UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Mejoramiento del proceso de síntesis lógica llevada a cabo para la elaboración de un circuito integrado a escala nanométrica

Trabajo de graduación presentado por Karol Sophia Cardona Polanco
para optar al grado académico de Licenciada en Ingeniería Electrónica

Guatemala,

2021

Vo.Bo.:

(f) _____

MSc. Carlos Esquit

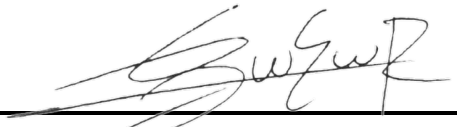
Tribunal Examinador:

(f) _____

MSc. Carlos Esquit

(f) _____

Ing. Jonathan de los Santos



(f) _____

Ing. Guilmar Escobar

Fecha de aprobación: Guatemala, 22 de junio de 2021.

Agradecimientos

Primero que todo quiero agradecer a mis papás, Sergio Cardona y Flor de María de Cardona, por haberme dado la oportunidad de estudiar y desarrollarme en mi carrera en esta casa de estudios. Por apoyarme siempre e impulsarme a seguir adelante aprovechando mis capacidades y cualidades confiando en mí para llegar a ser lo que soy hoy.

Agradezco a mis hermanos por el apoyo incondicional que me brindaron estos años, la compañía durante mis proyectos y tareas, los ánimos que me daban cada vez que sentía que no podía seguir.

Agradezco a toda mi familia que siempre creyó en mí y aplaude cada uno de mis logros y me apoya en mis fracasos. Gracias por hacerme sentir capaz de lograr lo que deseo, por celebrar conmigo cada uno de mis triunfos, estar al pendiente de mis avances y apoyarme a iniciar la carrera que siempre quise.

Por último quiero agradecer a Luis Estuardo Abadia López, por su apoyo incondicional durante los últimos años de mi carrera, los ánimos que me brindó en cada momento que sentí no poder hacer un proyecto. Su compañía en las horas de estudio y explicaciones en las clases que se me dificultaron, por confiar en mí y hacerme saber todos los días que soy capaz de lograr lo que quiero en todos los ámbitos de mi vida.

Índice

Agradecimientos v

Lista de figuras x

Resumen xii

Abstract xiii

1. Introducción 1

2. Antecedentes 3

3. Justificación 5

4. Objetivos 7
4.1. Objetivo general 7
4.2. Objetivos específicos 7

5. Alcance 9

6. Marco teórico 11
6.1. Lenguaje descriptivo de hardware, *HDL* 11
6.2. Flujo de diseño 12
6.3. Synopsys 12
6.4. Design Vision 13
6.5. Librerías 13
6.6. Síntesis lógica 14
6.7. Netlist 14
6.8. IC Validator 15
6.9. NetTran 15

7. Síntesis lógica 17
7.1. Flujo de diseño 17
7.2. Diseño de circuito en HDL 18

7.3. Configuración de Design Vision 19
7.3.1. Análisis de comandos *script* original 20
7.3.2. Librerías 21
7.3.3. Importación de archivos 24
7.3.4. Configuración de restricciones de diseño 27
7.3.5. Revisión de diseño y compilación 33
7.3.6. Archivos y reportes generados 35
7.4. Diseño de circuito con pines de entrada y salida 40
7.5. Nuevo *script* y *netlist* finales 45

8. Traducción de netlist **53**
8.1. Proceso 54
8.2. Pruebas 55
8.2.1. Not 55
8.2.2. Full Adder 57

9. Conclusiones **59**

10. Recomendaciones **61**

11. Bibliografía **63**

Lista de figuras

1. *HDL* para un sumador de un bit . . . 11
2. Flujo de diseño para un circuito integrado [5] . . . 12
3. Esquemático de un sumador después del proceso de síntesis lógica . . . 13
4. Netlist de un sumador de un bit sintetizado . . . 14
5. Proceso de traducción de *Netlist* [14] . . . 15

6. Diagrama de flujo para Síntesis lógica . . . 18
7. Interfaz gráfica y consola de la herramienta de *Design Vision* . . . 19
8. Primer paso para incluir las librerías . . . 23
9. Interfaz para agregar los archivos de librerías . . . 23
10. Importación de archivos . . . 23
11. Configuración para cargar el archivo que describe el circuito . . . 25
12. Ventana para cargar el archivo . . . 25
13. Formatos disponibles . . . 25
14. Incorporación del archivo para mostrar en la herramienta . . . 26
15. Último paso para completar la carga del circuito a *Design Vision* . . . 26
16. Selección para tener una vista del esquemático . . . 27
17. Circuito *Not* en el nivel jerárquico más alto, compuerta . . . 27
18. Circuito interno de la compuerta *Not* . . . 27
19. Pasos a seguir para configurar las *constraints* desde la interfaz gráfica . . . 28
20. Opciones que se pueden configurar en la sección de restricciones desde la interfaz gráfica . . . 28
21. Listado de reportes disponibles desde la interfaz gráfica . . . 37
22. Ventanas para generar reporte de área, reloj y diseño . . . 37
23. Reporte de área para una compuerta Not . . . 38
24. Ejemplo de reporte de Diseño . . . 39
25. Reporte de jerarquía . . . 39
26. Reporte de potencia . . . 40
27. *Netlist* compuerta *Not* sintetizada . . . 41
28. Tabla de verdad pin PDDW0204SCDG . . . 42
29. Representación de pin PDDW0204SCDG configurado como entrada . . . 42
30. Configuración para pines de entrada . . . 43
31. Representación de pin PDDW0204SCDG configurado como salida . . . 43

32. Configuración para pines de salida 43
33. Descripción de circuito *Not* con pines de entrada y salida 44
34. *Netlist* final producto de la síntesis lógica a una compuerta *Not* 47
35. *Netlist* final producto de la síntesis lógica a una compuerta *Nand* de dos entradas 47
36. *Netlist* final producto de la síntesis lógica a una compuerta *Nor* de tres entradas 48
37. *Netlist* final producto de la síntesis lógica a un sumador de dos *bits* 48
38. *Netlist* final producto de la síntesis lógica a un sumador de ocho *bits* 51

39. Archivo descriptivo de las celdas para llevar a cabo la traducción de *netlist* . . 54
40. Terminal de la herramienta *IC Validator* 55
41. *Netlist* generado de la síntesis circuito *Not* 56
42. Traducción *netlist* circuito *Not* 56
43. Traducción a nivel transistores de circuito *Not* 56
44. Síntesis de circuito *Full Adder* con instancias de VDD y VSS 57
45. Traducción a *spice* de *netlist* e inclusión de librería 57
46. *Netlist* a nivel transistor de circuito *Full Adder* 57

Resumen

El presente proyecto abarca el estudio y reconfiguración de la etapa de síntesis lógica para llevar a cabo la fabricación del primer *chip* diseñado en Guatemala. Esta etapa es la primer fase del flujo de diseño con el que se está trabajando, por lo que es importante lograr una configuración que produzca resultados satisfactorios. Este flujo de diseño fue propuesto por los estudiantes predecesores ex alumnos de la Universidad del Valle de Guatemala. El objetivo principal de este proyecto es desarrollar la sección de síntesis lógica adecuada para poder implementarla en las etapas posteriores del flujo de diseño.

Para continuar con la línea de investigación existente, se hizo uso de la herramienta *Design Vision* perteneciente al grupo de *Synopsys*. A través de esta se llevaron a cabo distintas pruebas con el objetivo de verificar el funcionamiento del *script* proporcionado al inicio de la segunda fase del proyecto. Así mismo también se realizó una revisión del manual de la herramienta para poder entender el entorno de trabajo. Los crcuitos que se utilizaron para comprobar la funcionalidad de la configuración fueron las compuertas *Not*, *Nand* de dos entradas, *Nor* de tres entradas, y a parte sumadores de 2, 8 y 32 *bits*.

El proceso de síntesis lógica se dividió en dos secciones. En la primer sección se llevó a cabo la síntesis del circuito descrito en *HDL* generando un primer archivo en 'verilog', mientras en la segunda se procedió a elaborar la síntesis pero ahora con el nuevo archivo descrito en 'verilog' junto a la instancia a los pines de entradas y salidas. Esta última configuración incluye la creación y asignación de pines de entrada y salida físicas al diseño, para ello también se revisó el proceso ya existente y se adecuó según los resultados que se obtuvieron en las pruebas.

