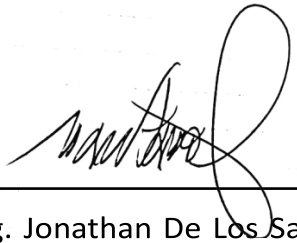
Vo.Bo.:



(f) _____

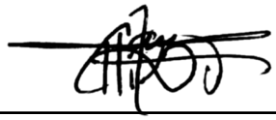
Ing. Jonathan De Los Santos

Tribunal Examinador:



(f) _____

Ing. Jonathan De Los Santos



(f) _____

M. Sc. Carlos Esquit

(f) _____

Ing. Juan Ricardo Girón Rubio

Fecha de aprobación: Guatemala, 08 de 12 de 2022.

Prefacio

La elaboración de esta tesis ha sido posible gracias al apoyo colaborativo de la Universidad del Valle y familia, estando conmigo desde el primer día de carrera hasta el último motivándome.

Quiero agradecer, en primer lugar, al catedrático Jonathan De Los Santos; su orientación y enseñanzas en el campo de diseño e innovación, sin el cual no pudiera haberse llevado a cabo. En segundo lugar, quiero mostrar mi más sincero agradecimiento al Ing. Carlos Esquit por su magnífica dirección y compañía a lo largo de los últimos años trabajando en la carrera de electrónica. En tercer lugar, es preciso agradecer a mis compañeros de departamento; Luis Gómez, Carlos Letona y Diego Equite, por sus consejos y observaciones experimentales. En el plano personal, no puedo olvidarme de todas aquellas personas que me han acompañado hasta la conclusión de esta tesis. Primeramente, agradezco a mis padres, y a toda mi familia su incondicional apoyo, en todos los sentidos posibles, finalmente, a mis amigos, un largo etcétera que me muestra, cada día, lo afortunado que soy; a Luis, Carlos, Rodrigo, Marco, Mario, Diego, Allison, Israel y José por estar a mi lado y en la distancia. Espero que si alguien queda fuera de esta breve lista sepa excusarme. A todos ellos reitero mi más sincero agradecimiento

Índice

Lista de figuras

Lista de cuadros

Resumen

En el presente trabajo se explica el proceso que se llevó a cabo en la implementación de un nuevo flujo de diseño para la creación de un nanochip con tecnología de 180 nm, proporcionado por Synopsys, apoyándonos con las librerías de TSMC. El propósito de un nuevo flujo de diseño es poder dejar esta implementación para iteraciones nuevas con las siguientes generaciones. Como resultado se obtuvieron más ventajas al momento de la creación de un nanochip, gracias a la guía estructurada de pasos y así mismo por tener herramientas no descontinuadas y actualizadas.

Para realizar la implantación del flujo de diseño en este trabajo, se utilizaron las herramientas provisionadas por la empresa de Synopsys. Se necesitaron de varias herramientas con las cuales, en su mayoría, fueron utilizadas con anterioridad. Sin embargo, hay cuatro nuevas herramientas (*Prime time, Tetra Max, Design Compiler, Formality*), que se repartieron en el grupo de trabajo. Las herramientas a desarrollar en este trabajo son: *Formality* para la etapa de síntesis lógica; comparando los resultados de dos diseños y detectando diferencias inesperadas con el archivo verilog. *Design Compiler* se utilizó en la etapa de síntesis lógica y la optimización de diseño representado una parte pequeña de su lógica.

Abstract

This paper explains the process that we will carry out in the implementation of a new design flow for the creation of a nanochip with 180 nm technology, provided by Synopsys, supported by TSMC libraries. The purpose of a new design flow is to be able to leave this implementation for new iterations with the next generations. As a result, to obtain more advantages at the time of creating a nanochip, thanks to the structured guide of steps and also to have tools that are not discontinued and updated.

To implement the design flow in this work, the tools provided by Synopsys will be used. Several tools will be needed, most of which have been used previously. However, there are four new tools (Prime time, Tetra Max, Design Compiler, Formality), which will be distributed to the working group. The tools to be developed in this work are: *Formality* for the logic synthesis stage; comparing the results of two designs and detecting unexpected differences with the verilog file. *Design Compiler* will be used in the logic synthesis stage and the design optimization represented a small part of its logic.

CAPÍTULO 1

Introducción

Mucho se ha dicho respecto al alcance de la nanoelectrónica en los tiempos contemporáneos. Desde los revolucionarios aportes de Taniguchi Norio hasta Ari Aviram, investigador de IBM, y por Mark Ratner químico teórico durante los años de 1974 a 1988 hasta las recientes experiencias con chips de 4nm. Nuestro entendimiento de la electrónica ha variado tanto, en sentidos tan impredecibles, que sorprende al mundo con investigaciones con geles conductores. La electrónica, en su empeño por descoser sus límites, nos ha llevado a entender y aceptar que la tecnología siempre avanza. Por lo tanto, en el desarrollo del trabajo nos ocuparemos de modo puntual en describir y aplicar los conocimientos bases para fomentar un nanochip.

CAPÍTULO 2

Antecedentes

El trabajo del nanochip "El Gran Jaguar" tuvo grandes avances en el año 2021 con su respectiva promoción, la cual se enfocó en sus etapas de diseño. La promoción presente se encargó de replicar los mismos trabajos de graduación anteriores, además se brindó un apoyo de material escrito, vídeos y una demostración de todo su proceso. Gracias a este aporte se logró replicar dichos trabajos con más facilidad, con las explicaciones de comandos, librerías, funciones, entre otras. El trabajo se desarrolló en 5 etapas las cuales se explicarán a continuación.

2.1. Creación del archivo verilog

El trabajo del nanochip se desarrolló en la herramienta de verilog [1], el desarrollo fue a partir de un circuito funcional. Como es conocido, verilog se encargará de crear el hardware, sin embargo, al ser un trabajo muy largo y repetitivo de las conexiones de compuertas, se decidió trabajar el programa en Python con sus dos textos respectivos y actualizados con la promoción presente. Este código genera un nuevo verilog que posee una nube combinacional con un contador de entrada y un bus de 8 bits de salida. El objetivo de la nube combinacional es tomar un valor del contador y repartir las señales al bus de 8 bits para transformarlo en un binario. Con el apoyo de la herramienta de DVE se lograron observar los binarios, con sus pausas del contador de cada nueva palabra y corroborar que dichos binarios sean equivalentes a lo escrito en ambos textos y mostrarlos cada cierto tiempo.

```
module chip_SP(q_out, reset, clk, EN, clk_s, select);
output [7:0] q_out;
input reset;
input clk;
input [1:0]select;
reg [8:0] contador;
reg [7:0] q;
input EN;
output clk_s;
wire W_1;
wire W_2;
wire W_3;
wire W_4;
wire W_5;
wire W_6;
wire W_7;
wire W_8;
wire W_9;
wire W_10;
wire W_11;
wire W_12;
wire W_13;
wire W_14;
wire W_15;
wire W_16;
wire W_17;
wire W_18;
wire W_19;
wire clk_G;
assign clk_s = clk_G;
AND2 U1(EN,clk_G,W_1);
INV  U2(W_1,W_2);
INV  U3(W_2,W_3);
INV  U4(W_3,W_4);
INV  U5(W_4,W_5);
INV  U6(W_5,W_6);
```

Figura 1: Archivo de verilog generado.

2.2. Síntesis lógica

La síntesis lógica posee dos subetapas dentro de ella, en las cuales se enfocan en obtener tres archivos: ".ddc" ".sdc" ".v".

Primera subetapa, síntesis lógica cell: Se utilizó el último verilog generado por el Python, con el cual se apoyó con la herramienta de Design Vision, para conectar las compuertas que realizarán la funcionalidad del chip, es decir, sintetizar el Core como se ve en la Figura 2.

Segunda subetapa, síntesis lógica cell-IO: Para esta última subetapa teniendo el Core ya sintetizado, se utilizó el mismo archivo de verilog, pero con una modificación, ya que se le agregaron las entradas y salidas de los bloques principales generadas en la anterior subetapa. Con el objetivo de obtener bloques más grandes y crear el layout físico teniendo una vista del esquemático 3 y una más profunda donde vemos las entradas y salidas 4.

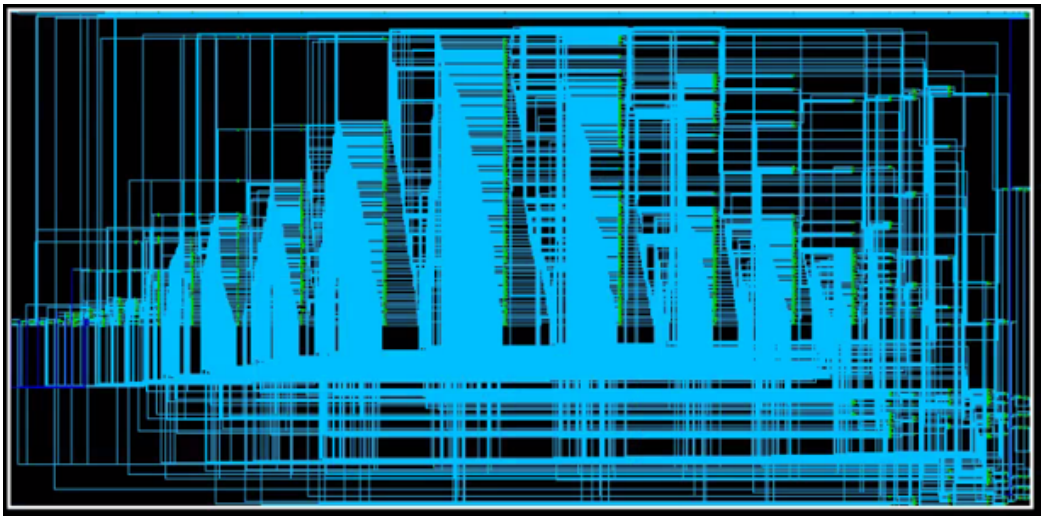


Figura 2: Core sintetizado.

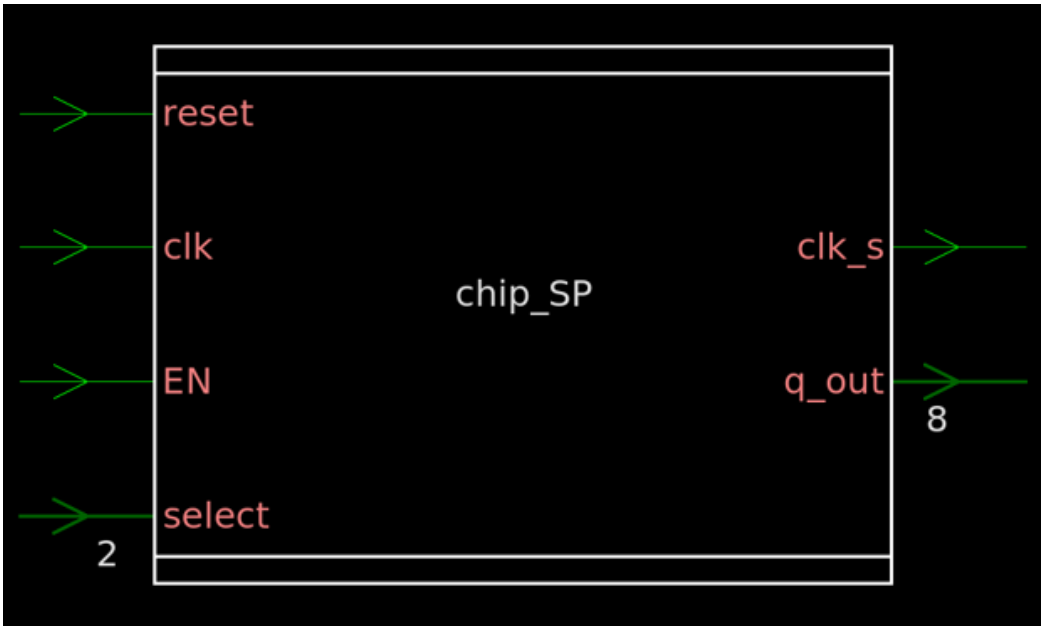


Figura 3: Schematic View.

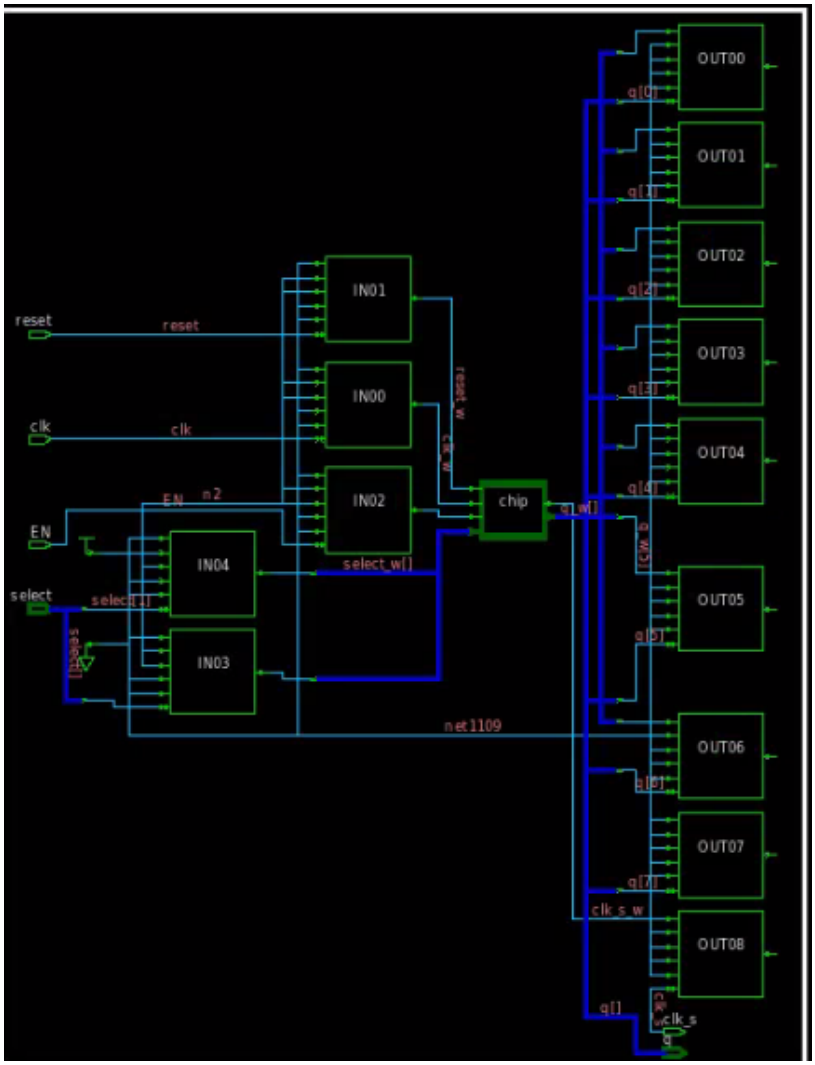


Figura 4: Chip con entradas y salidas.

En la Figura 5 podemos observar el comportamiento en bits de como se muestra en el chip los textos que se describieron en el programa de Python.

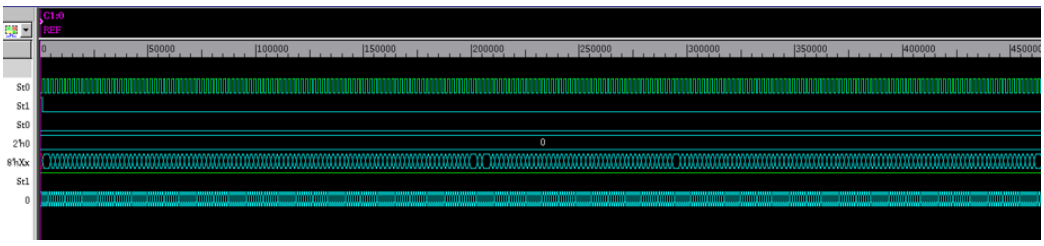


Figura 5: Información de texto en bits.

2.3. Síntesis física

La etapa de síntesis física, es una de las partes más importante para el progreso de este proyecto, ya que aquí se transformará la síntesis lógica a un conjunto de componentes físicos posicionados en silicio y unidos a través de interconexiones 6. Cabe mencionar que para esta etapa se nos fue proporcionado un archivo de comando de ICC2 y haciendo posibles dichas conexiones en el circuito físico.

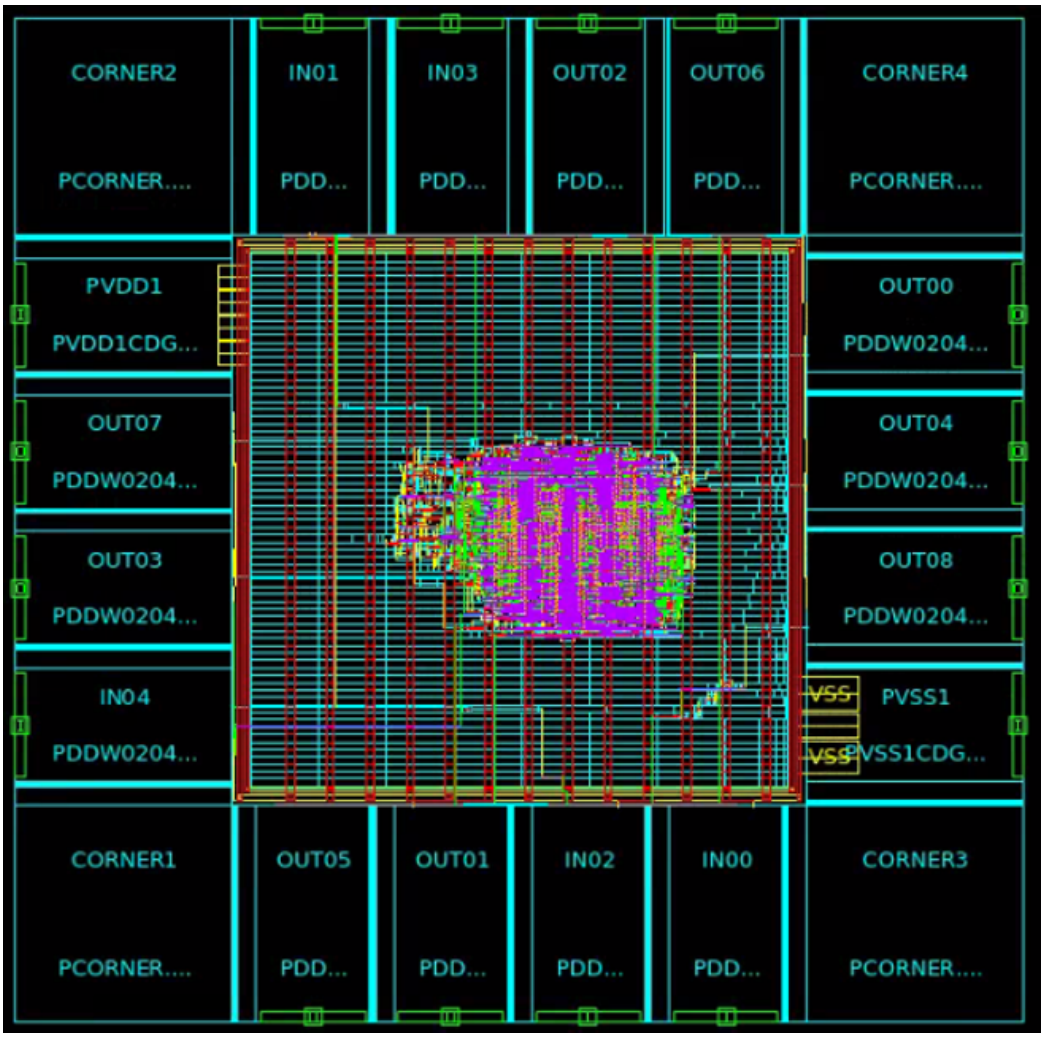


Figura 6: Circuito físico del NanoChip.

2.4. Verificaciones

Esta última parte se compone de tres secciones:

- Rule Check (DRC):

  Cuando se obtiene un layout del circuito a implementar, es necesario cumplir ciertas

restricciones para poder fabricarlo. Este proceso se llama, Design Rule Check (DRC); cuyo objetivo es que el layout generado se compare contra un conjunto de restricciones establecidas por el fabricante con el que se trabaja, es decir, estas condiciones pueden variar dado el fabricante, dichas reglas deben cumplir para que un diseño pueda ser manufacturado.

Figura 7: Verificación de DRC.

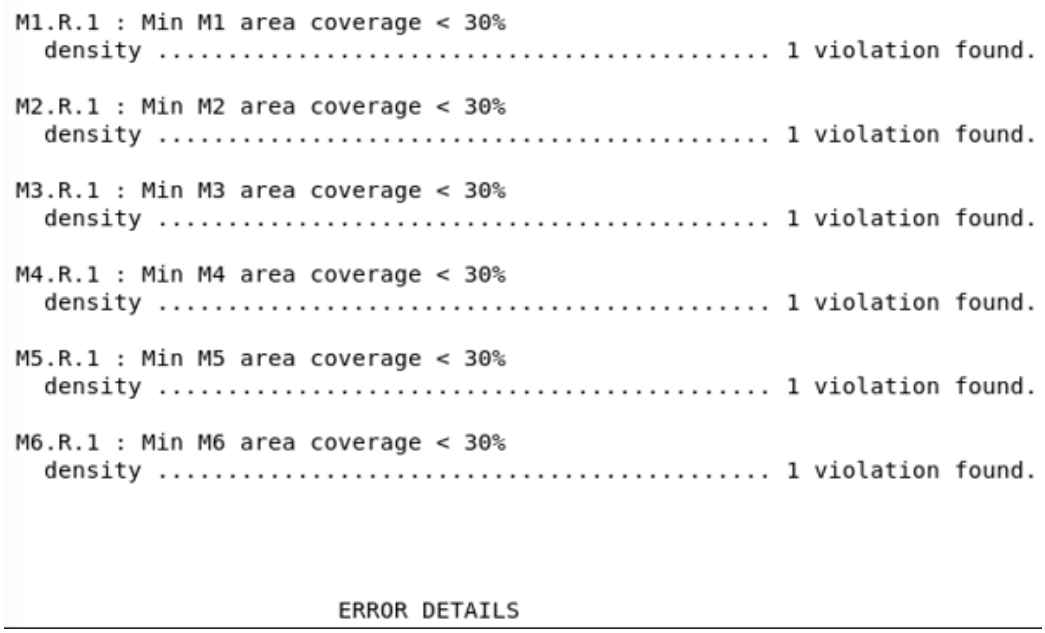


Figura 8: Verificación de DRC 2.

- Electrical Rule Check (ERC): El ERC es una manera de comprobar la robustez de algún diseño en forma de esquemático o de layout. Este utiliza reglas que no son revisadas en el proceso de DRC o el LVS (Layout vs Schematic). Puede revisar el diseño para encontrar dispositivos flotantes, pines flotantes, wells flotantes, entre otras reglas más.

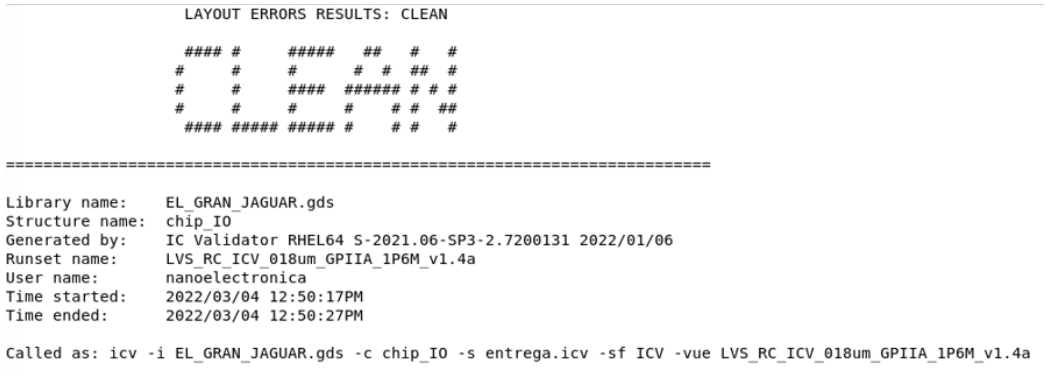


Figura 9: Verificación de ERC.

- Antenna Rule Check (ARC): El Antenna Rule Check se encarga de encontrar conexiones entre metales y gates. Las reglas que utiliza para verificar el diseño se basan en una proporción entre el área del metal que se está conectando y el área del gate del transistor. Mientras más área se tenga en el gate, se permite una mayor cantidad de carga acumulada antes de que ocurra una descarga.

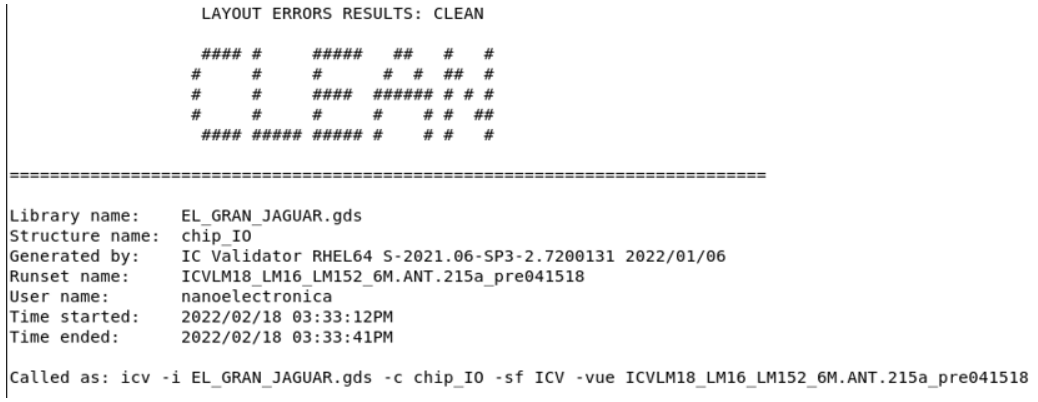


Figura 10: Verificación de Antenna.

2.5. Extracción de parásitos

Como lo dice su nombre, se encarga de extraer los parásitos generados de los componentes geométricos a nivel nanométrico, sus parámetros eléctricos, es decir, capacitancias y resistencias. Para esta verificación del circuito, se ingresa un netlist y con la ayuda de la herramienta de StarRC se genera un HSPICE con todas estas capacitancias y resistencias parásitas presentes en el layout.

```
*
*|DSPF 1.3
*|DESIGN chip_IO
*|DATE "Fri Mar 11 01:51:28 2022"
*|VENDOR "Synopsys"
*|PROGRAM "StarRC"
*|VERSION "S-2021.06-SP4"
*|DIVIDER /
*|DELIMITER :
**FORMAT NETNAME
*

** COMMENTS

** OPERATING_TEMPERATURE 25
** DENSITY_OUTSIDE_BLOCK 0
** GLOBAL_TEMPERATURE 25
**
** TCAD_GRD_FILE /home/nanoelectronica/Desktop/Trabajos/LPE/cm018g_1p6m_4x1u_mim5_40k_cbest.nxtgrd
**   TCAD_TIME_STAMP Tue Apr 23 22:51:06 2019
**   TCADGRD_VERSION 62
```

Figura 11: HSPICE con capacitancias y resistencias.

```
.SUBCKT chip_IO

*|GROUND_NET 0

*|NET ln_N_10 0.0450721PF
*|I (ln_N_10:F1 chip/U212 ln_N_4 B 0 238.9775 216.8400)
*|I (ln_N_10:F2 chip/U203 ln_N_4 B 0 244.5750 248.2000)
*|I (ln_N_10:F3 chip/U220 ln_N_7 B 0 246.5200 244.6200)
*|I (ln_N_10:F4 chip/U393 ln_N_5 B 0 281.5050 295.2400)
*|I (ln_N_10:F5 chip/U344 ln_N_4 B 0 258.0150 212.9200)
*|I (ln_N_10:F6 chip/U164 ln_N_4 B 0 285.1325 200.8875)
*|I (ln_N_10:F7 chip/U40 ln_N_6 B 0 248.4650 240.3600)
*|I (ln_N_10:F8 chip/U492 ln_N_5 B 0 248.4800 286.7250)
*|I (ln_N_10:F9 chip/U565 ln_N_7 B 0 252.9600 291.8800)
```

Figura 12: HSPICE con capacitancias y resistencias 2.

CAPÍTULO 3

Justificación

En la Universidad del Valle de Guatemala (UVG), los estudiantes de Ingeniería Electrónica poseen una característica a diferencia de las demás universidades; es su conocimiento en el área de la nanoelectrónica. El objetivo principal es, al finalizar el proyecto del nanochip, concluir con la implementación de un nuevo flujo de diseño propuesto por Synopsys. Representado a Guatemala como el primer país centroamericano en crear un nanochip.

Por consiguiente, sabiendo la importancia de finalizar el proyecto, se buscará un nuevo flujo de diseño documentado por Synopsys, para tener otra alternativa al momento de la creación de un nanochip; adicional a lo anterior, con dicha elaboración se espera tener grandes aportes al aprender un flujo más sencillo que permita evitar errores de densidad para el fabricante y diseñador. Se dejarán documentaciones de usos de las herramientas, vídeos de instalaciones, etc. Posteriormente, al tener la alternativa recomendada como se ha mencionado anteriormente, garantizar un correcto funcionamiento para interacciones de las nuevas generaciones.

Al mandar a fabricar el nanochip se puede invertir tiempo únicamente en mantener actualizado el flujo de diseño y permitir que aquellos que se quieran beneficiar de este, puedan crear proyectos de alto nivel en los departamentos de Electrónica, Mecatrónica y Biomédica de la UVG. Además, a largo plazo, que otras carreras puedan colaborar de forma interdisciplinaria para la creación de proyectos más ambiciosos y con un mayor alcance para beneficio de la comunidad UVG y del país.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

- Continuar el proyecto de diseño de un nanochip de 180 nm implementado una propuesta alternativa de flujo de diseño dada por Synopsys, usando dos diferentes aplicaciones con distintas características al diseño original, utilizando las librerías educativas de Synopsys y replicando el proceso con las librerías de TSMC.

4.2. Objetivos específicos

- Documentar una guía de la herramienta Formality para realizar diversas comparaciones de diseño y evaluar si hubo algún cambio en la etapa del flujo de diseño exitosamente.
- Documentar una guía de la herramienta Design Compiler para realizar la síntesis lógica y optimización del circuito en la etapa del flujo de diseño exitosamente.
- Documentar las instalaciones de las dos nuevas herramientas de Formality y Design Compiler para iteraciones de otros años.
- Creación e implementación de decks para la simulación y manufactura del circuito.
- Mantener una comunicación efectiva con los otros grupos de trabajo y miembros del mismo grupo.
- Resolver tres errores de densidad de los 6 que se obtuvieron en la iteración pasada en la línea de investigación.
- Actualizar el flujo de diseño provisionado por Synopsys.

CAPÍTULO 5

Alcance

El nuevo flujo de diseño se podrá utilizar para las siguientes generaciones, con la finalidad de garantizar un mejor desarrollo y fabricación de un nanochip, con tecnologías de 14nm, 90nm y 180nm. El desarrollo de este será más eficiente gracias a las guías proporcionadas por Synopsys y actualizadas por el grupo encargado, teniendo una mejor manufactura de las herramientas aprovechando de mejor manera las áreas de cada una de ellas. Con esto se evitarán problemas de densidad debido a que estas herramientas no son descontinuadas.

CAPÍTULO 6

Marco teórico

Para elaborar y diseñar un chip sin errores, es necesario seguir ciertos pasos para poder agilizar el proceso. El objetivo de su elaboración y diseño es poder verlo funcionar correctamente, con un tiempo y presupuesto apropiado.

6.1. Synopsys

Compañía especializada en la creación de programas con el fin de desarrollar y realizar verificación de circuitos de nanométricos fabricados en silicio[1]. Con la cual se apoyará elaborando un flujo de diseño como el que se presenta en la siguiente Figura. 13

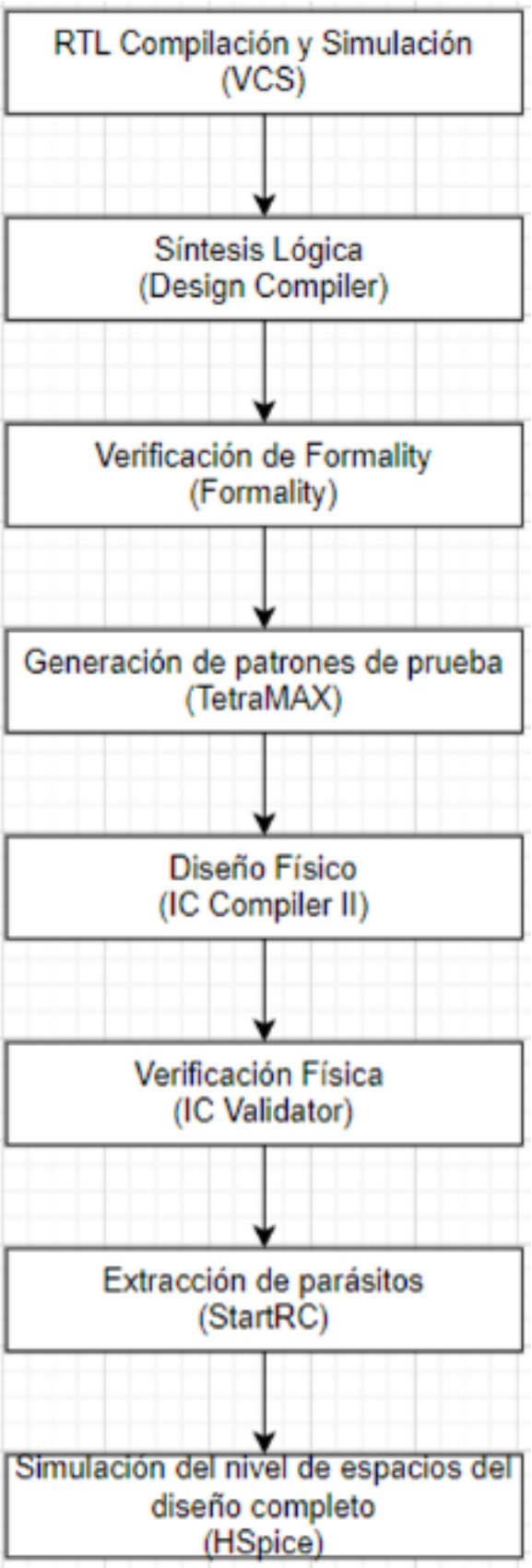


Figura 13: Flujo de diseño

Se logró observar que este flujo está compuesto por nueve herramientas, las cuales se explicarán más adelante.

6.2. Logic Simulation (VCS):

La herramienta de VCS,[2] es un simulador de verilog de alto rendimiento y alta capacidad que incorpora tecnologías avanzadas de verificación de abstracción de alto nivel en una única plataforma abierta. Asimismo, incluye una solución de verificación inteligente con tecnología de cobertura avanzada, todo integrado dentro de una plataforma de simulación HDL. Admite verilog, HDL y VHDL.

Para simular un modelo de verilog en VCS se debe realizar a cabo la realización de dos pasos:

1. Compilar los archivos de origen, es decir el archivo verilog compuesto de su circuito o circuitos y su testbench, transformándolos en un archivo binario ejecutable.

2. Ejecución del archivo binario ejecutable(simv).

Este enfoque de dos pasos simula más rápido y utiliza menos memoria que los simuladores interpretativos. En la siguiente figura se muestra el Flujo de trabajo de VCS.

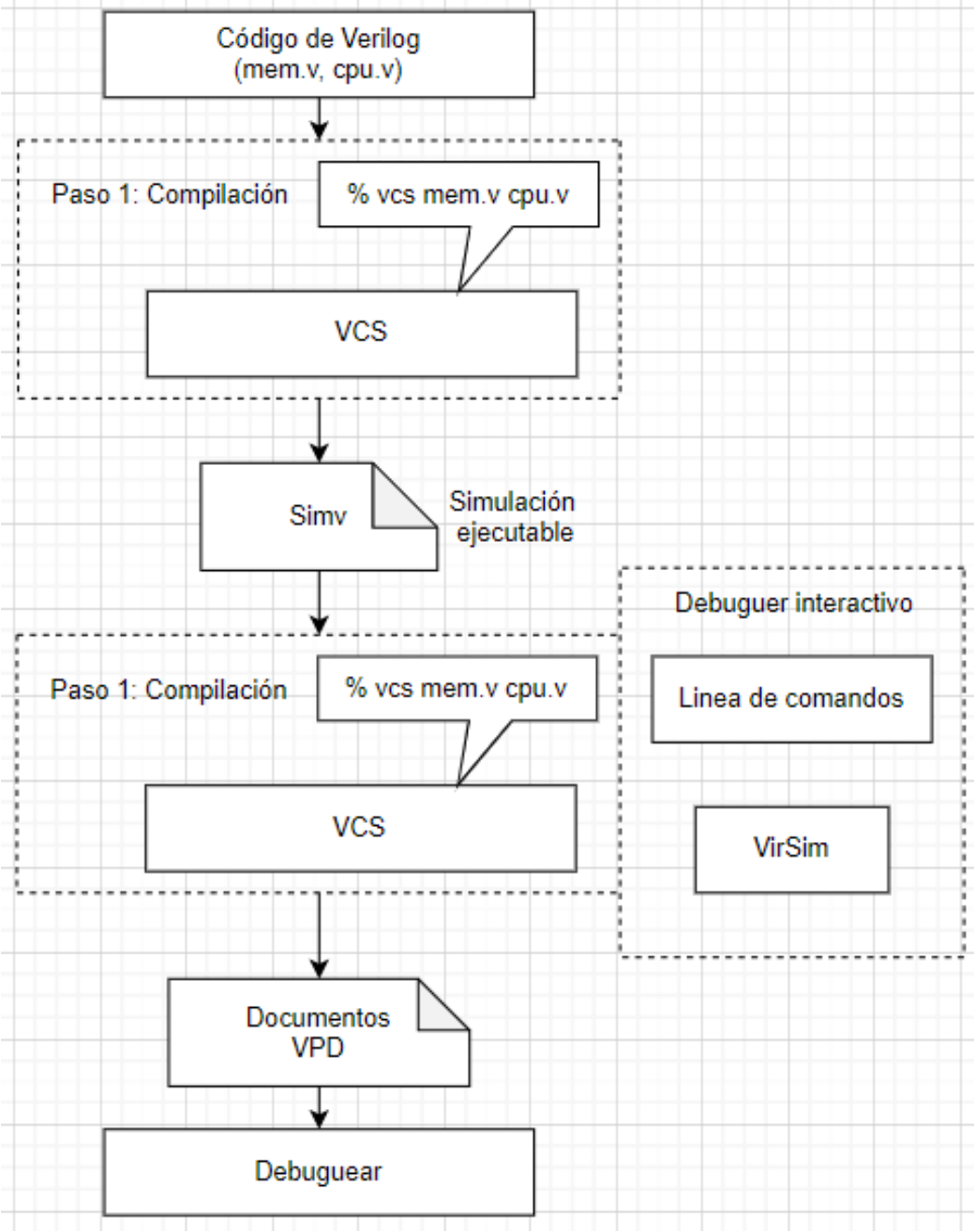


Figura 14: Flujo de trabajo de VCS

Algunos comandos importantes para la preparación de VCS se muestran a continuación:

6.2.1. CMD para inicializar la herramienta

```
dve
```

6.2.2. CMD para compilar archivos de origen

```
vcs nombre_del_archivo_testbench.v nombre_del_archivo_verilog.v
```

6.2.3. CMD para correr una simulación

```
simv -l log
```

6.3. Design Compiler:

En la siguiente herramienta,[3] cabe resaltar que su función es para la síntesis lógica combinacional o secuencial optimizando los diseños para brindar la representación lógica más pequeña y rápida de una función dada. Además comprende herramientas que sintetizan sus descripciones en verilog en diseños optimizados, dependientes de la tecnología y a nivel de compuerta bajo restricciones simultáneas de tiempo y área, también bajo diferentes condiciones de carga, temperatura y voltaje.

Flujo de la herramienta:

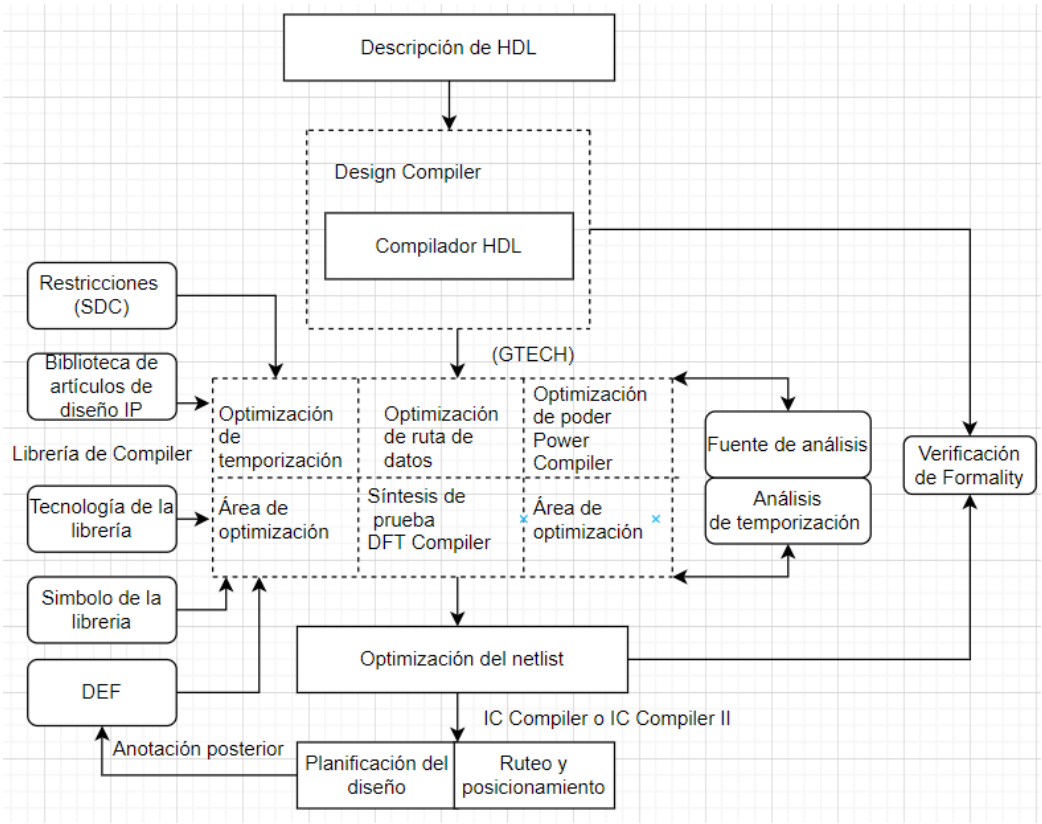


Figura 15: Flujo de trabajo de Design Compiler

Algunos comandos importantes para la preparación de *Design Compiler* se muestran a continuación:

6.3.1. CMD para inicializar la herramienta

```
dc_shell -gui
```

6.3.2. CMD para buscar archivos

```
-f source nombre_de_archivo
```

6.4. IC Compiler II (ICC2):

En la tercera herramienta tendremos ICC2,[4] es para la creación de diseños físicos y síntesis, combinando la planificación de la síntesis física, la síntesis de árbol de reloj y el enrutamiento para la implementación de diseño lógico y físico.

Flujo de la herramienta:

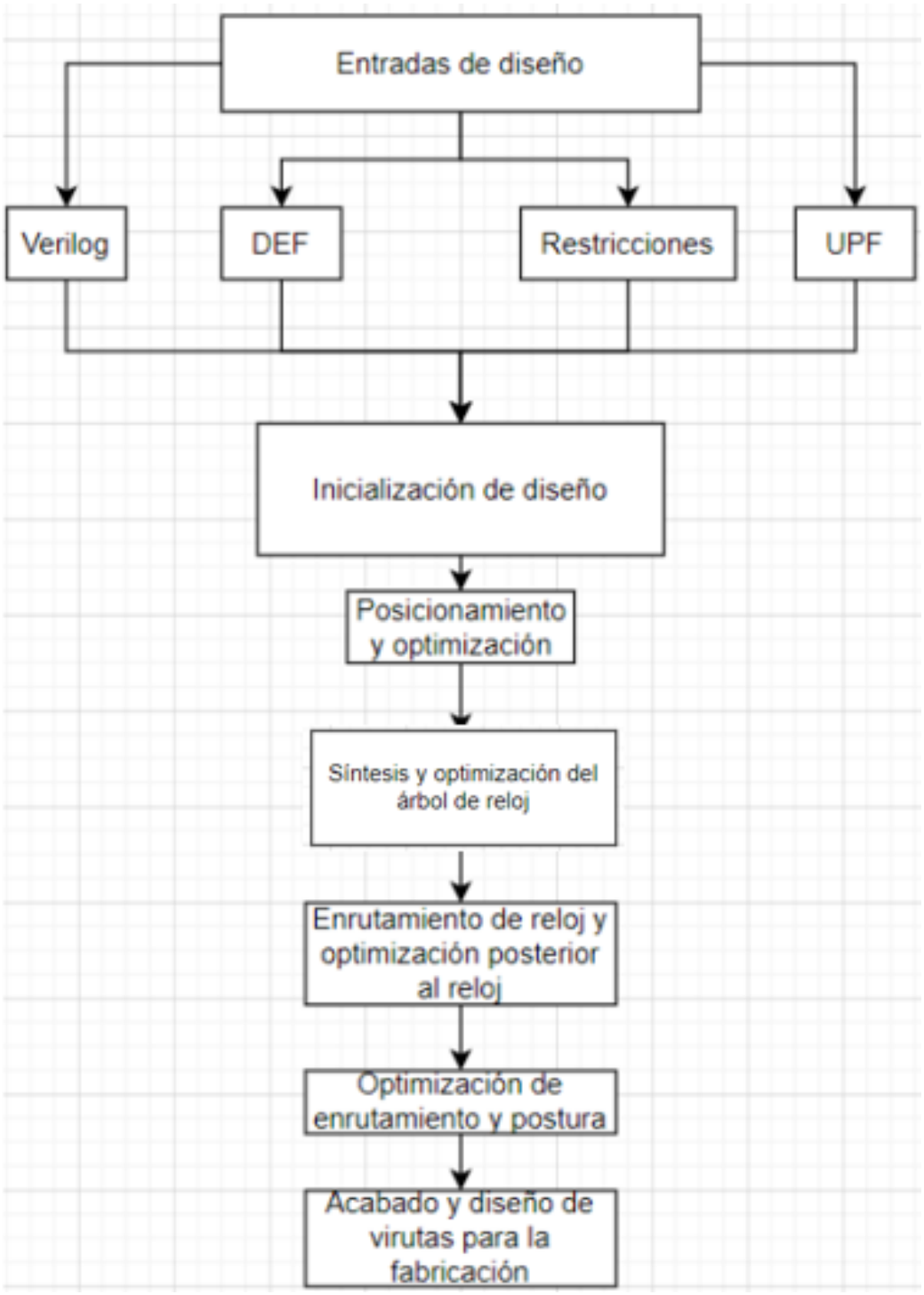


Figura 16: Flujo de trabajo de IC Compiler II

Algunos comandos importantes para la preparación de *IC Compiler II* se muestran a continuación:

6.4.1. CMD para inicializar la herramienta

```
icc2_shell -gui
```

6.4.2. CMD para crear librerías

```
create_lib ../mi_lib_dir/mi_lib
```

6.4.3. CMD para guardar librerías

```
save_lib lib_A
```

6.5. Prime Time:

Esta herramienta[5] se utiliza para realizar análisis de tiempo estático a nivel de compuerta de un chip completo, validando solamente el rendimiento de sincronización de un diseño comprobando todas las rutas en busca de infracciones de tiempo, sin utilizar simulaciones lógicas o vectores de prueba.