Todo el proceso detallado en el documento se realizó a través de líneas de comando descritas en la consola de *Design Vision*, estas pruebas y resultados fueron documentados en el presente trabajo de investigación, así mismo se muestra dicho proceso si se desea realizar a través de la interfaz gráfica de la herramienta. Esto se decidió realizar para brindarle al estudiante una guía del proceso paso a paso de tal forma que pueda entenderlo y ejecutarlo sin ninguna dificultad.

Como parte extra al proceso de síntesis se obtuvo la traducción del *netlist* a nivel transistor con ayuda de los archivos brindados por *Synopsys* con fines académicos. De igual forma

el desarrollo se encuentra debidamente documentado en uno de los capítulos del trabajo escrito.

Abstract

The present project covers the study and configuration of the stage of logical synthesis to carry out the fabrication of the fisrt chip designed in Guatemala. This stage is the first step in the design flow with which we are working, so it is important make a configuration that produce satisfactory results. These design flow was proposed by the predecessors students of the project at the Universidad del Valle de Guatemala. The main objective is develop the phase of logical synthesis adequate to be implemented in the next stages.

To continue with the investigation, it was used the Design Vision tool that is part of the Synopsys software. Through this were carried out a series of tests with the objective to verify the results of the script proposed at the begining of the investigation. Also were made a review of the manual tool to understand the enviroment of work. The circuit that was used to corroborate the function of the conofigurations, was a Not gate, Nand gate with two inputs and Nor gate with three inputs, also worked with two, eight and 32 bits adders.

The process of logical synthesis was divided in two sections. In the first it takes place de synthesis of the circuit described in HDL thus generating a first file in verilog description. In the second one we proceed to elaborate the synthesis of new circuit that include the last verilog and the instances of in and out physical pins. This last configuration include the creation and asigment of the pins to the design. For this reason, it was also reviewd the process and it was adecuated according to the results obtained in the tests, since there were some errors.

All the process was made through command lines described in the console of Design Vision. These tests and results were documented in the present work, it was also described the process if the student want elaborate the synthesis through the graphic interface tool. This was decided to do to provide the student a guide step by step of the process in such a way he can understand and run it without any difficulty.

Extra to the main objective, it has been made the translate of the netlist obtained as a product of the synthesis. This traduction was made at transistor level with the help of information and files provided for Synopsys group. In the same way de development it was captured in one chapter of the document, it conatins all the information and instructions to execute it in the best way.

CAPÍTULO 1

Introducción

Con el paso de los años el porcentaje de dispositivos electrónicos que hay en el mundo ha ido en aumento, esto se debe a la demanda que el ser humano ha creado, ya que se han vuelto algo indispensable para el día a día. Este crecimiento se ha visto tanto en las empresas de manufactura como en las empresas desarrolladoras de tecnología. En estas últimas es donde los encargados del área de diseño se han dado a la tarea de crear soluciones que satisfagan al cliente, es decir, mejorar las características y propiedades del producto que fabrican.

Los avances tecnológicos no habrían sido posibles si no se hubiera dado el nacimiento del primer circuito electrónico en el año 1958. En ese año el Ingeniero Jack St. Clair Kilby que laboraba en **Texas Instruments** llevó a cabo este gran desarrollo para la humanidad. Previo a esta invensión los componentes con los que se construían los distintos equipos electrónicos eran tubos al vacío [1].

Un circuito integrado es un componente electrónico que incorpora e interconecta múltiples dispositivos electrónicos a escala nanométrica, mayormente transistores, estos son fabricados en varias capas de materiales semiconductores, metales y dieléctricos [2]. Para llevar a cabo la fabricación de un circuito electrónico es necesario seguir una serie de pasos que aseguren la obtención de resultados satisfactorios, normalmente este proceso se describe a través de un flujo de diseño [3]. La estructura de este depende del diseñador y el objetivo que se quiera cumplir.

Actualmente la fabricación de *chips* se realiza en grandes cantidades, esto es realizado por empresas como *IBM*, *Intel*, *Texas Instruments*, *Samsung*, *TSMC*, por mencionar algunos. Entre ellos varios se dedican tanto al diseño lógico como a la creación en físico de los componentes. Sin embargo, también se dedican a fabricar los diseños de grupos o compañías pequeñas que deseen crear un chip.

Se conoce de grandes países pioneros en el diseño de circuitos integrados y está cada vez más cerca que Guatemala pase a formar parte de esta lista, esto como producto de la dedicación y perseverancia de las personas involucradas en el proyecto. Será en la Universidad del Valle de Guatemala donde se lleve a cabo el diseño del primer chip. Han sido años de

investigación y preparación por parte del director del departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la universidad, Ingeniero Carlos Esquit. Así también ha existido preparación por parte de los estudiantes involucrados en este gran proyecto.

Como se mencionó existe un flujo de diseño para poder ejecutar la elaboración del diseño del circuito integrado. El proyecto se basará en el sugerido por el grupo de estudiantes involucrados en el año 2019 [3][4]. La primer etapa que se debe cumplir es la elaboración del circuito a implementar descrito en *hdl* para poder darle así paso a la fase de síntesis lógica, en la cual se convierte la descripción del circuito en términos de celdas [5]. Este archivo es el diseño que sienta las bases para posteriormente, en la etapa de síntesis física, se lleve a cabo el proceso de colocación, interconexión y así la creación del chip como tal.

CAPÍTULO 2

Antecedentes

En el año 2013 se impartió por primera vez el curso de introducción al diseño de sistemas VLSI presentado por el Ing. Carlos Esquit en la Universidad del Valle de Guatemala. El objetivo de este era presentar algunos de los conceptos empleados en el diseño de sistemas electrónicos a escala nanométrica. En el año 2014 se logró llegar a un acuerdo con la compañía de Synopsys para poder hacer uso de las herramientas de simulación que ofrecen. A través de estos recursos los estudiantes podrían reforzar sus conocimientos. Ese mismo año el estudiante de Ingeniería Electrónica, Jonathan Santos, realizó su trabajo de graduación en el cual acopló los conceptos que tenía junto con el paquete de *software* de Synopsys [6].

Durante el año 2019, cuatro estudiantes de la carrera de Ingeniería Electrónica de la Universidad del Valle de Guatemala trabajaron en conjunto para desarrollar su trabajo de graduación. Este tenía como finalidad la creación del flujo de diseño para la fabricación de un circuito integrado a escala nanométrica [4][3].

CAPÍTULO 3

Justificación

Para llevar a cabo la fabricación de un circuito electrónico a escala nanométrica se necesita que este cumpla con ciertos requerimientos de diseño, los cuales son proporcionados por la empresa fabricante. Este proceso necesita realizarse a través de una serie de pasos ordenados de forma jerárquica, los cuales se describen en el flujo de diseño. Cada una de las partes del flujo posee de igual manera pasos a seguir para obtener resultados satisfactorios, pero estos no se pueden llegar a completar si no se tiene algún conocimiento de la herramienta y el entorno donde se está desarrollando. Para ello es necesario dedicar tiempo a cada una de las etapas, estudiando procesos previos y tratando de mejorarlos.

Con una base sólida ya creada será posible desarrollar el presente trabajo de investigación, con el que se pretende mejorar el proceso de síntesis lógica dándole paso a que se eviten errores y se genere un flujo de diseño limpio, acercándose más a la fabricación del primer chip desarrollado en Guatemala. Esto será un incentivo tanto para Universidades como para instituciones y personas ajenas a dedicar su tiempo en investigación y experimentación para el desarrollo del país y de ellos mismos.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar la sección del flujo de diseño dedicada a la síntesis lógica de un circuito elaborado en lenguaje descriptivo de *hardware*, utilizando las herramientas de *software* de Synopsys.

4.2. Objetivos específicos

- Rediseñar la estructura de los comandos para obtener mejores resultados y un proceso sin errores, en los que se incluyen mapeos que no se realizaron correctamente e incompatibilidad con las librerías utilizadas.
- Comparar el proceso de síntesis lógica rediseñado a través de pruebas contra el que se nos proporcionó al inicio del proyecto y así poder observar si hubo cambios en la estructura del *netlist* generado, el área utilizada y la cantidad de compuertas.
- Buscar una herramienta en el *software* proporcionado por Synopsys para generar el *netlist* del circuito sintetizado a nivel de transistores.

CAPÍTULO 5

Alcance

Lo que se quiere lograr con esta investigación, es la mejora del proceso de síntesis lógica. Esto se pretende realizar con base a la documentación y archivos proporcionados por los estudiantes antecesores de dicho proyecto. Para poder llevar a cabo la reconfiguración se analizará el *script* elaborado por el Ingeniero Luis Arturo Nájera, archivo con el cual se elaboró la síntesis previamente. Esto constará del análisis de cada uno de los comandos que se implementaron.

Así mismo se revisarán posibles comandos que pueden hacer el proceso más limpio, la fuente que se utilizará será el listado de comandos brindado por *Synopsys*, en donde se encuentra la información necesaria. Si llega a encontrarse alguna configuración útil para el proceso, se agregará a la estructura de instrucciones ya definida. Las decisiones que se tomen serán con base a pruebas y revisión de documentación.

Por otra parte se desea completar satisfactoriamente el proceso completo de síntesis para poder generar un *netlist* funcional para las siguientes etapas del flujo de diseño. Así también se pretende llevar a cabo la traducción de dicho archivo a nivel transistores para posibles comparaciones futuras. Entre estas la revisión física de los transistores así como la cantidad total de componentes que se estarán utilizando.

CAPÍTULO 6

Marco teórico

6.1. Lenguaje descriptivo de hardware, *HDL*

Es un lenguaje, tipo de programación, que le permite al diseñador describir un circuito en forma de texto en lugar de un esquema. *HDL* permite que el *hardware* se describa en una amplia gama de estilos a través de módulos con descripción de compuertas, funciones, etc, como se puede observar en la Figura #1. En esta parte la claridad en la descripción es de vital ayuda para la calidad del diseño, ya que así será más fácil de depurar, mantener y actualizar el código [7]. Uno de los estándares más utilizados en la actualidad es *Verilog*, este lenguaje nos permite la descripción del diseño en diferentes niveles denominados niveles de abstracción: nivel de compuerta, nivel de transferencia [8].

- Nivel de compuerta: Corresponde a una descripción a bajo nivel también denominada modelo estructural. El diseñador realiza la descripción mediante el uso de primitivas lógicas (AND, OR, NOT, etc.) y conexiones lógicas [8].
- Nivel de transferencia: Los diseños descritos a este nivel especifican las características de un circuito mediante operaciones lógicas y la transferencia de datos entre registros. Mediante el uso de especificaciones de tiempo las operaciones se realizan en instantes determinados [8].

```
module FullAdder (cin, A, B, suma, cout);
    input A,B,cin;
    output suma,cout;

    wire w1,w2,w3;

    //Haciendo Suma
    xor Xor1(suma,A,B,cin);
    //Haciendo carrie out
    and And1 (w1,A,B);
    xor Xor2 (w2,A,B);
    and And2 (w3,w2,cin);
    or  Or1  (cout,w3,w1);

endmodule
```

Figura 1: *HDL* para un sumador de un bit

6.2. Flujo de diseño

El flujo de diseño es un término utilizado para referirse al conjunto de pasos que se deben realizar para convertir las especificaciones funcionales de un sistema digital en un plano que será enviado a alguna fábrica de circuitos integrados para su manufactura. Se pueden plantear diferentes flujos de diseño según las herramientas con las que se cuenten, así como también el tipo de circuito integrado (analógico, digital, mixto, etc.), la complejidad del sistema, etc. Particularmente se le llama flujo de síntesis cuando se emplea un lenguaje de descripción de *hardware* para obtener el diagrama estructural mediante herramientas de síntesis [9].

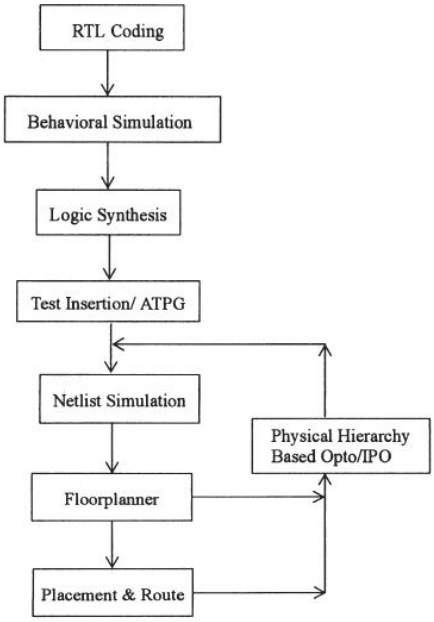


Figura 2: Flujo de diseño para un circuito integrado [5]

6.3. Synopsys

Synopsys es el proveedor líder mundial de soluciones para diseñar y verificar chips de silicio avanzados, brindar seguridad de *software*, llevar a cabo pruebas de calidad y diseñar los procesos y modelos de próxima generación necesarios para fabricar esos chips. Cuenta con una amplia gama de herramientas disponibles para llevar a cabo el diseño de un circuito integrado y todo el flujo de diseño que este conlleva. Así mismo brinda documentación de soporte como guías de usuario para que la interacción del cliente con cada uno de los programas sea la mejor y se aprovechen al máximo los recursos.

6.4. Design Vision

Este *software* es parte del paquete del grupo de *Synopsys*, el cual provee herramientas para la vista y el análisis de un diseño a nivel de tecnología y de compuertas. *Design Vision* ofrece dos interfaces: una interfaz solo de texto donde son ingresadas las líneas de comando y una interfaz gráfica, la cual proporciona menús con los comandos de síntesis de uso más frecuente y herramientas de análisis visual. A través de esta herramienta se lleva a cabo el proceso de síntesis lógica con el cual se puede observar el esquemático generado del circuito sintetizado. En la Figura #3 se observa un ejemplo de ello del proceso llevado a cabo con Design Vision.

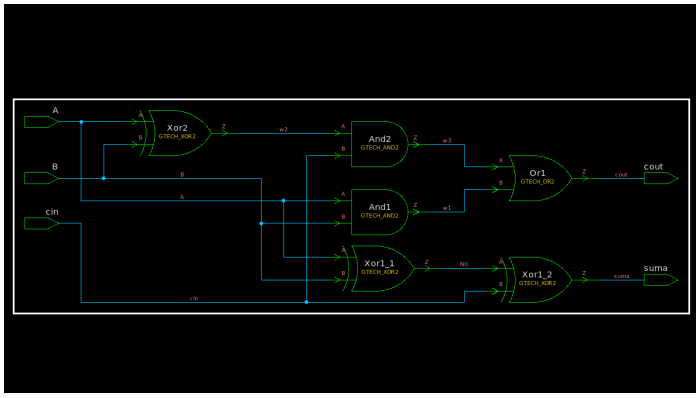


Figura 3: Esquemático de un sumador después del proceso de síntesis lógica

La vista del esquemático generado es de ayuda para realizar análisis de tiempo y de lógica, las rutas pueden estar representadas en diseños y sub-diseños dependiendo qué tan específico se requiera. El sub-diseño más pequeño que se puede obtener en Design Vision es a nivel de compuertas [10].

6.5. Librerías

Las librerías son referencias para que la síntesis lógica se lleve a cabo, ya que en ellas está contenida la información de las celdas para generar el esquemático y el *netlist*. Existen tres tipos de librerías las cuales son necesarias para el proceso de síntesis lógica, estas son la *Target Library, Link Library* y *Symbol Library*. Cada una de ellas tiene una función distinta pero que a la vez hace que se complementen.

- *Target Library* es la librería de los proveedores en donde se encuentran las celdas disponibles para utilizar en la generación de la *netlist* [5].
- *Link Library* posee la referencia a pines descritos en el circuito, por ejemplo los pines de entrada y salida [5].
- *Symbol Library* son las representaciones pictóricas de las celdas, si se desea ver el esquema del diseño entonces la herramienta requiere una librería de símbolos que

contenga las representaciones gráficas reales de todas las celdas [5].

Se necesita que las celdas sean las mismas en las tres librerías, es decir, que el nombre y el tipo sea igual, ya que si no se da el caso entonces habrá errores de diseño y se tendrá que optar por utilizar otra librería o corregir ese error.