Flujo de la herramienta:

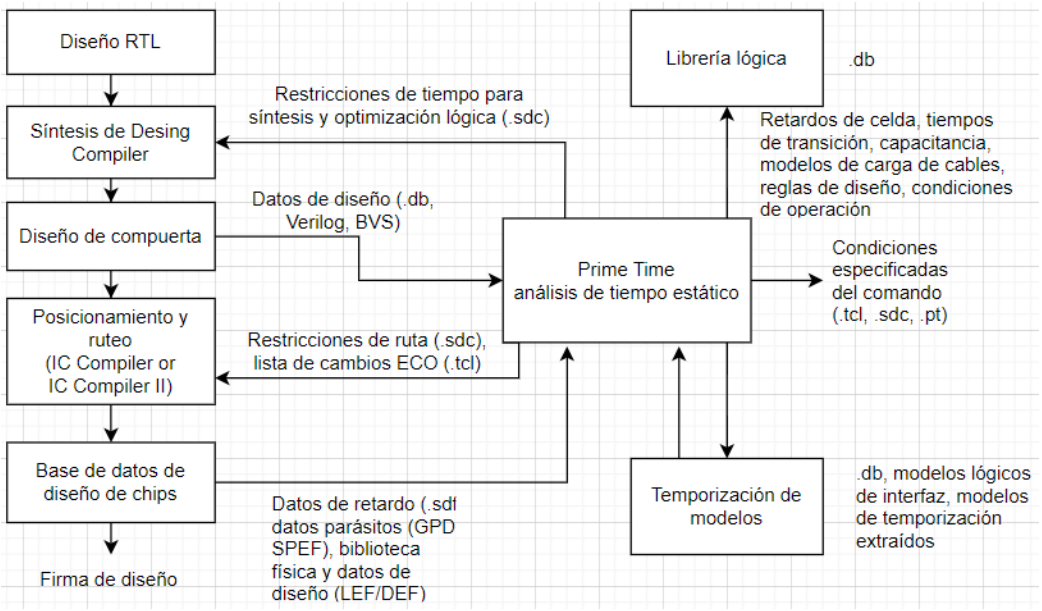


Figura 17: Flujo de trabajo de Prime Time

Algunos comandos importantes para la preparación de *Prime Time* se muestran a continuación:

6.5.1. CMD para inicializar la herramienta

```
pt_shell -gui
```

6.5.2. CMD para leer archivos

```
read_db

read_verilog

read_sdf
```

6.6. Formality:

La funcionalidad de Formality,[6] es proporcionar información acerca de los conceptos como; procedimiento, tipos de archivos, elementos de menú y metodologías. Además de esto su objetivo es determinar si dos vesiones de diseño son funcionalmente equivalentes.

Flujo de la herramienta:

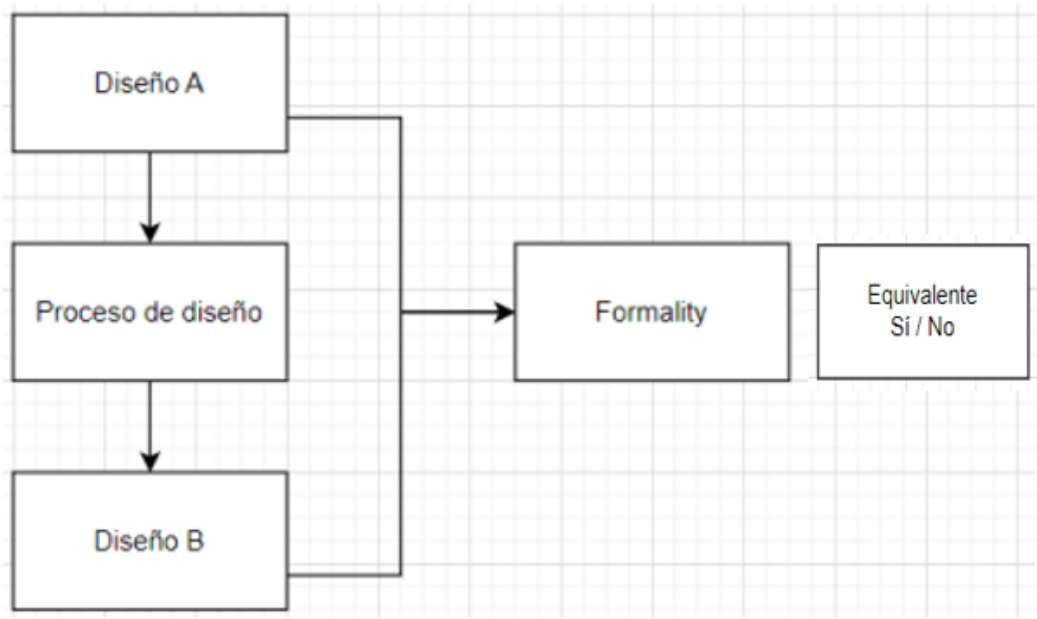


Figura 18: Flujo de trabajo de Formality

Algunos comandos importantes para la preparación de *Formality* se muestran a continuación:

6.6.1. CMD para inicializar la herramienta

```
fm_shell -gui
```

6.6.2. CMD para leer archivos

```
read_verilog -r top.v
```

6.6.3. CMD para compara puntos

```
match
```

6.7. Tetramax:

Para la quinta herramienta se trabajará con Tetramax,[7] se utilizará para comprobar las reglas de diseño de estabilidad; cumpliendo con sus objetivos de calidad y costo de prueba con una velocidad sin precedentes y para generar automáticamente vectores de prueba de fabricación para un diseño lógico.

Flujo de la herramienta:

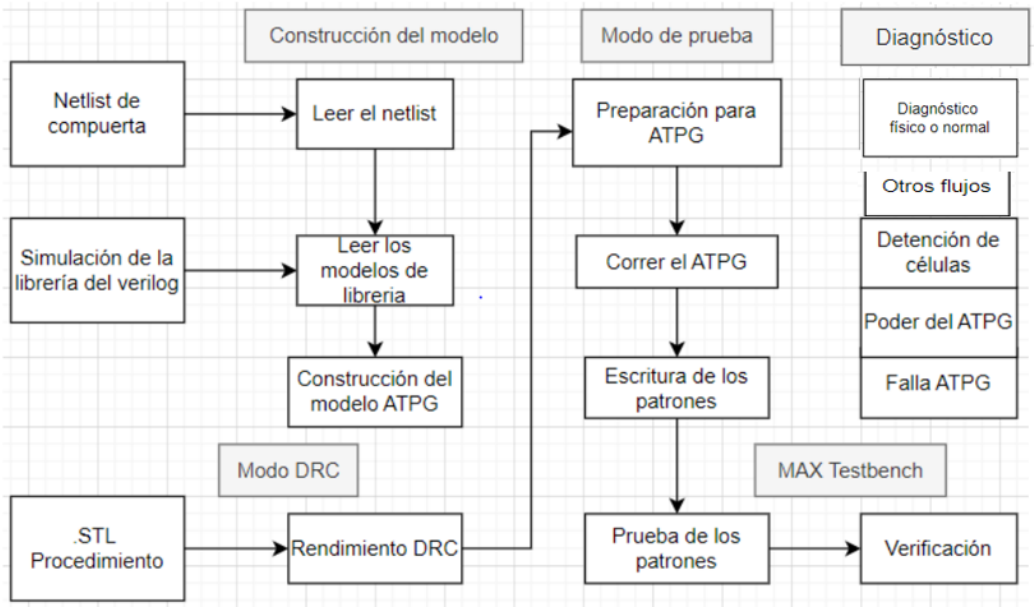


Figura 19: Flujo de trabajo de Tetramax

Algunos comandos importantes para la preparación de *Tetramax* se muestran a continuación:

6.7.1. CMD para inicializar la herramienta

```
tmax -tcl
```

6.7.2. CMD para crear un modelo

```
run_build_model nombre_del_archivo
```

6.8. IC Validator (ICV):

ICV,[8] es una herramienta que se utiliza para establecer una base sólida de uno de los módulos de verificación física en el flujo de diseño de un chip a nano escala, Layout Versus Schematic (LVS) y Design Rule Check (DRC). El proceso de LVS se encarga de verificar la integridad del layout de un circuito integrado mientras que el DRC de que cumpla las reglas del circuito. En el proceso de comparación se llevan a cabo modificaciones globales en los netlists, en donde se generan puntos de equivalencia (celdas equivalentes) y posteriormente se comparan.

Flujo de la herramienta se divide en dos:

- LVS

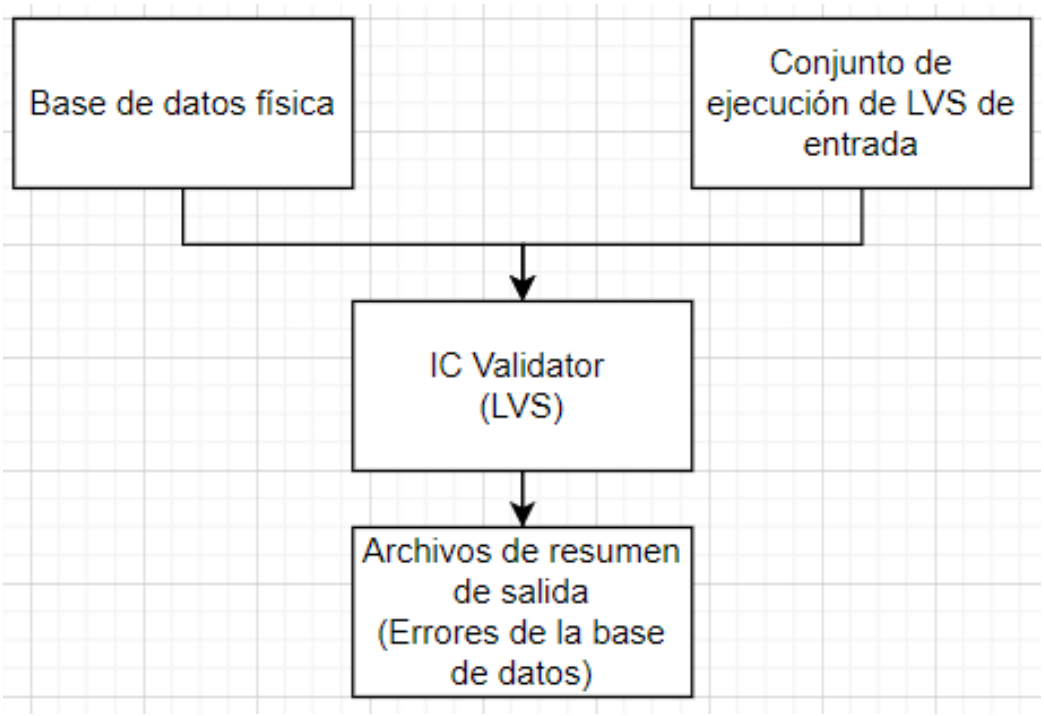


Figura 20: Flujo de trabajo de LVS

- DRC

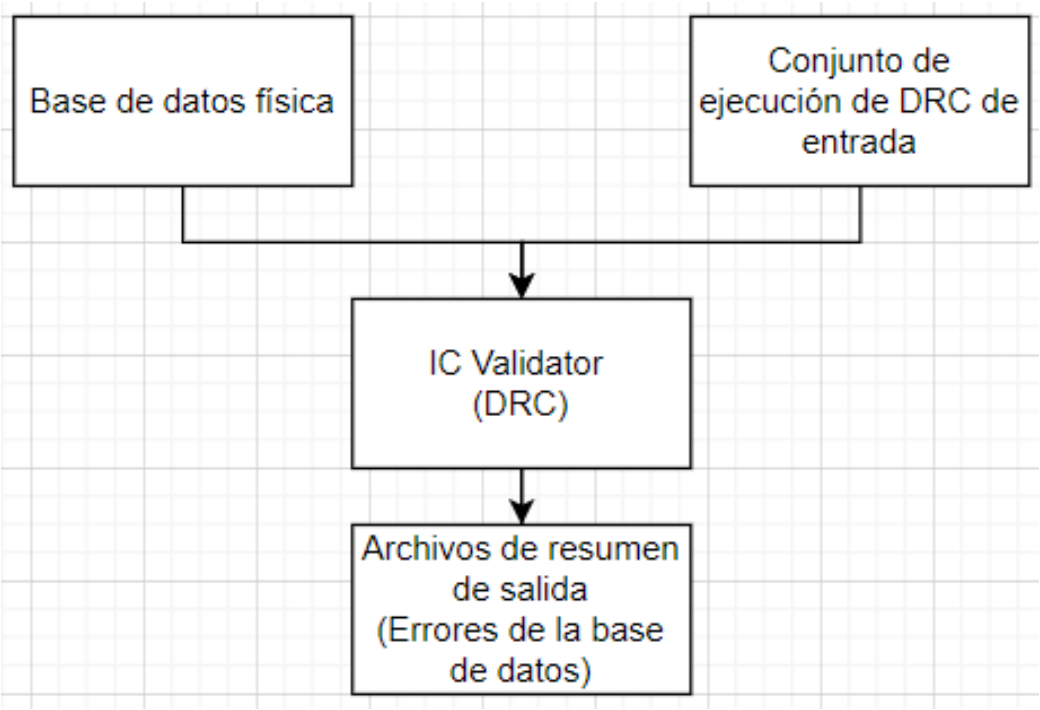


Figura 21: Flujo de trabajo de DRC

Algunos comandos importantes para la preparación de *IC Validator* se muestran a continuación:

6.8.1. CMD para compilar archivos

```
icv nombre_del_archivo
```

6.9. StarRC:

Como octava herramienta tendremos a StarRC,[9] la cual nos servirá para extraer parásitos como resistencias, condensadores e inductores de bases de datos que representan diseños de diseño de circuitos integrados. Se puede generar listas de redes para muchos tipos de análisis, como el tiempo, el ruido y la electromigración.

Flujo de la herramienta:

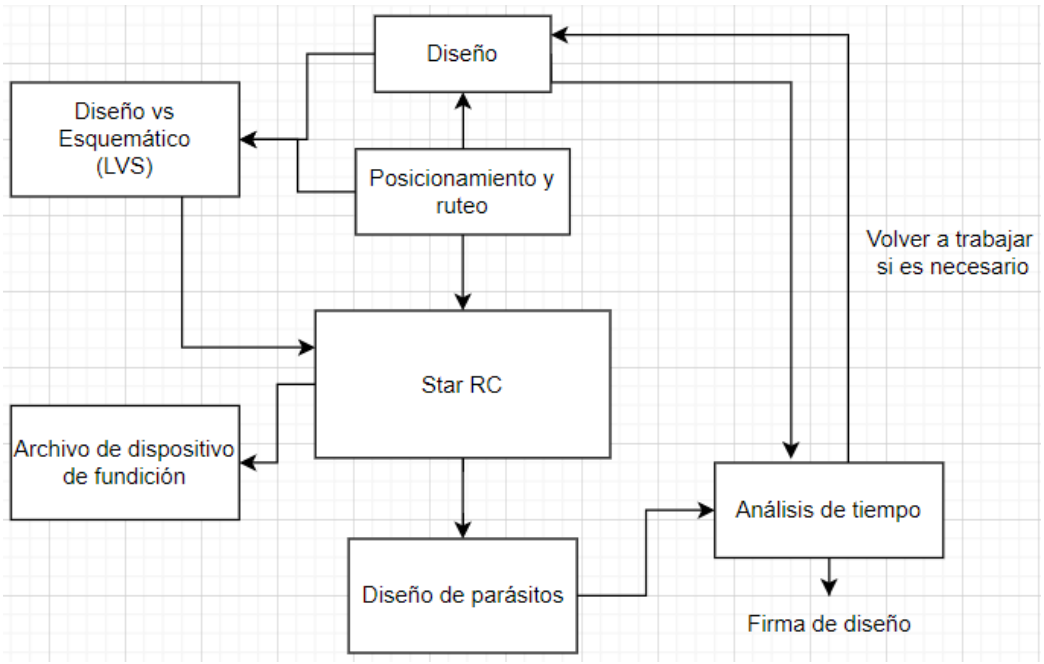


Figura 22: Flujo de trabajo de StartRC

Algunos comandos importantes para la preparación de *StartRC* se muestran a continuación:

6.9.1. CMD para inicializar la herramienta

```
starxtractor nombre_del_archivo.cmd
```

6.10. HSPICE:

Por último, tenemos la herramienta de HSPICE,[10] es un simulador de circuito analógico optimizador. Se utiliza en dominios de estado estacionario, transitorio y de frecuencia. Siendo una simulación rápida y precisa de circuitos y comportamiento. Facilita el análisis a nivel de circuito mediante el uso de Monte Carlo, el peor de los casos, el barrido paramétrico y los análisis de barrido de tabla de datos.

Flujo de la herramienta:

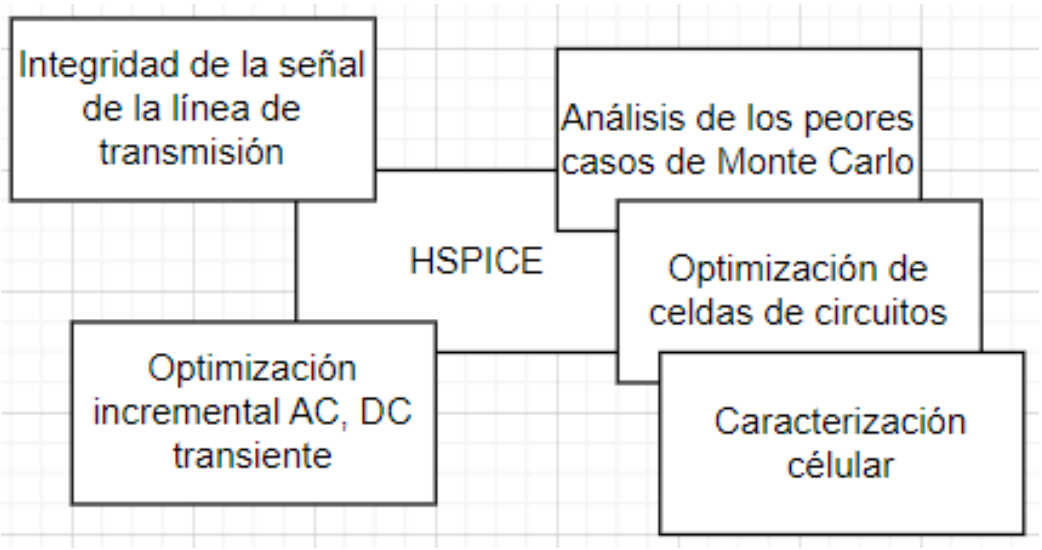


Figura 23: Flujo de trabajo de HSPICE

Algunos comandos importantes para la preparación de *HSPICE* se muestran a continuación:

6.10.1. CMD para compilar archivos

```
hspice nombre_del_archivo.sp > nombre_del_archivo.out
```

CAPÍTULO 7

Librerías educativas de Synopsys

7.1. Simulación lógica en VCS

Para la simulación lógica se trabajó con el puro archivo verilog del Gran Jaguar actualizado junto con su testbench que se actualizó de igual manera al agregarle más cantidad de caracteres y que cumplieran los requisitos necesarios de ambos textos del verilog como se muestra en las siguientes figuras. 24 25

```
module chip_SP(q_out, reset, clk, EN, clk_s, select);
output [7:0] q_out;
input reset;
input clk;
input [1:0]select;
reg [11:0] contador;
reg [7:0] q;
input EN;
output clk_s;
wire W_1;
wire W_2;
wire W_3;
wire W_4;
wire W_5;
wire W_6;
wire W_7;
wire W_8;
wire W_9;
wire W_10;
wire W_11;
wire W_12;
wire W_13;
wire W_14;
wire W_15;
```

Figura 24: Archivo verilog

```
module tb_FINAL;
        reg clk, reset, EN;
        reg [1:0]select;
        wire [7:0] q_out;
        wire clk_s;
        integer i;

        chip_SP
celda(.clk(clk), .reset(reset), .EN(EN), .q_out(q_out), .clk_s(clk_s), .select(select));


        initial begin

                $fsdbDumpfile("final.fsdb");
                $fsdbDumpvars(0, tb_FINAL);

                $dumpfile("final.vcd");
                $dumpvars(0, tb_FINAL);

                clk <= 0;

                for (i = 0; i<13848; i=i+1) begin
                        #1 clk <= ~clk;
```

Figura 25: Archivo testbench

Con el apoyo del flujo nuevo provisionado por Synopsys, fue más eficiente partir al momento de trabajar siguiendo los pasos de su guía, iniciando con compilar los archivos

verilog y testbench 27 y así crear el archivo simv como se ve en la Figura 26. También se pudo obtener una vista del esquemático 32 en el cual observó cada una de sus partes como: clock, selector, reset etc.

```
[nanoelectronica@uvgiemtbmcit11803 vcs]$ vcs circuito.v final_tb.v -full64 -debug_access+all
                        Chronologic VCS (TM)
      Version S-2021.09-SP1-1_Full64 -- Thu Nov  3 20:11:42 2022

                    Copyright (c) 1991 - 2022 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
   or distribution of this software is strictly prohibited.  Licensed Products
     communicate with Synopsys servers for the purpose of providing software
    updates, detecting software piracy and verifying that customers are using
    Licensed Products in conformity with the applicable License Key for such
  Licensed Products. Synopsys will use information gathered in connection with
    this process to deliver software updates and pursue software pirates and
                                  infringers.

 Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
            Inclusivity and Diversity" (Refer to article 000036315 at
                        https://solvnetplus.synopsys.com)

Parsing design file 'circuito.v'
Parsing design file 'final_tb.v'
Top Level Modules:
       tb_FINAL
No TimeScale specified
Starting vcs inline pass...

3 modules and 0 UDP read.
        However, due to incremental compilation, no re-compilation is necessary.
rm -f _cuarc*.so _csrc*.so pre_vcsobj_*.so share_vcsobj_*.so
ld -shared  -Bsymbolic   -o .//../simv.daidir//_cuarc0.so objs/amcQw_d.o
rm -f _cuarc0.so
if [ -x ../simv ]; then chmod a-x ../simv; fi
g++  -o ../simv      -rdynamic  -Wl,-rpath='$ORIGIN'/simv.daidir -Wl,-rpath=./simv.daidir -W
=./  /usr/lib64/libnuma.so.1     _8909_archive_1.so _prev_archive_1.so _cuarc0.so  SIM_l.o
c -lvfs    -lvcsnew -lsimprofile -luclinative /usr/synopsys/vcs/S-2021.09-SP1-1/linux64/lib/
9-SP1-1/linux64/lib/vcs_save_restore_new.o /usr/synopsys/verdi/S-2021.09-SP1-1/share/PLI/VCS
../simv up to date
CPU time: 4.095 seconds to compile + .198 seconds to elab + .148 seconds to link
```

Figura 26: Compilación de ambos archivos

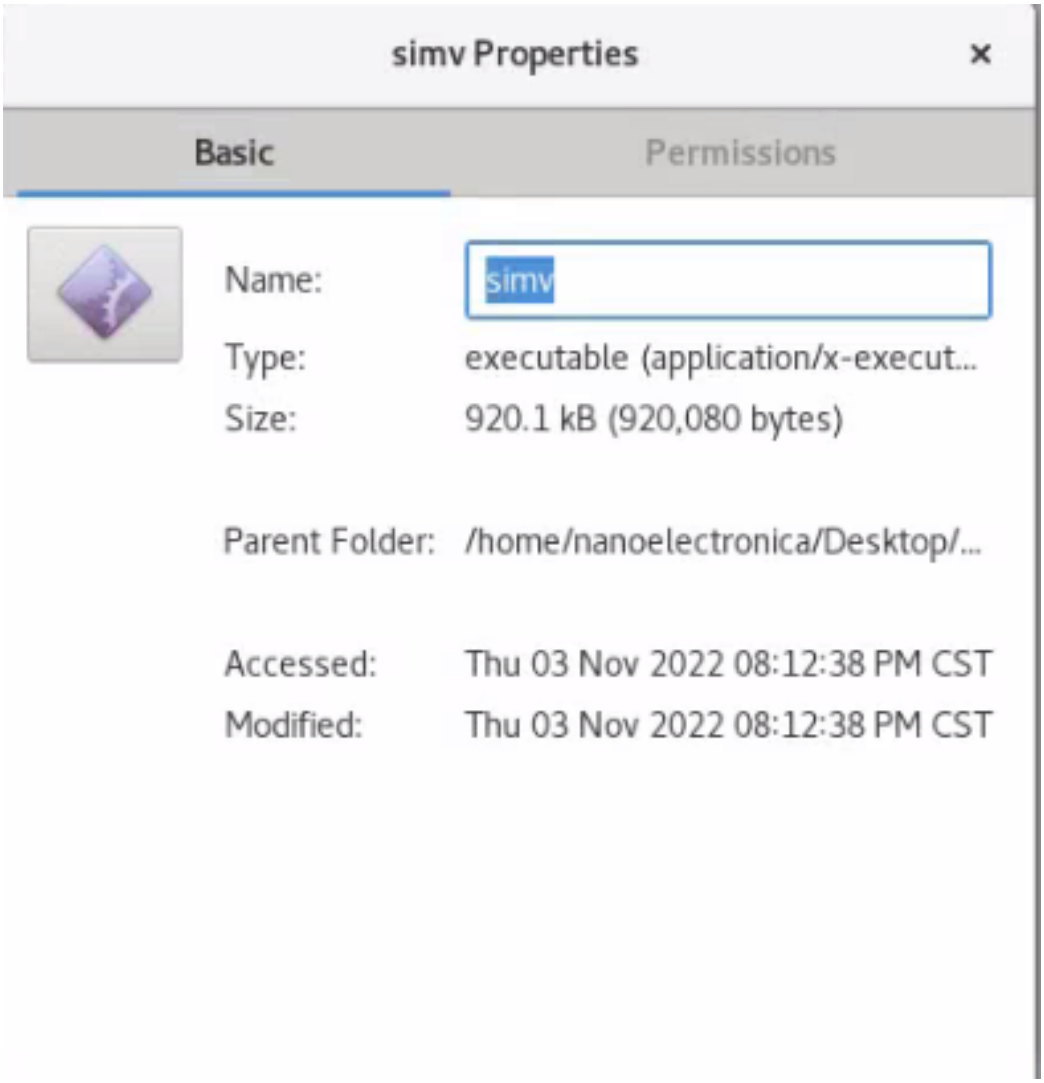


Figura 27: Ejecutable simv

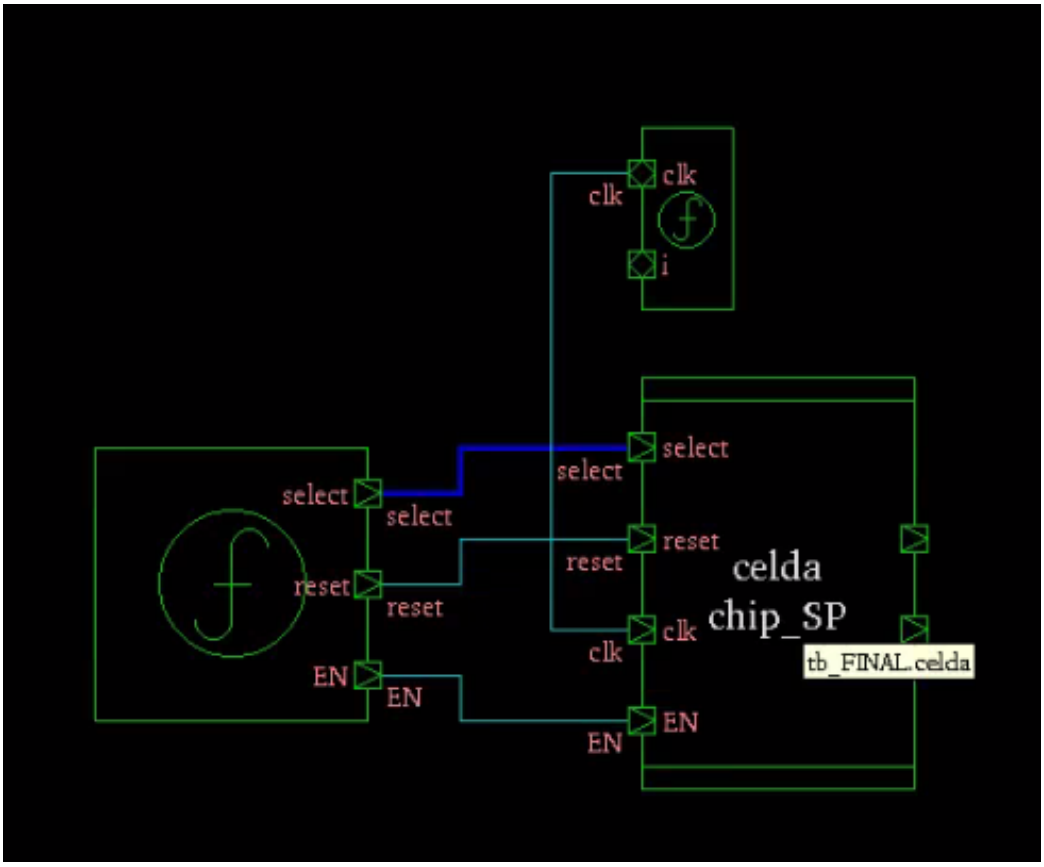


Figura 28: Esquemático

Luego de tener el archivo ejecutable, se configuró en DVE, tal como se observa en la Figura 29, se inició la simulación al agregar las vistas wave luego se ejecutó el simulador en la flecha color azul, en el cual se puede observar el comportamiento del archivo verilog, también así mismo se puede examinar el texto del archivo en hexadecimal 33. Estos valores en hexadecimal se compararon en un convertidor de hexadecimal-texto vía internet, en el cual se obtuvo el resultado esperado comparándolo con los textos del archivo verilog.

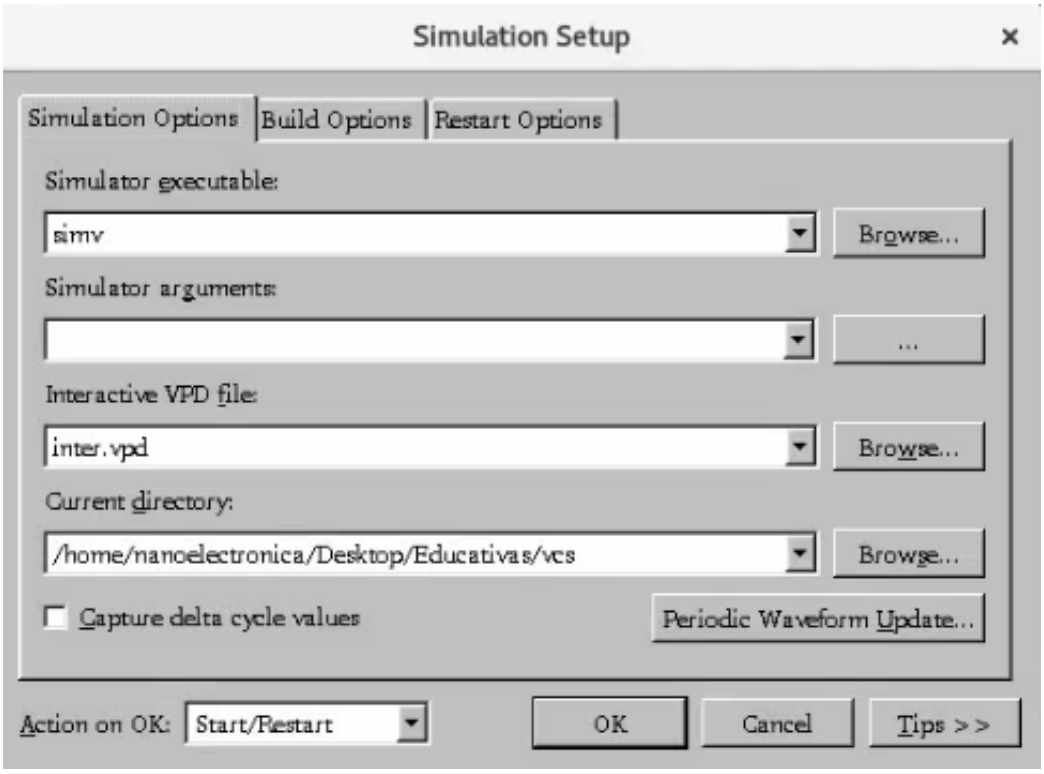


Figura 29: Configuración simv

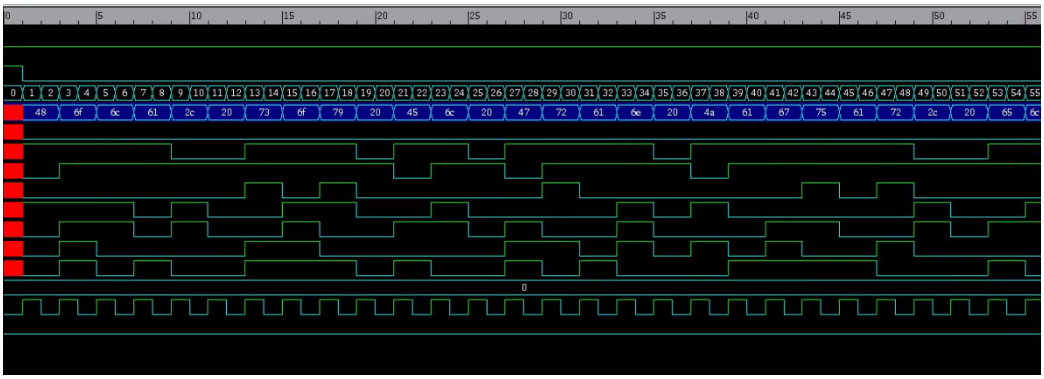


Figura 30: Vista ampliada del mensaje

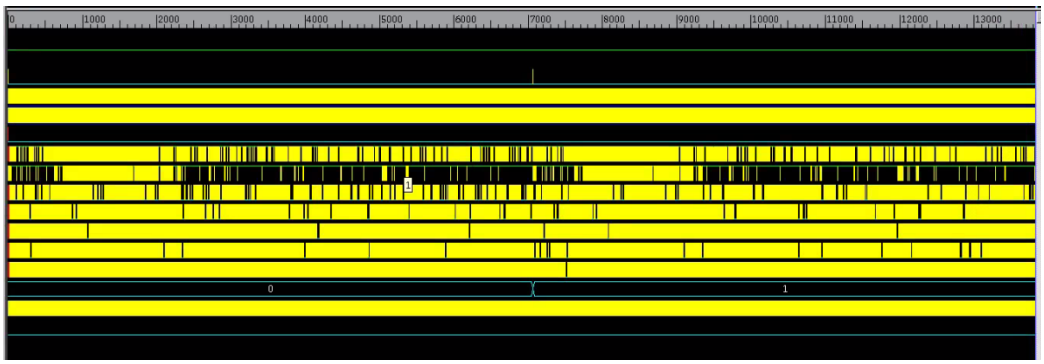


Figura 31: Vista completa del comportamiento

7.2. Síntesis lógica en Design Compiler

En la elaboración de la síntesis lógica se utilizaron dos archivos; circuito.v el cual estaba compuesto del chip con sus dos textos y su testbench, es importante mencionar que estos archivos no son los mismo que se utilizaron para la simulación lógica, modificando los dos textos por una palabra:

Texto uno: Hola

Texto dos: Adios

No se tomó en cuenta la tilde en la palabra evitando problemas futuros en los caracteres ASCII. Partiendo con el apoyo del laboratorio se ejecutó el script dado, modificando los nombres de los archivos al igual que sus localidades. Se obtuvo una exitosa síntesis como se muestran en las siguientes figuras:

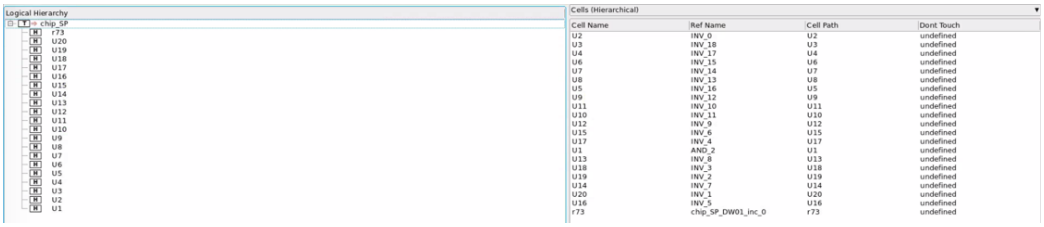


Figura 32: Vista de cell's

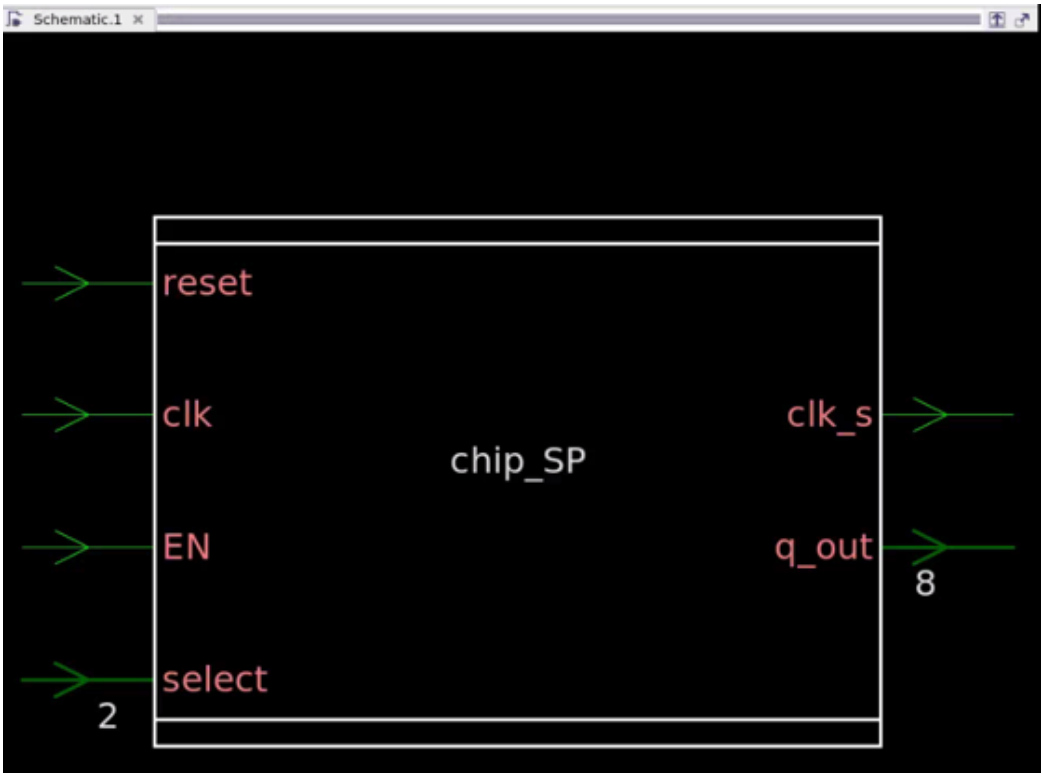


Figura 33: Esquemático superior

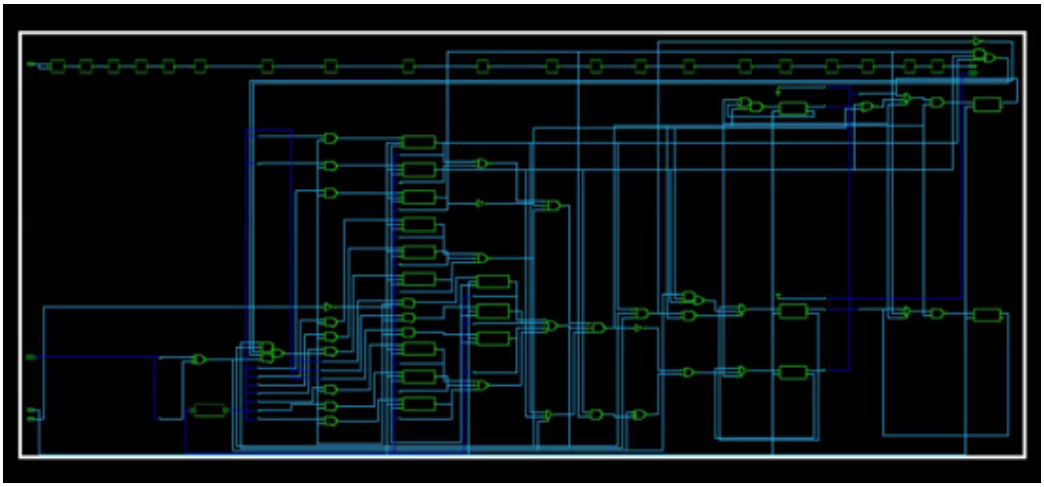


Figura 34: Esquemático completo

Como se mostró en las figuras anteriores se puede observar un esquemático correcto sin errores, los pines de entrada y salidas, los cells etc. por lo cual se procedió a realizar un archivo con extensión .sdc y .ddc como requerimiento para realizar la síntesis física.

7.3. Síntesis física en IC Compiler II

En el proceso de síntesis física utilizando los archivos generados en Design Compiler se utilizó nuevamente el scrip fabricado para que este proceso fuera automatizado. En este proceso se obtuvo una gran cantidad de obstáculos iniciando con los voltajes (vss y vdd). El cual se solucionó modificando el script de pg_cores, en otras palabras, este scrip describía la cantidad de voltajes que tendría el foorplaan con una separación determinada, por lo cual se modificó está separación llegándola a los límites de ambos extremos del foorplaan como se muestra en la siguiente figura:

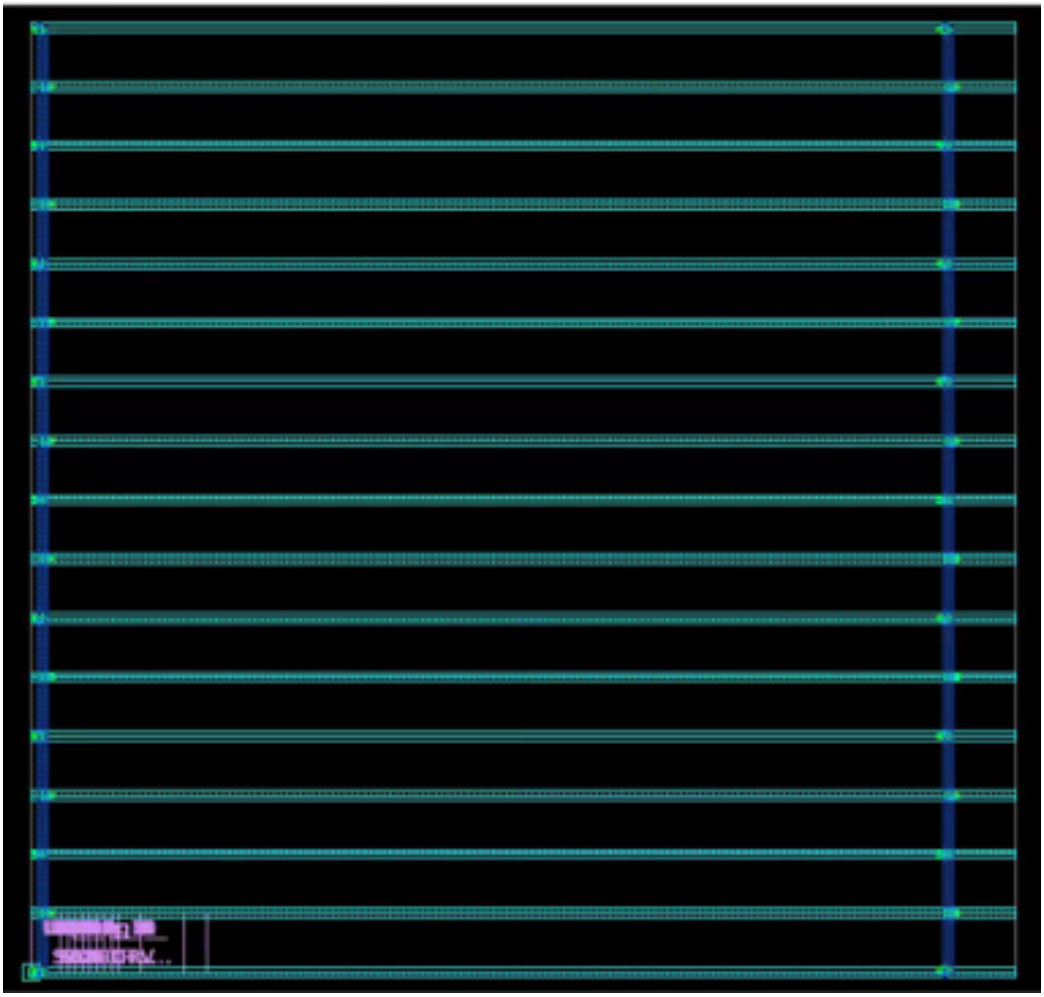


Figura 35: Separación de voltajes

Como segundo obstáculo nos encontramos con la ausencia de 10 cells denominadas .ªDDH_OP5", al inició se buscaron dentro de las librerías, sin embargo, estás no fueron encontradas por lo cual con el apoyo de Jonathan se decidió por navegar en la plataforma de solvnet hasta encontrar la carpeta, está carpeta se adjuntó a la carpeta IPDK para no tener este conflicto de nuevo.