6.6. Síntesis lógica

En la síntesis el código descrito en *HDL* se asigna a las compuertas lógicas de *hardware* para una tecnología en específico las cuales son descritas en las librerías empleadas. El proceso de síntesis lógica consta de dos pasos: traducción y optimización. La traducción implica transformar una descripción *HDL* en compuertas, mientras la optimización implica seleccionar la combinación óptima de la biblioteca de tecnología y las celdas para lograr la funcionalidad requerida [5]. Durante esta fase el diseñador puede agregar restricciones de diseño en la herramienta de síntesis permitiendo que la asignación de componentes sea más eficiente,esta optimización se realiza solo si es necesario para obtener mejores resultados [11].

Como producto del proceso de síntesis lógica se generarán tres formatos de archivos, estos son '.sdc', '.ddc' y '.v'. El primero agrupa todas las limitaciones del *netlist*, estas son configuradas por el usuario durante el proceso de síntesis en donde se incluyen los límites de área y reloj [12]. Ahora el formato de salida '.ddc' almacena información sobre el *netlist* generado así como las restricciones asignadas al circuito [10].

6.7. Netlist

Este documento describe el circuito en orden jerárquico como una red de dispositivos con nombres y pines según lo especificado por las librerías utilizadas [13]. El formato con el que se almacena este documento es en *Verilog*, ya que en un futuro es necesitado para continuar con el flujo de diseño para la fabricación de un circuito a nano escala. El tamaño de este archivo dependerá tanto de la cantidad de componentes utilizados como de los niveles de jerarquía que se utilizaron, esto se puede ver afectado por el proceso de síntesis que el usuario decida emplear.

```
module FullAdder ( cin, A, B, suma, cout );
  input cin, A, B;
  output suma, cout;
  wire   n2;

  XOR2X1 U4 ( .IN1(cin), .IN2(n2), .Q(suma) );
  AO22X1 U5 ( .IN1(A), .IN2(B), .IN3(cin), .IN4(n2), .Q(cout) );
  XOR2X1 U6 ( .IN1(A), .IN2(B), .Q(n2) );
endmodule
```

Figura 4: Netlist de un sumador de un bit sintetizado

6.8. IC Validator

IC Validator es una herramienta utilizada para la validación de un circuito realizada a través de la comparación del diseño en silicio y el esquemático del circuito [14]. Este *software* ofrece una mejor escalabilidad de procesamiento distribuido de la industria a más de 2000 núcleos de CPU. El rendimiento y la escalabilidad de la herramienta han permitido algunos de los chips de límite de retícula más grandes de la industria con miles de millones de transistores, la verificación de reglas de diseño el mismo día con la opción de diseño *versus* esquemático [15].

6.9. NetTran

Es una herramienta que se encuentra dentro del *software* de IC Validator, esta permite leer y traducir una *netlist* al formato que se desee, el formato que se requiere en este caso es un circuito a nivel transistores para poder realizar satisfactoriamente el proceso de *LVS*.

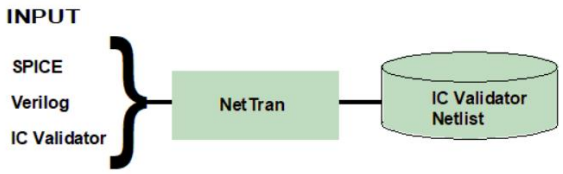


Figura 5: Proceso de traducción de *Netlist* [14]

En la Figura #5 se observa el formato de los archivos que son soportador por IC Validator para realizar la traducción de *netlist* a su formato. Estos archivos deben tener declarados cada uno de los puertos de entradas, salida o cables que son utilizados ya que sino es así entonces no se puede realizar la conversión. Si el *software* detecta algún pin flotante o un puerto sin conexión, generará una advertencia al usuario.

CAPÍTULO 7

Síntesis lógica

7.1. Flujo de diseño

Para llevar a cabo el proceso de síntesis lógica se necesita seguir una serie de pasos, con los cuales se podrá llegar a generar un *netlist* en formato *verilog*, este será utilizado en las siguientes etapas de diseño. El flujo de diseño que se utilizó para la fase de síntesis se muestra en la Figura #6. Más adelante se detallará cada uno de los pasos con la descripción de lo que se realizó, las herramientas que se implementaron, así como las pruebas y resultados obtenidos durante la realización del presente proyecto. Comenzando es de gran importancia poseer algún conocimiento sobre las herramientas a utilizar del paquete del *software* de Synopsys. Esto se puede realizar a través de la lectura de los manuales y guías brindadas por la compañía en *SolvNet*. Para poder acceder a esta documentación es necesario contar con un usuario y contraseña.

El diagrama de flujo que se presentará es uno de los caminos que se pueden seguir para cumplir el objetivo del proyecto. Sin embargo este se puede llegar a optimizar o construir de una manera más completa. Cualquier persona con un poco de conocimiento en la materia puede ser capaz de replicar los pasos que se mencionarán más adelante e incluso poder modificarlos si lo cree pertinente.

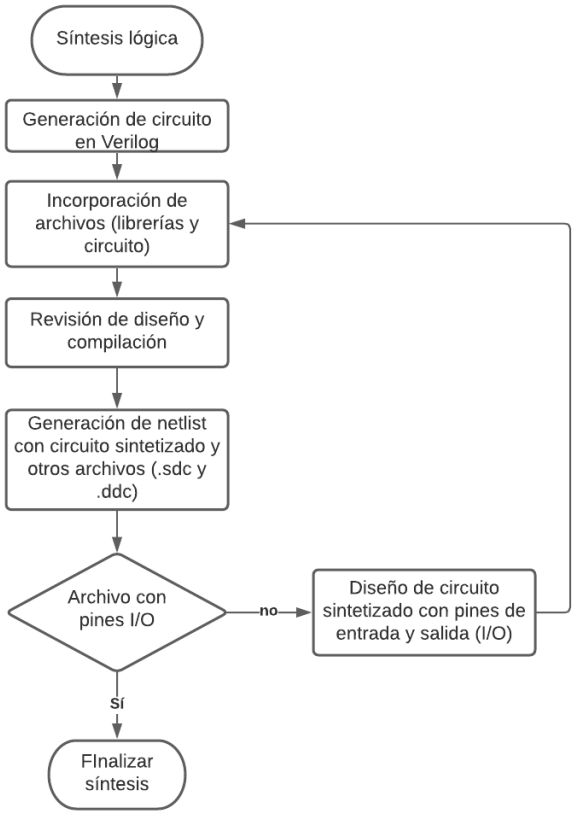


Figura 6: Diagrama de flujo para Síntesis lógica

7.2. Diseño de circuito en HDL

El primer paso a tomar en cuenta es la elaboración de un circuito descrito a nivel compuerta o a nivel de transferencia para comenzar con las pruebas. Dependiendo de la decisión que se tome respecto al nivel de abstracción la estructura jerárquica final se puede tornar sencilla o al contrario más compleja. Esto se debe a que el software de *Design Vision* se basa en este para la síntesis en conjunto con las librerías y las celdas disponibles. En caso el diseño se realice a nivel compuerta, se sugiere que sea utilizando configuraciones de únicamente dos entradas en cada una de las compuertas ya que algunas configuraciones, por ejemplo utilizar tres entradas en una compuerta *and*, no se encuentran descritas en las librerías y el *software* lo reemplaza con una conexión lógica similar un poco más compleja.

En este proceso se comenzó realizando pruebas con un circuito bastante sencillo, una compuerta *not*.

```
1 module Not (A, Y);
2         input A;
3         output Y;
4         assign Y = ~A;
5 endmodule
```

Se decidió comenzar de esta manera para que fuese más sencilla la depuración y debido a que si la síntesis lógica se lograba exitosamente con este modelo, entonces se escalaría a un circuito más detallado, para ser más específicos, un *Full Adder*. Con estas pruebas lo que se pretendía era tener una mejor visión de lo que es el flujo de diseño y posibles fuentes de errores provenientes de la descripción del circuito. Para evitar que la revisión se vuelva un proceso largo, se sugiere realizar pruebas con un *test* en alguna herramienta externa en donde se observe que el comportamiento de dicho circuito sea el esperado. Existen varios documentos que se vuelven guías para el entendimiento del diseño en *HDL*, una de las referencias que se utilizó es [8]. Este tutorial ejemplifica los conceptos de sintaxis, las distintas representaciones de constantes y definición de variables.