Se muestra en las siguientes figuras, las cells legalizadas en el foorplaan, seguidamente

el ruteo de las cells, añadir los fillers y por ultimó una agregación de vías redundantes.

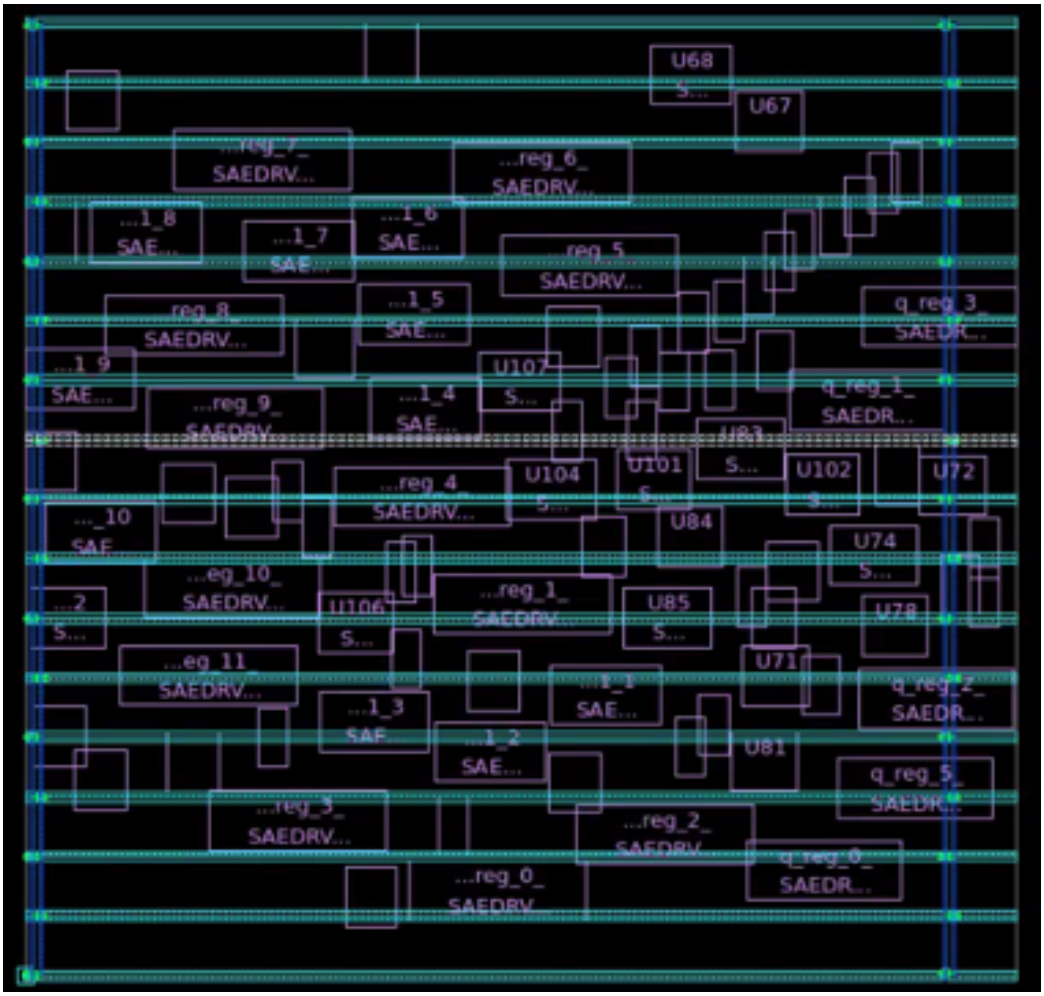


Figura 36: Legalización de cells

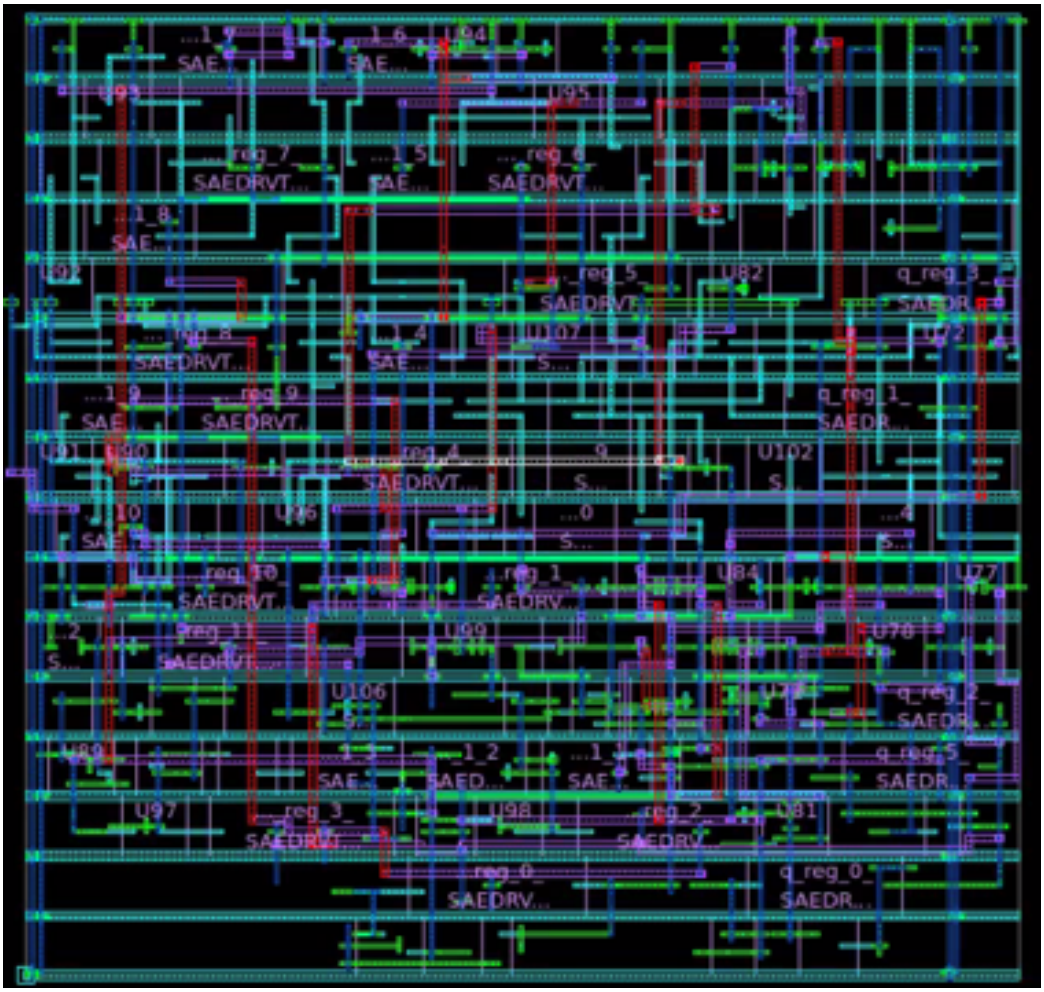


Figura 37: Ruteo

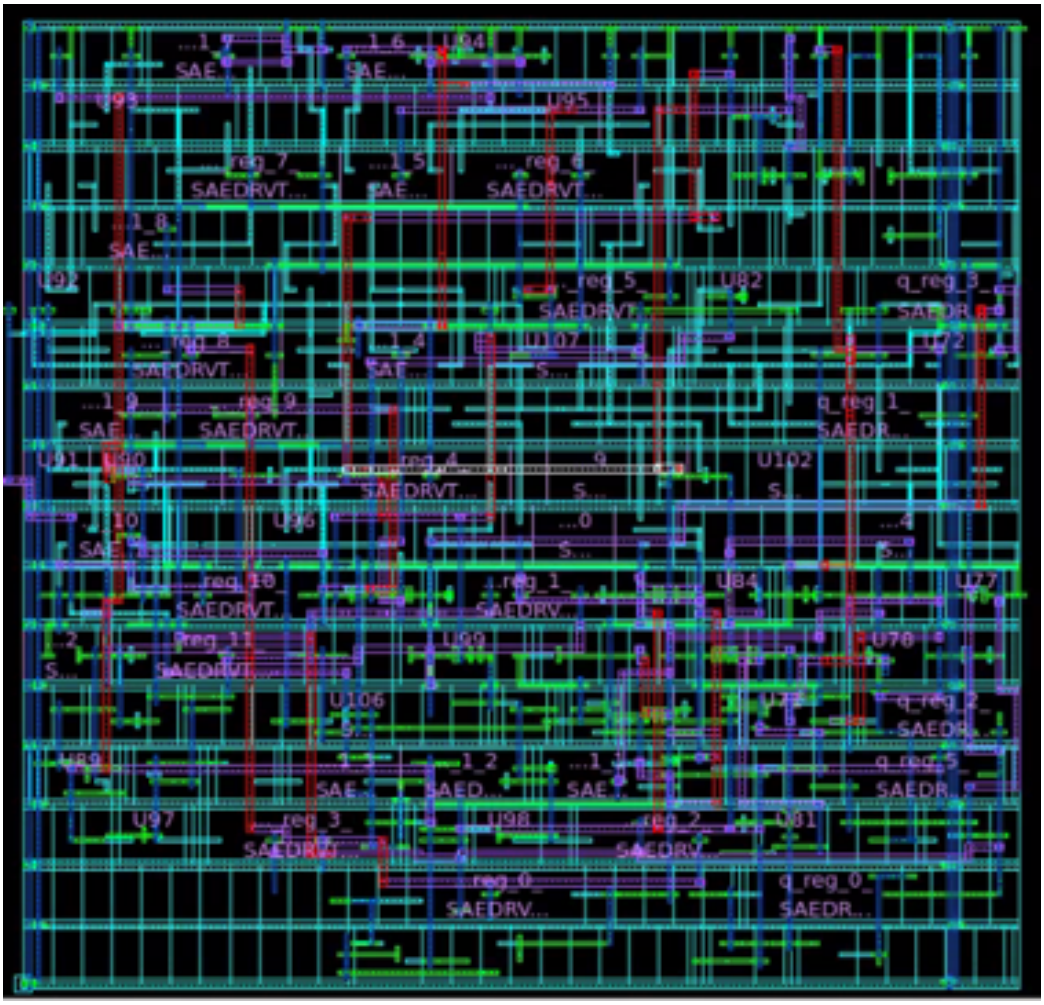


Figura 38: Complemento de fillers

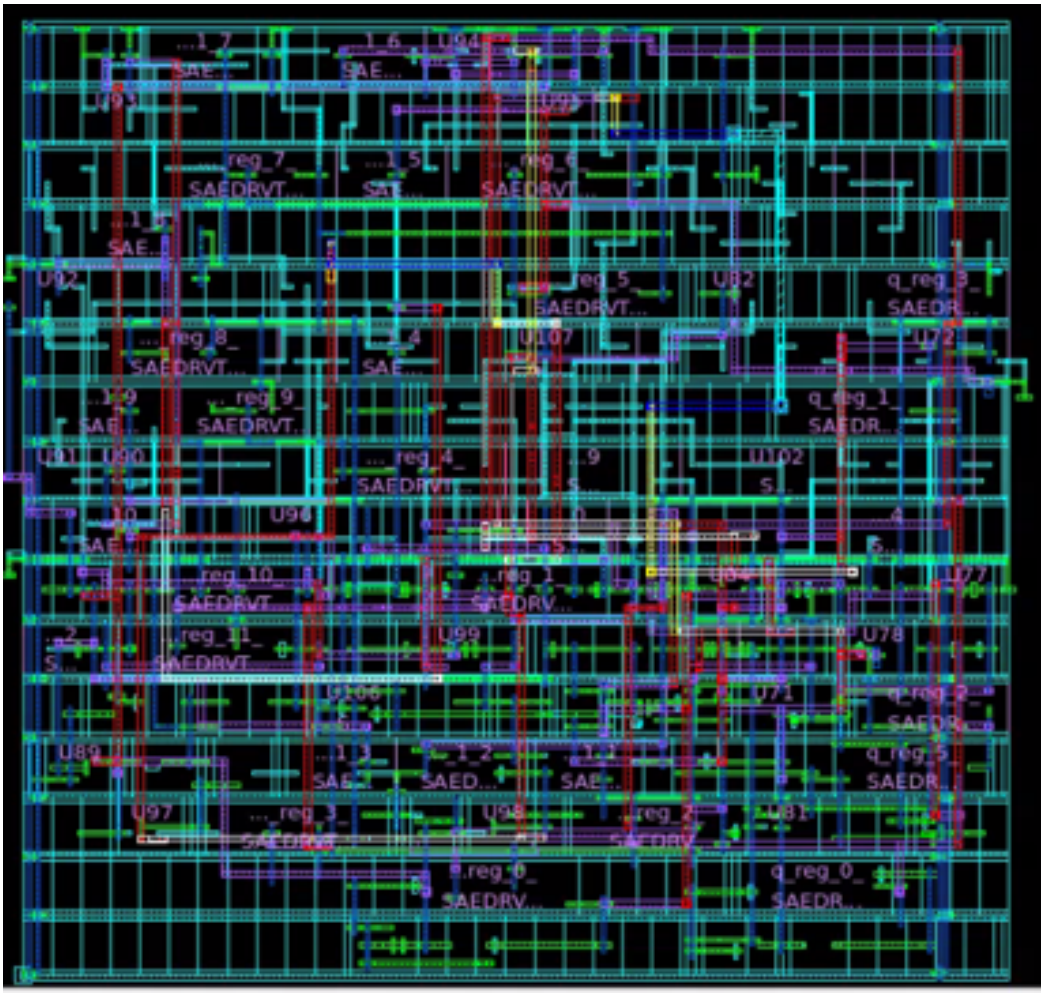


Figura 39: Complemento de vías redundantes

Por último se decidió realizar desde la síntesis física, el flujo de IC Compiler para la extracción de parásitos:

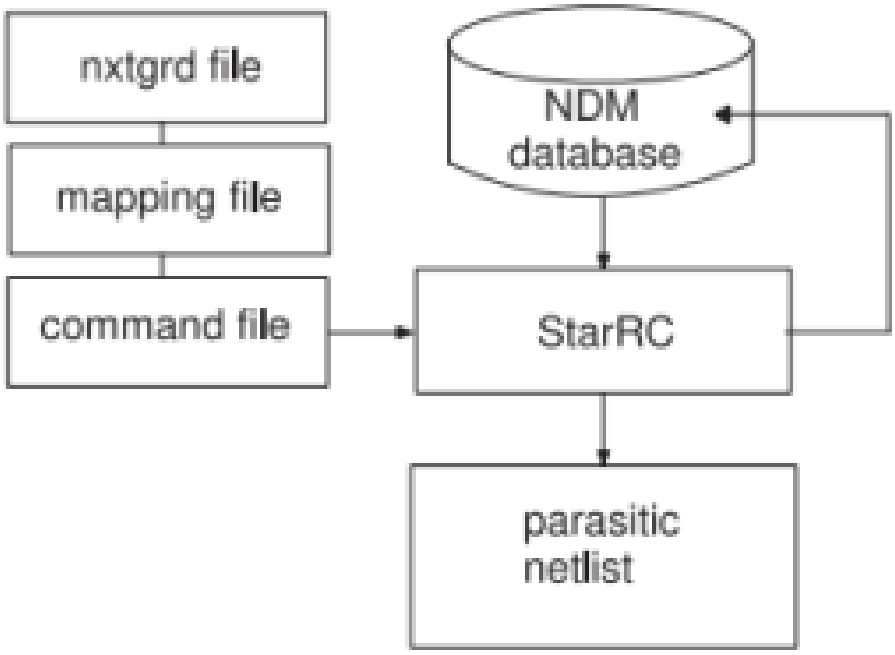


Figura 10: ICC II Flow.

```
NDM_CELL_REPORT_FILE: ndm_report
```

Figura 40: Flujo ICC II

El objetivo de realizar este flujo es el obtener un archivo .spf en el que se detallé un subcircuito que mostrara los pines de entrada y salida que tendrá nuestro chip. A diferencia del trabajo del año pasado es que el flujo por la herramienta de IC Compiler utiliza los archivos NDM compuesto como se ve en la Figura 41 obtenidos de la síntesis física, es decir, este método utiliza vistas de frame encargadas de realizar un pin sin importarle errores obtenidos en el design rule check, layout vs schematic o verificación de antena, lo que anteriormente no era posible por las cajas negras.

- NDM_CELL_REPORT_FILE
- NDM_DATABASE
- NDM_DESIGN_VIEW
- NDM_LAYOUT_VIEW
- NDM_EXPAND_HIERARCHICAL_CELLS

Figura 41: Archivos NDM

CAPÍTULO 8

Documentación de la herramienta Formality

En este capítulo se detalla una guía breve del manejo adecuado de la herramienta de Formality para realizar diversas comparaciones de diseño y evaluar si hubo algún cambio en la etapa del flujo de diseño. La cual estará conformada de las secciones presentes describiendo los más importante de cada una de ellas para obtener un resultado exitoso.

8.1. Descripción

Como se mencionó anteriormente, Formality se utiliza para comparar dos versiones de diseño.6.6

8.2. Verificación de diseños por un chequeo equivalente

La verificación del diseño es un proceso de cuatro fases:

1. Leer y elaborar descripciones del lenguaje en representaciones lógicas.

2. Prepararse para adelantarse a las diferencias.

3. Mapear señales correspondientes entre pares de diseños (Matching).

4. Comparar los conos lógicos que impulsan las señales mapeadas (Verification). Conos lógicos:

8.3. Conceptos de diseño:

Diseño de referencia:

Diseño de implementación:

Luego de que Formality demuestre la equivalencia del diseño de implementación con un diseño de referencia conocido, puede establecer el diseño de implementación como el nuevo diseño de referencia. Manteniendo los tiempos de verificación generales al mínimo durante las pruebas de regresión. De lo contrario trabajar a través de una metodología de diseño completa y luego, verificar la lista de conexiones aprobada con el RTL original. (Puede resultar en verificaciones difíciles y tiempos de verificación más prolongados.)

8.4. Comandos útiles para manejo de librerías

8.4.1. CMD para definir librerías

```
define_design_lib -r -path ./directorio
```

8.4.2. CMD para leer en las librerías

```
read_db -technology_library nombre_libreria.db
```

8.5. Flujo de verificación de Formality

La Figura 42 describe el flujo del proceso de verificación del diseño de Formality. Representa pasos específicos para realizar una verificación de equivalencia usando Formality.

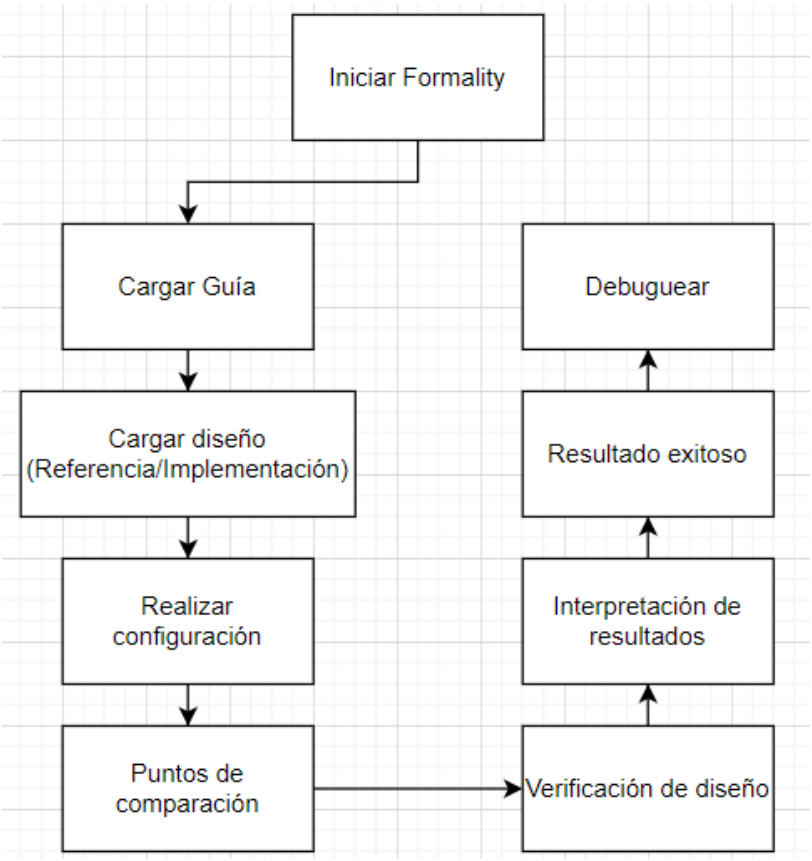


Figura 42: Flujo específico de Formality

8.6. Iniciar Formality

Para iniciar Formality se puede realizar de dos maneras: Primero desde la línea de comando; segundo abriendo la interfaz:

8.6.1. CMD para inicializar la herramienta por línea de comando

```
fm_shell
```

8.6.2. CMD para inicializar la herramienta con interface

```
fm_shell (setup) > star_gui
```

8.7. Cargar guía

El paso de guía de carga del flujo del proceso Formality es el punto en el que puede optar por proporcionar información de configuración sobre los cambios de diseño causados por otras herramientas utilizadas en el flujo de diseño. Lo cual se puede realizar con la herramienta de Design Compiler proporcionado información de configuración en forma de un archivo de instalación automatizado (.svf)

8.7.1. CMD para crear un archivo SVF en Design Compiler

```
dc_shell> set_svf miarchivo.svf
```

8.7.2. CMD para leer un archivo SVF en Formality

```
fm_shell> set_svf miarchivo.svf
```

8.8. Cargar diseños(Referencia/Implementación)

Para realizar la verificación, primero debe proporcionar el trámite de dos diseños; El diseño de referencia, el que se sabe que es funcionalmente correcto, el segundo diseño es una versión modificada del diseño de referencia y se conoce como diseño de implementación; Este es el diseño que desea verificar con el diseño de referencia para la equivalencia funcional.

Ambas cargas se componen de la misma forma como se muestra en la siguiente figura:

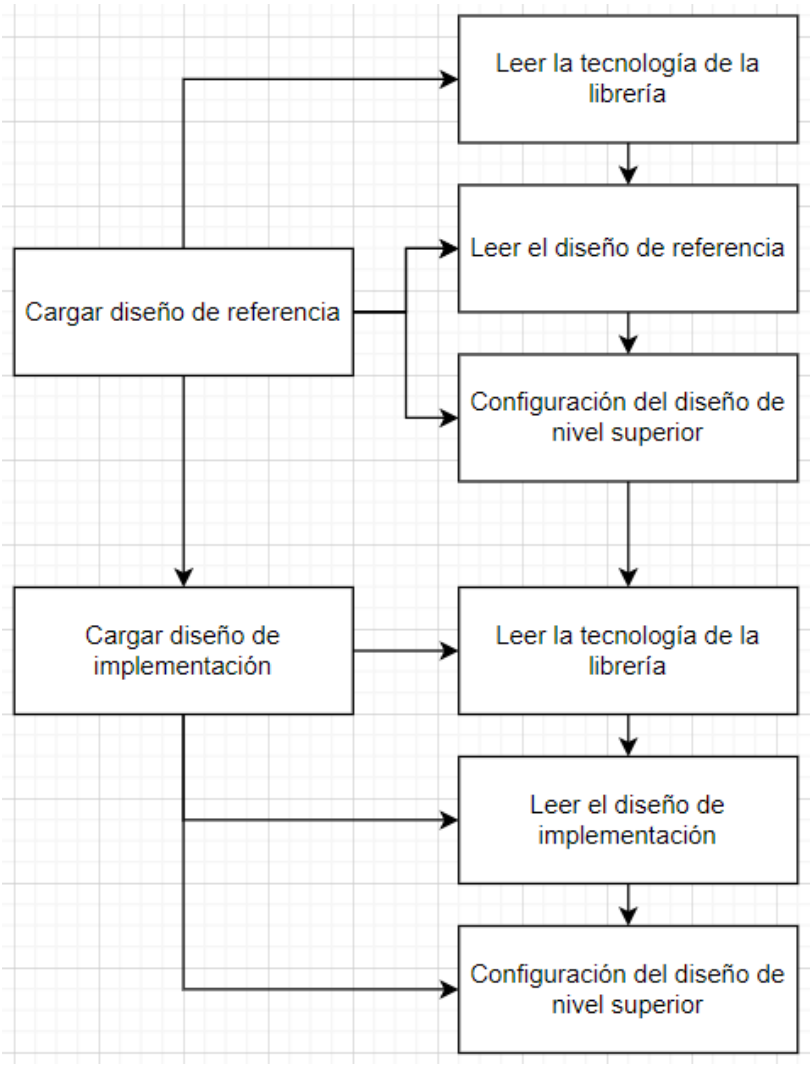


Figura 43: Flujo de diseños

8.8.1. Leyendo la tecnología de la librería

8.8.2. CMD para leer una tecnología

```
fm_shell> read_db nombre_archivo [-libname library_name ]
```

8.8.3. Leyendo el diseño de referencia

Tres posibles opciones:

8.8.4. CMD para leer un diseño

```
fm_shell> read_verilog -r nombre_archivo
```

```
fm_shell> read_vhdl -r nombre_archivo_referencia
```

```
fm_shell> read_db -r nombre_archivo_referencia
```

8.8.5. Configuración del diseño de nivel superior

8.8.6. CMD para leer un diseño

```
set_top [-vhdl nombre_archivo ]
```

8.8.7. Leyendo la tecnología de la librería

Si ya se especificó una librería .db para el diseño de referencia, se comparte automáticamente con el diseño de implementación. Es decir se utilizan los mis comandos.

8.8.8. Leyendo el diseño de implementación

Dos posible opciones:

8.8.9. CMD para leer un diseño

```
fm_shell> read_verilog -r nombre_archivo_implementacion
```

```
fm_shell> read_vhdl -r nombre_archivo_implementacion
```

8.8.10. Configuración del diseño de nivel superior

La herramienta debe leer la referencia o el diseño de implementación antes de ejecutar el comando set_top utilizando los comando anteriores para el nivel superior de referencia. No se leerá el diseño de implementación hasta que se haya especificado el comando set_top para el diseño de referencia.

8.9. Realizar configuración

El paso de configuración implica proporcionar información a Formality para dar cuenta de los problemas específicos del diseño que no se solucionaron automáticamente durante el paso de orientación. Casos de configuración más utilizados:

8.9.1. Manejo de cajas negras

Las cajas negras pueden causar fallas de verificación porque los pines de entrada se convierten en puntos de comparación en el diseño. Si las cajas negras del diseño de referencia no coinciden con los del diseño de implementación, los puntos de comparación no coinciden.

8.9.2. CMD para marcar un diseño como caja negra

```
fm_shell> set_black_box designID
```

8.9.3. CMD para informes de cajas negras

```
fm_shell> report_black_boxes
```

8.9.4. Especificación de constantes

Formality reconoce dos tipos de constantes; de diseño y definidas por el usuario. Las constantes de diseño son redes en su diseño que están vinculadas a un valor lógico 1 o 0; Las constantes definidas por el usuario son puertos o redes a las que adjunta un valor lógico 1 o 0 mediante comandos de Formality.

8.9.5. CMD para definir constantes

```
fm_shell> set_constant [-type tipo_de_la_constante ]
```

8.9.6. CMD para definir remover constantes

```
fm_shell> remove_constant
```

```
fm_shell> remove_constant -all
```

8.10. Puntos de comparación

Durante este paso, la herramienta Formality intenta hacer coincidir cada punto de comparación en el diseño de referencia con un punto de comparación correspondiente en el diseño de implementación.

8.10.1. CMD para una comparación

```
fm_shell> match
```

También podemos obtener un reporte de cuantos puntos no tuvieron coincidencia con el siguiente comando:

8.10.2. CMD para una comparación

```
fm_shell> report_unmatched_points
```

8.11. Verificación de diseño

El paso de verificación sigue a los pasos de carga, configuración y coincidencia de puntos de comparación.

8.11.1. CMD para verificar el diseño

```
fm_shell> verify
```

8.11.2. CMD para verificar con puntos de quiebre

```
fm_shell> svf_breakpoint
```

8.12. Interpretación de resultados

Al final de la verificación o en cualquier momento durante el proceso, si se elige interrumpir el proceso antes de que finalice, los resultados de la verificación se informan como APROBADO (todos los puntos de comparación son equivalentes), FALLO (algunos puntos de comparación no son equivalentes) o NO CONCLUYENTE (algunos puntos de comparación no están verificados o están terminados). Algunos reportes posibles a realizar y poder interpretar los mensajes de resultado son:

8.12.1. CMD para reportes

```
fm_shell> report_passing_points
```

```
fm_shell> report_failing_points
```

```
fm_shell> report_aborted_points
```

8.13. Resultado exitoso

Si el resultado es exitoso el trabajo ha concluido, de lo contrario se realizará una depuración para localizar el error.

8.14. Debuguear

El paso de depuración es necesario si la verificación del diseño no se realiza correctamente; Durante la depuración, se utilizan los resultados de verificación para identificar resultados fallidos o no concluyentes. Este paso ayuda a determinar dónde y posiblemente por qué los resultados no fueron exitosos. Proceso para el cual se armó un diagrama de flujo de las posibles pruebas a realizar.

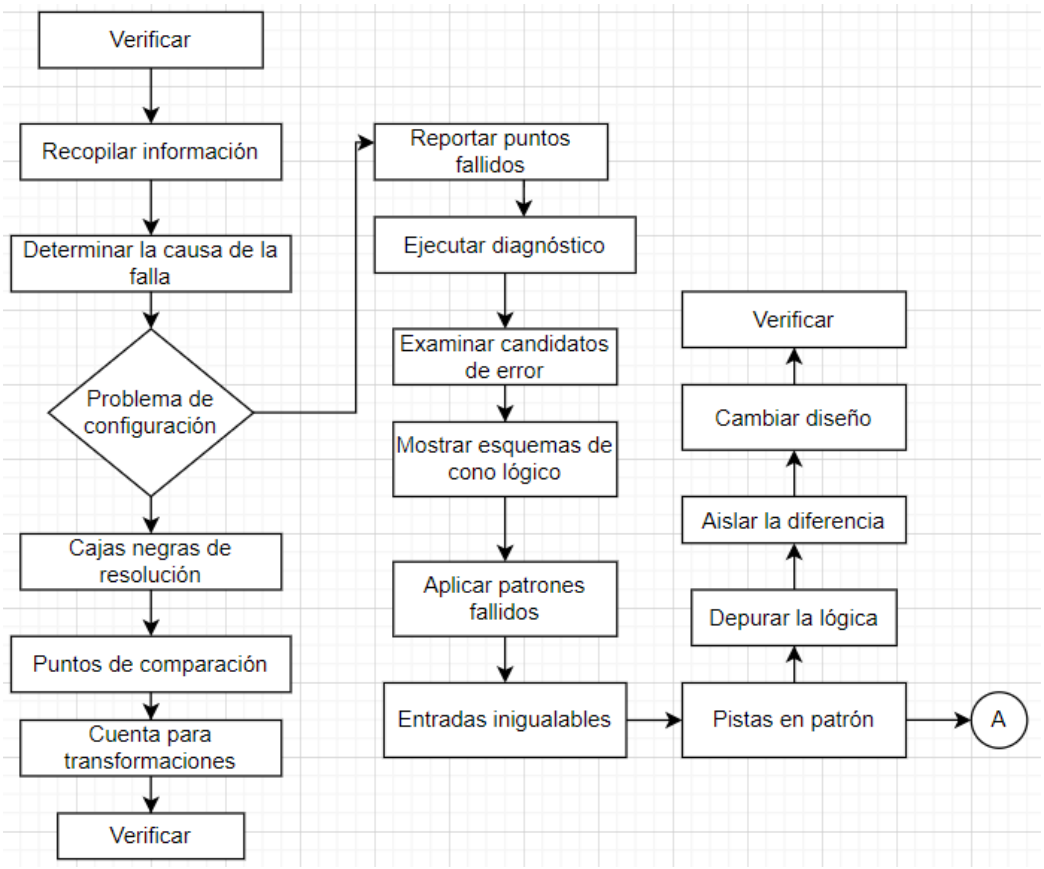


Figura 44: Flujo para debuguear

CAPÍTULO 9

Documentación de la herramienta Design Compiler

En este capítulo se detalla una guía breve del manejo adecuado de la herramienta de Design Compiler para realizar la síntesis lógica y optimización del circuito en la etapa del flujo de diseño. La cual estará conformada de las secciones presentes, describiendo lo más importante de cada una de ellas para obtener un resultado exitoso.

9.1. Descripción:

Como se mencionó, anteriormente, Design Compiler se utiliza para optimizar los diseños, brindando la representación lógica más pequeña y rápida de una función dada.6.3

9.2. Flujo de verificación de Design Compilar

La Figura 45 describe el flujo del proceso de Design Compiler, representando pasos específicos para realizar una síntesis lógica y optimización del circuito usando Design Compiler.

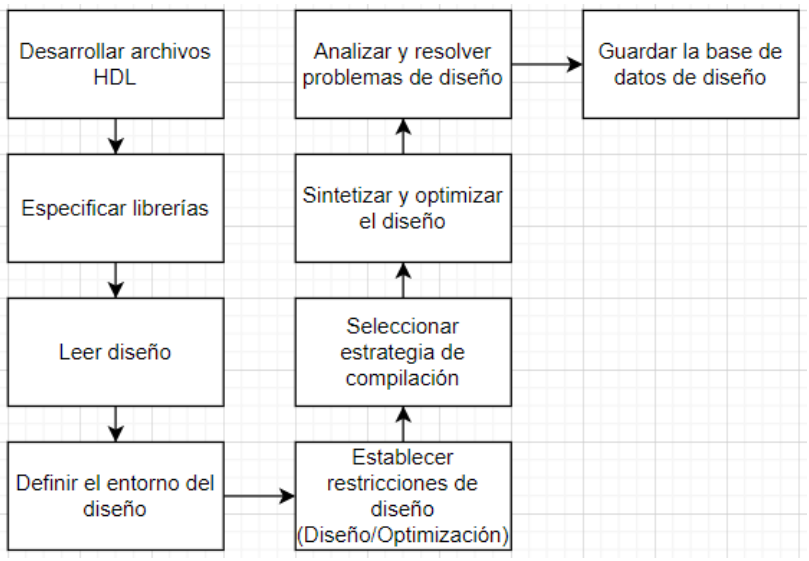


Figura 45: Flujo especifico de Design Compiler

9.3. Iniciar Design Compiler

Para iniciar Design Compiler se puede realizar de dos formas: una de ellas es desde la linea de comando; y la segunda abriendo la interfaz:

9.3.1. CMD para inicializar la herramienta por linea de comando

```
dc_shell
```

9.3.2. CMD para inicializar la herramienta con interfase

```
dc_shell (setup) > star_gui
```

9.4. Desarrollar archivos HDL

El formato de los archivos que se permiten en Design Compiler son los siguientes:

1. .ddc
2. .db
3. Verilog
4. VHDL

9.4.1. CMD para leer archivos

```
dc_shell > read_file verilog miarchivo.v
```

```
dc_shell > read_file vhdl miarchivo.v
```

9.5. Especificar librerías

Design Compiler utiliza bibliotecas de lógica, símbolos y DesignWare para implementar funciones de diseño y mostrar gráficamente los resultados de Synthesis; Las bibliotecas lógicas a las que se asigna Design Compiler durante la optimización se denominan; bibliotecas de destino. Las bibliotecas de destino contienen las celdas utilizadas para generar la lista de conexiones y definiciones para las condiciones operativas del diseño.

9.5.1. CMD para especificaciones generales de librerías

```
dc_shell> link_library
```

```
dc_shell> target_library
```

```
dc_shell> symbol_library
```

```
dc_shell> synthetic_library
```

```
dc_shell> create_mw_library
```

9.6. Leer diseño

La herramienta Design Compiler lee diseños en la memoria desde archivos de diseño. Muchos diseños pueden estar en la memoria en cualquier momento. Después de leer un diseño, se puede modificar de muchas maneras, como: agrupar o desagrupar sus sub-diseños o cambiar las referencias de los sub-diseños.

9.6.1. CMD para lecturas generales de diseño

```
dc_shell> read_file
```

```
dc_shell> analyze
```

```
dc_shell> elaborate
```

9.7. Definir el entorno del diseño

Antes de poder optimizar un diseño, se debe definir el entorno en el que se espera que opere el diseño. El entorno se define especificando las condiciones operativas, las características de la interfaz del sistema y los modelos de carga de cables (solo se usa cuando Design Compiler no está funcionando en modo topográfico). Las condiciones de operación incluyen variaciones de temperatura, voltaje y proceso. Las características de la interfaz del sistema incluyen controladores de entrada, cargas de entrada y salida y cargas de abanico. El modelo de entorno afecta directamente a los resultados de la síntesis de diseño.

9.7.1. CMD para definiciones generales en un diseño

```
dc_shell> set_operating_conditions
```

```
dc_shell> set_drive
```

```
dc_shell> set_driving_cell
```

```
dc_shell> set_load
```

```
dc_shell> set_min_library
```

```
dc_shell> set_wire_load_model
```

9.8. Establecer restricciones de diseño

Las restricciones son declaraciones que definen los objetivos de su diseño en características de circuito medibles, como: tiempo, área y capacitancia. Design Compiler necesita estas restricciones para optimizar efectivamente el diseño.

9.8.1. Restricciones de las reglas de diseños

Las restricciones de las reglas de diseño reflejan las restricciones específicas de la tecnología que su diseño debe cumplir para funcionar según lo previsto.

9.8.2. CMD para restricciones de diseño

```
dc_shell> set_max_transition
```

```
dc_shell> set_max_fanout
```

```
dc_shell> set_capacitance
```

9.8.3. Restricciones de optimización de diseño

Las restricciones de optimización representan objetivos y restricciones de diseño de velocidad, área y potencia que desea, pero que pueden no ser cruciales para el funcionamiento de un diseño.

9.8.4. CMD para restricciones de optimización

```
dc_shell> create_clock
```

```
dc_shell> set_clock_latency
```

```
dc_shell> set_input_delay
```

```
dc_shell> set_output_delay
```

```
dc_shell> set_max_area
```

9.9. Seleccionar estrategia de compilación

Al seleccionar una estrategia, de arriba hacia abajo y de abajo hacia arriba no son comandos. Se refieren a dos estrategias de compilación de uso común que utilizan diferentes combinaciones de comandos.

9.10. Sintetizar y optimizar el diseño

La optimización es el paso de síntesis del compilador de diseño que asigna el diseño a una combinación óptima de celdas de biblioteca de lógica de destino específicas, en función de los requisitos funcionales, de velocidad y de área del diseño. Para iniciar el proceso de compilación se utilizan los comandos que se observan abajo, que sintetizan y optimizan el diseño.

9.10.1. CMD para compilación

```
dc_shell> compile_ultra
```

```
dc_shell> compile
```

9.11. Analizar y resolver problemas de diseño

Utilizando los informes generados por Design Compiler podemos analizar y depurar su diseño. Puede generar informes tanto antes como después de compilar su diseño. Se recomienda generar informes antes de compilar para verificar que haya establecido correctamente los atributos, las restricciones y las reglas de diseño.

9.11.1. CMD para compilación

```
dc_shell> check_design
```

```
dc_shell> report_area
```

```
dc_shell> report_constraint
```

```
dc_shell> report_timing
```

9.12. Guardar la base de datos de diseño

Para finalizar luego de obtener los resultados esperados se recomienda guardar la base de datos de su diseño.

9.12.1. CMD para guardar el diseño

```
dc_shell> write_file
```

CAPÍTULO 10

Documentar las dos nuevas herramientas de Formality y Design Compiler parar iteraciones de otros años.

10.1. Guía de instalaciones para los programas *Design Compiler* y *Formality*

10.1.1. *Formality*

- Abrir una terminal en la dirección que usted tenga ubicado los archivos de descarga.
- Ingresar el siguiente comando:

10.1.2. CMD para Inicialización del installer_gui

```
installer -gui
```

Figura 46: Paso uno.

- Se desplegará el siguiente menú

Figura 47: Paso dos.

- Dejar por default como se muestra en la imagen.

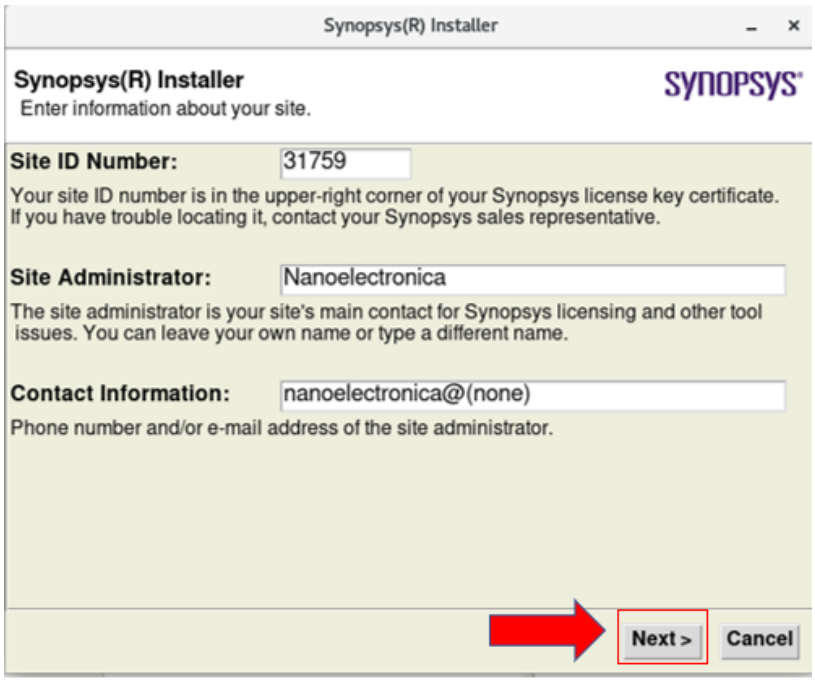


Figura 48: Paso tres

- Recomendado dejar por default como se muestra en la imagen.

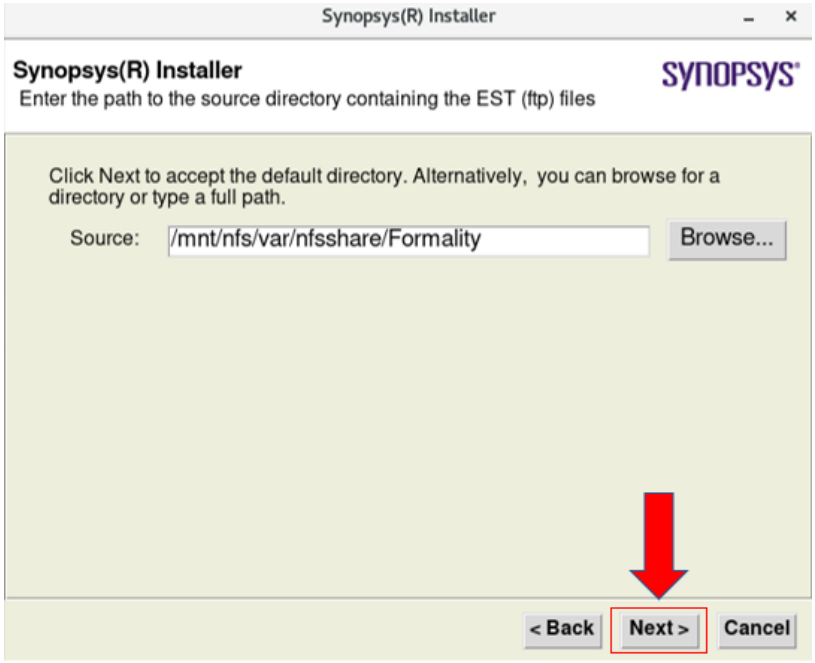


Figura 49: Paso cuatro

- Recomendado dejar por default como se muestra en la imagen.

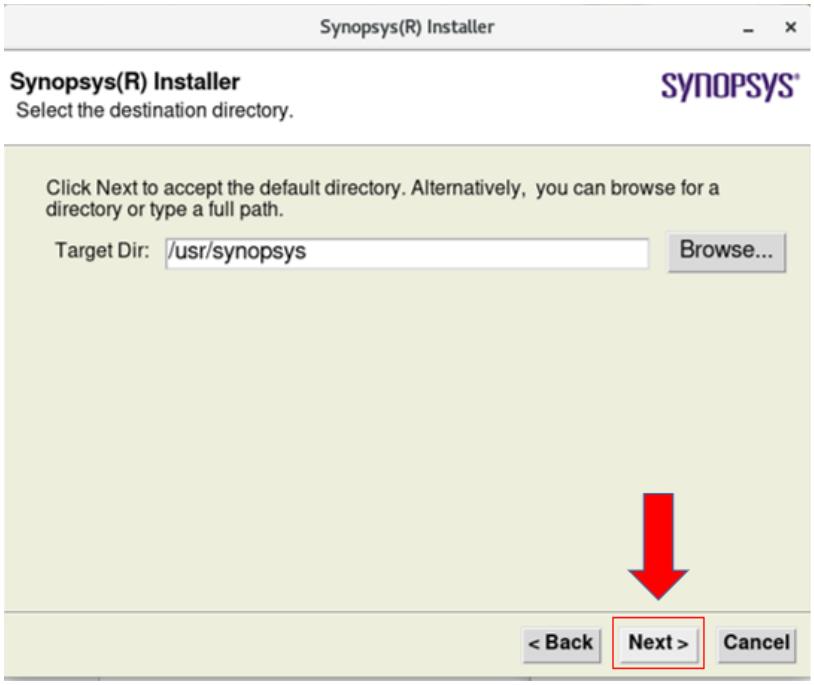


Figura 50: Paso cinco

- Se escogerá la siguiente opción.

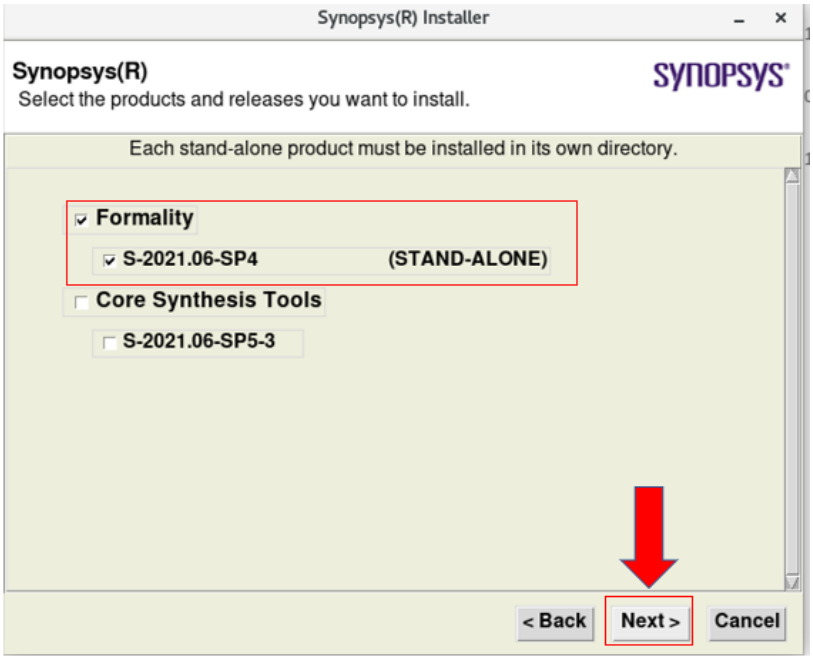


Figura 51: Paso seis

- Recomendado dejar por default como se muestra en la imagen.

Figura 52: Paso siete

- ¡Listo para instalar!

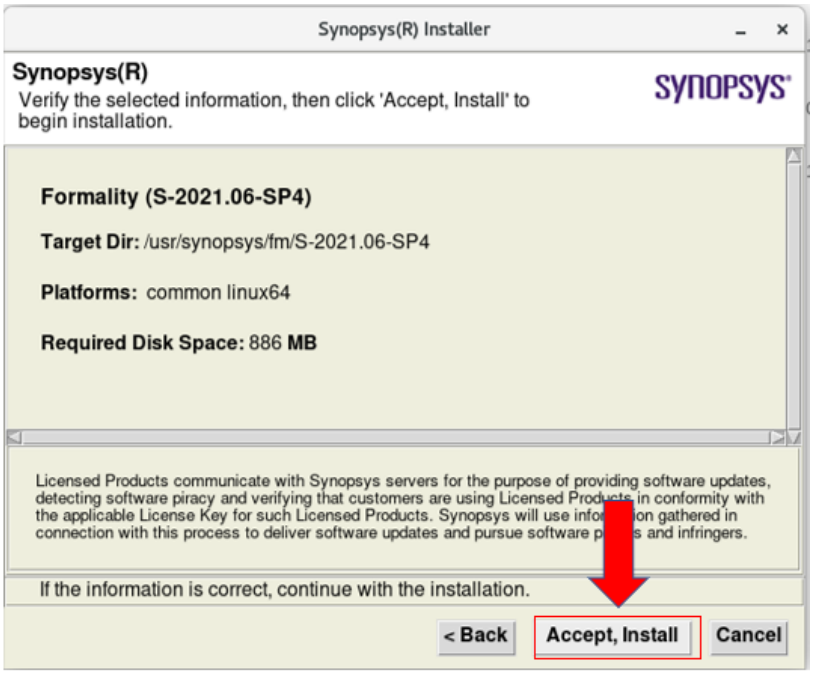


Figura 53: Paso ocho

- Instalación finalizada

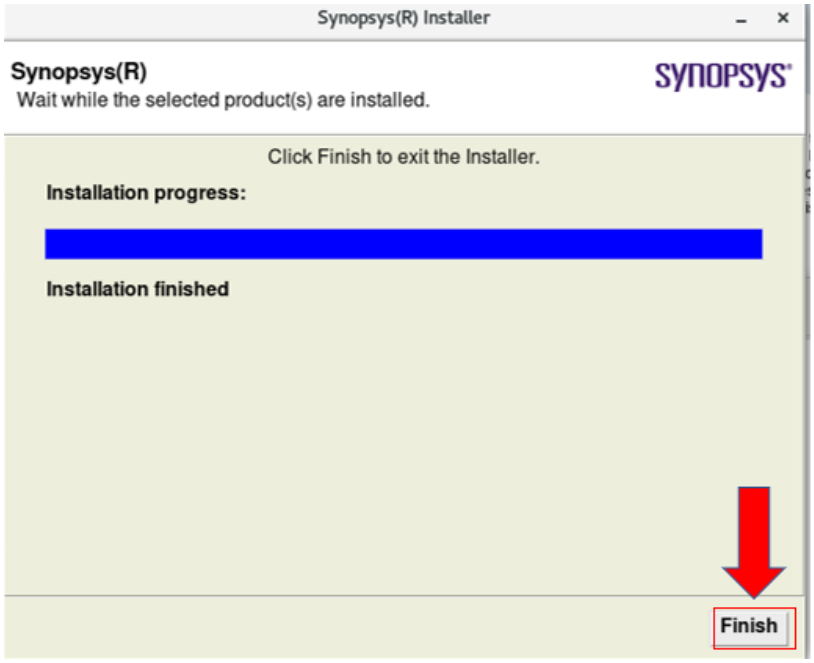


Figura 54: Paso nueve

- Ambos mensajes se deben de omitir.

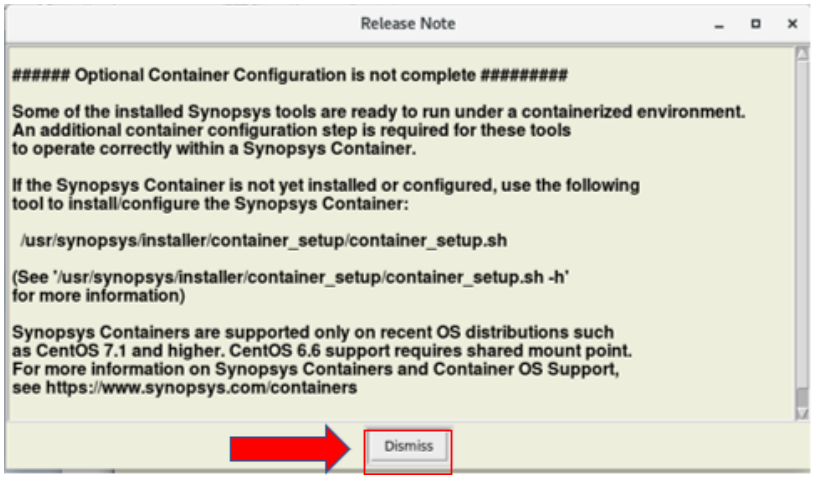


Figura 55: Paso diez

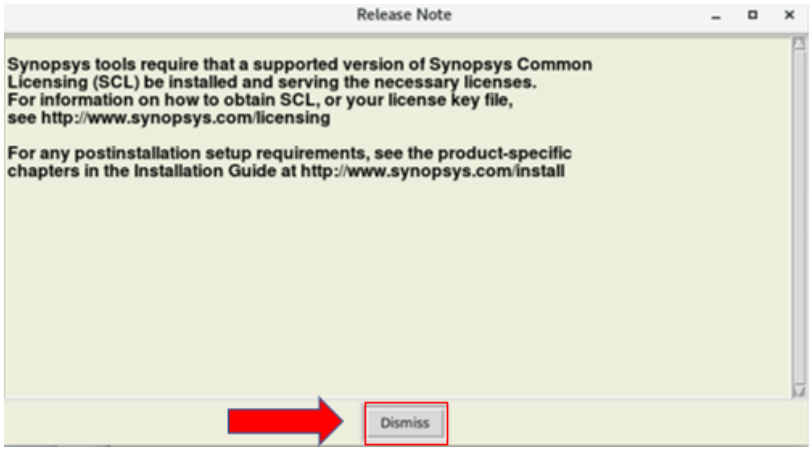


Figura 56: Paso once

- Verificar en la .bash que se exporte el path.
- Si no se encuentra colocarlo copiando la dirección de algún archivo dentro de la carpeta de bin

Figura 57: Paso doce

- Vista de la aplicación.

10.1.3. CMD para inicialización de la herramienta

```
fm_shell

start_gui
```

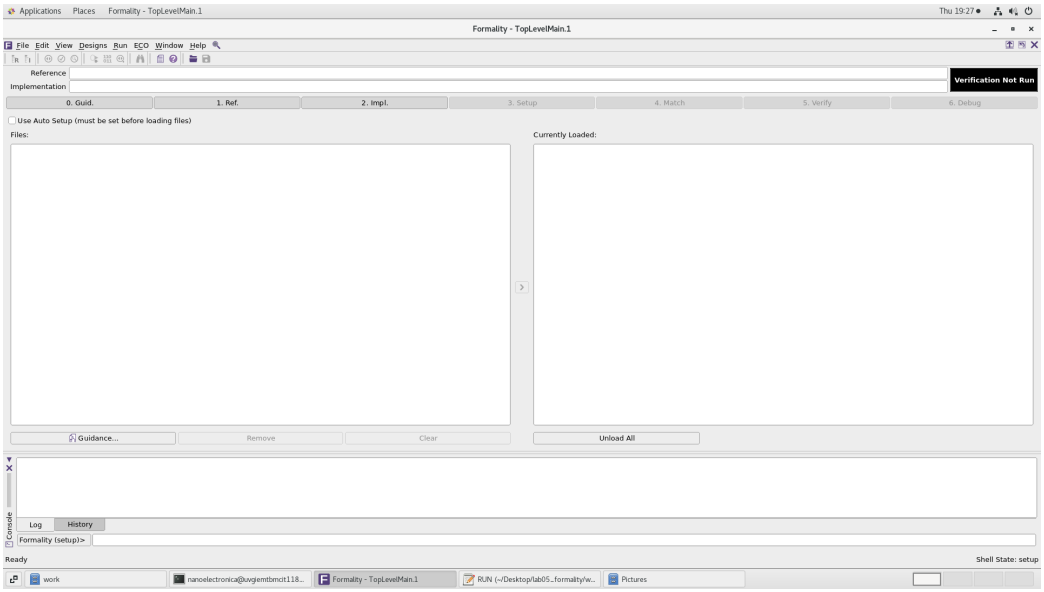


Figura 58: Paso trece

10.1.4. *Design Compiler*

- Abrir una terminal en la dirección que usted tenga ubicado los archivos de descarga.
- Ingresar el siguiente comando:

10.1.5. CMD para Inicialización del installer_gui

```
installer -gui
```

- Se desplegará el siguiente menú

Figura 59: Paso uno

- Dejar por default como se muestra en la imagen.

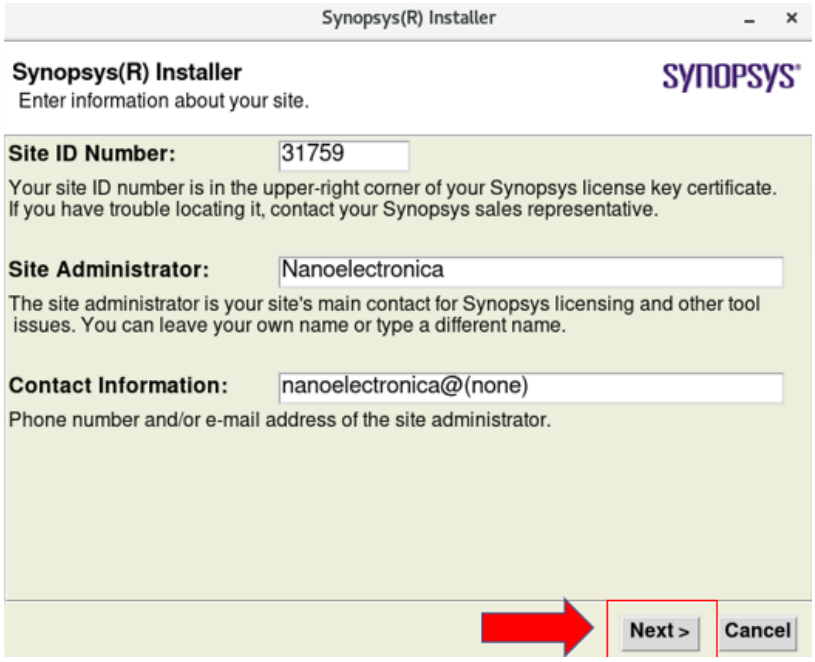


Figura 60: Paso dos

- Recomendado dejar por default como se muestra en la imagen.

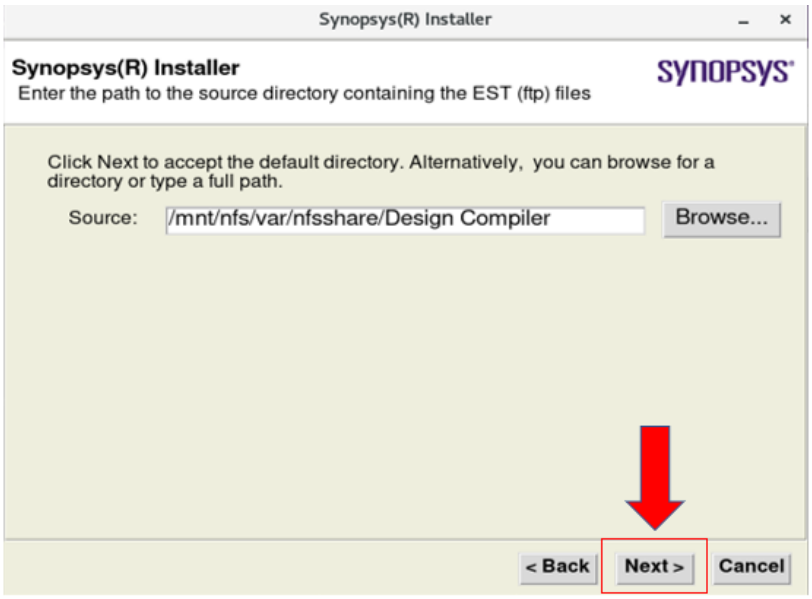


Figura 61: Paso tres

- Recomendado dejar por default como se muestra en la imagen.

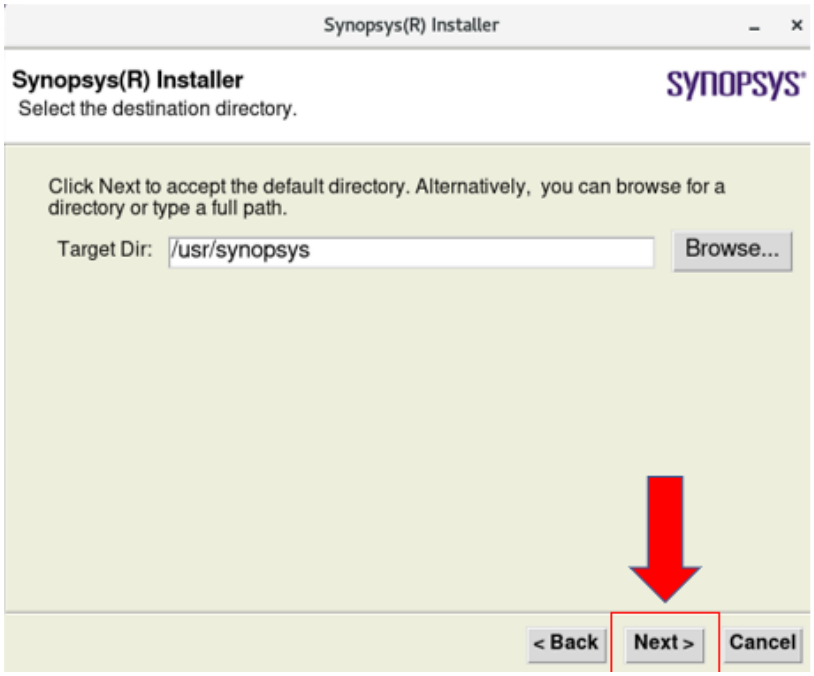


Figura 62: Paso cuatro

- Se escogerá la siguiente opción..

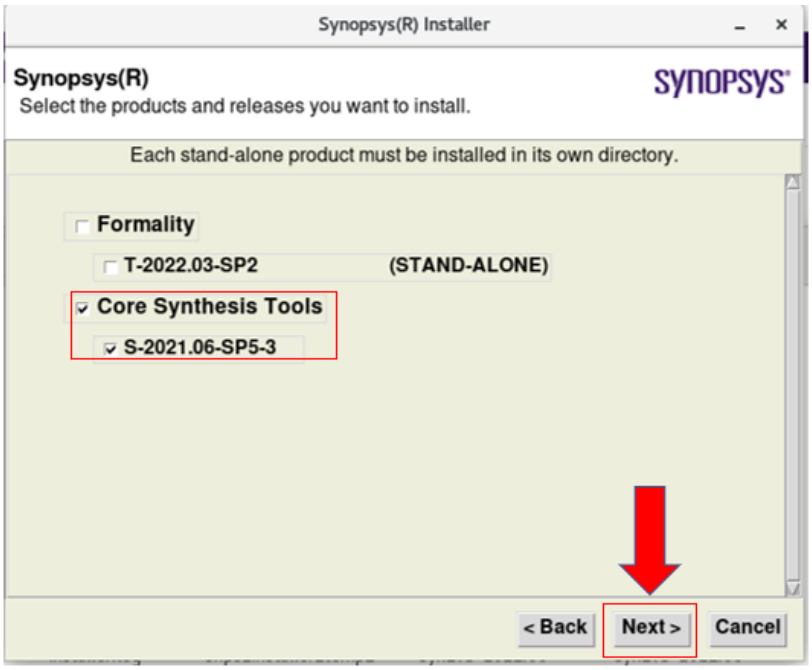


Figura 63: Paso cinco

- Recomendado dejar por default como se muestra en la imagen.

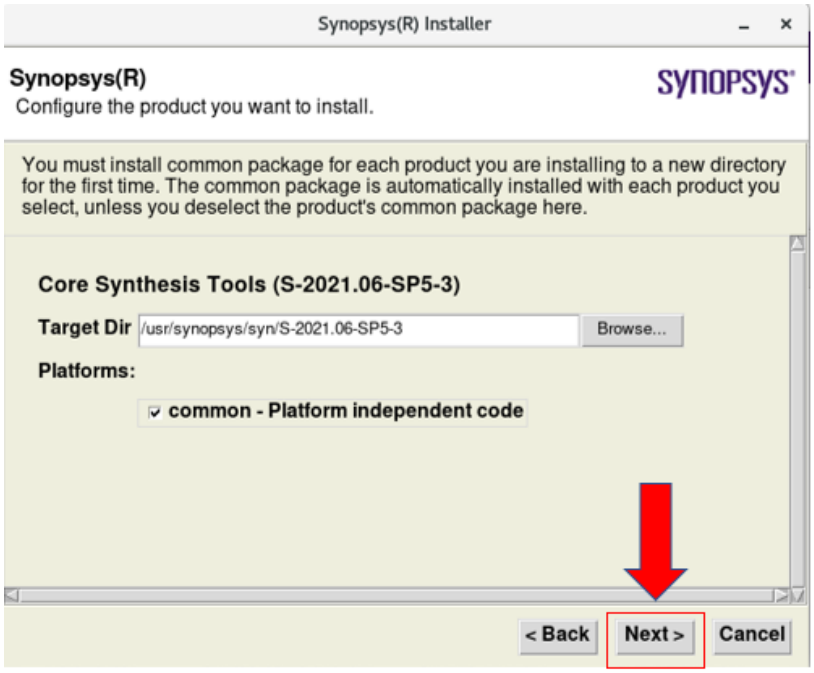


Figura 64: Paso seis

- ¡Listo para instalar!

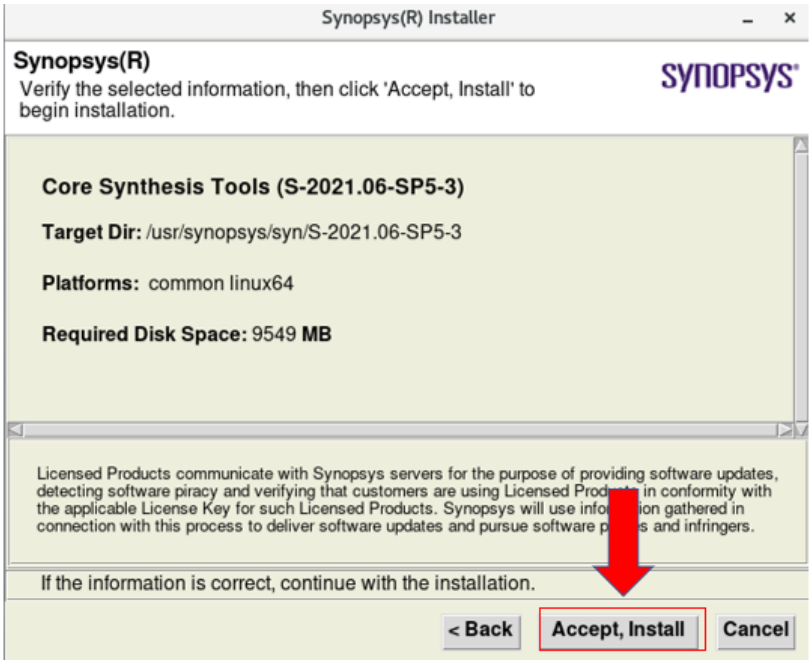


Figura 65: Paso siete

- Instalación finalizada.

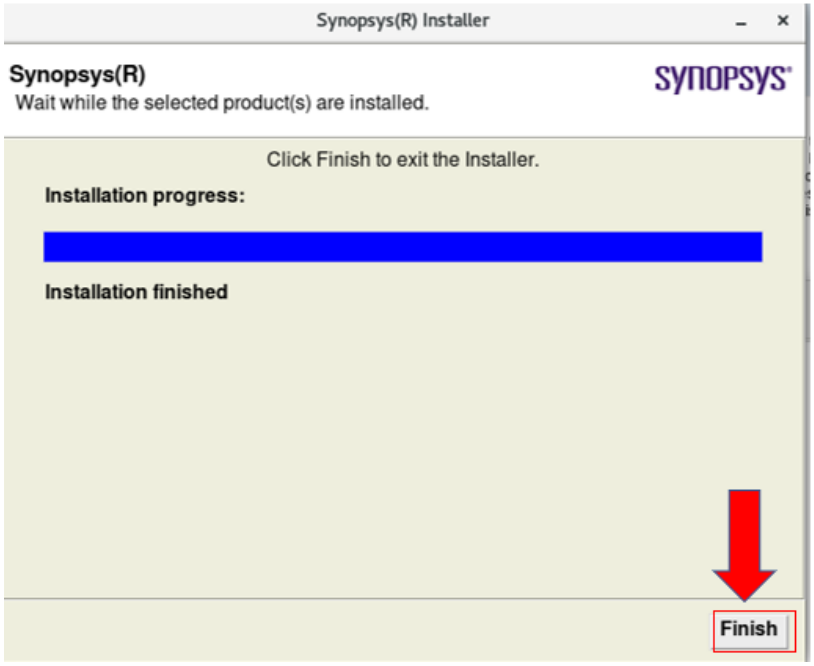


Figura 66: Paso ocho

- Ambos mensajes se deben de omitir.

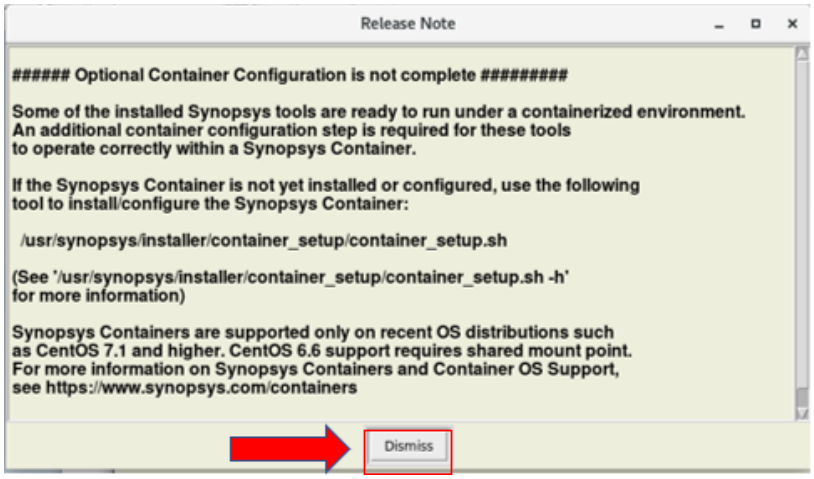


Figura 67: Paso nueve

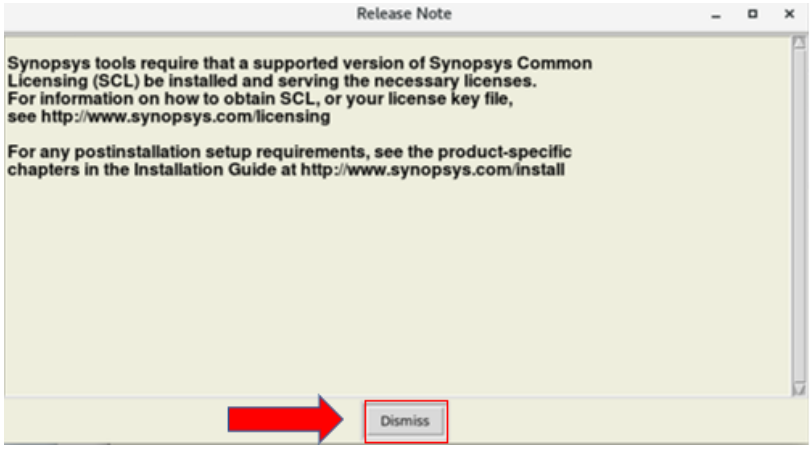


Figura 68: Terminal iniciando la gui.

- Verificar en la .bash que se exporte el path.
- Si no se encuentra colocarlo copiando la dirección de algún archivo dentro de la carpeta de bin

```
  GNU nano 2.3.1        File: /home/nanoelectronica/.bashrc            Modified

# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

LM_LICENSE_FILE=27020@192.168.6.252; export LM_LICENSE_FILE

PATH=/usr/synopsys/installer:$PATH
PATH=/usr/synopsys/customcompiler/R-2020.12/bin:$PATH
PATH=/usr/synopsys/cscope64/P-2019.06/ai_bin
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging featur$
# export SYSTEMD_PAGER=

# User specific aliases and functions
```

Figura 69: Paso diez

- Vista de la aplicación.
- Comandos:

10.1.6. CMD para inicialización de la herramienta

```
dc_shell

start_gui
```

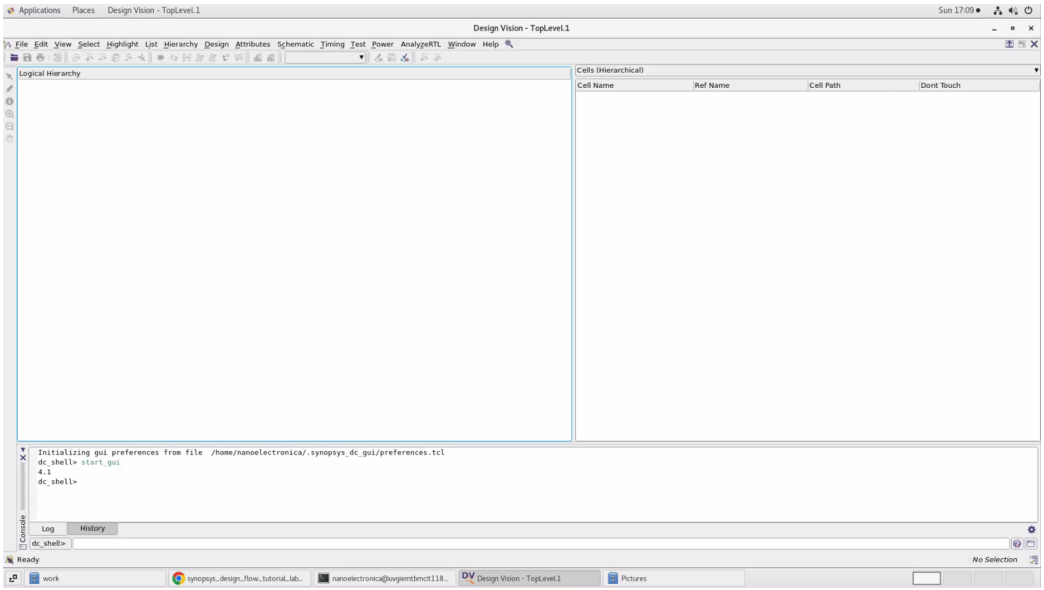


Figura 70: Paso once

CAPÍTULO 11

Creación e implementación de decks para la simulación y manufactura del circuito.

En este capitulo se detalla la construcción de 11 compuertas lógicas que conforman al nanochip, con el propósito de observar su comportamiento y evaluar que las entradas y salidas fueran las correctas. A continuación se presentará el esquemático de las compuertas[11],[12], la tabla de verdad que describe a la compuerta y su simulación en la herramienta de HSPICE.

11.1. Primera compuerta: AO21D0BWP7T

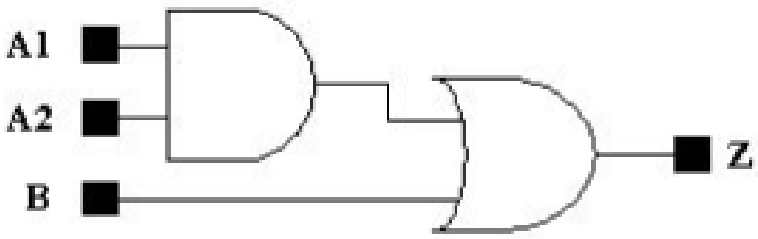


Figura 71: Compuerta AO21D0BWP7T

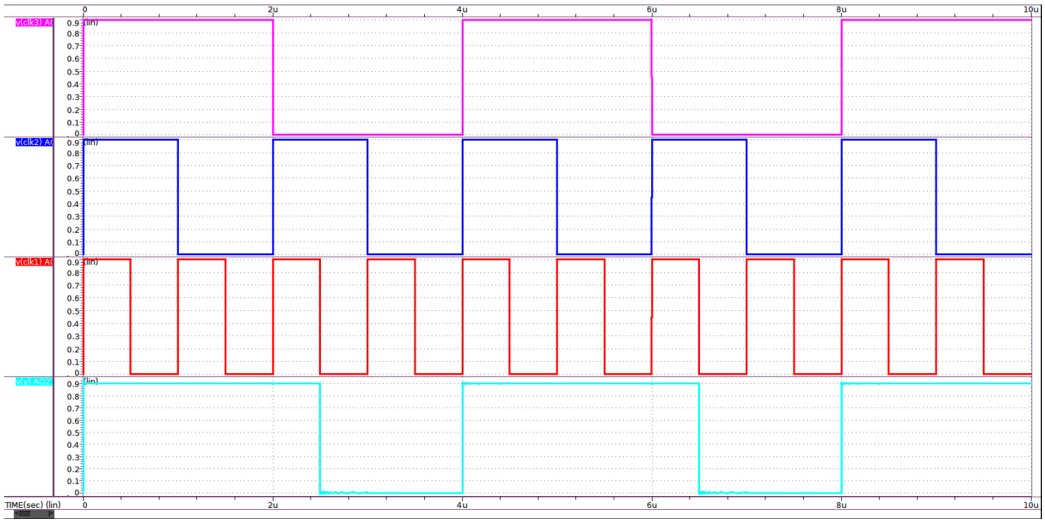


Figura 72: Simulación de AO21D0BWP7T

INPUT			OUTPUT
A1	A2	B	ZN
1	1	x	1
x	x	1	1
0	x	0	0
x	0	0	0

Cuadro 1: Comportamiento lógico de la compuerta AO21D0BWP7T

Para la primera compuerta 71 se observa que está conformada con tres entradas hacia una AND(2) y una OR(1). En el funcionamiento podemos ver que al tener dos entradas en uno lógico la salida será un uno, de otra manera si la entrada de la compuerta NOR está en uno su salida será un uno nuevamente, en caso contrario a las dos últimas pruebas que sí tienen un cero, la NOR y otro cero, cualquiera de la entrada de la AND, su salida será un cero.

11.2. Segunda compuerta: AOI221D0BWP7T

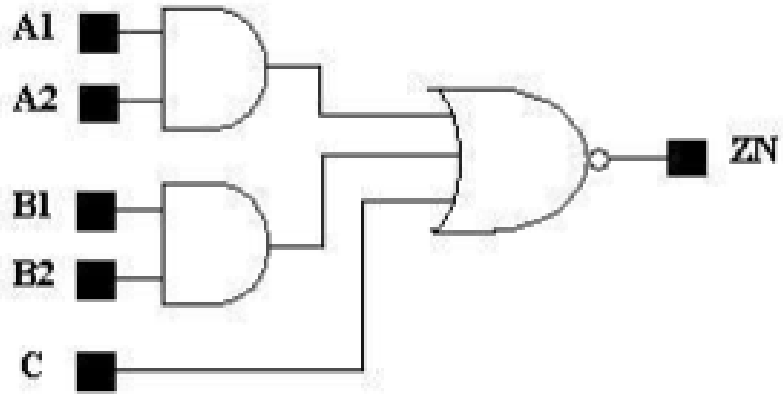


Figura 73: Compuerta AOI221D0BWP7T

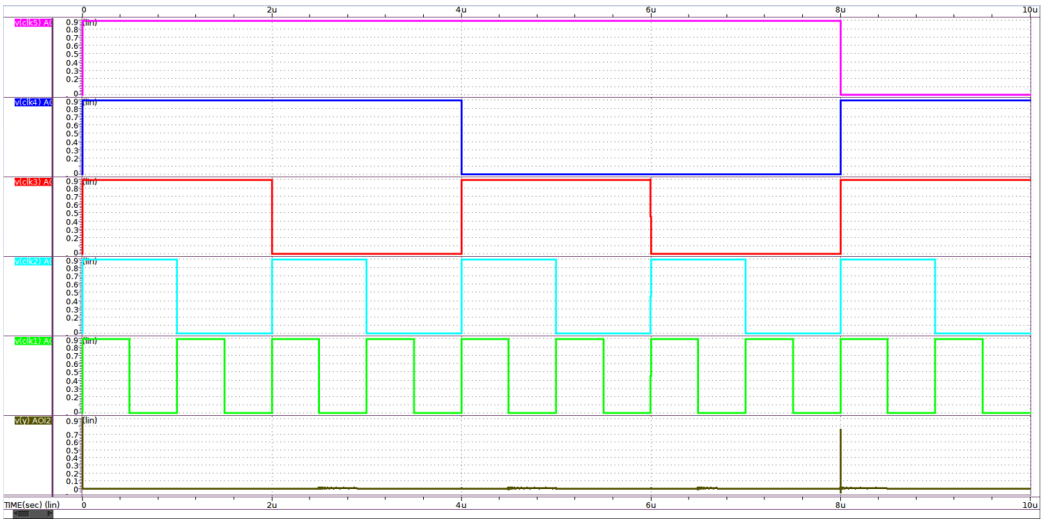


Figura 74: Simulación de AOI221D0BWP7T

INPUT			OUTPUT		
A1	A2	B1	B2	C	ZN
1	1	x	x	x	0
x	x	1	1	x	0
x	x	x	x	1	0
0	x	x	0	0	1
x	0	x	0	0	1
0	x	0	x	0	1
x	0	0	x	0	1

Cuadro 2: Comportamiento lógico de la compuerta AOI221D0BWP7T

En la segunda compuerta 73 se observa que está conformada por 5 entradas hacia dos AND(2,2) y una NOR(1). En el funcionamiento podemos ver que en las dos AND's su salida va hacia la NOR, por lo cual sabemos para el funcionamiento que solo obtendremos un uno en la salida si sus entradas son uno, sin embargo, la NOR para obtener un uno en salida todas sus entradas deben ser cero, como se ve en la simulación la salida nunca se obtuvo un uno.

11.3. Tercera compuerta: IIND4D0BWP7T

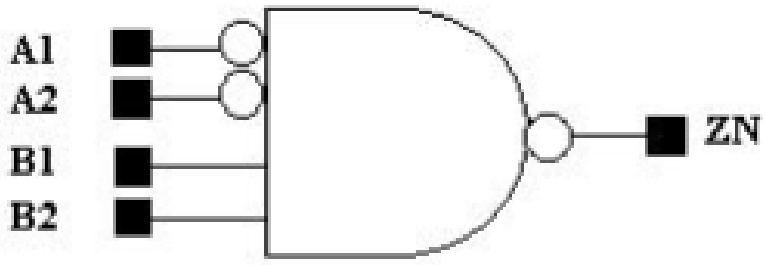


Figura 75: Compuerta IIND4D0BWP7T

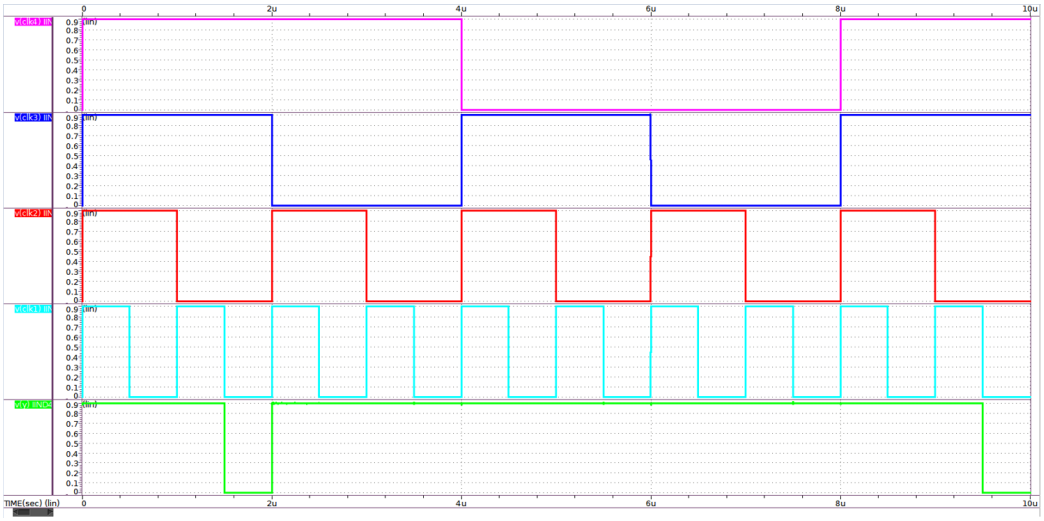


Figura 76: Simulación de IIND4D0BWP7T

INPUT			OUTPUT	
A1	A2	B1	B2	ZN
0	0	1	1	0
1	x	x	x	1
x	1	x	x	1
x	x	0	x	1
x	x	x	0	1

Cuadro 3: Comportamiento lógico de la compuerta IIND4D0BWP7T

La tercera compuerta 75 está conformada únicamente por una NAND con cuatro entradas, dos de ellas están negadas. En el funcionamiento podemos ver que al tener solamente un uno en cualquiera de las cuatro entradas nuestra salida será un uno efectivamente, además podemos observar que en la simulación la gráfica de la salida está en cero.

11.4. Cuarta compuerta: IND3D1BWP7T

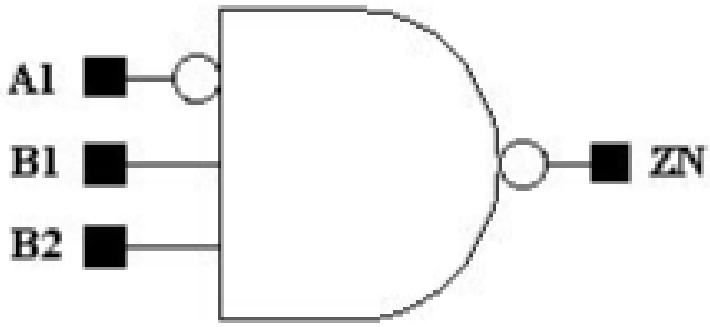


Figura 77: Compuerta IND3D1BWP7T

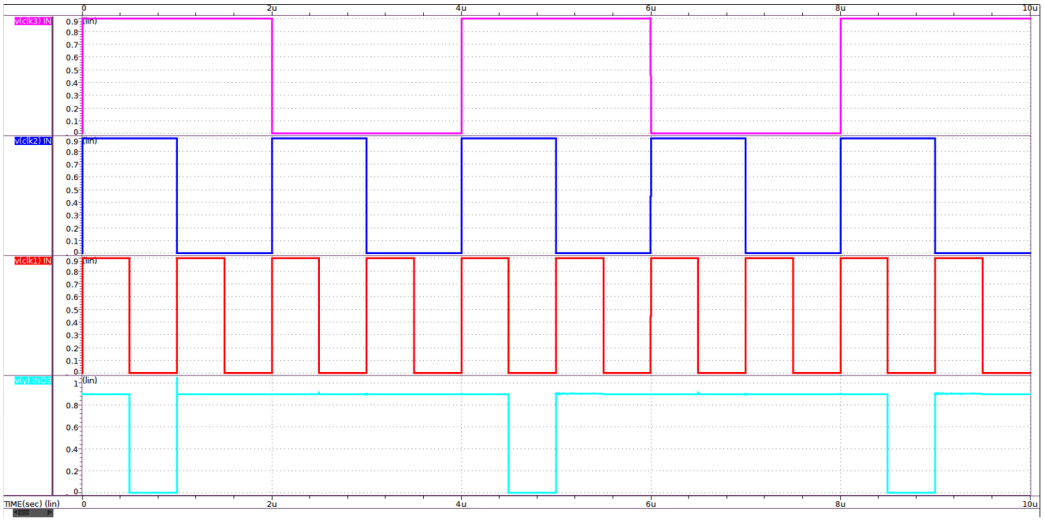


Figura 78: Simulación de IND3D1BWP7T

INPUT			OUTPUT
A1	B1	B2	ZN
0	1	1	0
1	x	x	1
x	0	x	1
x	x	0	1

Cuadro 4: Comportamiento lógico de la compuerta IND3D1BWP7T

Para esta compuerta 77 es una NAND similar a la descrita anteriormente, con la diferencia de tener tres entradas y una de ellas negada. En la simulación podemos ver el mismo comportamiento teniendo un uno en la salida debido a que más de alguna entrada tiene un uno y cuando tenemos un cero es porque las entradas también están en cero.

11.5. Quinta compuerta: IND4D0BWP7T

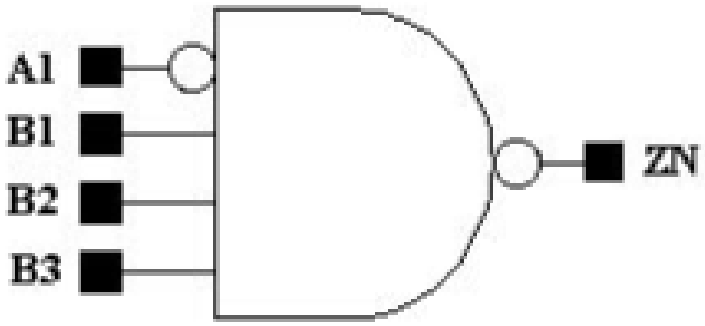


Figura 79: Compuerta IND4D0BWP7T

Figura 80: Simulación de IND4D0BWP7T

INPUT			OUTPUT	
A1	A2	B1	B2	ZN
0	0	1	1	0
1	x	x	x	1
x	1	x	x	1
x	x	0	x	1
x	x	x	0	1

Cuadro 5: Comportamiento lógico de la compuerta IND4D0BWP7T

La quinta compuerta 79 nuevamente es similar a las anteriores con la diferencia que tiene cuatro entradas y nada más una de ellas negada, en las cual el funcionamiento vuelve a ser el esperado.