7.3. Configuración de Design Vision

Esta herramienta puede ser manejada de dos formas, a través de comandos o a través de la interfaz gráfica. El grupo de Synopsys brinda un manual en donde describe todos los comandos que se pueden utilizar y la función que tienen. Para un mejor manejo de ellos se sugiere una lectura en paralelo de las guías de Design Vision y Design Compiler, ya que en cada una de las secciones se hace referencia a cómo funciona el *software*, cómo hacer uso de todas sus características y posibles errores que surgen durante el proceso.

Para llevar a cabo el presente proyecto se optó por utilizar las propiedades de la herramienta a través de líneas de comandos, ya que es una manera más sencilla de controlar los procesos. Para ingresar a la herramienta se debe ir a la ubicación de la carpeta donde se encuentra la herramienta, en este caso se encuentra en la siguiente ruta: `Aplicaciones/Archivos/OtrasUbicaciones/Equipo/usr/Synopsys/Syn/bin`

Una vez en la carpeta de *bin* se abre una terminal en dicha ubicación. Luego se ingresa el siguiente comando el cual se encarga de abrir la interfaz gráfica de la herramienta.

```
./design_vision
```

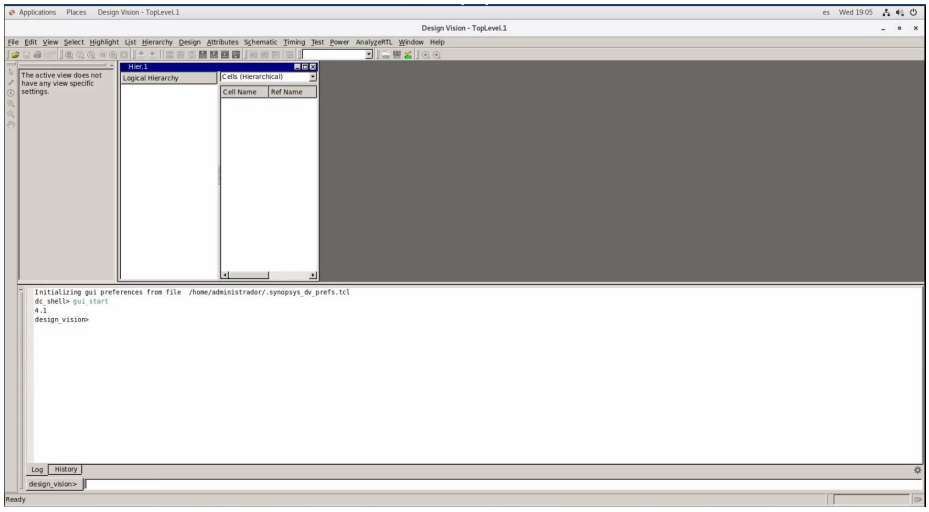


Figura 7: Interfaz gráfica y consola de la herramienta de *Design Vision*

A este punto ya puede hacerse uso de *Design Vision*, los comandos para llevar a cabo el proceso se ingresan en la consola de la herramienta, Figura #7. Para cerrar el *software* únicamente se debe ejecutar el comando `exit` tanto en la consola de la herramienta como en la terminal desde donde se ejecutó la apertura de *Design Vision*.

La complejidad de la configuración de la herramienta depende de los parámetros que el usuario decida utilizar, ya que en sí el *software* no necesita de mucha configuración. Los procesos más importantes son la importación de las librerías y la carga del archivo a analizar. Los siguientes pasos tienen mayor relación con el entorno que se desea crear para llevar a cabo la síntesis, con esto nos referimos a la activación de ciertas características a través de comandos, los cuales cada uno tiene una tarea específica que se detallará en las siguientes secciones.

Algunas de las tareas pueden configurarse de tal manera que el proceso demande en gran parte la capacidad y rendimiento de *Design Vision*, principalmente las tareas en las que se puede seleccionar la calidad del proceso, *low*, *medium* o *high*. Dentro de estas configuraciones se mencionarán la aplicación de restricciones al diseño, la revisión del diseño y la compilación, el cual es el paso donde se lleva a cabo la síntesis y optimización.

7.3.1. Análisis de comandos *script* original

A través de líneas de comandos se puede realizar todo el proceso de síntesis lógica, desde lo que es abrir la herramienta hasta cerrarla una vez se concluya con las tareas. Al inicio de la investigación se analizó el *script* proporcionado por los exalumnos de Ingeniería Electrónica, Ingeniero Luis Arturo Nájera y el Ingeniero Steven Rubio. Este archivo incluye los pasos de importación de librerías, la lectura de archivos en *verilog*, la revisión del diseño y la generación de los archivos '.sdc', '.ddc' y el *netlist*.

Este documento es un punto de partida funcional para proceder a ejecutar la síntesis lógica, sin embargo si se desea un proceso más elaborado se debe tomar en cuenta la estructuración de los comandos, ya que algunos llevan un orden y no respetarlo puede generar errores. Estos se pueden dar tanto en el diseño como en los archivos exportados o reportes del circuito.

A continuación se listará cada uno de los comandos cronológicamente utilizados en el *script* original con una breve descripción del mismo. En las siguientes secciones se profundizará respecto a la tarea que cumplen dependiendo de cómo se configuren.

Script

El primer comando que se utiliza sirve para indicarle a la herramienta la dirección de la carpeta en donde debe buscar los archivos que se utilizarán durante el proceso [16]. En este caso se implementa esta instrucción para ubicar la carpeta que contiene las librerías *link library* y *target library*.

```
lappend search_path
```

A través del siguiente par de instrucciones se apunta a las librerías de la tecnología que se estarán utilizando, aquí solo se indica el nombre de cada uno de los archivos, ya que la

dirección se declara con el comando anterior [16].

```
set link_library "Varname ... Varname"
set target_library "Varname"
```

El siguiente comando se utiliza para la lectura del archivo que contiene la descripción del circuito al cual se le realizará la síntesis lógica [16]. En la opción de formato se indica con el que se estará trabajando, en este caso se diseñó en *verilog*. Después de declarar el formato se procede a indicar la dirección en donde se ubica este archivo y el nombre del mismo. Si se incluye solo el nombre del archivo la herramienta no podrá hallarlo.

```
read_file -format verilog path
```

Ahora se debe revisar el diseño, específicamente la estructura con la que se describió. Esto se realiza para saber si existe alguna inconsistencia en el circuito, entre ellas puertos o pines sin conexión, celdas sin entradas o salidas, inexistencia de referencia hacia celdas entre el diseño y la librería, etc [16].

```
check_design
```

Para finalizar el proceso se hace uso del comando `compile`, el cual es el encargado de realizar la síntesis del circuito en un formato de red descrito a nivel compuerta [16]. También es el encargado de llevar a cabo en paralelo la optimización del circuito dependiendo de las restricciones de diseño previamente configuradas.

```
compile
```

Estos últimos comandos que se mencionarán cumplen con la función de almacenar el esquema del circuito después del proceso de síntesis en distintos formatos [12]. Estos archivos son útiles más adelante en el flujo de diseño. Los formatos que se emplearán en este caso son '.ddc', '.v' y '.sdc'.

```
write -format ddc -h -o
write -hierarchy -format verilog -output
write_sdc
```

Algunas de las líneas de comandos anteriores se pueden describir de una manera más completa, ya que estas poseen parámetros o características configurables que podrían mejorar el proceso de síntesis. Para ello como se mencionó a un inicio, se debe realizar una lectura detallada de los manuales referenciados para aprovechar los recursos y lograr como resultado un diseño completo. Esto influirá en las etapas posteriores.

7.3.2. Librerías

Esta es una de las secciones más importantes, ya que es la base para dar inicio al proceso de síntesis lógica. Al realizar la selección de las librerías que se usarán, se tienen que tomar en cuenta varios aspectos. Entre ellos se encuentra la tecnología a implementar, en este caso se utilizará una tecnología de 180nm, también se debe saber el voltaje con el que se alimentará el circuito, para el proyecto se eligió 3.3V.

Otro aspecto es el propósito con el que se llevará a cabo el proceso, educacional o para

fabricación. Hay que tomar también en cuenta el tipo de celdas que se necesitarán, normalmente estas son celdas estándar y celdas de entrada y salida. Y por último, si es el caso, tomar en cuenta la empresa con la que se llevará a cabo la fabricación.

Al inicio del desarrollo del proyecto se comenzó a trabajar con librerías de propósito educativo proporcionadas por el grupo de *Synopsys*. Esto se debe a que no se tenía un panorama aún muy claro del flujo de diseño que se implementaría. Sin embargo, más adelante con avances en investigación y pruebas, se realizó la búsqueda de una compañía dedicada a la fabricación de circuitos integrados. Con esta conexión universidad-fabricante se obtuvo otro grupo de librerías, pero estas ya con propósito de fabricación, de este proceso se encargaron los estudiantes predecesores en el año 2019.

Como se mencionó, las primeras librerías utilizadas fueron las de *Synospsys*, a continuación se hace referencia hacia cuáles se emplearon junto a la ruta en donde se encuentran ubicadas en el *software*. Cabe mencionar que para este caso en particular se escogió una tecnología de 90nm debido a que no había disponible otra tecnología superior en las referencias.

- *Link Library*:
  - usr/Synopsys/PDK/SAED90_EDK/SAED_EDK90nm/Digital_Standar_cell_-library/Synopsys/models/saed90nm_max.db
  - usr/Synopsys/PDK/SAED90_EDK/SAED_EDK90nm/Digital_Standar_cell_-library/Synopsys/models/saed90nm_min.db
  - usr/Synopsys/PDK/SAED90_EDK/SAED_EDK90nm/Digital_Standar_cell_-library/Synopsys/models/saed90nm_typ.db
- *Target Library*:
  - usr/Synopsys/PDK/SAED90_EDK/SAED_EDK90nm/Digital_Standar_cell_-library/Synopsys/models/saed90nm_typ.db
- *Symbol Library*:
  - usr/Synopsys/PDK/SAED90_EDK/SAED_EDK90nm_REF/references/ChipTop/ref/icons/saed90nm.sdb

Los archivos mencionados con anterioridad son librerías estándar, es decir, aquellas en donde se encuentran las celdas que utilizará el circuito. Entre ellas se encuentran compuertas, *mux*, *Flip Flops*, entre otros elementos. En este paquete no se incluyen las librerías de celdas de entradas y salidas. En el caso de las librerías proporcionadas por el fabricante, TSMC, *Taiwan Semiconductor Manufacturing Company*, se encuentran agrupadas tanto las librerías estándar como las de entradas y salidas.

La instancia hacia estas referencias se puede realizar a través de la interfaz gráfica o a través de la consola. Si se lleva a cabo el proceso en la consola se utiliza el comando `lappend search_path` junto a su ubicación y luego se incluyen con el comando `set`. Cabe mencionar que todas deben estar ubicadas en un mismo directorio, caso contrario la herramienta no encontrará el archivo y generará un error. Si el proceso se realiza por el primer método entonces se deben seguir los siguientes pasos:

1. Dirigirse a File > Setup

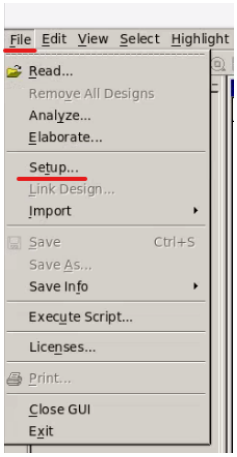


Figura 8: Primer paso para incluir las librerías

2. Configuración de librerías: Una vez ingresado a *Setup*, se mostrará la interfaz en donde se configuran cada una de las librerías, la incorporación de *Symbol Library* no es completamente necesaria.

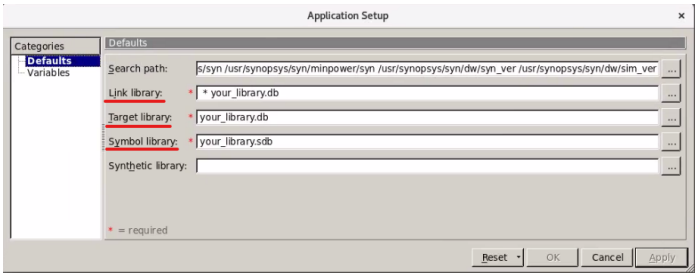


Figura 9: Interfaz para agregar los archivos de librerías

3. Importación de archivos: En este paso primero se deben eliminar los archivos que ya existan en cada una de las librerías con la opción de *Delete* en la interfaz. Luego a través de la opción *Add* se pueden agregar las instancias a los archivos que se necesiten en cada uno de los casos.

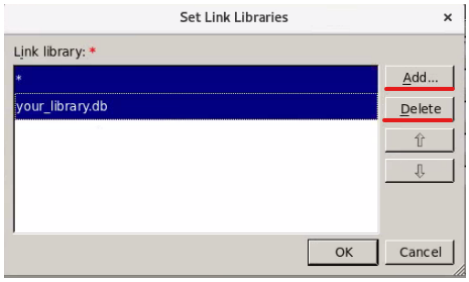


Figura 10: Importación de archivos

4. El último paso es aplicar los cambios y con eso ya se tienen establecidas las librerías que se utilizarán.

Como se puede observar es un proceso algo extenso. Esto se puede reducir solamente a tres líneas de comandos mencionadas anteriormente en el apartado **'Análisis de comandos *script* original'**.

Respecto a las librerías de entradas y salidas estas deben pertenecer a la misma tecnología que las anteriores, así como utilizar el mismo voltaje de alimentación. Una discrepancia entre estos dos grupos puede provocar, en el peor de los casos, que no funcione el circuito integrado o que bien las simulaciones y pruebas que se realicen tengan una gran cantidad de errores y no se lleguen a concluir los procesos. El archivo de esta librería debe ser incluido en la configuración de *Link Library*.

La librería de pines de entrada y salida hace una descripción de las terminales que serán utilizadas para las entradas y salidas de las señales del circuito. Esta configuración es necesaria porque sin ella existirán problemas en las etapas posteriores, y el circuito integrado no estará cumpliendo con su función. En la sección de **'Diseño de circuito con pines de entrada y salida'** se explicará de manera más detallada la implementación e incorporación de esta librería. Así mismo el uso y configuración correcta de estos pines.

7.3.3. Importación de archivos

La importación de archivos es el paso siguiente a la configuración de librerías, aquí es donde se debe incluir el archivo que contiene la descripción del circuito con el que se trabajará. Por defecto los formatos de archivos aceptados por la herramienta de *Design Vision* son *Verilog*, *SystemVerilog* y *VHDL*. Si se está trabajando con estos tipos de archivos entonces al importarlos se debe de realizar de la siguiente manera `read_file path`. El *software* admite también otro tipo de archivos [12] los cuales se listan a continuación:

- ddc - *Synopsys internal database format*
- db - (Librerías)
- equation - *Synopsys equation format*
- pla - *Berkeley PLA format*
- st - *Synopsys State Table format*

Para poder leer estas extensiones se debe agregar la configuración de un parámetro al comando utilizado anteriormente, específicamente la opción de `-format format_name`. Por ejemplo si se desea importar un archivo 'ddc', entonces la línea de comando tendrá el siguiente formato.

```
read_file -format ddc path
```

Es importante que en este paso la estructura del circuito esté bien definida, ya que la herramienta solo muestra el mensaje de error si el comando no fue ingresado de la forma correcta o si el formato no es aceptado. Si existe alguna falla o incongruencia en la definición del circuito esto se podrá observar al momento en que se realice la revisión de diseño. Para

evitar que el proceso se vuelva tedioso es recomendable que previo a la importación del archivo se realicen pruebas del mismo con un *testbench*.

El proceso de importación de archivo a través de la interfaz gráfica se realiza de la siguiente manera. Primero se debe dirigir a la pestaña *File* y seleccionar la opción de *Analyze* tal como se muestra en la Figura #11. Seguido se mostrará una ventana en donde se realiza la respectiva carga del archivo, esto se muestra en la Figura #12. A través del botón *Add* se debe dirigir a la ruta donde se encuentre el archivo y seleccionarlo, seguido en la opción *Format* se debe elegir con qué formato se trabajó la descripción, Figura #13. Por último es importante tomar en cuenta el nombre de la librería en donde se almacenó, en la opción *Work Library* y posterior aplicar los cambios.