11.6. Sexta compuerta: INR2D1BWP7T

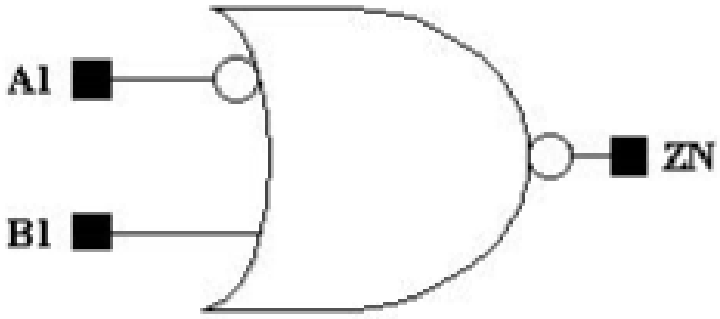


Figura 81: Compuerta INR2D1BWP7T

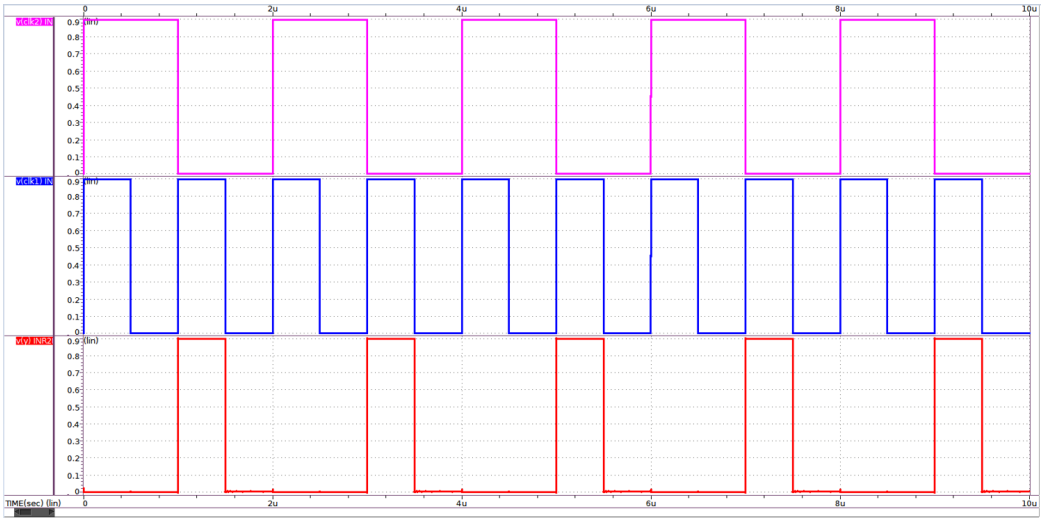


Figura 82: Simulación de INR2D1BWP7T

INPUT			OUTPUT
A1	B1	ZN	
1	0	1	
0	x	0	
x	1	0	

Cuadro 6: Comportamiento lógico de la compuerta INR2D1BWP7T

La siguiente compuerta 81 se obtuvo una NOR de dos entradas, una de ellas negada, la entrada negada se observa cuando esta en uno será un eventual cero en la salida y cuando coincide con la otra señal de entrada, se vuelve un cero nuestra salida.

11.7. Séptima compuerta: ND3D0BWP7T

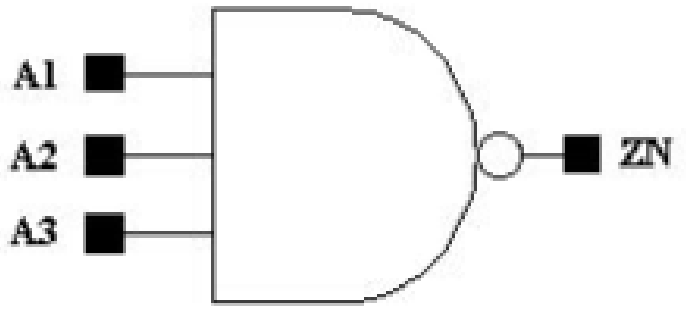


Figura 83: Compuerta ND3D0BWP7T

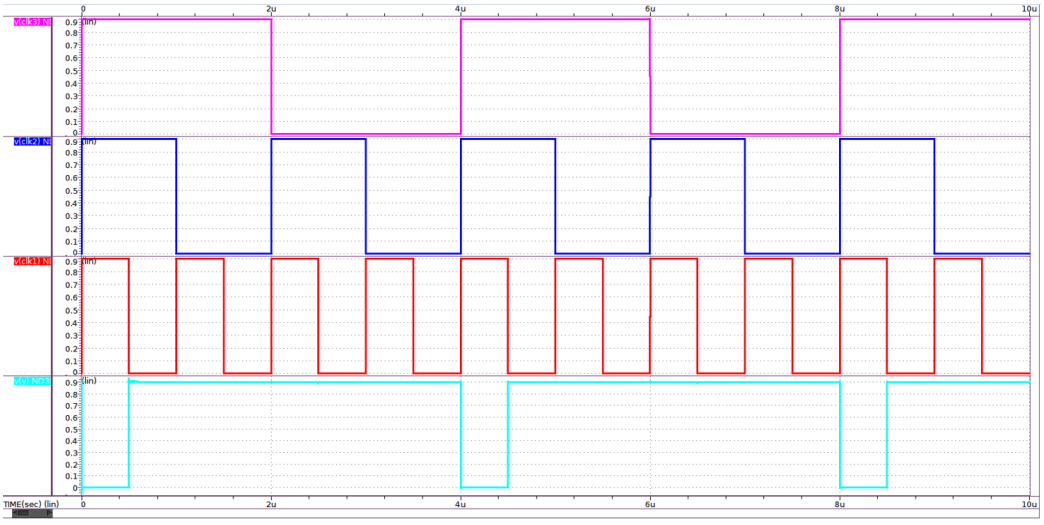


Figura 84: Simulación de ND3D0BWP7T

INPUT			OUTPUT
A1	B1	B2	ZN
0	1	1	1
1	x	x	x
x	0	x	x
x	x	0	0

Cuadro 7: Comportamiento lógico de la compuerta ND3D0BWP7T

La compuerta 83 nuevamente se obtuvo una NAND, ninguna de sus entradas es negada, vemos que el funcionamiento en las tres señales mostradas en la Figura anterior, cuando están en uno nuestra salida es cero como se observó en la señal de salida identificada del color celeste.

11.8. Octava compuerta: OA221D0BWP7T

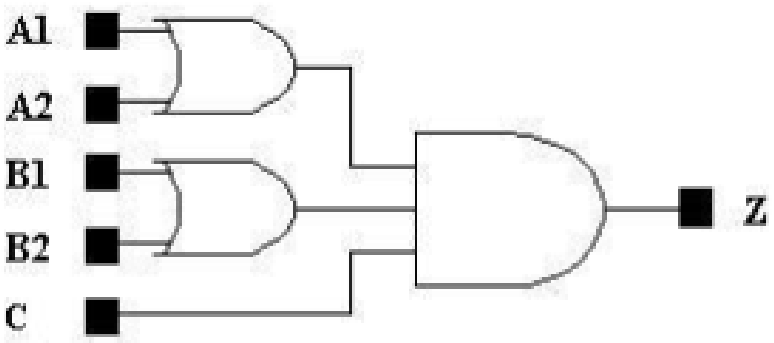


Figura 85: Compuerta OA221D0BWP7T

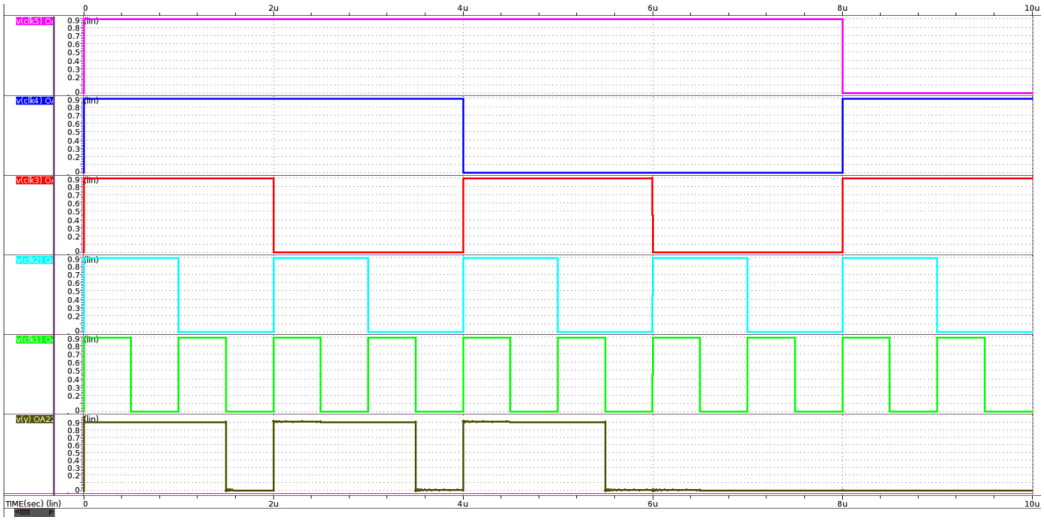


Figura 86: Simulación de OA221D0BWP7T

INPUT			OUTPUT		
A1	A2	B1	B2	C	ZN
0	0	x	x	x	0
x	x	0	0	x	0
x	x	x	x	0	0
1	x	x	1	1	1
x	1	x	1	1	1
1	x	1	x	1	1
x	1	1	x	1	1

Cuadro 8: Comportamiento lógico de la compuerta OA221D0BWP7T

La octava compuerta 85 está conformada con cinco entradas hacia dos OR(2,2) y una AND(1). El funcionamiento es el esperado ya que podemos ver que las salidas de las OR's siempre dieron uno siempre y cuando sus entradas posean un uno, la señal verde es determinante cuando estén en uno para completar la tabla de verdad que sabemos que necesita todas sus entradas en uno para que su salida sea uno.

11.9. Novena compuerta: OA222D0BWP7T

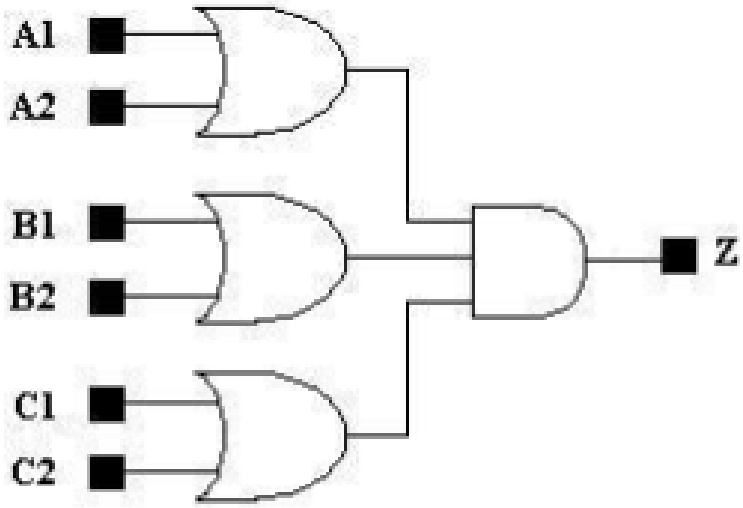


Figura 87: Compuerta OA222D0BWP7T

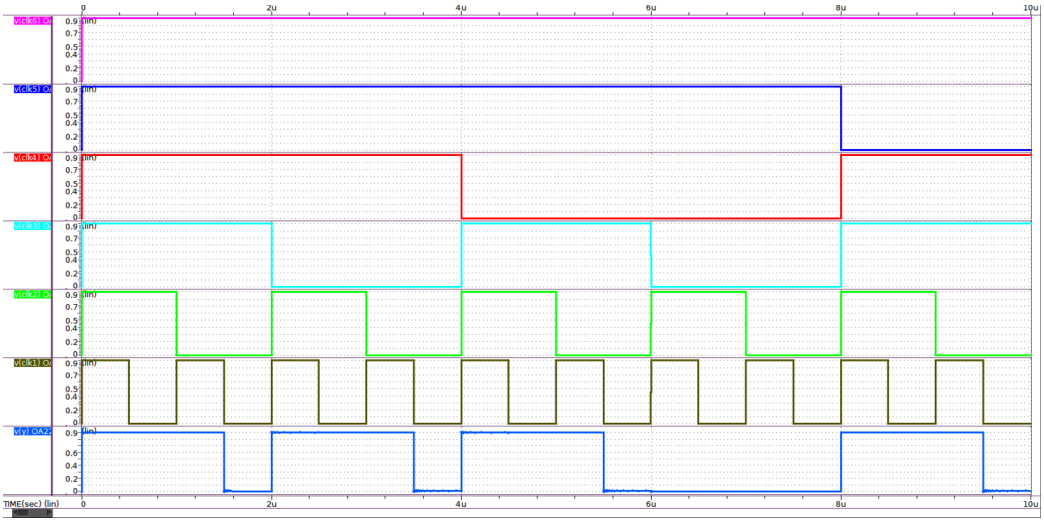


Figura 88: Simulación de OA222D0BWP7T

INPUT			OUTPUT			
A1	A2	B1	B2	C1	C2	ZN
0	0	x	x	x	x	0
x	x	0	0	x	x	0
x	x	x	x	0	0	0
1	x	x	1	x	1	1
x	1	x	1	x	1	1
1	x	1	x	x	1	1
x	1	1	x	x	1	1
1	x	x	1	1	x	1
x	1	x	1	1	x	1
1	x	1	x	1	x	1
x	1	1	x	1	x	1

Cuadro 9: Comportamiento lógico de la compuerta OA222D0BWP7T

La siguiente compuerta 61 está conformada con seis entradas todas hacia tres OR's(2,2,2) y conectadas hacia una AND. El funcionamiento es fácil de observar ya que para que la salida de una AND sea uno todas sus entradas deben ser uno, con la cual en la simulación vemos cuatro momentos en los que no todas sus entradas son uno.

11.10. Décima compuerta: OAI21D0BWP7T

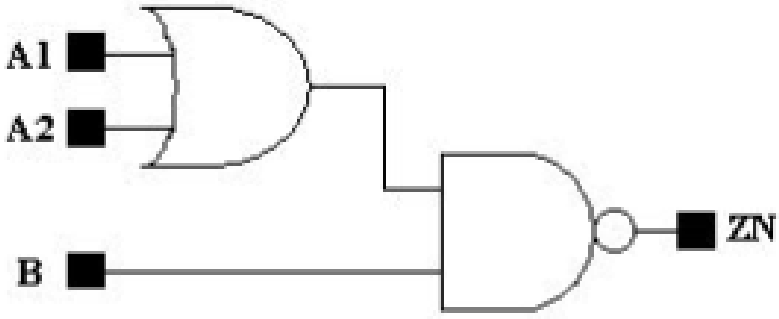


Figura 89: Compuerta OAI21D0BWP7T

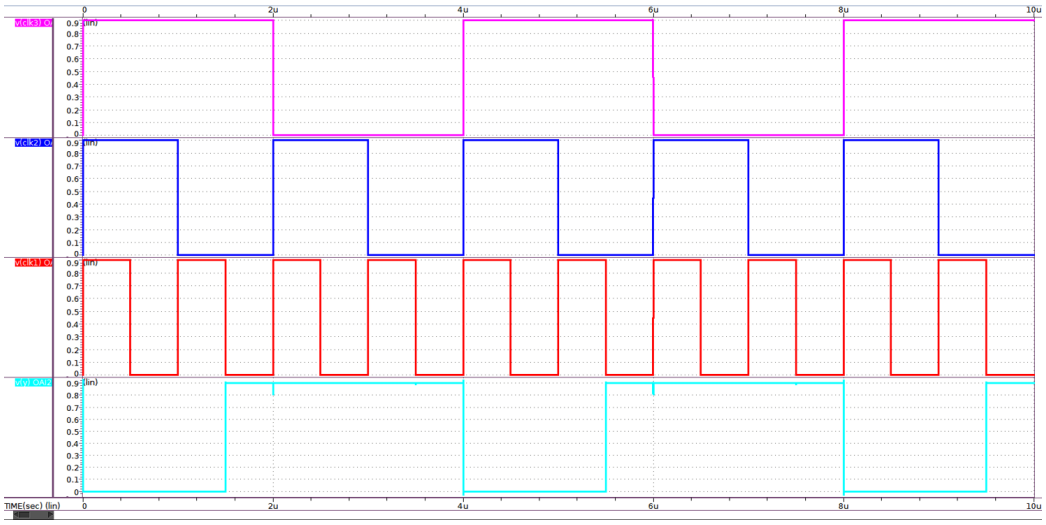


Figura 90: Simulación de OAI21D0BWP7T

INPUT			OUTPUT
A1	B1	B2	ZN
0	0	x	1
x	x	0	1
1	x	1	0
x	1	1	0

Cuadro 10: Comportamiento lógico de la compuerta OAI21D0BWP7T

Para la penúltima compuerta 61 vemos en la señal celeste que su salida siempre será uno toda vez las dos entradas de la compuerta NAND no sean uno.

11.11. Onceava compuerta: OR4D1BWP7T

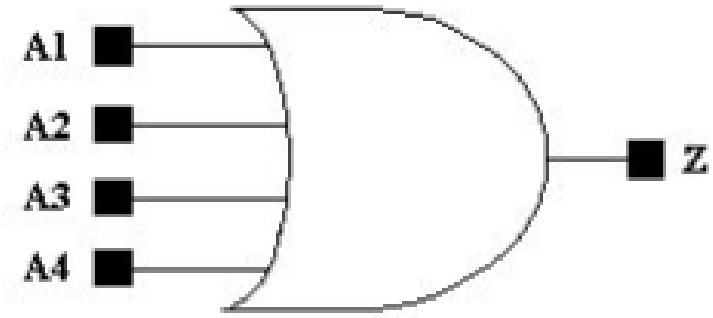


Figura 91: Compuerta OR4D1BWP7T

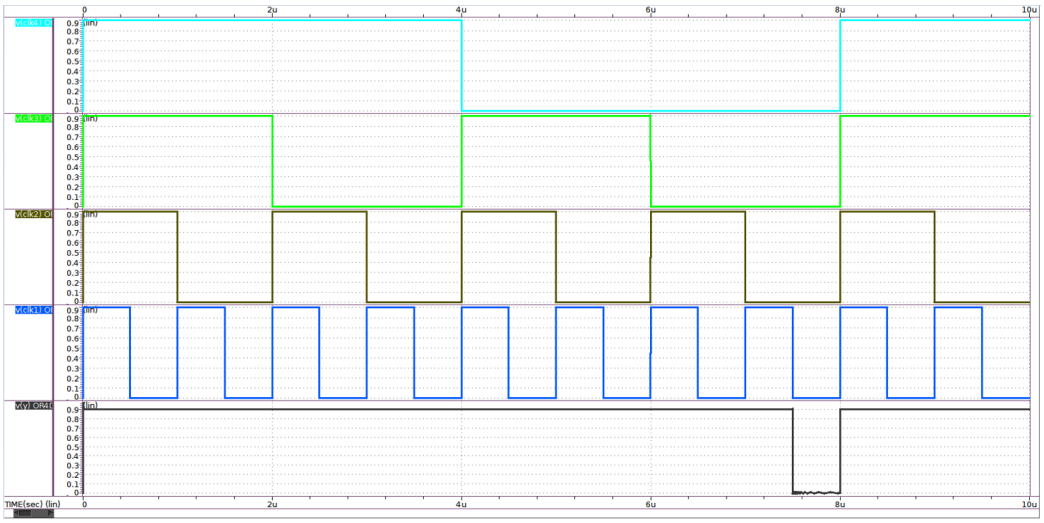


Figura 92: Simulación de OR4D1BWP7T

INPUT			OUTPUT	
A1	A2	B1	B2	ZN
0	0	0	0	0
1	x	x	x	1
x	1	x	x	1
x	x	1	x	1
x	x	x	1	1

Cuadro 11: Comportamiento lógico de la compuerta OR4D1BWP7T

Para la última compuerta 91 se observó un funcionamiento siempre activo ya que únicamente en un slot de tiempo de la simulación las cuatro entradas están en cero, dando como resultado que la salida sea un cero de igual forma.

CAPÍTULO 12

Resolución de tres errores de densidad

En este capítulo se detalla la resolución de tres errores de densidad de los seis que se tenían inicialmente. Primero es importante aclarar que un error de densidad es la falta de metal en las capas del diseño del chip. Como se mencionó anteriormente la ausencia de metal en ciertas secciones del diseño se ve reflejado en la siguiente Figura 93.

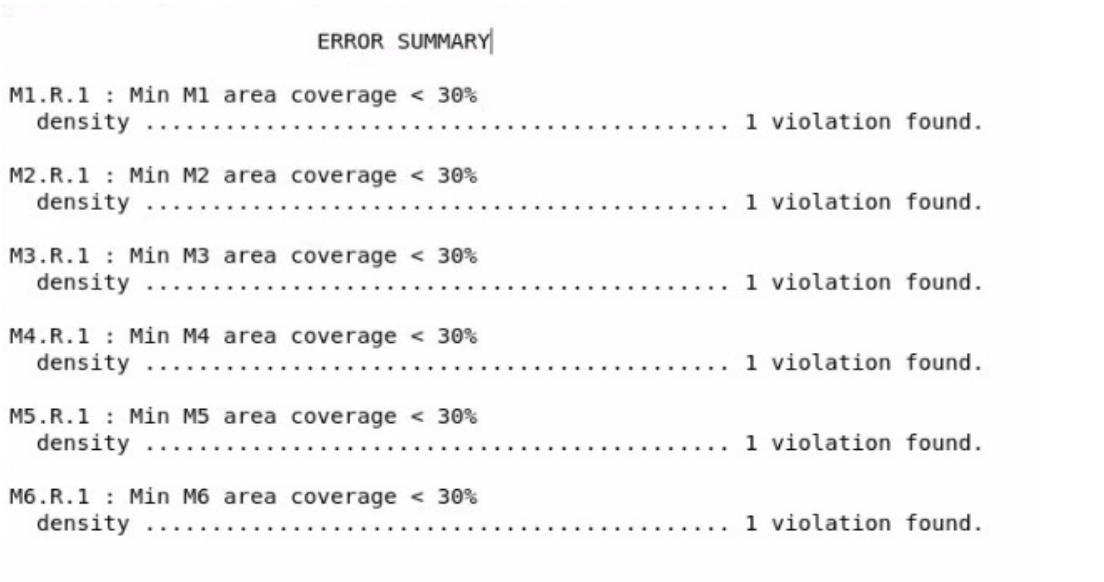


Figura 93: Errores de densidad

En las cuales se encuentran los seis errores con menos de 30 % de densidad y el valor que posee cada uno de los ellos 94, 95, 96, 97, 98, 99 se presenta a continuación:

```
--------------------------------------------------------------------------------
M1.R.1 : Min M1 area coverage < 30%
--------------------------------------------------------------------------------

/home/nanoelectronica2021/Documentos/El_Gran_Jaguar2/ICVLM18_LM16_LM152_6M.215a_pre041518:6723:density
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure Window (x1,y1) (x2,y2)                        Report = Value
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
chip_IO   (107.7700, 113.7700) (447.6050, 418.4700)
                                                        ratio  = 0.1530
                                                        areaL1 = 14420.0092
                                                        areaW  = 94270.7384
```

Figura 94: Valor del metal 1

```
--------------------------------------------------------------------------------
M2.R.1 : Min M2 area coverage < 30%
--------------------------------------------------------------------------------

/home/nanoelectronica2021/Documentos/El_Gran_Jaguar2/ICVLM18_LM16_LM152_6M.215a_pre041518:6732:density
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure Window (x1,y1) (x2,y2)                        Report = Value
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
chip_IO   (107.7200, 113.7200) (447.6050, 418.5200)
                                                        ratio  = 0.0888
                                                        areaL1 = 8370.9209
                                                        areaW  = 94302.9944
```

Figura 95: Valor del metal 2

```
--------------------------------------------------------------------------------
M3.R.1 : Min M3 area coverage < 30%
--------------------------------------------------------------------------------

/home/nanoelectronica2021/Documentos/El_Gran_Jaguar2/ICVLM18_LM16_LM152_6M.215a_pre041518:6741:density
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure Window (x1,y1) (x2,y2)                        Report = Value
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
chip_IO   (113.7200, 113.7200) (420.7600, 418.5200)
                                                        ratio  = 0.2249
                                                        areaL1 = 20789.7169
                                                        areaW  = 92445.3632
```

Figura 96: Valor del metal 3

```
--------------------------------------------------------------------------------
M4.R.1 : Min M4 area coverage < 30%
--------------------------------------------------------------------------------

/home/nanoelectronica2021/Documentos/El_Gran_Jaguar2/ICVLM18_LM16_LM152_6M.215a_pre041518:6750:density
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure Window (x1,y1) (x2,y2)                        Report = Value
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
chip_IO   (113.7200, 113.7200) (420.7600, 418.5200)
                                                        ratio  = 0.0371
                                                        areaL1 = 3425.3160
                                                        areaW  = 92445.3632
```

Figura 97: Valor del metal 4

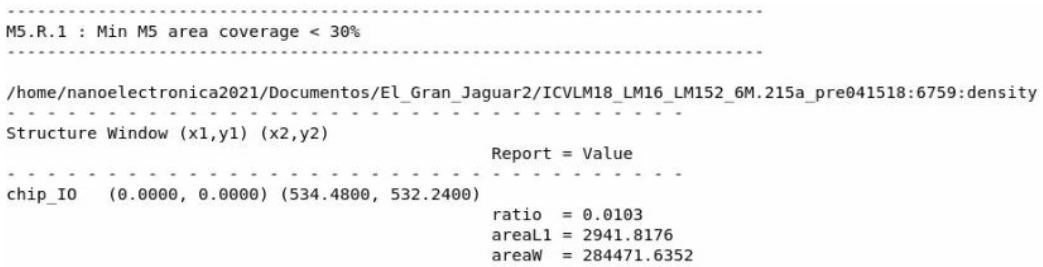


Figura 98: Valor del metal 5

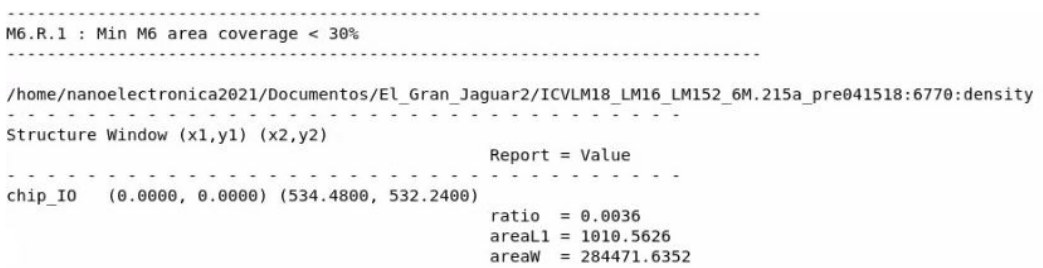


Figura 99: Valor del metal 6

Se buscó incrementar cada metal para cumplir el requisito de DRC, esto se logró a partir de la aplicación de IC Compiler II junto con Diego Equite, los tracks inicialmente se observaban como en la Figura 100 se cambiaron manualmente la anchura(widths) de los metales 101, teniendo en cuenta las reglas como la separación entre cada uno que se requiere.

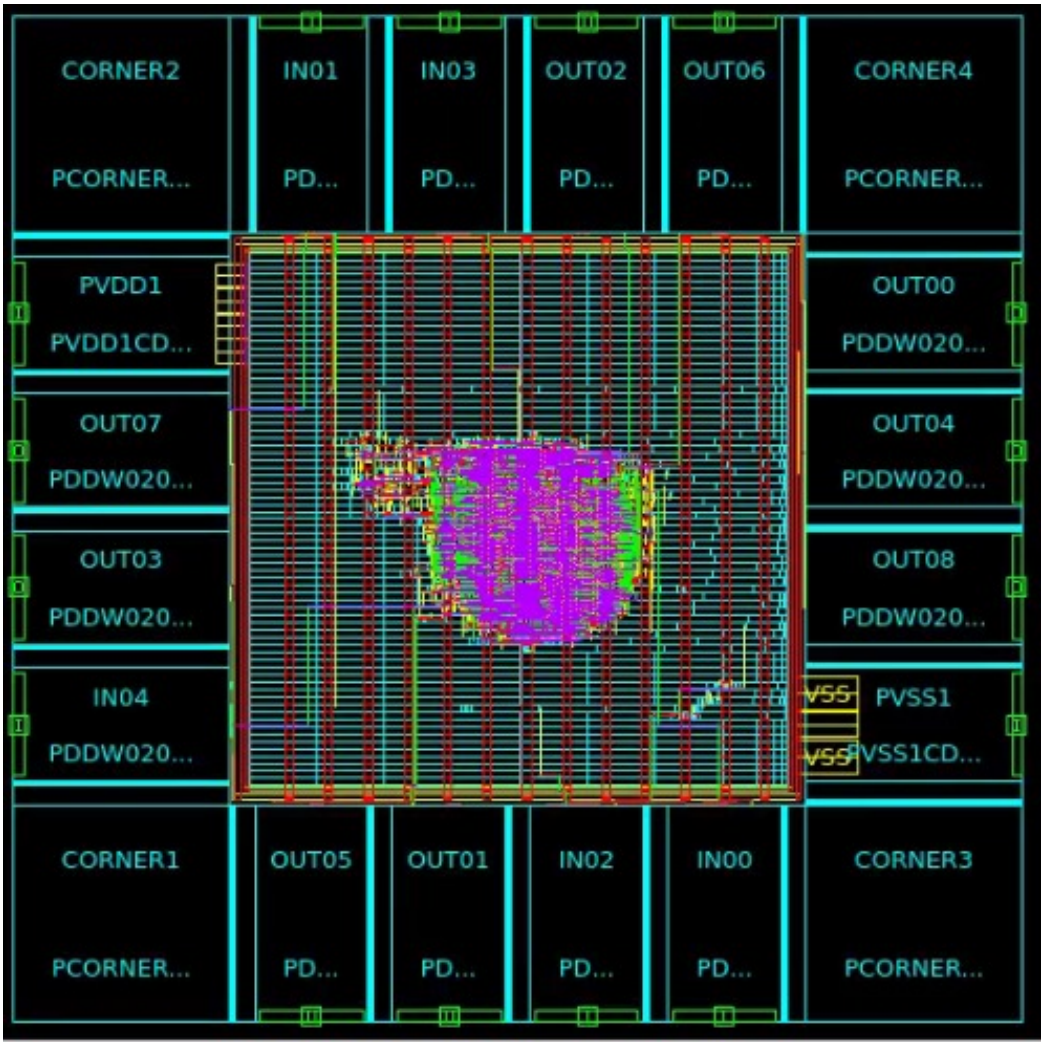


Figura 100: Tracks sin modificar

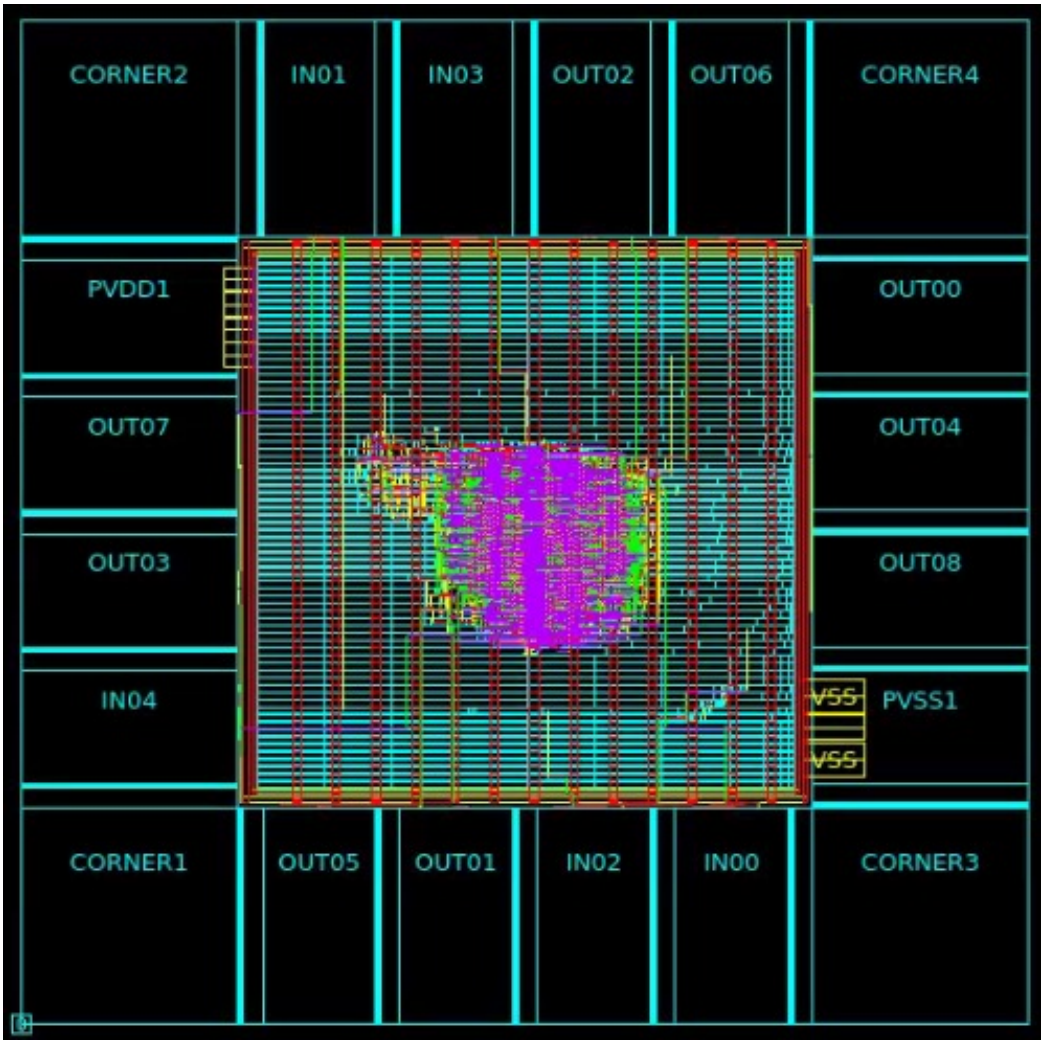


Figura 101: Tracks modificados

Sin embargo, se decidió mejor modificar el código ya que cambiar directamente la anchura en la aplicación como se mostró en las anteriores imágenes era muy tedioso y complicado al momento de no juntar los tracks, en cambio en el código como se muestra en la Figura 102 solo se modificó el valor y teniendo también un valor de separación modificable.

```
set_attribute -objects [get_shapes PATH_16_61] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_62] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_64] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_65] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_66] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_37] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_2] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_1] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_40] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_68] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_67] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_61] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_62] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_64] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_65] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_66] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_37] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_2] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_1] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_40] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_3] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_45] -name width -value 0.47
set_attribute -objects [get_shapes PATH_16_8] -name width -value 2.3
set_attribute -objects [get_shapes PATH_16_44] -name width -value 2.4
set_attribute -objects [get_shapes PATH_16_62] -name width -value 2.5
set_attribute -objects [get_shapes PATH_16_61] -name width -value 2.5
```

Figura 102: Código fuente

Al realizar el DRC se logró superar el requisito de los primeros tres metales 103, sin embargo como vemos en la Figura 104 se generaron más errores de diseño por lo cual se decidió no modificar de está manera para cumplir el requisito.

```
                    ERROR SUMMARY

M4.R.1 : Min M4 area coverage < 30%
  density ..................................... 1 violation found.

M5.R.1 : Min M5 area coverage < 30%
  density ..................................... 1 violation found.

M6.R.1 : Min M6 area coverage < 30%
  density ..................................... 1 violation found.
```

Figura 103: Solución de tres errores de densidad

```
M1.S.1 : Min. M1 space < 0.23
  external1 ..................................... 117 violations found.

M1.W.1 : Min. M1 width < 0.23
  internal1 ..................................... 62 violations found.

M2.R.1 : Min M2 area coverage < 30%
  internal1 ..................................... 1 violation found.

M2.S.1 : Min. M2 space < 0.28
  external1 ..................................... 28 violations found.

M3.W.1 : Min. M3 width < 0.28
  internal1 ..................................... 14 violations found.

M4.R.1 : Min M4 area coverage < 30%
  density ....................................... 1 violation found.

M5.R.1 : Min M5 area coverage < 30%
  density ....................................... 1 violation found.

M6.R.1 : Min M6 area coverage < 30%
  density ....................................... 1 violation found.

VIA1.S.1 : Min. VIA1 space < 0.26
  external1 ..................................... 18 violations found.
```

Figura 104: Nuevos errores generados

CAPÍTULO 13

Conclusiones

1. Al utilizar la herramienta de Formality se evidenció que brinda un análisis comparativo equivalente entre varios diseños y no hubo una diferencia entre ambos circuitos en el estudio.

2. Con la herramienta de Design Compiler se logró obtener un circuito pequeño y rápido volviendo está etapa más eficiente.

3. Al realizar el análisis de las guías de Synopsys del 2019 se evidenció que los comandos e imágenes era obsoletos por lo que se actualizaron a los más recientes.

4. Al realizar las pruebas de las compuertas descritas en HSPICE por separado analizándolas con las tablas de verdad se comprueba que el funcionamiento final del chip está dentro de lo esperado, sin embargo, no es la única alternativa.

5. Se logró trabajar en conjunto con el grupo de automatización de la línea de investigación mejorando los tiempos de trabajo para lograr el desarrollo del trabajo.

6. Al tratar de resolver la mínima cantidad de errores de manera forzada puede ocasionar una mayor cantidad de errores de los que se esperaban solucionar.

7. Al continuar el proyecto de diseño de un nanochip de 180 nm implementando la variante de un nuevo flujo se establece que permitirá a las futuras generaciones una mejor fabricación de chips gracias al proceso de las herramientas aprovechando el máximo de cada una de ellas, sin importar la tecnología con la que se desarrolle el chip.

CAPÍTULO 14

Recomendaciones

1. Trabajar los laboratorios en el escritorio de su ordenador, haciendo una copia de estos mismo de la carpeta de Synopsys. Evitando trabajar con archivos descargados desde algún ordenador, esto genera una corrupción en los documentos.

2. A pesar de no tener una herramienta designada para estudiar a profundidad, es necesario siempre apoyarse en las guías de usuario de cada herramienta para tener un mejor manejo de los comandos y las múltiples opciones que poseen.

3. Es importante separar las tareas por secciones ya sea por muchas tareas individuales asegurando o esperar un mejor resultado.

CAPÍTULO 15

Bibliografía

[1] *Documentation of Synopsys*, ver. 220I, Synopsys, Inc, jul. de 2022.

[2] Synopys, "VCS User Guide," *Synopsys*, 2022.

[3] Synopys, "Design Compiler® User Guide," *Synopsys*, 2022.

[4] Synopys, "IC Compiler™ II Implementation User Guide," *Synopsys*, 2022.

[5] *PrimeTime™ User Guide*, ver. K-2015.12, December 2015, Synopsys, Inc, sep. de 2021.

[6] Synopys, "Formality® User Guide," *Synopsys*, 2022.

[7] *TestMAX ATPG and TestMAX Diagnosis User Guide*, ver. Version T-2022.03, March 2022™, Synopsys, Inc, sep. de 2021.

[8] Synopys, "IC Validator User Guide," *Synopsys*, 2022.

[9] *StarRC™ User Guide and Command Reference*, ver. Version T-2022.03, March 2022, Synopsys, Inc, sep. de 2021.

[10] *PrimeSim™ HSPICE S-2021.09-SP2 User Documentation*, ver. S-2021.09-SP2, Synopsys, Inc, sep. de 2021.

[11] *TCB018GBWP7T TSMC 0.18um Standard Cell Library Databook*, ver. 270a, Synopsys, Inc, mayo de 2009.

[12] *TPH018NV3 SL TSMC 0.18um Standard I/O Library Databook*, ver. 280B, Synopsys, Inc, jul. de 2010.

CAPÍTULO 16

Anexos

16.1. Flujo de diseño tutorial de Synopsys

16.2. Laboratorio 1: Simulación lógica. VCS

Synopsys Design Flow Tutorial

Laboratorio 1: Simulación lógica. VCS

Objetivo

Aprenda a simular RTL y diseño de nivel de compuertas 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

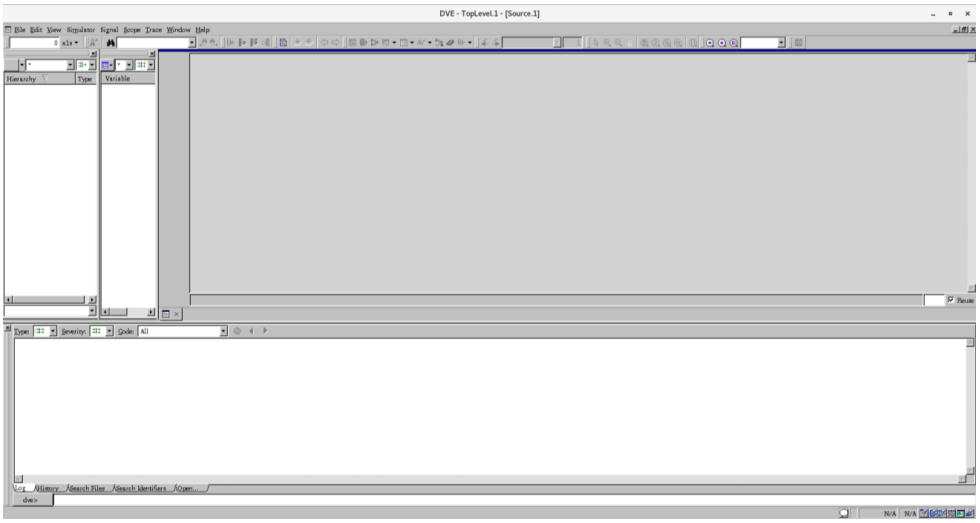
1. Primero compile johnson.v RTL source y Johnson_test.v testbenches usando VCS con el siguiente comando:
*Nota: se trabajará en la carpeta de work. Utilizar el siguiente comando localizado en el archivo RUN.txt o aquí abajo:

```
%       vcs ../source/johnson_test.v ../source/johnson.v -full64 –debug_access+all
```

*Nota: Se crearán carpetas necesarias en la carpeta de work.
-gui Habilita DVE
-debug_all Habilita la depuración de la línea de comandos, incluido el seguimiento de línea (opcional)

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

2. Configurar DVE

En la barra de menú, seleccione *Simulador* > *Configuración*,
En "Simulador ejecutable" pulse "Examinar..." y seleccione el archivo "SIMV" en el directorio actual

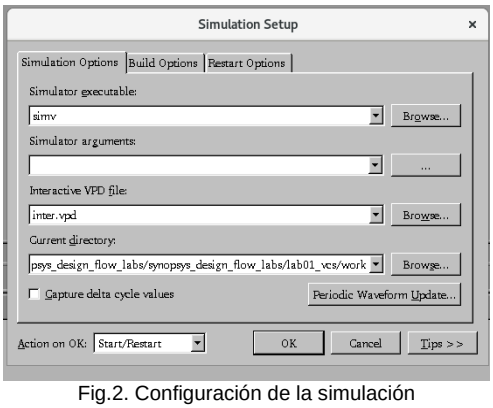


Fig.2. Configuración de la simulación

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

- **Simulador ejecutable:** especifica el nombre de un ejecutable de simulador.
- **Argumentos del** simulador: identifica los argumentos del simulador.
- **Archivo VPD interactivo**: 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 aserción de estos archivos.
- **Directorio actual**: especifica la ruta completa del ejecutable del simulador.

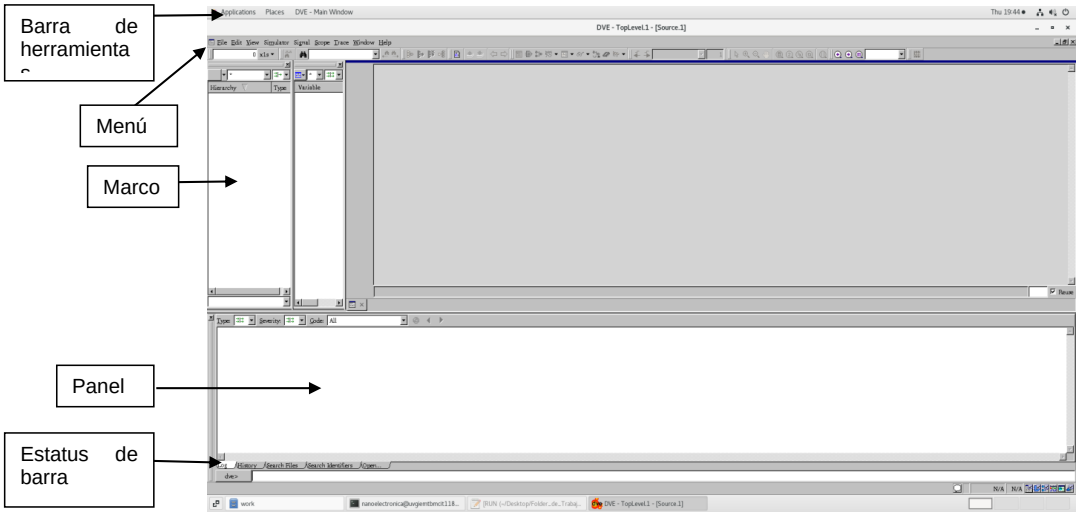


Fig.3. Descripción general de la ventana DVE

Una ventana TopLevel contiene:

1. Marco
2. Menús
3. Barras de herramientas
4. Barra de estado
5. Destinos de panel.

Visualización de una forma de onda

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 controlador o la carga, 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 la información de forma de onda de las señales en la vista onda (Fig.4)
Ahora abriremos las nuevas vistas (Add to waves), daremos click derecho es nuestro archivo Johnson_test, localizado en la sección de de Hierarchy.

Uso de comandos de menú del simulador Ejecuta la simulación hasta que se alcanza un punto de interrupción, finaliza la simulación o durante la duración especificada en el cuadro de diálogo Tiempo de

mantenimiento establecido o en las entradas de tiempo de la barra de herramientas.