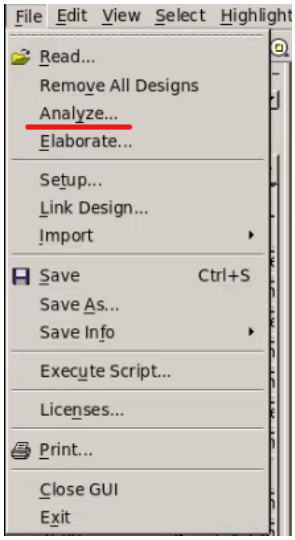


Figura 11: Configuración para cargar el archivo que describe el circuito

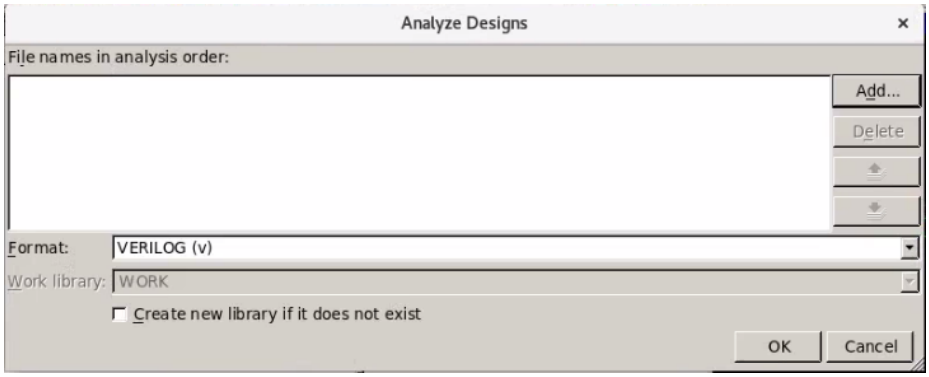


Figura 12: Ventana para cargar el archivo

Figura 13: Formatos disponibles

Con estos pasos se lleva a cabo la incorporación del archivo a la librería de trabajo, sin embargo hace falta realizar la carga del mismo a la herramienta. Este proceso se realiza de la siguiente forma, primero se dirigirá a la pestaña *File* y se debe seleccionar la opción *Elaborate*, como se muestra en la Figura #14. Una vez ahí se mostrará una ventana, Figura #15, en donde se debe primero ingresar el nombre de la librería de trabajo en la opción *Library* y seguido seleccionar en *Design* el nombre del archivo que se incorporó, este se debe buscar entre el listado de todos los archivos que se han almacenado en esa librería.

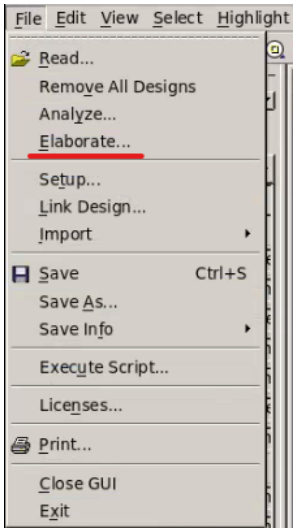


Figura 14: Incorporación del archivo para mostrar en la herramienta

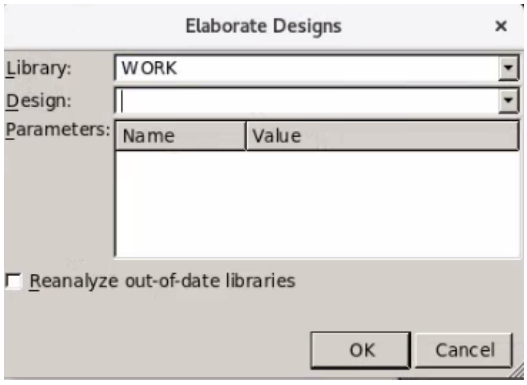


Figura 15: Último paso para completar la carga del circuito a *Design Vision*

Una vez configurado el diseño, se mostrará la vista del mismo en la interfaz gráfica de la herramienta. Para hacer esto se debe ir a *Logical Hierarchy* seleccionar el circuito y a través de *click* derecho seleccionar *Schematic View*, esto se muestra en la Figura #16. La Figura #17 y la Figura #18, muestran el circuito de una compuerta Not a nivel esquemático desde dos niveles de jerarquía distintos.

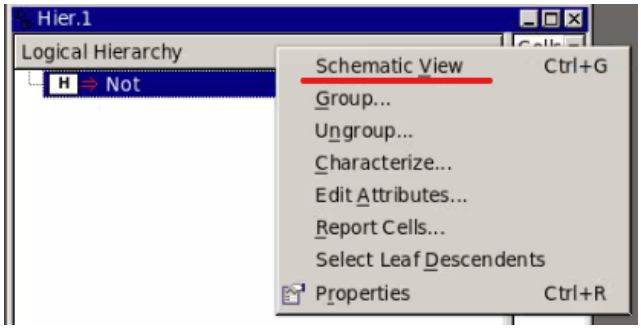


Figura 16: Selección para tener una vista del esquemático

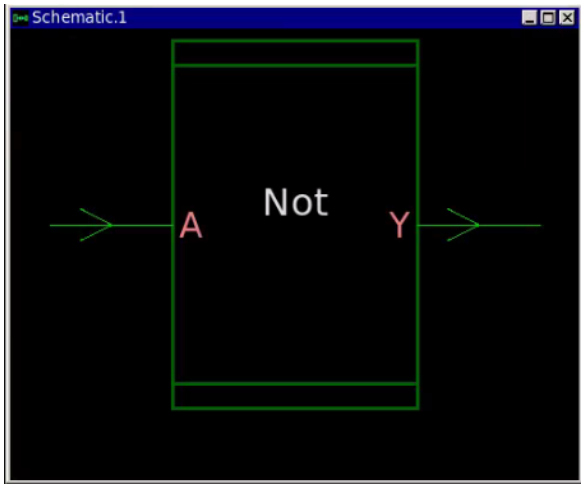


Figura 17: Circuito *Not* en el nivel jerárquico más alto, compuerta

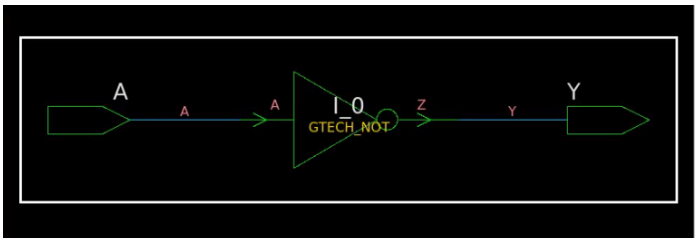


Figura 18: Circuito interno de la compuerta *Not*

7.3.4. Configuración de restricciones de diseño

La configuración de restricciones, *constraints*, harán que se pueda tener un mejor control en el proceso de síntesis del diseño. Los objetivos de estas restricciones es poder optimizar ciertos parámetros del circuito, como el reloj, retrasos en las señales de entrada y salida y la minimización del área de diseño. Hay que tomar en cuenta que estas configuraciones se deben hacer a través de valores realistas [10][17].

El proceso de configuración puede extenderse a través de una larga lista de comandos,

por lo que la mayoría de diseñadores optan por agruparlas en un nuevo *script*. Con esto luego se crea la instancia hacia su ubicación para que la herramienta puede incorporarlos en el diseño. Sin embargo la declaración de parámetros se puede llevar a cabo en el mismo archivo donde se encuentra configurado el proceso de síntesis.