Cuando la simulación se está ejecutando, este icono se activa. Haga clic para detener la simulación.
El icono de stop, de la simulación puede variar depende la versión or lo tanto puede salir ese icono o no cambiar y volver a apachar la flecha y terminar la simulación.

*Nota: Centraremos la simulación con la tecla F (Zoom full).

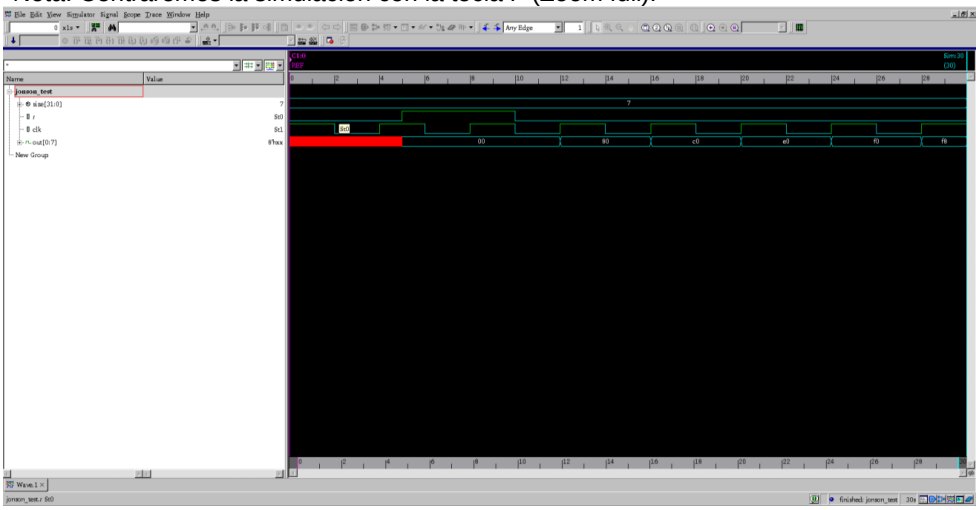


Fig.4. Ventana WaveView

Seguimiento de controladores y cargas

1. Rastree los controladores y las cargas de una señal en cualquier momento para ver los controladores cargas que causaron un cambio de valor y ver todos los controladores / cargas que posiblemente contribuyeron a un valor de señal.
2. Seleccione una señal en una vista o panel. Por ejemplo, panel Datos, vista Origen, vista Lista, etc.
3. Haga clic con el botón derecho y seleccione Controladores de seguimiento o Cargas de seguimiento. Cuando se realiza el seguimiento de un controlador, se creará un nuevo panel Controlador si no existe ninguno en el marco de nivel superior actual. Si existe un panel de controladores, la información del controlador se agregará a la parte superior de la lista.

Además, el primer controlador se resaltará en la vista Fuente y se anotará con un nodo azul en la canaleta. En la vista Onda, haga doble clic en una forma de onda para ver sus controladores. (Figura 5)

Vincule los paneles del controlador a la vista Origen en el mismo marco de nivel superior y en la vista Esquema de trazado. Los botones de opción Vincular a, en la parte superior derecha del panel, muestran las ventanas vinculadas actuales. Al vincular una vista Origen y Esquema, cuando se selecciona el objeto en el panel Controladores, el objeto también se seleccionará en las vistas vinculadas.

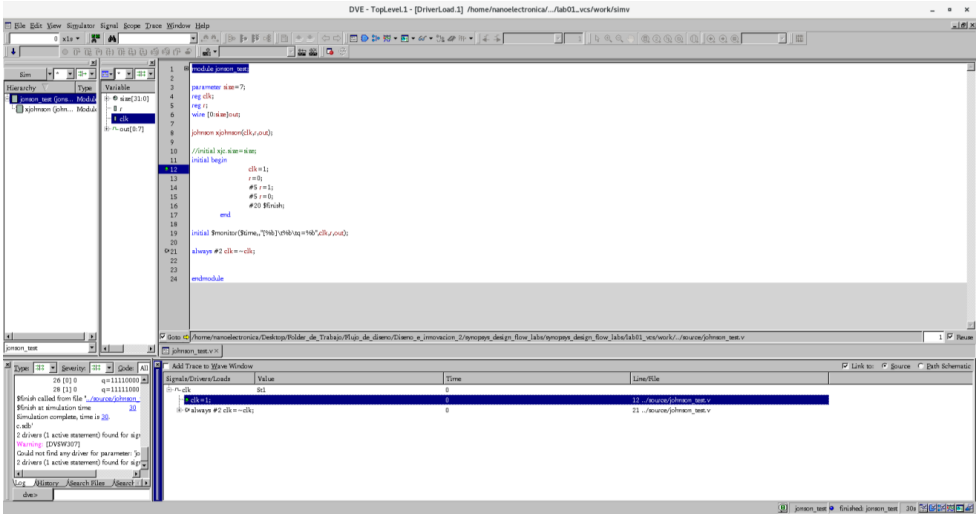


Fig.5.Ventana de nivel superior con controladores de seguimiento y cargas

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 Onda.
2. Haga clic con el botón derecho y seleccione Comparar.

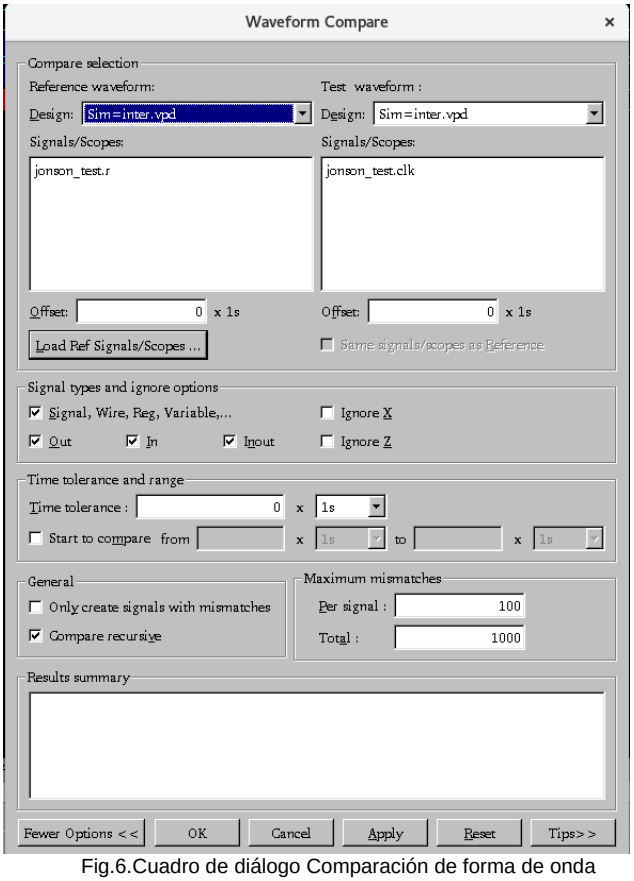


Fig.6.Cuadro de diálogo Comparación de forma de onda

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 Más opciones (Fig. 6).
5. El cuadro de diálogo se expande y se muestran opciones adicionales.
6. 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, independientemente de los valores de la señal de prueba.
Introduzca una tolerancia de tiempo para filtrar los valores de falta de coincidencia 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 que se van a 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 Wave actual.
10. Seleccione un resultado en la vista Onda, haga clic con el botón derecho y seleccione Mostrar información de comparación.

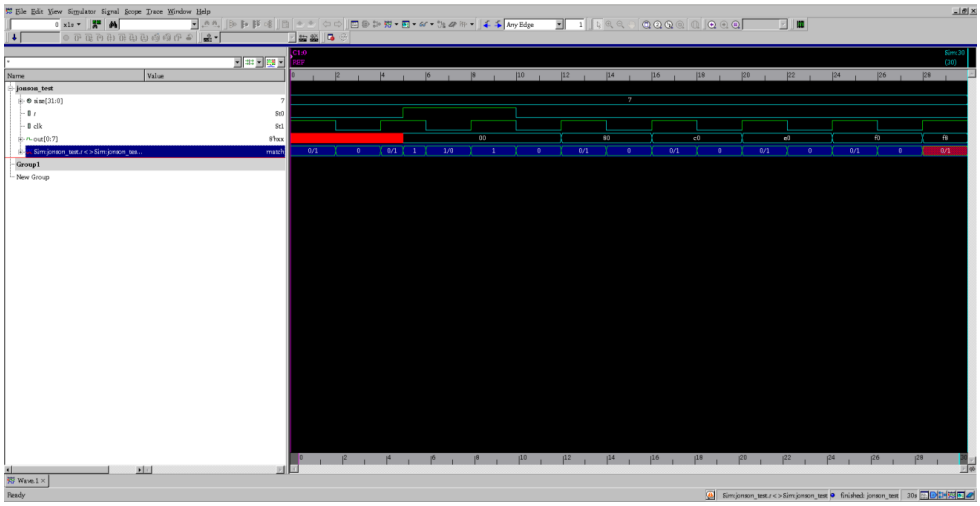


Fig.7. Grupos de señales comparados en la vista de onda

Selección de señal en la vista esquemática

Para seleccionar una señal en la vista Esquema:

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

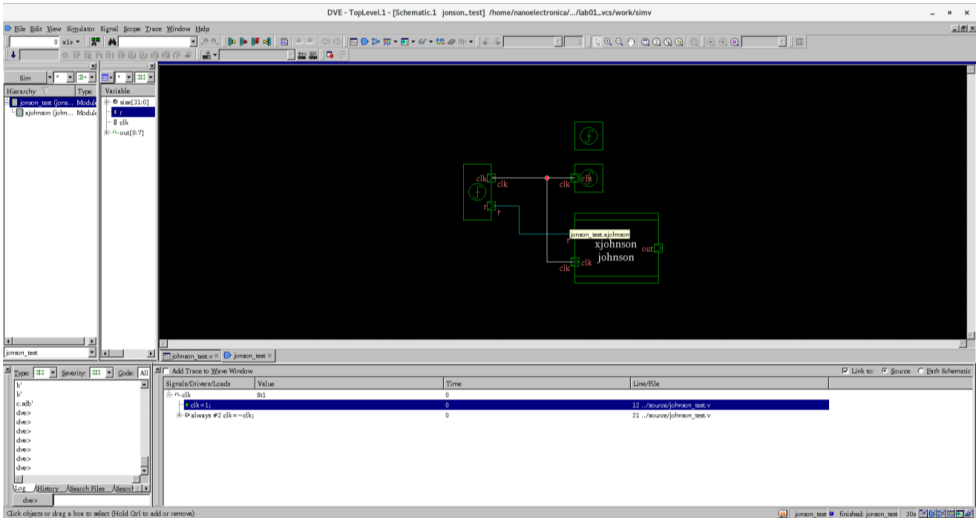


Fig.8. Grupos de señales resaltados en WaveView

Uso de radix´s definidos por el usuario

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

Para crear, eliminar, importar y exportar un radio 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 realice 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.

El radix está editado.

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 radio se exporta.
9. Active 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 Onda.
10. Haga clic en Aceptar o aplicar para guardar el radio definido por el usuario.

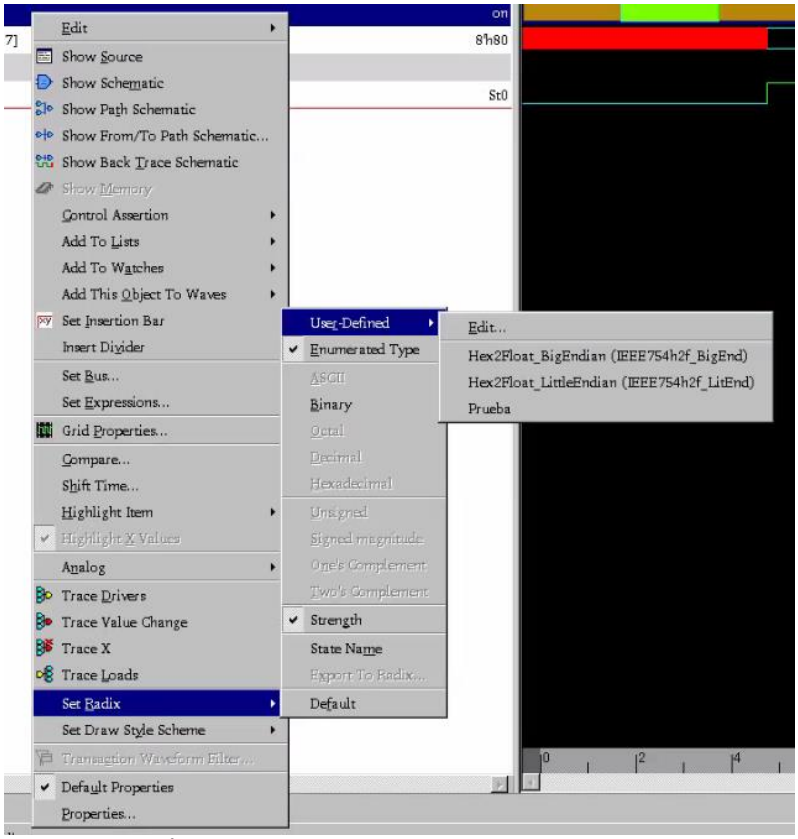


Fig.9. Cuadro de diálogo Editar radix definido por el usuario con ventana de vista de onda

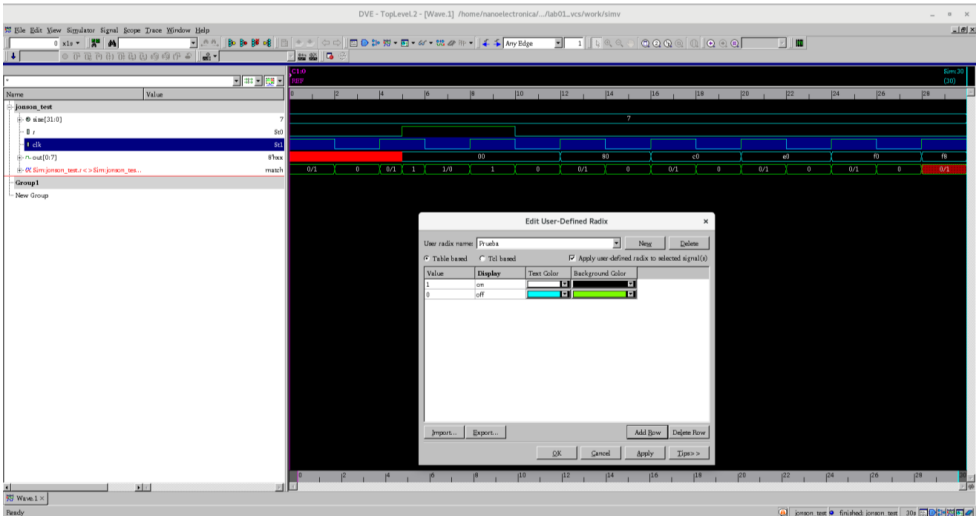


Fig.10. Editor de Radix

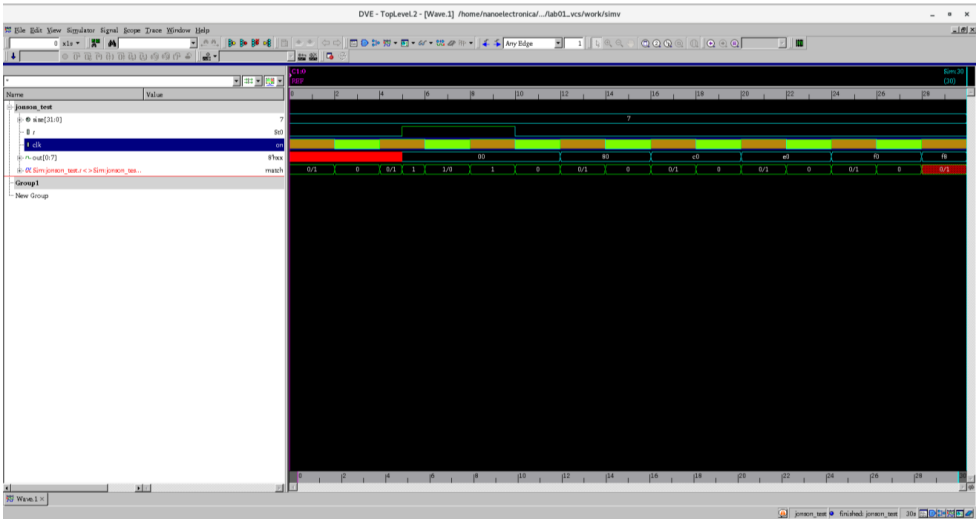


Fig.11. Vista con radix aplicado.

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

```
exit
```

16.3. Laboratorio 2: Síntesis lógica. Design Compiler

Synopsys Design Flow Tutorial

Laboratorio 2: Síntesis lógica. Compilador de diseño

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:

1. Produzca diseños ASIC rápidos y eficientes en el área mediante el empleo de bibliotecas de celdas estándar o especificadas por el usuario
2. Traducir diseños de una tecnología a otra
3. Explore 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.
4. Sintetizar y optimizar máquinas de estado finito
5. Integre 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
6. Cree y divida esquemas jerárquicos automáticamente.

Flujo de síntesis básico

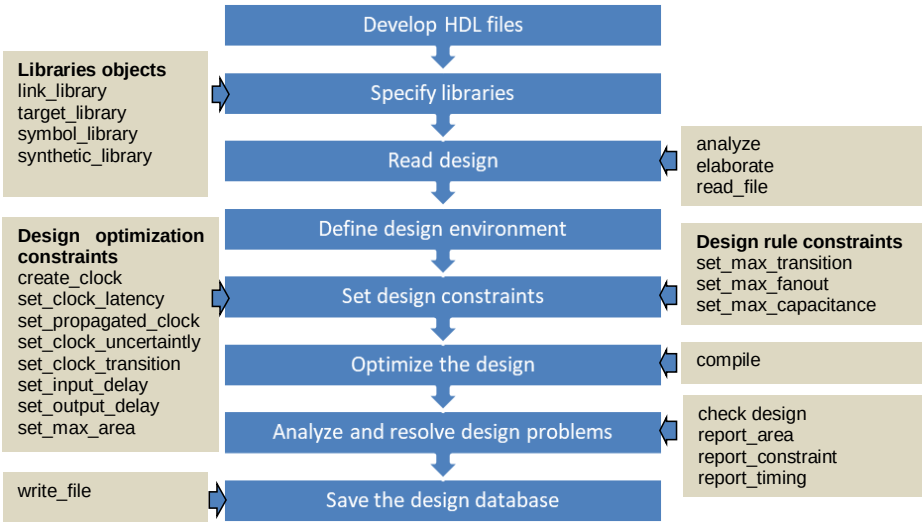


Fig.1.Flujo de síntesis básico

Design Compiler input and output files

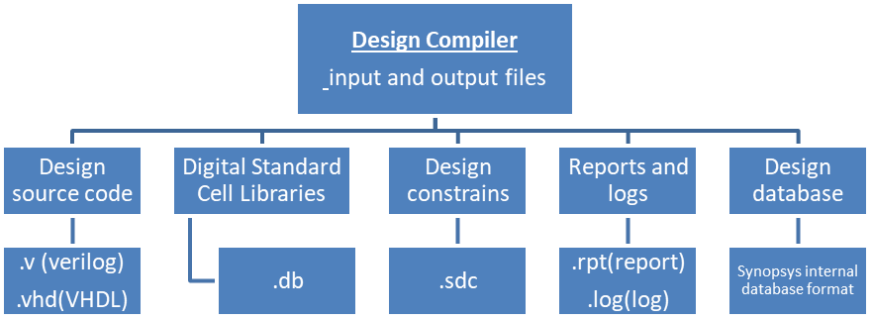


Fig.2.Salidas y entradas

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)

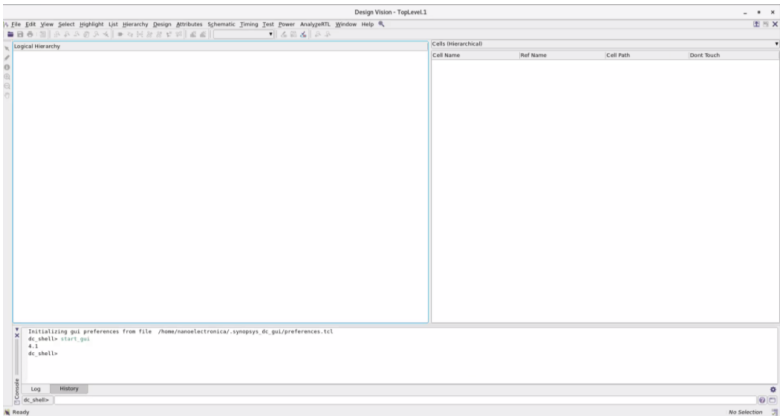


Fig.3. Ventana gui de nivel superior del compilador de diseño

2. Cuando se inicia Design Compiler, lee automáticamente el archivo ".synopsys_dc.setup" del directorio actual. En este laboratorio su 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 han configurado, abra **File>Setup** ventana (Fig. 2).

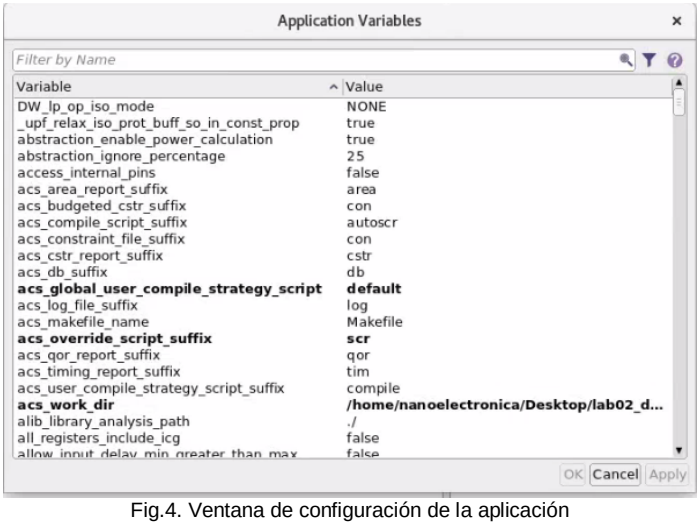


Fig.4. Ventana de configuración de la aplicación

Se debe rectificar que los archivos siguientes si estén correctamente:

- Search path: sim_ver
- Link library: saed14rvt_tt0p8v25c.db
- Target library: saed14rvt_tt0p8v25c.db
- Symboly library: your_library.sdb
- Synthetic library: “Vacia”

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> source ../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).

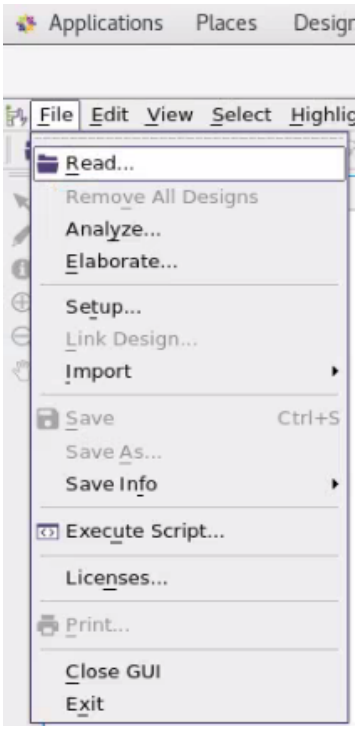


Fig.5. Read design

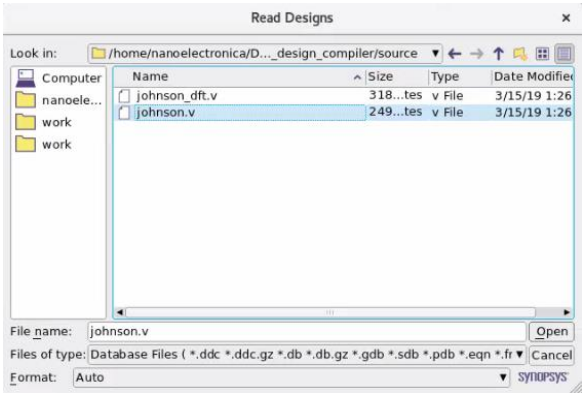


Fig.6. Read designs box

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 analizar una ganancia. Después de analizar un diseño, vuelva a analizarlo solo cuando cambie (Fig.5 y Fig.6).

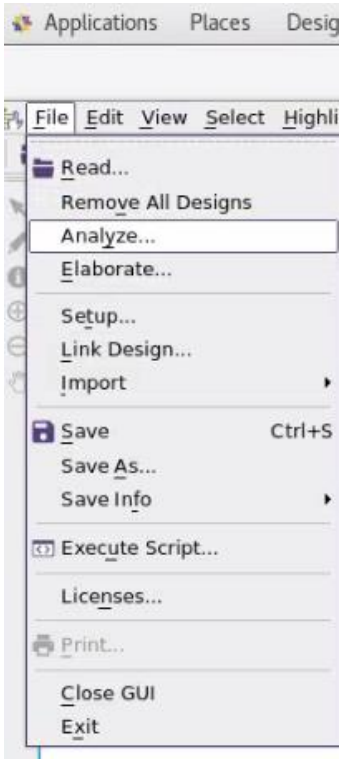


Fig.7. Read Analyze

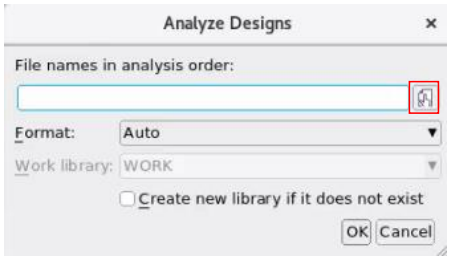


Fig.8. Analyze designs

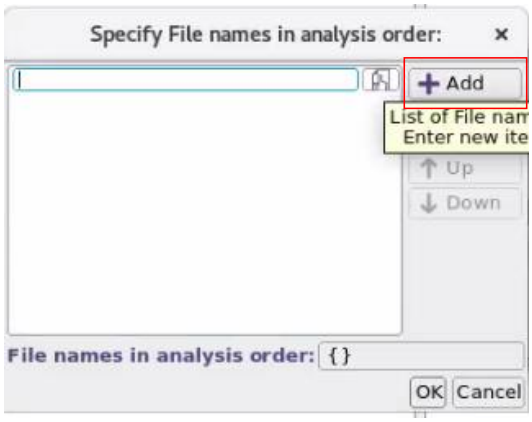


Fig.9. Analyze designs add

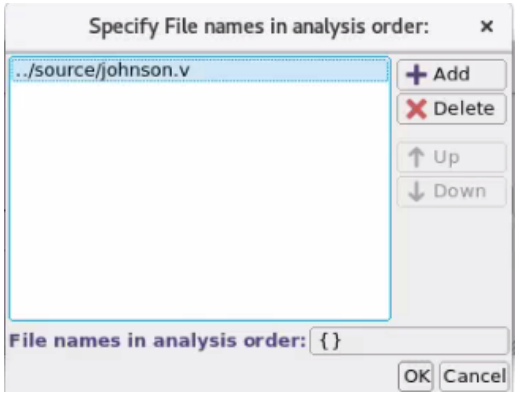


Fig.10. Analyze designs box

3.3 El comando **Elaborate** hace lo siguiente:

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

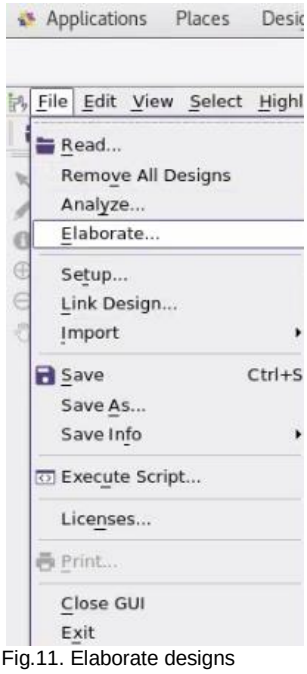


Fig.11. Elaborate designs

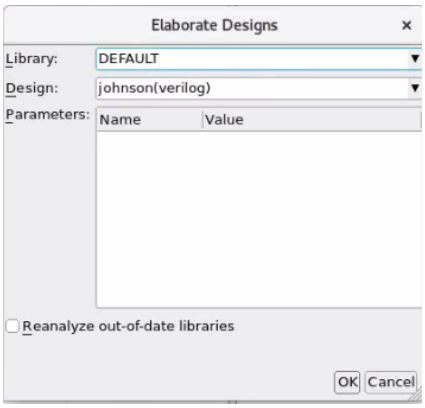


Fig.12. Elaborate designs box

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 el archivo johnson.sdc.
Las restricciones de optimización comprenden:

1. Limitaciones de tiempo (rendimiento y velocidad)
2. Retrasos de entrada y salida (rutas síncronas)
3. Retardo mínimo y máximo (rutas asincrónicas)
4. Superficie máxima (número de puertas)

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

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

Para obtener el esquemático damos click derecho y nos aparecerán las siguientes opciones como se muestra en la figura 13.

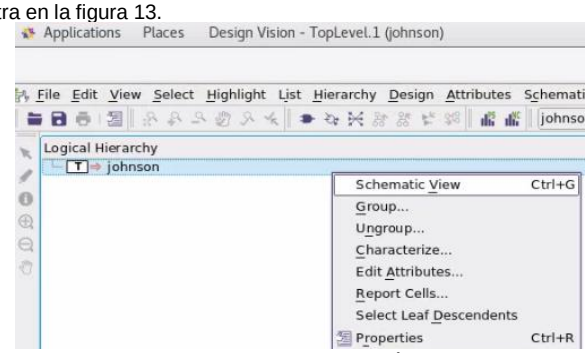


Fig.13. Vista del esquemático

Después de todos los pasos mencionados anteriormente, la vista de diseño del compilador de diseño se muestra en la Fig.14.

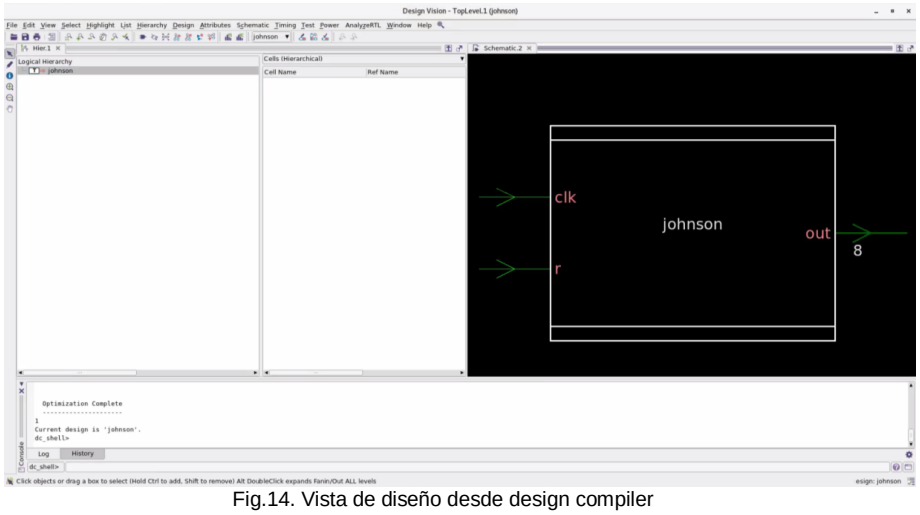


Fig.14. Vista de diseño desde design compiler

3.7 El comando **compile** realiza la síntesis y optimización lógica y a nivel de puerta en el diseño actual. La optimización se controla mediante 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.15).

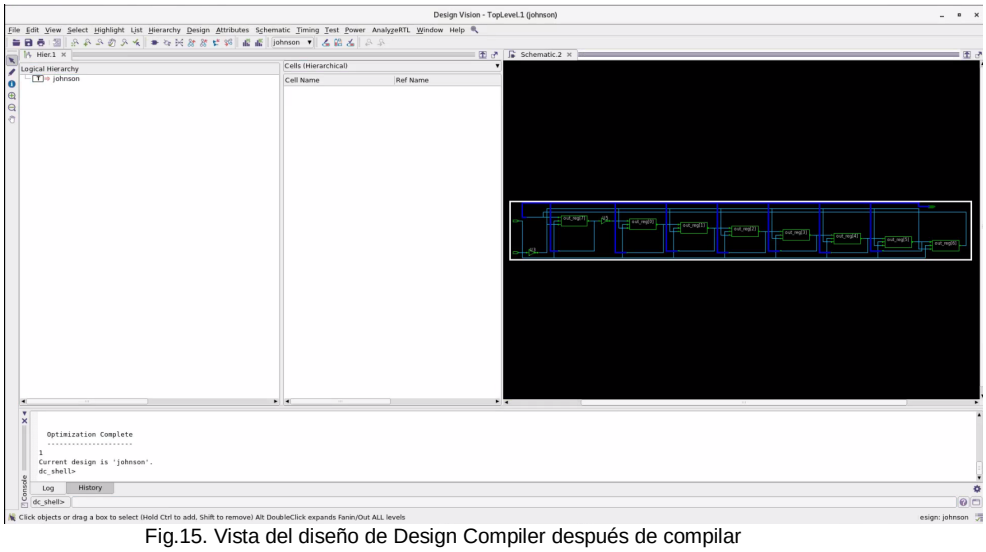


Fig.15. Vista del diseño de Design Compiler después de compilar

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.

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/saed14rvt_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 Escriba un diseño desde la memoria en formato .ddc (formato de base de datos interna synopsys), así como .v y .vhdl.

```
dc_shell> write -format verilog ../results/johnson.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 DFT Compiler.

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, elija de forma predeterminada el estilo de escaneo **multiplexed_flip_flop**.

Flujo básico de síntesis de escaneo

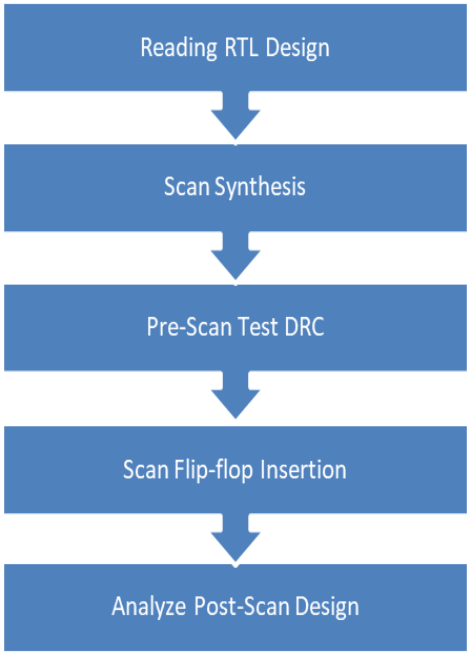


Fig.16. Flujo básico de síntesis de escaneo

Tareas de laboratorio

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

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

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

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

```
dc_shell> set test_default_scan_style
```

Con 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 Esquema de la barra de herramientas como se muestra en la Fig. 1.

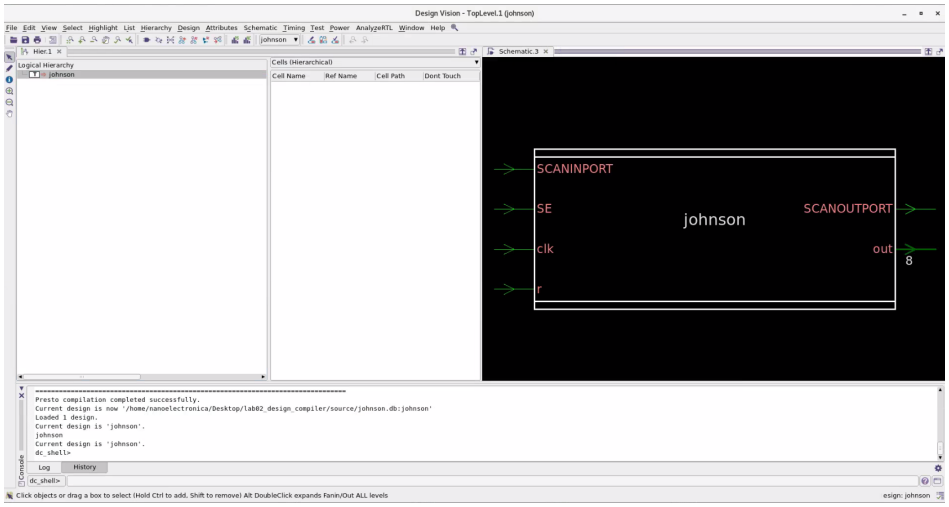


Fig.17.Vista esquemática de apertura de Johnson

La vista esquemática se muestra en la Fig. 18.

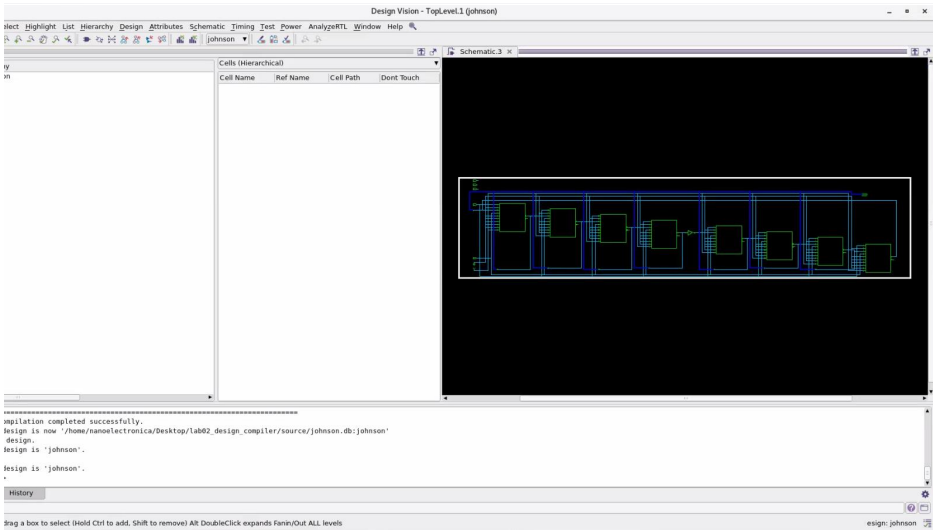


Fig.18. Vista esquemática de johnson después de DFT

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

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

5. Cree un protocolo test 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 infracciones 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.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 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 desencadenadores de escaneo para evitar los cambios de característica esquemática. 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.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 especifica 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 un circuito 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**. **-buffer_constraints** opción amortigua las constantes lógicas en lugar de duplicarlas.

```
dc_shell> set_fix_multiple_port_nets –all
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. Escriba un diseño desde la memoria en un archivo con el format.sdc (formato de base de datos interna synopsys), 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
```

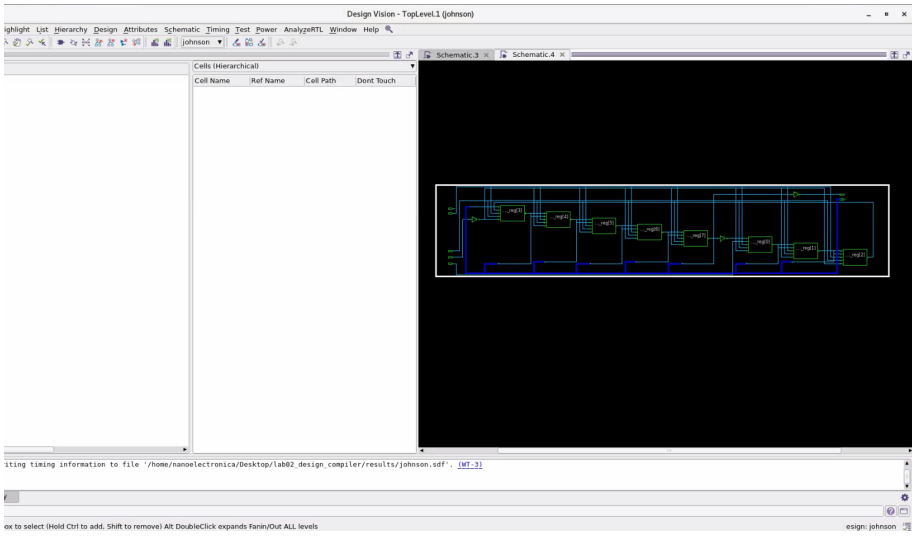


Figura 19. Vista esquemática de johnson final

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

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

```
dc_shell> exit
```

16.4. Laboratorio 3: Diseño físico. IC Compiler II

Synopsys Design Flow Tutorial

Laboratorio 3: Diseño Físico. IC Compiler II

Objetivo

Aprenda a compilar el diseño con IC Compiler II y qué tipo de comandos se necesitan para ello.

Introducción

IC Compiler II es una herramienta de diseño de síntesis única y convergente de netlist a GDSII para diseñadores de chips que desarrollan diseños submicrónicos muy profundos. Toma como entrada una lista de red a nivel de puerta, un plano de planta detallado, restricciones de tiempo, bibliotecas físicas y de tiempo, y datos de proceso de fundición, y genera como salida un archivo de formato GDSII del diseño.

IC Compiler II proporciona dos interfaces de usuario:
- Interfaz shell (icc2_shell)
La interfaz de línea de comandos de IC Compiler II se utiliza para scripts, modo por lotes y operaciones de pulsador.
- Interfaz gráfica de usuario (GUI)
La interfaz gráfica de usuario de IC Compiler II es una herramienta avanzada de análisis y edición física.

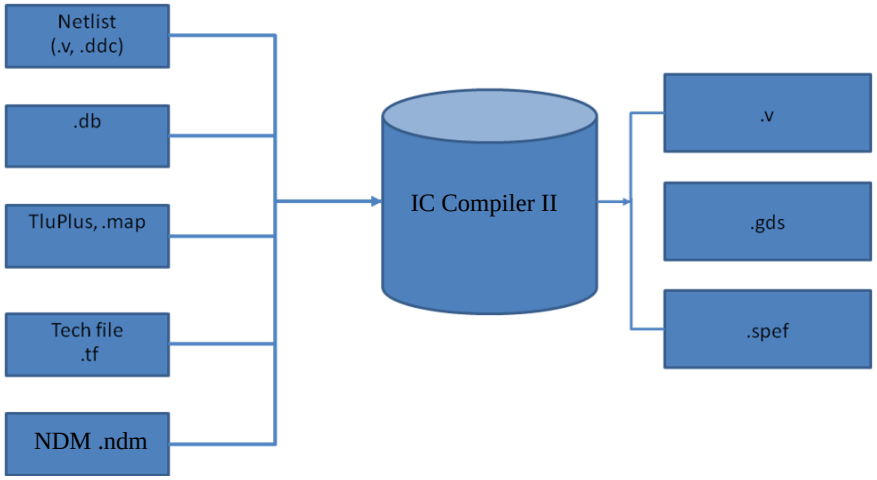


Fig.1.Archivos de entrada y salida del compilador IC II

La información de lógica, temporización y potencia de la celda suele estar contenida en un conjunto de bases de datos Synopsys (.**db**).

Ficha técnica
Un archivo de tecnología proporciona información específica de la tecnología, como nombres y características físicas y eléctricas de cada capa de metal y reglas de diseño de enrutamiento. IC Compiler II utiliza la biblioteca de diseño Milkyway para acceder a la información tecnológica.

TluPlus
Los atributos parásitos definen los parásitos de la capa metálica. En general, IC Compiler II obtiene la información parasitaria de los archivos TLUPlus en lugar de hacerlo del archivo de tecnología.

Simplified ICC II Design Flow

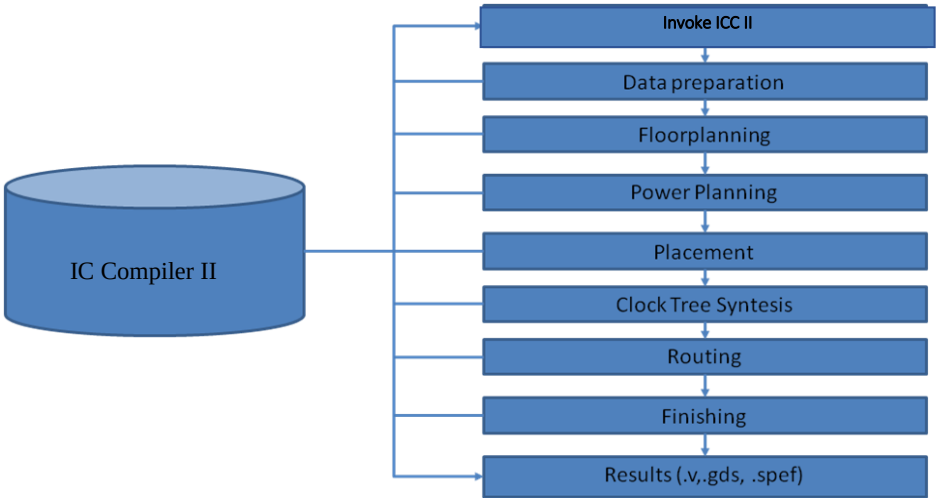


Figura 2. Flujo de diseño simplificado del compilador IC II

1. Inicie la interfaz gráfica de usuario (GUI) de IC Compiler II en el directorio de **Work**. Para iniciar la interfaz de línea de comandos de IC Compiler II, escriba el siguiente comando en el símbolo del sistema UNIX o Linux:

```
%icc2_shell -gui
```

Esto abre la ventana gui de nivel superior de IC Compiler II. (Figura 1)

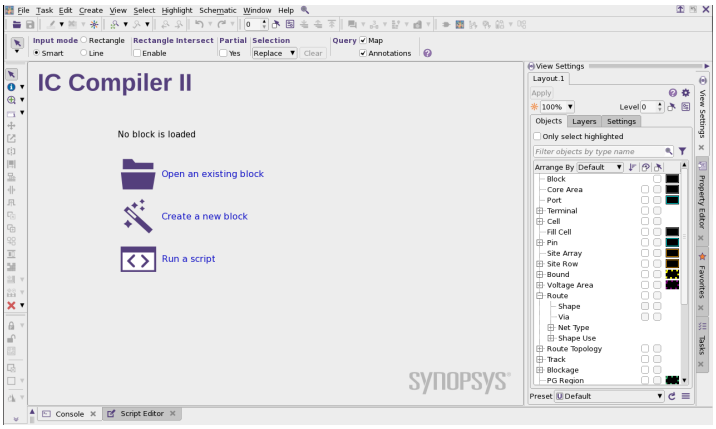


Fig.3. Ventana gui de nivel superior de IC Compiler II

2. Cuando se inicia IC Compiler II, lee automáticamente el archivo ".synopsys_icc.setup" del directorio actual. En este laboratorio, este archivo se encuentra en la carpeta **Work**.

Inicie IC Compiler II desde ese directorio para leer automáticamente el archivo y configurar las bibliotecas. Para comprobar si se han establecido las bibliotecas, abra **File> > Setup > Aplication Setup** (Fig. 4).

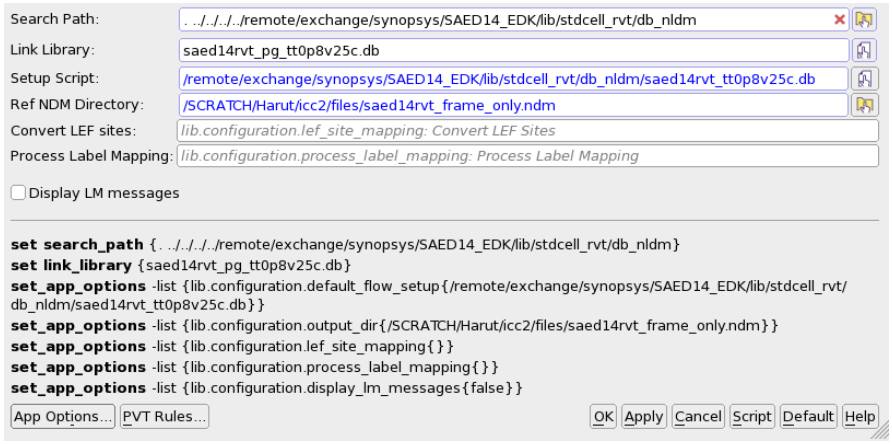


Figura 4. Ventana de configuración de la aplicación

Si no se establecen estos valores, salga e inicie IC Compiler II desde el directorio "work" o complete los archivos manualmente.

En IC Compiler II, especifique los archivos .db que se usarán para el diseño estableciendo **target_library** y **link_library** variables.
target_library variable especifica los archivos de biblioteca .db que contienen las celdas lógicas que se pueden usar para la optimización, por ejemplo, diferentes puertas NAND que tienen varias áreas, intensidades de unidad, retrasos y uso de energía.
link_library variable especifica las bibliotecas de .db que contienen todas las celdas lógicas que se pueden usar para resolver referencias jerárquicas en el diseño durante la ejecución del comando link.

Flujo de diseño

1. **Crear una nueva biblioteca de modelos de diseño**

Para crear la biblioteca de diseño NDM, elija **File > Create library**. Aparecerá el cuadro de diálogo Crear biblioteca. (Fig.5)

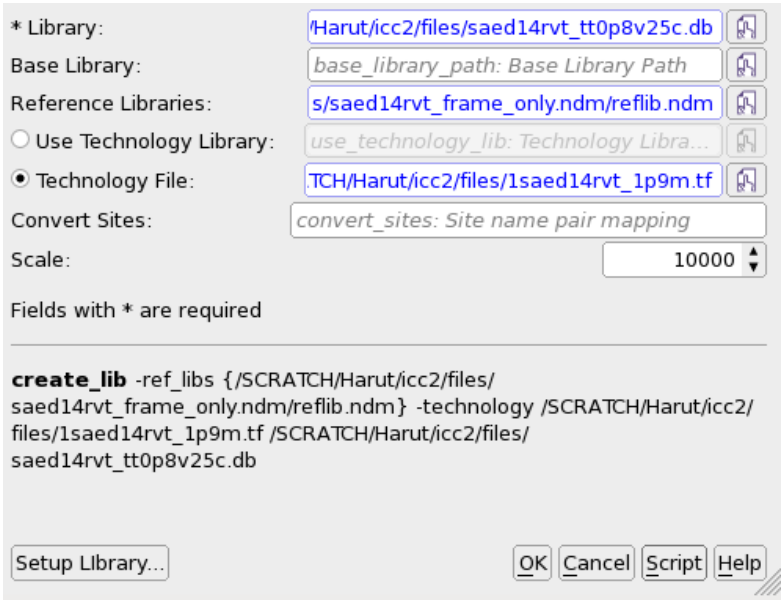


Figura 5. Ventana Crear biblioteca

El comando *create_mw_lib* y el cuadro de diálogo Crear biblioteca tienen opciones para especificar el nombre de la nueva biblioteca NDM, el nombre del archivo de tecnología asociado, los nombres de las bibliotecas de referencia NDM asociadas y el estilo de nomenclatura de bus.

2. **Establecer TLU+ archivos**

Establezca rutas de archivo TLUPlus. TLUPlus es una tabla binaria que almacena coeficientes RC. Los modelos TLUPlus permiten resultados precisos de extracción RC al incluir los efectos de ancho, espacio, densidad y temperatura en los coeficientes de resistencia. Configúrelo usando **View > Map > Rail parasitics** (Fig.6).

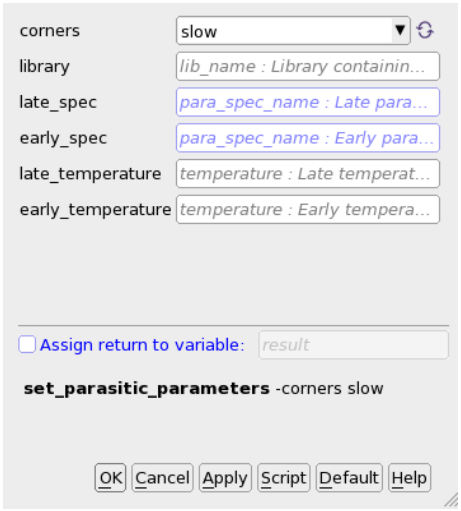


Figura 6. Establecer ventana TLU+

Escriba o busque el archivo de mapa en el cuadro de texto "Archivo de mapa". Escriba o busque los archivos de modelo TLUPlus máximos y mínimos.

3. Importar diseño

Lea el archivo Verilog para el diseño eligiendo **Task > Create block > Create Design Library** y especificando el archivo Verilog netlist, que es un diseño de nivel de puerta en uno o más archivos (Fig.7 y Fig.8).

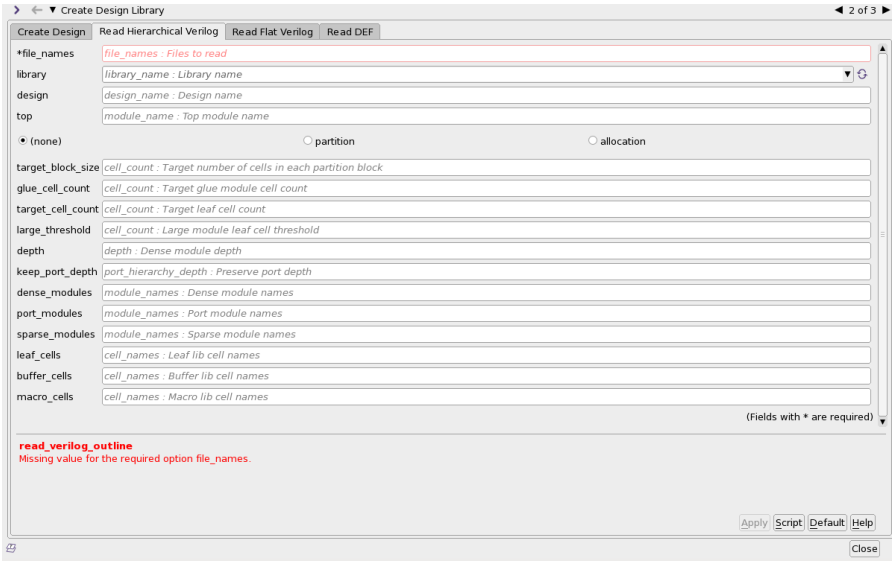


Figura 7. Ventana Importar diseño

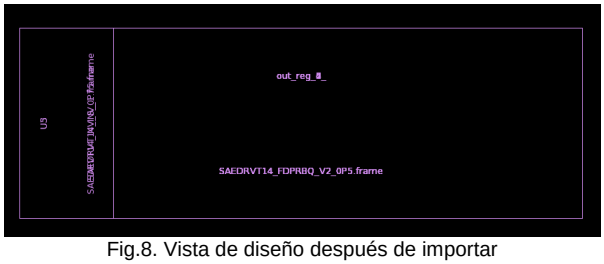
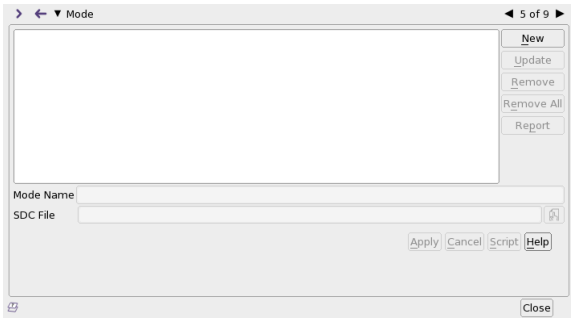


Fig.8. Vista de diseño después de importar

4. Restricciones de lectura

Para leer las restricciones, vaya a **Task > Timing > Mode.**



Se ha completado la instalación.

Después de configurar el diseño en IC Compiler II, siga estos pasos:

4.1. Inicializar el plano de planta,
4.2. Crear anillos rectangulares
4.3. Crear correas de alimentación, enrutar celdas estándar
4.4. Colocación
4.5.CTS (Síntesis del árbol del reloj)
4.6. Enrutamiento
4.7. Acabado.

5. Inicializar plano de planta

La información del plano de planta incluye el área central, los puertos de nivel superior y los sitios de colocación. Para planificar el diseño, elija Plano de planta>Inicializar plano de planta (Fig.8 y Fig.9).

Especifique los parámetros de control.

Indique el método que especifica el tamaño del área central:
1. Relación de aspecto: una relación de altura dividida por el ancho (el valor predeterminado)
2. Ancho y alto: el ancho y la altura exactos
3. Número de fila: varias filas
4. Límite: un tamaño fijo de acuerdo con el límite definido en la planificación del diseño.

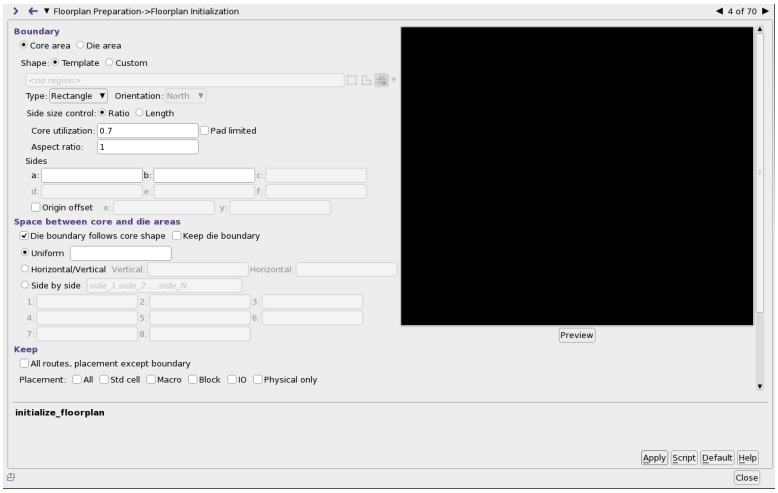


Fig.9. Inicializar ventana de plano de planta

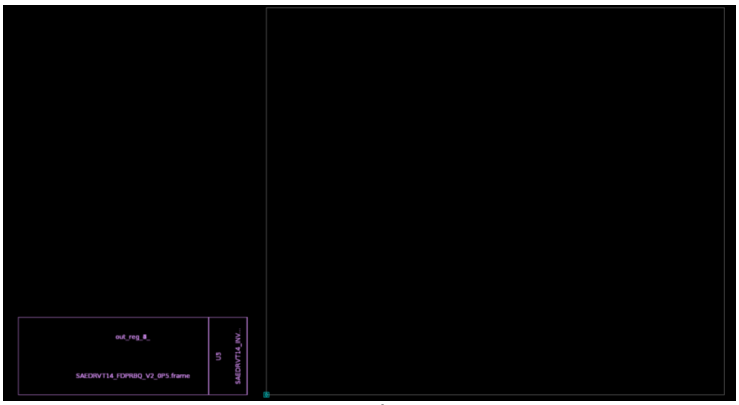


Fig.10. Vista de diseño después de initialize_floorplanning

6. Crear correas eléctricas

Después de la creación de anillos rectangulares, las correas se conectan automáticamente a la potencia más cercana y al anillo de tierra en, o más allá, de ambos extremos de las correas.

El comando *create_pg_strap* crea correas de alimentación en un diseño. Use algunas correas anchas en lugar de muchas correas delgadas para mejorar la calidad de la colocación y disminuir el tiempo de ejecución de la colocación.

Lo mismo que en la barra de menús, elija **Task > Design Planning > RIng/Hard Macro / Std cell Pattern** (Fig.13 y Fig.14).

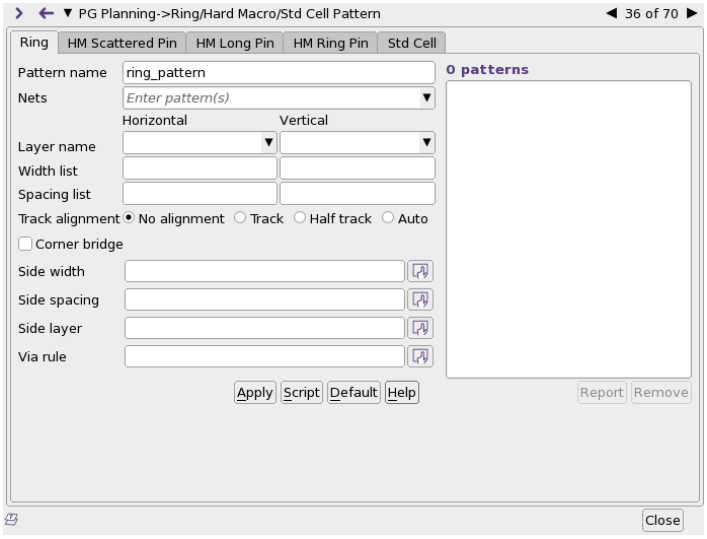


Fig.13. Ventana de herramienta crear objeto

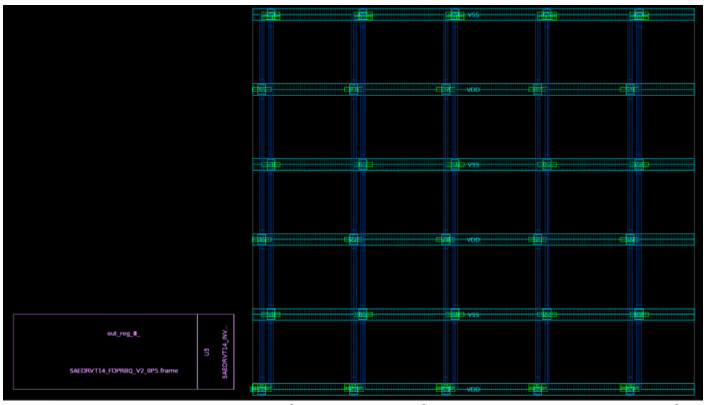


Fig.14. Vista de diseño después de la creación de correas de alimentación

7. Colocación

place_opt : el comando realiza la colocación, el enrutamiento y la optimización de la simulación en el diseño actual. Durante la fase de colocación, las celdas estándar de este diseño se han colocado automáticamente en filas de colocación horizontal.

Para ejecutar la colocación, elija **Placement > Core Placement and Optimization Task > Placement > Placement > Create Placement** (Fig.15 y Fig. 16).
Aparecerá el cuadro de diálogo Ubicación principal y optimización. Seleccione "Optimización de energía".

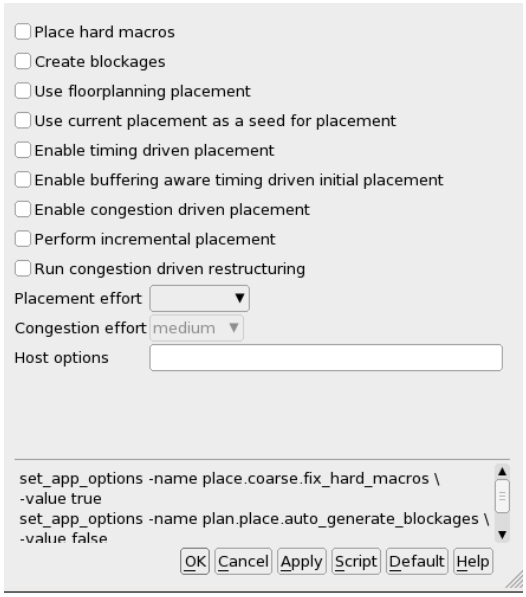


Fig.15. Ventana de colocación y optimización del núcleo

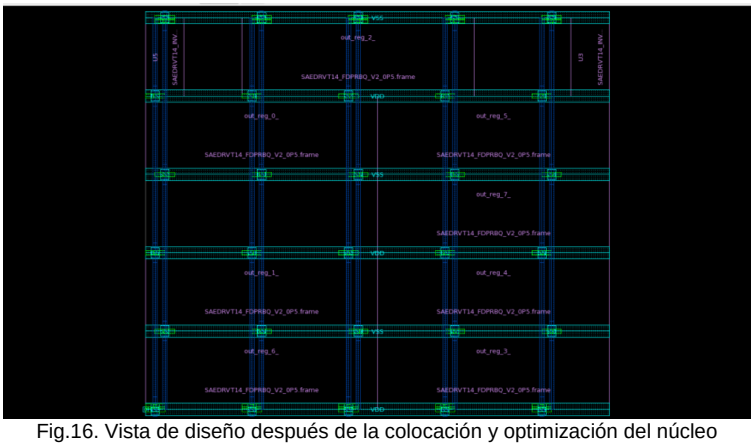


Fig.16. Vista de diseño después de la colocación y optimización del núcleo

8. Síntesis del árbol del reloj

Durante la síntesis del árbol de reloj, IC Compiler II construye árboles de reloj que cumplen con las restricciones de la regla de diseño del árbol de reloj mientras equilibra las cargas y minimiza el sesgo del reloj. IC Compiler II corrige la colocación de los receptores de reloj, realiza la lógica incremental y la optimización de la ubicación, y corrige la ubicación de los búferes y registros en el árbol de reloj. El comando *clock_opt* realiza la síntesis del árbol de reloj, el enrutamiento de las redes de reloj, la extracción, la optimización y, opcionalmente, la fijación de infracciones de tiempo de retención en el diseño actual. Si se produce un error en la síntesis del árbol de reloj o se produce un error en el enrutamiento de las redes de reloj, el comando devuelve con un valor de 0.

Para realizar la síntesis y optimización del árbol de reloj, utilice el comando *clock_opt* o elija CTS > Core CTS y Optimization en la GUI. (Fig.17 y Fig.18). Aparece el cuadro de diálogo CtS principal y optimización y seleccione celdas de reloj de ruta. Para realizar la síntesis del árbol del reloj, elija **Task > Clock tree > Chack Clock Trees**

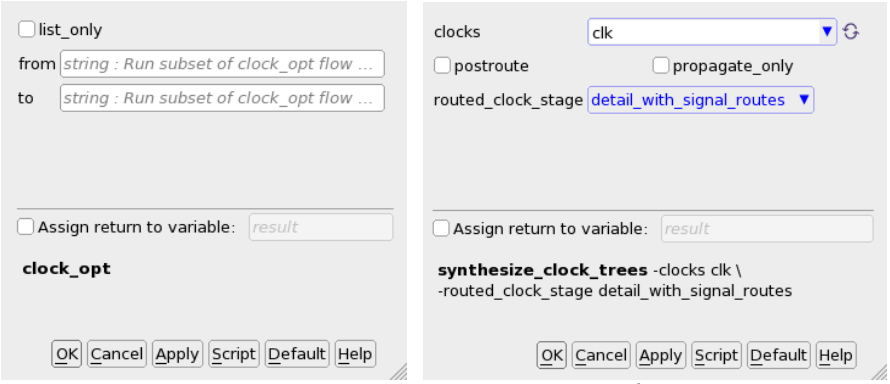


Fig.17. CTS central y ventana de optimización

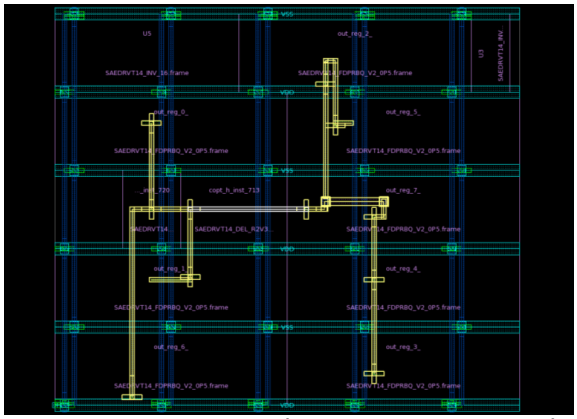


Fig.18. Vista de diseño después de Core CTS y Optimización

9. Celdas estándar previas a la ruta

Conecte los pines de alimentación y de tierra de las celdas estándar a las correas y anillos y conecte los rieles de alimentación y de tierra en las celdas estándar. Para asegurarse de que el enrutador global puede reconocer la obstrucción del enrutamiento, enrute las celdas estándar antes de realizar el enrutamiento global.

El comando *route_auto* conecta los pines de alimentación y tierra de las celdas estándar a los anillos o correas de alimentación y tierra, y conecta los rieles de alimentación y tierra en las celdas estándar.

Lo mismo que en la barra de menús, elija **Task > Routing > Create Routing Blockage** (Fig. 19 y Fig.20).

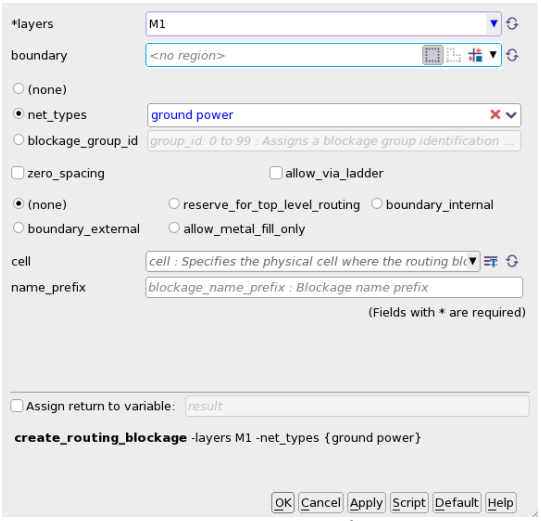


Fig.19. Ventana de celdas estándar de ruta

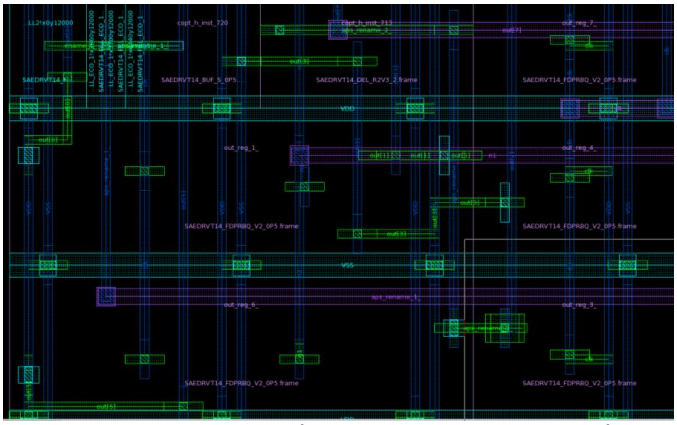


Fig.20. Vista de diseño después de Enrutamiento de celdas estándar

10. Enrutamiento

El *route_opt* este comando realiza enrutamiento simultáneo y optimización postura en el diseño actual. El resultado de este comando es un diseño optimizado postura.
Finalmente, para enrutar el diseño, elija **Route >Core Routing and Optimization**. Aparece el cuadro de diálogo enrutamiento y optimización principal (Fig.21 y Fig. 22).

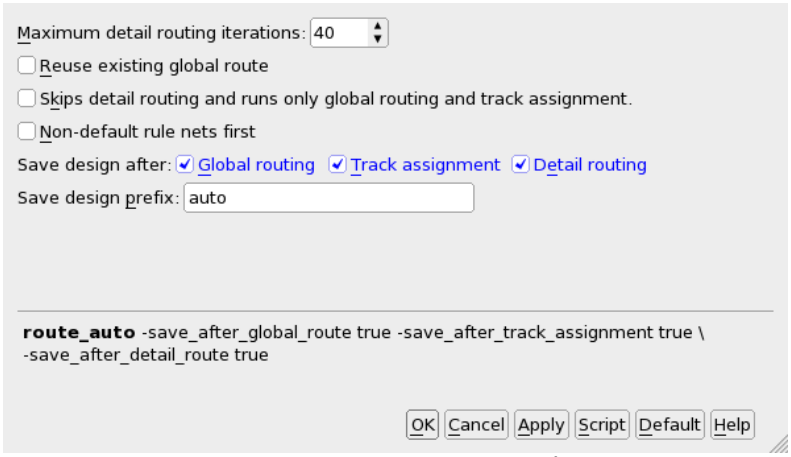


Fig.21. Ventana de enrutamiento y optimización principal

La CCI *tiene route_eco* mando. Este comando realiza el enrutamiento de las redes rotas. Ejecuta la ruta global, la asignación de pistas y la ruta de detalles.

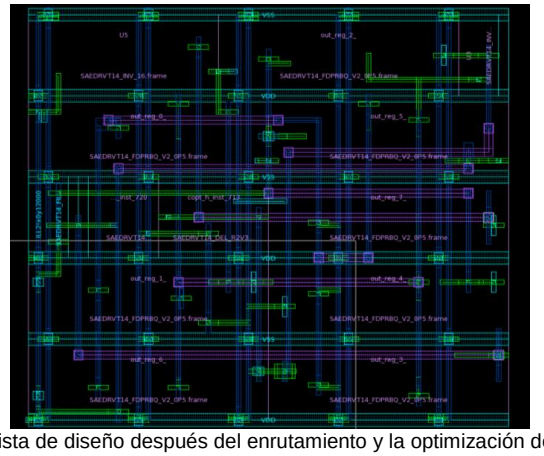


Fig.22. Vista de diseño después del enrutamiento y la optimización del núcleo

11. Acabado

create_stdcell_filler este comando rellena los espacios vacíos en las filas de celdas estándar con instancias de celdas de relleno maestras en la biblioteca (Fig.23).

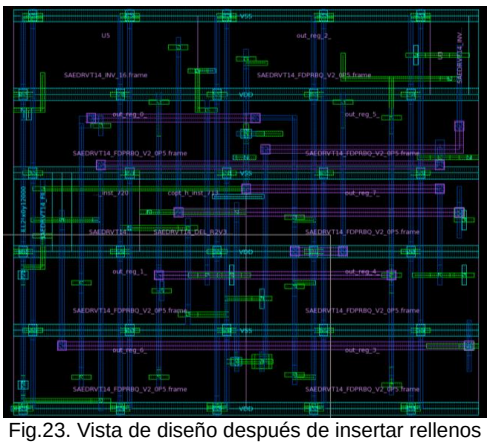


Fig.23. Vista de diseño después de insertar rellenos

1. Comprobación de reglas de diseño

Para comprobar DRC en IC Compiler II se utilizan herramientas externas como IC Validator.

Para hacer esto, se deben establecer las primeras opciones de firma física. Para ello, vaya a **Signoff > Set Physical Signoff** menú, se abrirá la ventana de configuración (Fig. 24). Debe configurarse con el nombre de la herramienta, el conjunto de tiradas y el archivo de mapa. Para comprobar DRC, elija **Verification > Signoff DRC.**

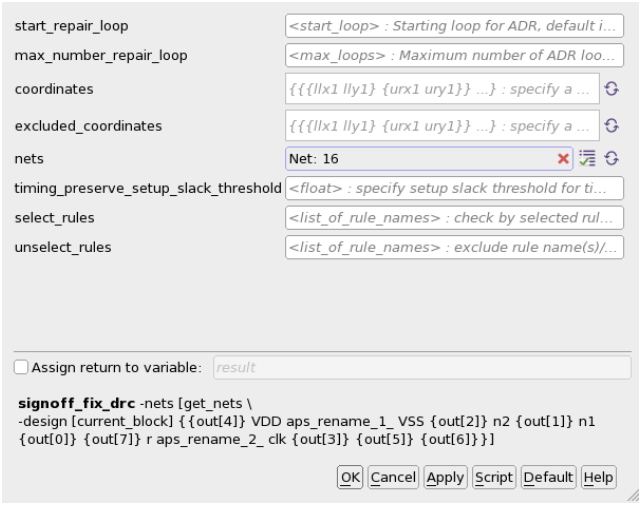


Figura 24. Ventana de configuración de DRC de firma

2. Comprobación de diseño frente a esquema (LVS)

Para verificar los errores de LVS, elija **Verification > LVS** (Fig.25).

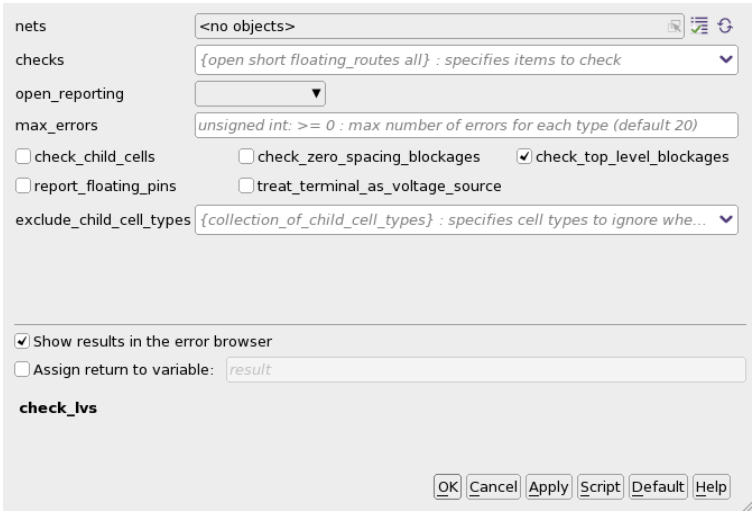


Fig.25. Compruebe la ventana LVS

Otras características.

IC Compiler II puede diferentes datos en forma de mapas superpuestos. Por ejemplo, elija **View > Map mode,** a continuación, en la barra lateral abierta, seleccione **Cell density**. A continuación, haga clic en **Reload** (Fig.26).

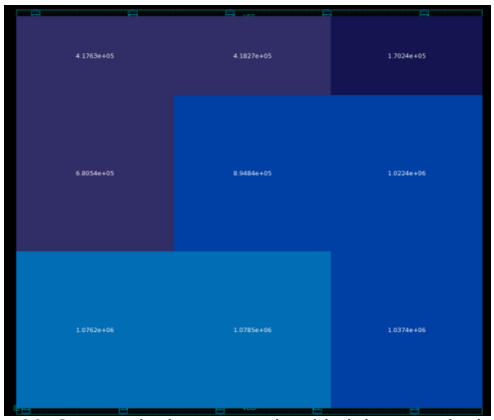


Figura 26. Compruebe la ventana densidad de potencia de fuga

3. Redacción de resultados

Escriba el diseño en Verilog a nivel de puerta: **Task > Design Planning > Write Floorplan & Verilog > Verilog** (Fig. 27).

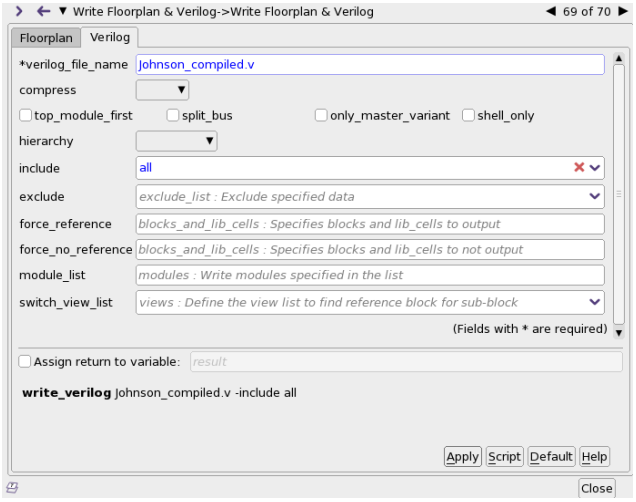


Fig.27. Escribir cuadro de formato verilog

4. Parásitos de exportación

Extraiga parásitos usando **file > Export > Write Parasitics**, en el cuadro de diálogo abierto presione OK. Para escribir el diseño con el formato de intercambio parásito estándar (SPEF) desde la barra de menús, elija Archivo > Exportar > Escribir parásitos (Fig. 28).

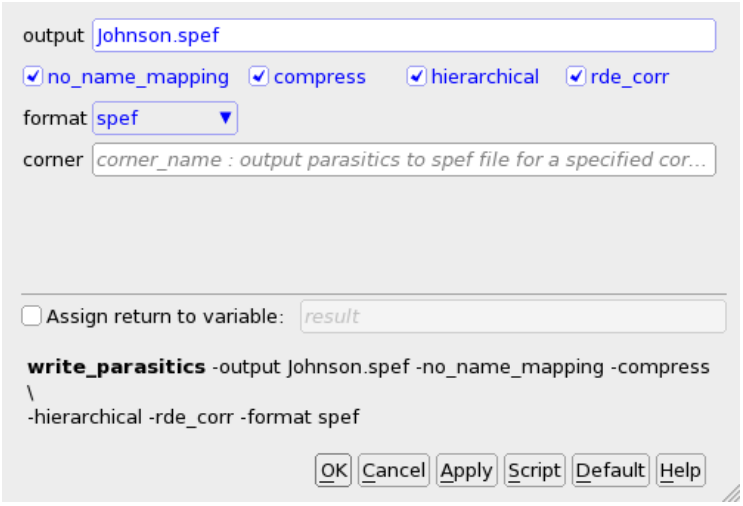


Fig.28. Escribir cuadro de formato .spef

4.1 Flujo de exportación (GDSII)

Para escribir los datos de la biblioteca especificada en un archivo en formato GDS, elija **File > Export > Write Stream** (Fig. 26).

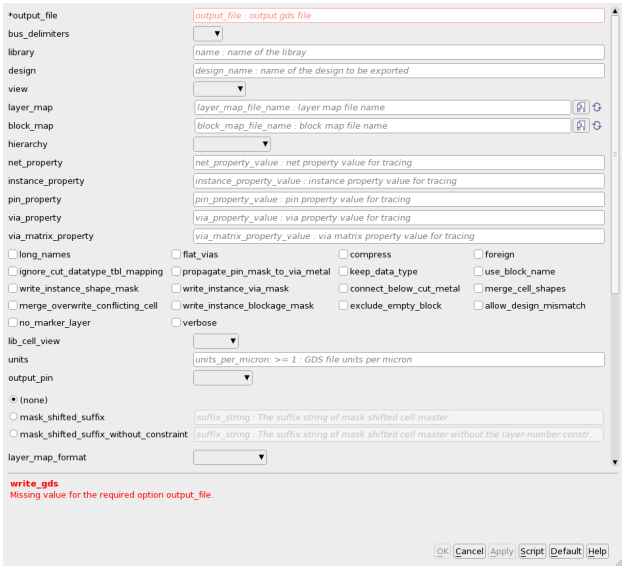


Fig.29. Cuadro de formato Write .gds

En este diseño tendrá los resultados de salida almacenados en él directorio ../**results**

Para salir de ICC II, seleccione **File > Close design**, luego **File > Close library** y Luego **File > Exit**.

Utilice el script creado para llevar a cabo todas estas operaciones mediante comandos. Hay un script **flow.tcl** en el directorio **scripts**. Para obtener el script, escriba en la ventana ICC recién abierta. Explorar el script para aprender comandos

```
Icc2_shell> source ../scripts/flow.tcl
```

Informe

El informe debe incluir:

1. Nivel de la puerta Verilog
2. Corriente
3. Archivo SPEF
4. Capturas de pantalla de diseños colocados y enrutados, así como resultados de DRC / LVS.

16.5. Laboratorio 4: Análisis de tiempo estático. PrimeTime

Synopsys Design Flow Tutorial

Laboratorio 4: Análisis de tiempo estático. Prime Time

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 parasitics netlist 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 **work**, 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)

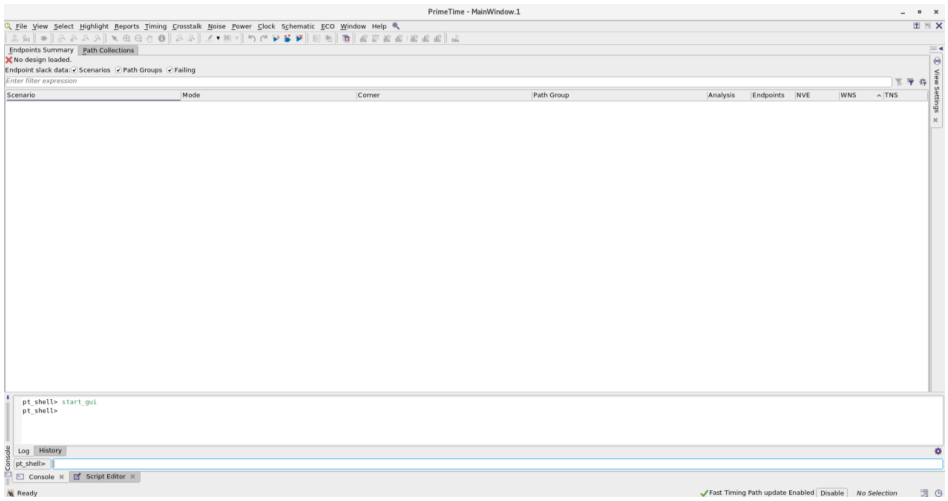


Fig.1. Ventana de GUI de nivel superior de PrimeTime

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

```
set link_path  [list ../ref/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 **post_lay** directorio.

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 la sincronización de retención, primero escriba **reset_design** comando 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 lista de diseño netlist. (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. 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.

```
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** comando. Especifica que un puerto o pin está en un valor lógico constante 1 o 0, o se considera con un tránsito ascendente o descendente. Los puertos de este laboratorio no están activos con el valor 0, por lo que el puerto r y 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   : Thu Aug 23 07:20:35 2022
****************************************
  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
```

Dando click + Ctrl, podremos seleccionar las salidas que deseemos ver sus tiempos.

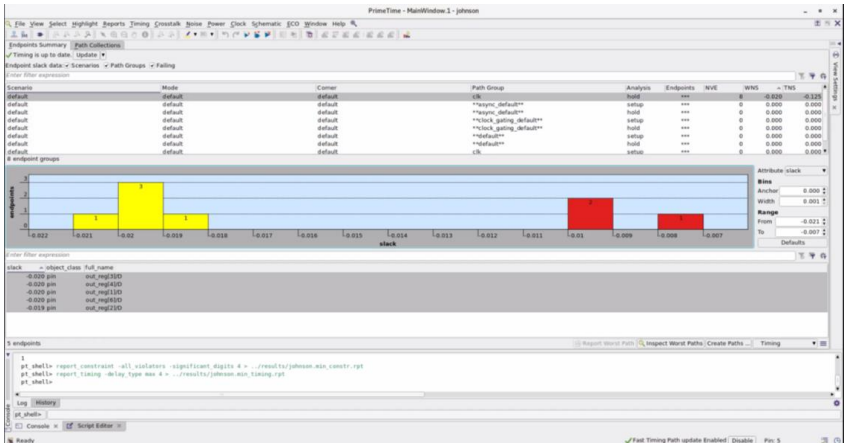


Figura 2. Salidas preferidas

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 las siguientes figuras:

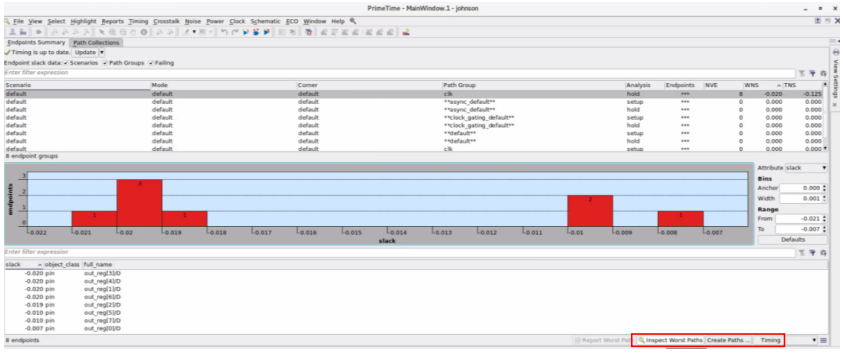


Figura 3. Inspección de peor camino, tiempo y crear paths

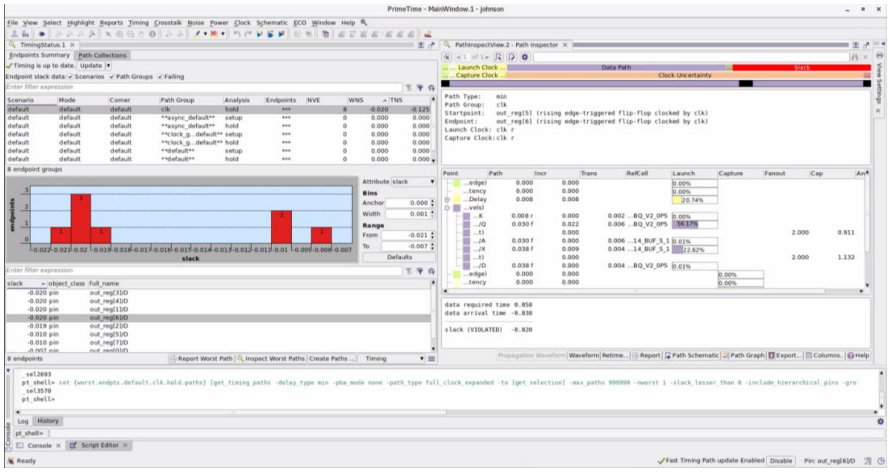


Figura 4. El controlador de análisis de temporización

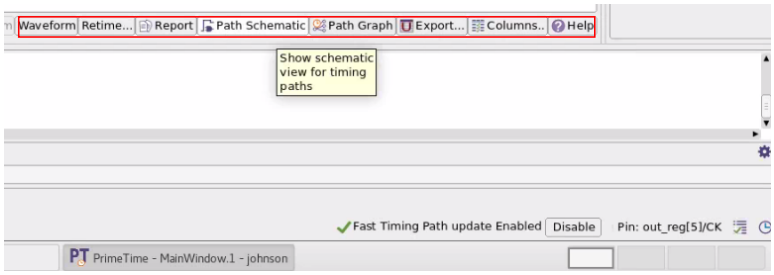


Figura 5. Opción de esquemático

Adicional tenemos más opciones como exportar, gráfica, ayuda etc.

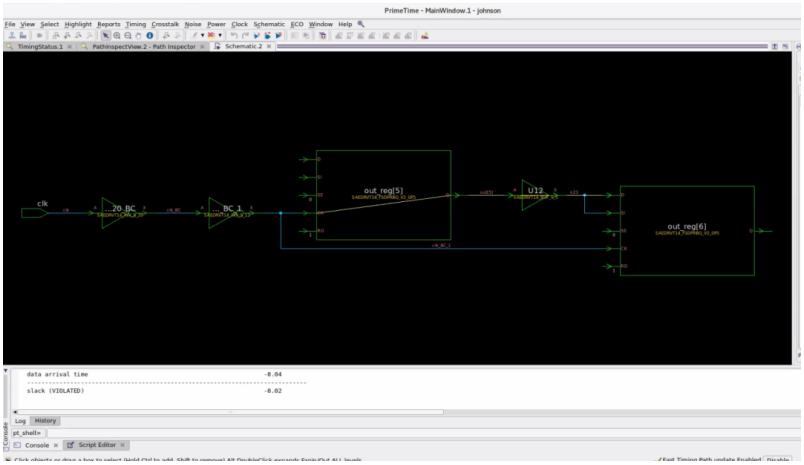


Figura 6. Consola del Inspector de rutas

3.7 El archivo output 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 ../**results** /directorio.

4. Para salir de PrimeTime escribe:

```
pt_shell> exit
```


16.6. Laboratorio 5: Verificación formal. Formality

Synopsys Design Flow Tutorial

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 Formality. Desde aquí, inicie la interfaz gráfica de usuario (GUI) de la siguiente manera:

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

Cuando se invoque a Formality, comience en el modo **setup**. El **setup** indica el modo en el que se encuentra actualmente cuando se utilizan comandos. Los modos son **setup**, **math** y **verify**.

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

Tareas de laboratorio

1. **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 Work **pre_lay**, y si a Formality se le da el resultado de ICC, entonces trabaje en el directorio **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 post_lay directorio. Para iniciarlo, introduzca:

```
% fm_shell -gui
```

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

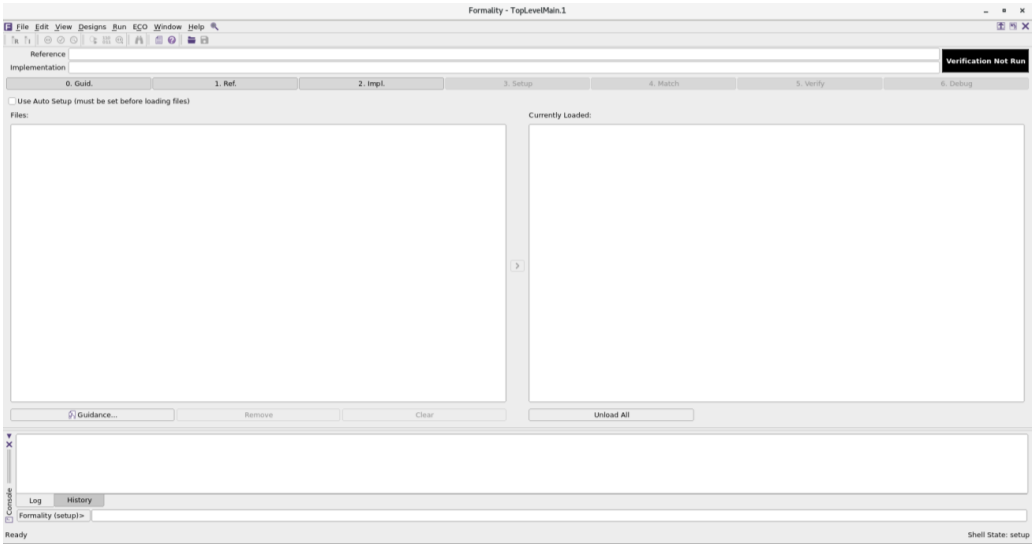


Fig.1. La ventana principal de la GUI de Formality

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

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

Ahora entienda las secciones generales de Formality.

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

*Nota: En la imagen No.1 se logra apreciar debajo de la barra de implementación varias secciones, se debe encontrar en la de referencia.

Especificar el diseño de referencia localizarse en 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_dft.v. Haga clic en Abrir y cargar este archivo (Fig. 2).

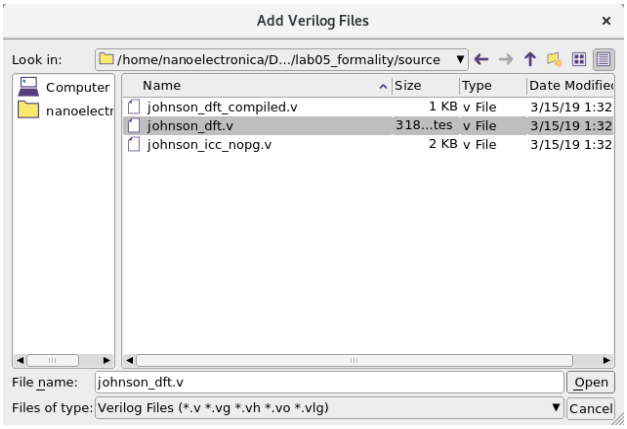


Fig.2. Configurar la fuente de nivel de compuerta de CC

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

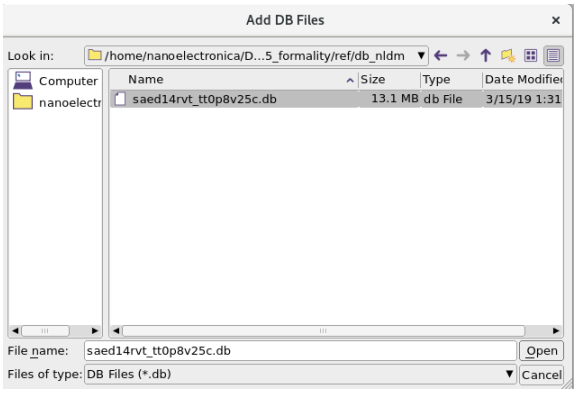


Fig.3. Setup.db file.

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

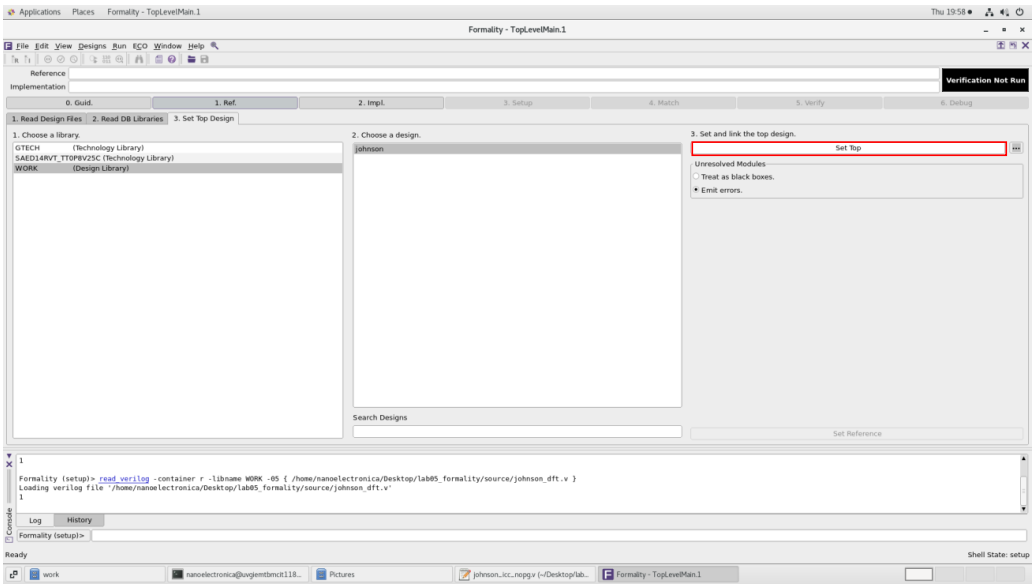


Fig.4. Set Top Referencia

Luego se podrá observar en la Fig.5, que en la opción de “1.Ref.” aparecerá un check verde indicado que todo está bien.

Fig.5. Verificar check verde.

3. 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 puerta después de que Design Compiler nombró a este archivo johnson_icc_nopg.v. Haga clic en Abrir y cargar este archivo (Fig. 5).

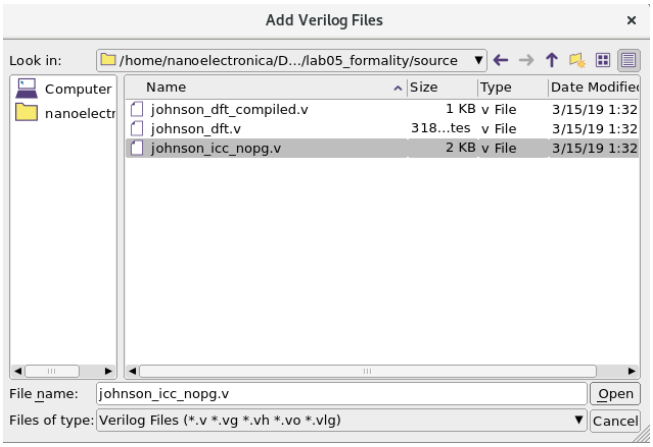


Fig.6. Configurar el nivel de la puerta ICC Verilog

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

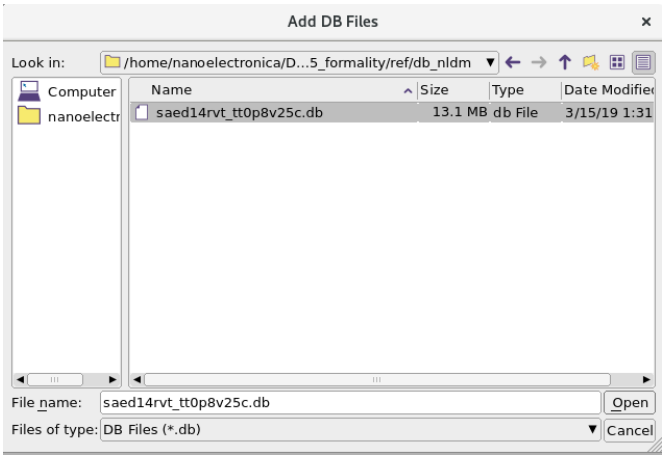


Fig.7. Establecer archivo de .db de puerta

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

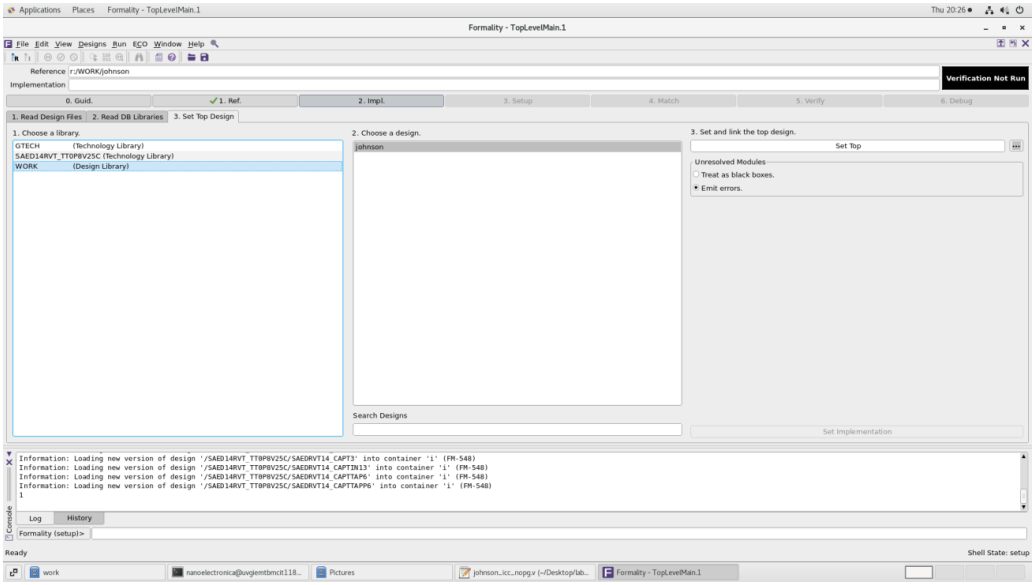


Fig.8. Set Top Implementación

Luego se podrá observar en la Fig.9, que en la opción de “2.Imp.” aparecerá un check verde indicado que todo está bien.

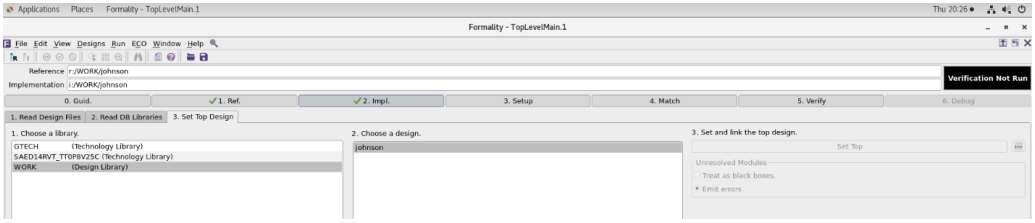


Fig.9. Verificar check verde.

4. Configurar (Configurar el diseño)

Para configurar el diseño, haga clic en 3.Setup (Fig.10 y Fig.11).

Presionamos la opción de Set, pegado a su mano derecha, luego seleccionar las opciones que se muestra en la siguiente figura.

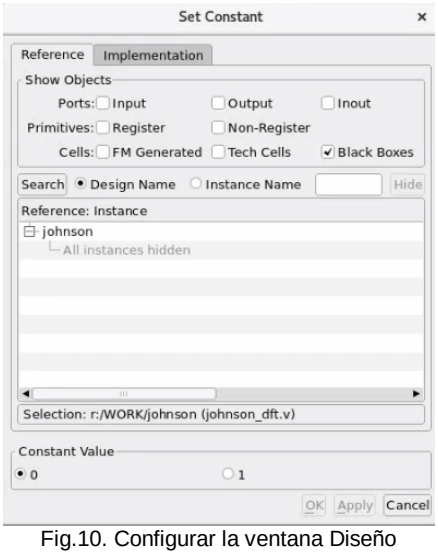


Fig.10. Configurar la ventana Diseño

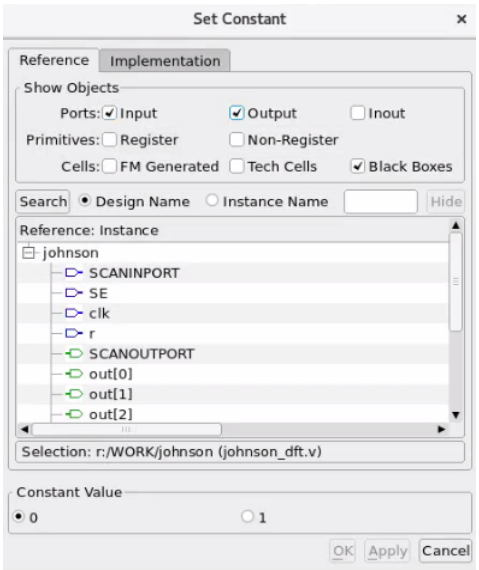


Fig.11. Configurar la ventana Diseño de Referencia

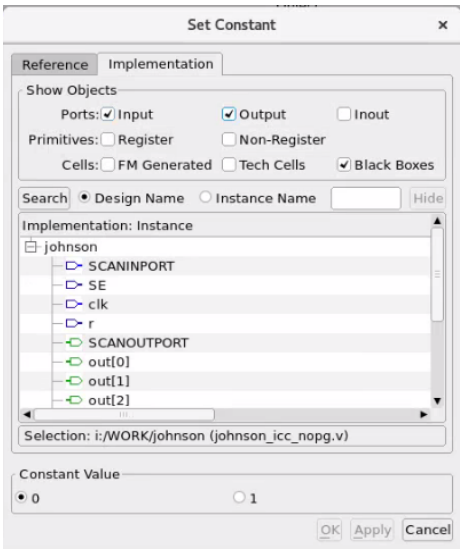


Fig.12. Configurar la ventana Diseño de Implementación

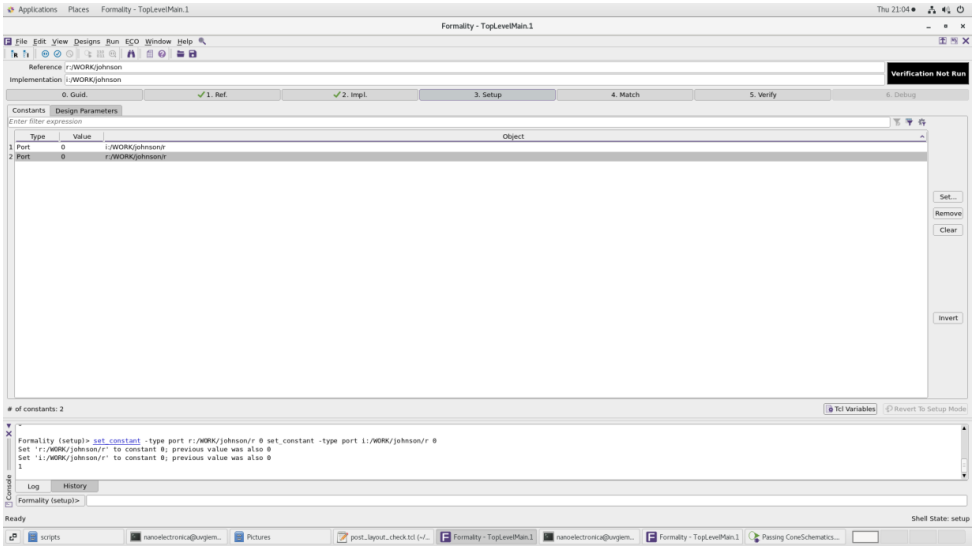


Fig.13. Después de configura el diseño

5. 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. Match por match compara puntos.

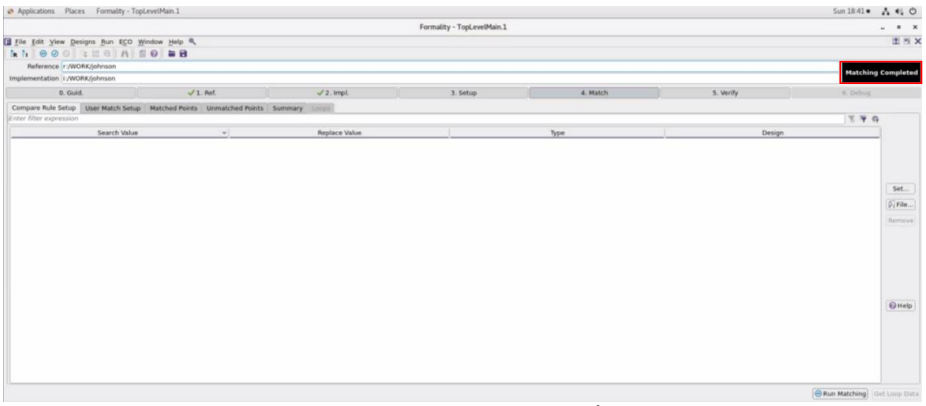


Fig.14. Puntos de comparación

Como se ve en el masaje que se muestra en la Fig.14, hay 0 puntos de comparación inigualables.

6. 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.

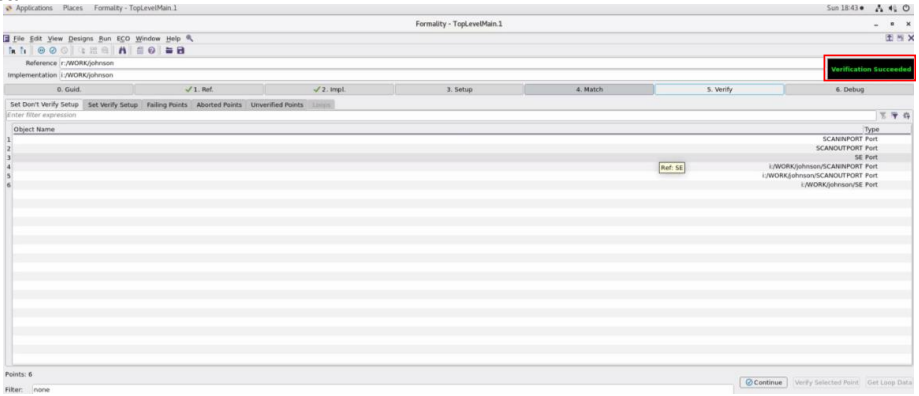


Fig.15. Verificar el diseño

Verificación del proceso completado correctamente.

7. Debug

Durante la depuración, se deben encontrar los puntos exactos en los diseños que exhiben la diferencia en la funcionalidad y luego corregirlos (Fig. 16).

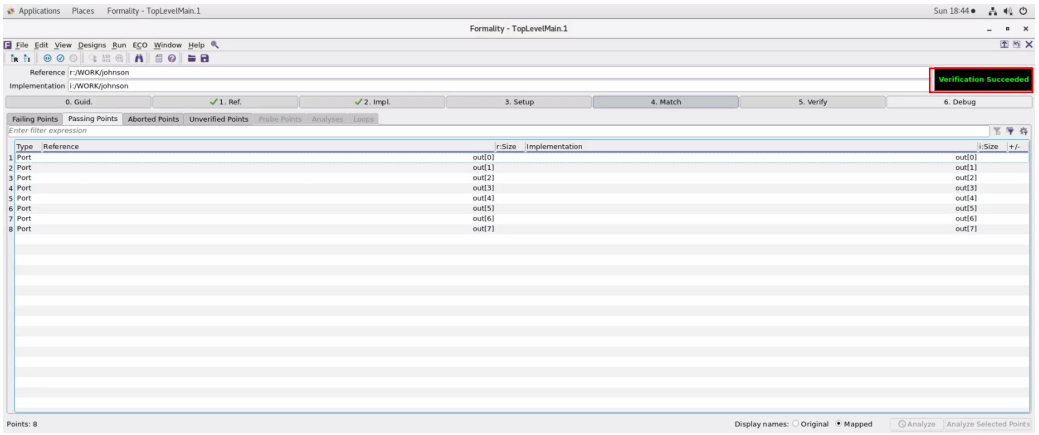


Fig.16. Debugear el diseño

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

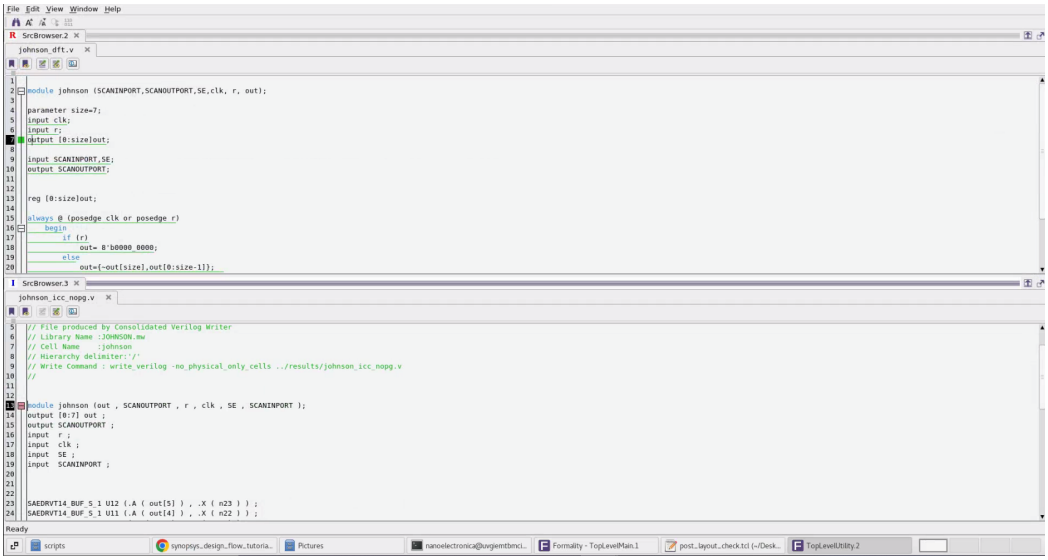


Fig.17. (a). Implementación Verilog y (b). Referencia Verilog

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

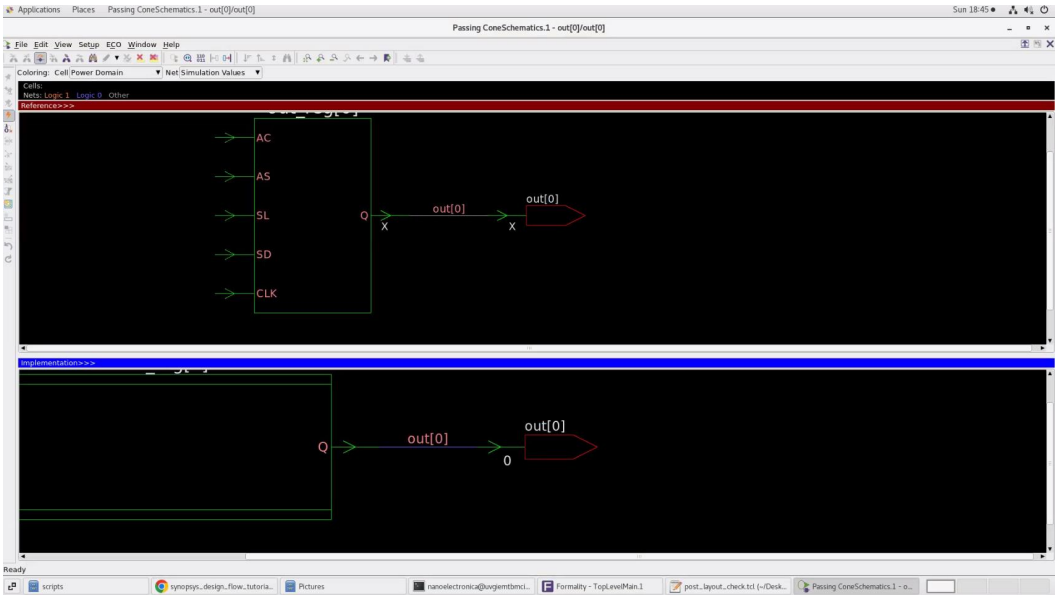


Fig.18. Ver objeto de referencia en esquema

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

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

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

16.7. Laboratorio 6: Generación automatizada de patrones de prueba. TetraMAX

Synopsys Design Flow Tutorial

Laboratorio 6: Generación automatizada de patrones de prueba. TetraMAX

Objetivo

Aprenda a generar patrones que maximicen la cobertura de las pruebas mientras usa un número mínimo de vectores de prueba para una amplia variedad de tipos de diseño y flujos de diseño.

Introducción

TetraMAX es una herramienta de generación automática de patrones de prueba (ATPG) de alta velocidad y alta capacidad.

Tareas de laboratorio

Si TetraMAX recibe los resultados del compilador de diseño, trabaje en el directorio **pre_lay**. Si TetraMAX recibe los resultados de IC Compiler, trabaje en el directorio **post_lay**. Este laboratorio se basa en los resultados de TetraMAX de IC Compiler. Los pasos para ejecutar TetraMAX con los resultados de Design Compiler son similares.

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

```
% tmax –tcl
```

Esto abre la ventana de gui de nivel superior de TetraMAX. (Figura 1)

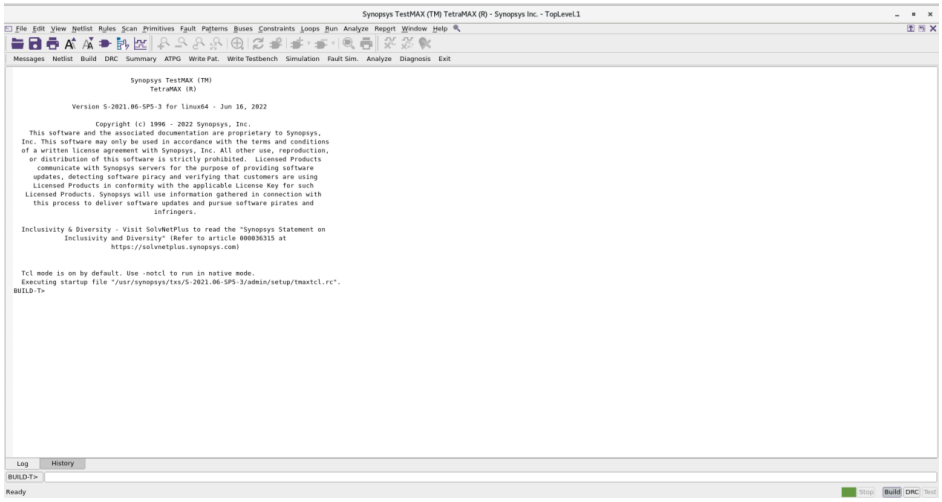


Fig.1. Ventana de GUI de nivel superior de TetraMAX

2. Lea Archivos de biblioteca y diseño. Utilice el comando **read_netlist** para leer la biblioteca y la lista de redes (Fig. 2).

```
BULD-T> read_netlist ../ref/lib/stdcell_rvt/verilog/saed14nm_rvt.tv
BULD-T> read_netlist ../source/johnson_dft_compiled.v
```

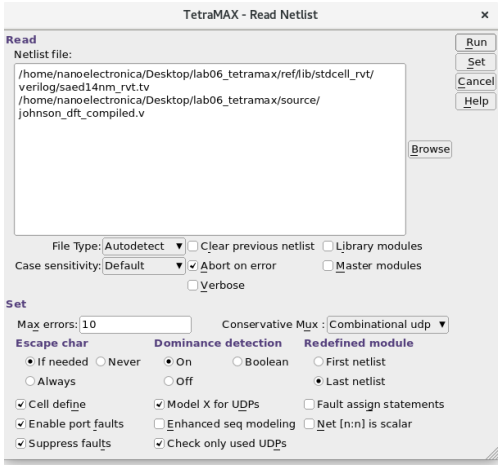


Fig.2. Consola de lectura de netlist

3. Diseño de construcción (Fig. 3). Este comando crea el modelo de simulación en memoria a partir de los módulos de diseño que se han leído. Una vez finalizado el proceso de compilación, TetraMAX realiza un proceso de aprendizaje. Si ya existe un modelo de simulación, se elimina automáticamente antes de crear el nuevo modelo.

```
BULD-T> run_build_model johnson
```

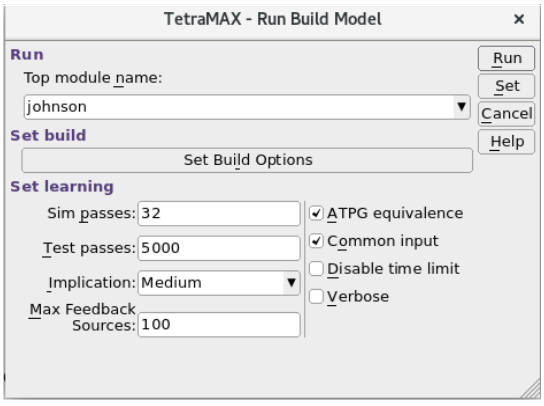
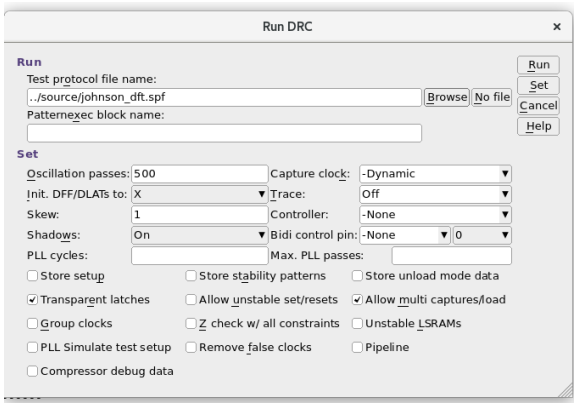


Fig.3. Ejecución de la consola del modelo de compilación

4. Ejecute drc con el archivo .spf, que se generó a partir de DFT. (Fig. 4)

```
DRC-T> run_drc ../source/johnson_dft.spf -test
```

Utilice el comando **run_drc** para realizar la comprobación de reglas de diseño, que es necesaria para entrar en el modo de comando TEST (**-test** (el predeterminado) donde se puede realizar la generación de pruebas y la simulación de errores.



1. Figura 4. TetraMAX DRC en ejecución

5. Se emite un mensaje cuando se ejecuta el comando **add_faults**, que indica cuántas fallas nuevas se agregaron a la lista.

```
TEST-T> add_faults -all
```

- all agregan fallas en todos los sitios de fallas potenciales en el diseño a la lista de fallas, excepto las fallas con un atributo no fault. Para los errores de transición, el comando **add faults -all** no agrega ningún error en el reloj ni la lógica de habilitación de escaneo de forma predeterminada. Esto se debe a que no es significativo probar fallas en dicha lógica a velocidades funcionales con patrones basados en escaneo.

6. Utilice el siguiente comando para establecer los parámetros que controlan el proceso ATPG. Esto también afecta a todos los análisis que utilizan la generación de pruebas (incluida la RDC y la justificación).

```
TEST-T> set_atpg -capture 10
```

-Capture Establece el nivel de esfuerzo del algoritmo ATPG de secuenciación rápida. Los valores aceptables son enteros entre 2 y 10, o 0. Un valor de 0 deshabilita el esfuerzo del patrón ATPG de secuenciación rápida y da como resultado patrones ATPG de escaneo básico (combinado), que es el valor predeterminado. Un valor de 2 o superior permite el esfuerzo de patrón ATPG de secuenciación rápida.

7. Utilice el siguiente comando para ejecutar ATPG automático (Fig.5):

```
TEST-T> run_atpg -auto
```


La opción **-auto_compression** del comando **run_atpg** puede ser útil para ayudar con la generación óptima de conjuntos compactos de patrones ATPG. Está diseñado para seleccionar automáticamente las mejores configuraciones y algoritmos para proporcionar resultados razonablemente buenos.

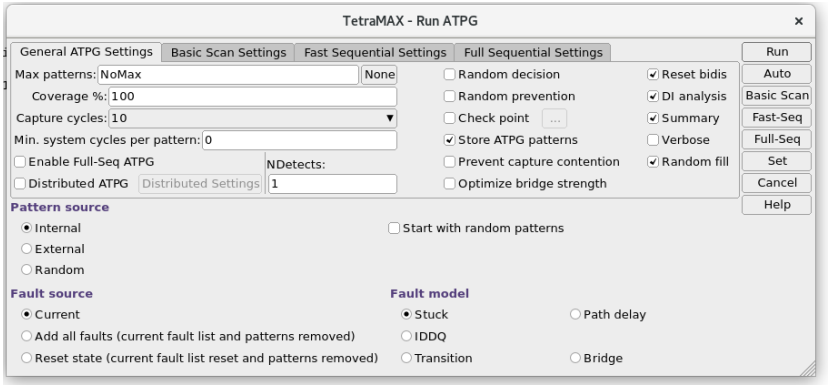


Figura 5. Ejecución de ATPG

8. Informe el resumen de ATPG.

```
TEST-T> report_summaries
```

9. Este comando ayuda a determinar por qué no se detectan los errores.

```
TEST-T> analyze_faults -class au
```

Cuando se usa en una o más clases de fallas, proporciona un resumen de cuántas veces se encontraron ciertos problemas de detección potenciales, como "conectado a un TLA" (pestillo transparente) o "restringido", y así sucesivamente.

Cuando se utiliza en un error seleccionado identificado por pin_path nombre, el comando realiza un análisis combinado detallado para ver si se puede detectar el error.

-Class au (ATPG incomprobable) selecciona una clase específica de fallas para su análisis.

10. Utilice el siguiente comando para informar de los datos de patrones del búfer de patrones interno o externo.

```
TEST-T> report_patterns -summary
```

-summary selecciona un informe de resumen para el conjunto de patrones internos.

11. Especifica el nombre de la ruta de acceso base del archivo en el que TetraMAX escribe los patrones. Tenga en cuenta que el **nombre de archivo** debe ser el primer argumento especificado en la línea **de comandos de patrones** de escritura.

```
TEST-T> write_patterns ../results/atpg_parallel.stil -format stil -parallel -
replace
```

```
TEST-T> write_patterns ../results/atpg_serial.stil -format stil  -serial -
replace
```

-format Verilog_tables opción especifica el formato de salida para los patrones que escribe TetraMAX. El valor predeterminado es binario.

Se admiten dos formas de Verilog: una table_format de datos multiarchivo (especificada por la opción **verilog_tables**) y un formato single_all en uno (especificado por la opción **verilog_single_file**). Ambas formas de Verilog admiten la selección de la aplicación en serie o en paralelo del desplazamiento de escaneo según lo especificado por las opciones **-serial / -parallel**. El valor predeterminado es la aplicación paralela de scan shift.

El comando **write_patterns -format stil** incluye un conjunto de opciones que generan un script de shell o un archivo de comandos que contiene los comandos para invocar el simulador respectivo.
Los resultados de salida del diseño se almacenan en él directorio ../**ubicación/resultados/**.

12. Para salir de TetraMAX, escriba exit en la línea de comandos.

```
TEST-T> exit
```

Para ver el test_bench de TetraMAX en DVE ingrese **run_sim** archivo que se encuentra en el directorio **de trabajo**.

Utilice el script creado para llevar a cabo comandos complejos. Los scripts se encuentran en el directorio **de scripts**, dentro del directorio **post_lay**. El nombre de este script es: **atpg_pre.tcl.** Para obtener el script, escriba:

```
BULD-T> source ../scripts/atpg_pre.tcl
```


16.8. Laboratorio 7: Verificación física. IC Validator

Synopsys Design Flow Tutorial

Laboratorio 7. Verificación física. IC Validator

Objetivo

Aprenda a verificarlas reglas de diseño (DRC) y el diseño frente al esquema (LVS) en el diseño de circuitos integrados (IC) utilizando la herramienta de verificación física jerárquica ICV.

Introducción

La herramienta de verificación física jerárquica de ICV realiza la comprobación de reglas de diseño (DRC) y el diseño frente al esquema (LVS) en el diseño de circuitos integrados (IC).

Tareas de laboratorio

1. Para ejecutar ICV DRC, busque el directorio **work**, que se encuentra en el directorio **drc**. En el directorio de work, el archivo **RUND_DRC** es saed14nm_1p9m_drc_rules.rs**.** En este archivo hay direcciones actuales con **johnson.gds,** output resultados y escribir Verilog actual Top nombre del módulo, en este trabajo de laboratorio es **johnson.**

Cambiar el archivo **saed14nm_1p9m_drc_rules.rs** se muestra a continuación:

```
#ifndef m_LIBRARY_PATH
#define m_LIBRARY_PATH
"/remote/exchange/synopsys/SAED14_PDK/icv/"
#endif


library(
        library_name  = ../source/Johnson.gds,
        format = GDSII,
        cell = "Johnson"

);

FINi    =       assign({{1}});
FINCUTi         =       assign({{2}});
NWELLi          =       assign({{3}});
DNWi            =       assign({{4}});
DIFFi   =       assign({{5}});
DDMYi           =       assign({{5,1}});
PIMPi   =       assign({{6}});
NIMPi   =       assign({{7}});
DIFF_15i        =       assign({{8}});
```

escriba el siguiente comando para compilar la librería:

```
icv ../ref/tech/icv_drc/saed14nm_1p9m_drc_rules.rs.
```

2. En la carpeta de **work** escriba los siguientes comandos:

```
–i ../source/johnson.gds
```


```
–c johnson -vue
```

3. Revise y depure los errores generados en la ejecución de ICV DRC utilizando archivos de error ICV. Vea los errores de la RDC en **Johnson. LAYOUT_ERRORS** archivo que se encuentra en el directorio de **resultados**.

Ejecute ICV para realizar la generación de patrones de relleno, incluidas las comprobaciones básicas de DRC en las capas que se llenan. Hay algunos errores que deben corregirse.
En este diseño, los resultados de DRC se almacenan en el directorio .../**location/results/**.

4. Si la netlist esquemática está en formato Verilog, NetTran la traducirá al formato ICV utilizando los siguientes comandos:

```
icv_nettran  -verilog rtl_dc_compiled.v
             -sp saed14nm_rvt.cdl
             -verilog-b1 VDD -verilog-b0 VSS
             -outName Johnson_icc.cdl  -outType SPICE
             -dupCell USE_MULTIPLE
             -sp-dupPort WARNING
             -sp-resolveDupInstances
             -globalNets VDD
             -forceGlobalsOn
```

5. Para ejecutar ICV layout-versus-schematic (LVS), abra el directorio **de trabajo** ubicado en el directorio **lvs**. En el directorio **de trabajo** localice el archivo de runset lvs (saed14nm_1p9m_lvs_rules.rs**).** En este archivo se encuentran las direcciones actuales con **Johnson.gds,** los resultados de salida, el nombre del módulo Top actual de Verilog y la netlist esquemática, que se generaron a partir de NetTran (**Johnson_icc.cdl**).

Cómo cambiar el archivo **saed14nm_1p9m_lvs_rules.rs** se muestra a continuación:

```
/*Defined all needed variables*/

library(
        library_name  = ../source/Johnson.gds,
        format = GDSII,
        cell = "Johnson"

);
DECK_TYPE : string = "PEX_DECK";
CROSS_REFERENCE : string = "yes";
ICV_RUN_FILE_DIR: string = ".";


openaccess_options(
   view="layout",
   layer_mapping_file= "./techfiles/saed14nm_1p9m_gdsout.map"
   );

library(
```


```
library_name= "CELLNAME.gds",
format=GDSII,
cell= "resistor"
);

layout_netl_fh:handle;
device_db:device_database;
xref_db_h:handle;

schem_netl_fh = schematic(

schematic_file    = {{filename = "CELLNAME.sp",format = SPICE}}

);

error_options(db_path="./TOPCELLNAME_err",
create_vue_output=true );

run_options(
instance_prefix="X",
uppercase=false
);
hierarchy_options(flatten={"*"});

resolution_options(snap_resolution=0.001);
text_options(
        net_prefix="N",
        layout_power={"VDD","VDD12","VDD22", "VDD2","VDDG", "VDDH",
"VDDL", "VDDIO", "VDDPAD"},
        layout_ground={"VSS","VSS12","VDD22", "VSSIO"}
);

net_options(
schematic_power={"VDD","VDD12","VDD22", "VDD2","VDDG", "VDDH",
"VDDL", "VDDIO", "VDDPAD"},
schematic_ground={"VSS","VSS12","VDD22", "VDD2", "VDDH", "VDDL",
"VDDIO"},
schematic_global={"VDD","VDD12","VDD22","VSS","VSS12","VDD22",
"VDD2","VDDG", "VDDH", "VDDL", "VDDIO", "VSSIO", "VDDPAD"}
);
prototype_options(false);
```

ICV LVS realiza un proceso de comparación que verifica si la implementación geométrica o de diseño de un circuito coincide con la representación esquemática.

6. Ejecute ICV LVS desde el directorio **de trabajo** con el siguiente comando:

```
icv –i lib_name.gds –c cell_name runset_name -vue
```

7. Revise y depure los errores generados en la ejecución de ICV LVS utilizando archivos de error ICV. Ver el

Los errores de LVS en **Johnson. LVS_ERRORS** archivo que se encuentra en el directorio **de resultados**.

En este laboratorio, los resultados de la comparación de bloques superiores contienen errores como los anteriores: Esos errores deben corregirse. Lo cual no está incluido dentro de este laboratorio.

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

16.9. Laboratorio 8: Extracción de parásitos de diseño. StarRC

Synopsys Design Flow Tutorial
Laboratorio 8: Extracción de parásitos de diseño. StarRC

Objetivo

Aprenda a usar StarRC para realizar extracciones parasitarias.

Introducción

StarRC es una herramienta de extracción parásita de diseño de próxima generación que extrae parásticos RC para un diseño. StarRC se puede utilizar en cualquier etapa del ciclo de diseño físico para extraer parásitos precisos.

Este laboratorio requiere un archivo de comandos StarRC que especifique rutas de acceso a los datos de entrada y referencias tecnológicas. A continuación, se muestra un ejemplo de dicho archivo:

```
BLOCK: johnson
TCAD_GRD_FILE: ../ref/tech/star_rcxt/saed14nm_1p9m_nominal.nxtgrd
ICV_RUNSET_REPORT_FILE: ../source/work_lvs/pex_runset_report
EXTRACTION: RC
NETLIST_FILE: ../results/johnson_rcx.spf

** spice netlist for port order
SPICE_SUBCKT_FILE: ../source/johnson_icc.cdl
XREF: YES
```

Tareas de laboratorio

1. Cambiar directorio ~/work/. Obtenga los resultados de LVS del trabajo de laboratorio de ICV, que es requerido por StarRC. Desde este directorio:

```
% StarXtractor saed14nm_1p9m_star_nominal.cmd
```

En el resultado que se está consiguiendo netlist parásito – Johnson_count.spf, que se puede incluir en deck, como en el trabajo de laboratorio de HSPICE, y después de eso corriendo circuito real con sus elementos parásitos.

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

16.10. Laboratorio 9: Simulación a nivel SPICE del diseño completado. HSPICE

Synopsys Design Flow Tutorial

Laboratorio 9: Simulación a nivel SPICE del diseño completado. HSPICE

Objetivo

Aprenda a usar HSPICE para ejecutar la simulación a nivel de dispositivo.

Introducción

Para la simulación HSPICE se necesitan los siguientes archivos de entrada netlist, archivo de modelo (saed14nm.lib), tipo de análisis, opciones y medidas. El archivo de presentación johnson_icc.sp en el directorio ~/tutorial/HSPICE/work/ incluye el archivo de modelo (saed14nm.lib), netlist (johnson_rcx.spf) y contiene opciones de tipo de análisis.

Tareas de laboratorio

1. Cambiar directorio a ~/work

```
% cd work
```

2. Ejecute HSPICE. Este comando ejecuta HSPICE y los resultados guardados del directorio de salida /results.

```
% hspice johnson_testbench.spi  –o ../results
```

3. Los resultados están en el directorio ~/results (johnson_icc.mtX, johnson_icc.trX). Los archivos binarios johnson_icc .trX se pueden abrir utilizando visores de forma de onda como WaveView o CosmosScope.

El comando OPTION establece una amplia variedad de opciones de simulación HSPICE. ACCURACY establece la precisión del solucionador en uno de los siguientes: BAJO, MEDIO, ALTO.

OPTION AUTOSTOP	AUTOSTOP
SYNTAX	.OPTION AUTOSTOP
DESCRIPTION	Stops a transient analysis in HSPICE, after calculating all TRIG-TARG, FIND-WHEN, and FROM-TO measure functions. This option can substantially reduce CPU time. You can use the AUTOSTOP option with any measure type.

OPTION	GMIN
SYNTAX	.OPTION GMIN=x
DESCRIPTION	Use this option to specify the minimum conductance added to all PN junctions for

	a time sweep in transient analysis. The default is 1e-12.
OPTION	POST
SYNTAX	.OPTION POST=[0|1|2|3|ASCII|BINARY]
EXAMPLE	.OPTION POST=2
DESCRIPTION	Use an .OPTION POST statement to display high-resolution AvanWaves plots of simulation results on either a graphics terminal or a high-resolution laser printer. Use .OPTION POST to provide output, without specifying other parameters.

OPTION	GSHUNT
SYNTAX	.OPTION GSHUNT=x
DESCRIPTION	Use this option to add conductance from each node to ground. Add a small GSHUNT to each node to help solve "timestep too small" problems caused by either high-frequency oscillations or numerical noise.

OPTION	CAPTAB
SYNTAX	.OPTION CAPTAB
DESCRIPTION	Use this option to print a compiled table of single-plate node capacitances for diodes, BJTs, MOSFETs, JFETs, and passive capacitors at each operating point.

In this design output results are stored in ../**results/** directory.

==========

CAPÍTULO 17

Glosario

Conos lógicos: Un cono lógico consiste en una lógica combinacional que se origina en un objeto de diseño específico y se abre en abanico hacia atrás para terminar en determinadas salidas del objeto de diseño.. 42

Diseño de implementación: Este diseño es el diseño modificado. Es el diseño cuya corrección desea probar. Por ejemplo, un diseño recién sintetizado es una implementación del diseño RTL de origen.. 43

Diseño de referencia: Este diseño es el diseño dorado, el estándar contra el cual la formalidad prueba la equivalencia.. 43