De igual manera a través de la interfaz gráfica se pueden configurar, solo que la cantidad de restricciones es menor y menos específicas [10]. Los parámetros que se configuran en esta parte aseguran que el circuito y posteriormente su síntesis cumpla con dichas especificaciones. Estas son reglas que no se pueden violar [10]. Para llevar a cabo la configuración a través de la interfaz gráfica se deben seguir los siguientes pasos, Figura #19 y Figura #20:

- Dirigirse a *Attributes*
- Seleccionar *Optimization Constraints*
- Configurar las restricciones según la opción u opciones que se quieran

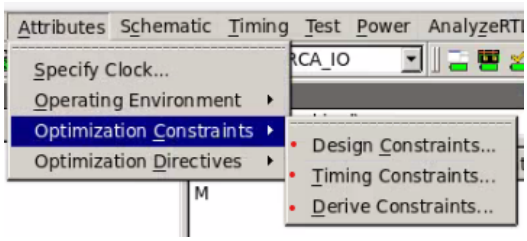


Figura 19: Pasos a seguir para configurar las *constraints* desde la interfaz gráfica

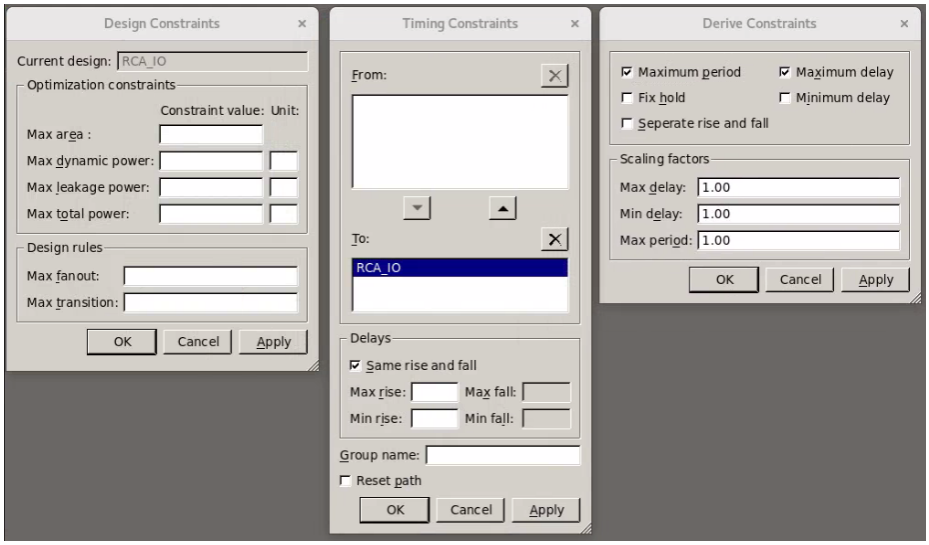


Figura 20: Opciones que se pueden configurar en la sección de restricciones desde la interfaz gráfica

Como se observa en la Figura #20 esas son algunas de las opciones que el usuario puede configurar, este proceso también puede llevarse a través de líneas de comando como se describirá a continuación.

Timing Constraints

Para poder configurar esta restricción se debe tomar en cuenta que existen rutas síncronas y asíncronas en el diseño. Las rutas síncronas se pueden restringir especificando un reloj. Para ello se ejecuta el comando `create_clock`. Una vez se tenga creado el reloj se recomienda que también se especifique el retraso en las entradas y salidas de los puertos. Esto se lleva a cabo con ayuda de los siguientes comandos, `set_input_delay` y `set_output_delay` [17].

Para las rutas asíncronas se considera el proceso de restricciones a través de configurar el tiempo de retraso máximo y mínimo de punto a punto. Esto se realiza a través de los comandos `set_max_delay` y `set_min_delay` [17].

A continuación se realizará una descripción de cada una de las configuraciones mencionadas con anterioridad. Si se desea obtener información más detallada se sugiere referenciarse al manual de comandos de la herramienta de *Design Vision* [12].

El comando `create_clock` crea un reloj en el circuito, si no se especifica la fuente para el mismo entonces se considera un reloj virtual. Debido a que es virtual tiene una sincronización ideal y no posee retraso alguno en la señal. Para llevar a cabo su configuración existe una variedad de parámetros [12].

Comenzando nos encontramos con el parámetro `-name clock_name`, esta opción especifica el nombre del reloj. Si se tiene una reloj ya especificado en el circuito entonces a este se le debe asignar el mismo nombre. Caso contrario se debe asignar uno para poder crear el reloj de manera virtual que no esté asociado a algún pin o puerto. El comando tendría la siguiente estructura final.

```
create_clock -name "CLK"
```

El siguiente parámetro es `add`, el uso de este se da cuando se implementan múltiples relojes en un mismo pin o puerto. Si se hace uso de esta configuración entonces se debe también utilizar el parámetro `-name` para nombrar a cada uno de ellos.

```
create_clock -name "CLK" -add
```

El siguiente parámetro es `source_objects`, el cual especifica un listado de pines o puertos en donde se aplicará el reloj creado. Si no se desea hacer uso de esta opción entonces se sugiere utilizar el primer parámetro, `-name`, sin asociarlo a ningún pin o puerto.

```
create_clock -name "CLK" {clk1 clk2 clk3}
```

Ahora procede la configuración de la señal del reloj. Para comenzar con ello se hace uso del parámetro `-period period_value`, con el se especifica el periodo con el que se quiera estar trabajando. Por ejemplo, si se desea un periodo de 50 entonces el comando tendría la siguiente estructura.

```
create_clock -name CLK -period 50
```

Terminando con la configuración de la señal se hace uso del parámetro `-waveform edge_list`. A través de esta instrucción se puede especificar el tiempo donde se de el flanco de subida y el flanco de bajada durante un periodo. Puede incluir varios tiempos si se desea más de un pulso en un periodo y la descripción del mismo se realiza por medio de una lista.

Si este parámetro no se configura, entonces para crear la forma de la señal la herramienta toma el flanco de subida en '0' y para calcular el tiempo donde se da el flanco de bajada toma el valor del periodo que se configuró previamente y lo divide dentro de dos.

A continuación se muestra un ejemplo especificando un periodo de 50 y dos pulsos en un solo periodo. El primer pulso con flanco de subida en 5 y flanco de bajada en 25, el segundo pulso con flanco de subida en 30 y flanco de bajada en 50. Esta configuración tendría la siguiente forma:

```
create_clock -name CLK -period 50 {5 25 30 50}
```

Con este parámetro ya se puede concluir la creación de uno o varios relojes. Ahora se procede a la configuración de los retrasos en las señales de entrada y salida en pines y puertos.

Al igual que el comando descrito con anterioridad, estos dos próximos también poseen varios parámetros para llevar a cabo su configuración. Se recalca que comparten los mismo parámetros solo que la configuración será distinta dependiendo si se desea retraso de entrada o de salida. Debido a esto se muestra a continuación la información de ambos agrupada para cada una de las configuraciones.

El primer parámetro es `delay_value`, con este se especifica el valor que se le dará de retraso a la señal. Indica el tiempo que la señal se mantiene estable en los puertos de entrada y salida antes y después de que se de un flanco en el reloj específicamente [12].

```
set_input_delay 2.5
```

```
set_output_delay 2.0
```

El siguiente parámetro es `-reference_pin pin_port_name`. Aquí se especifica el pin o puerto donde esté ubicado el reloj al que se le agregará el retraso, si no se utiliza el parámetro `-clock` entonces el retraso se aplica a los múltiples relojes que se encuentren en un mismo pin asignado.

```
set_input_delay -reference_pin in
```

```
set_output_delay -reference_pin out
```

Ahora el parámetro a configurar es `-clock clock_name`. Con este se especifica el reloj al que se relacionará el *delay*. La configuración puede ser a un solo objeto o a una lista de objetos. A continuación se muestra cómo se observa el parámetro ya declarado.

```
set_input_delay -clock clk1
```

```
set_output_delay -clock {clk2 clk3}
```

El parámetro `clock_fall` en este caso se especifica solo si el *delay* será en función al flanco de bajada. Por defecto ambos trabajan respecto al flanco de subida.

```
set_input_delay -clock_fall
```

El siguiente parámetro es `-level_sensitive`, si este se configura entonces se tratará a la ruta de la fuente del retraso como un *latch*. Si no se especifica esta configuración por

defecto la herramienta lo tratará como un *Flip-Flop*. Este parámetro se utiliza solo cuando se quiere una configuración más específica.

Si no se desea incluir la latencia de la señal de reloj o de la fuente del reloj al retraso de entrada o salida, se deben configurar los siguientes parámetros: `-network_latency_included` y `-source_latency_included`. Si no se utilizan estas configuraciones entonces por defecto la latencia del reloj se agrega a los valores de *delays* configurados.

Los parámetros `rise` y `fall` se utilizan para indicar a qué tipo de transición se referirá el valor del *delay*. Si no se hace uso de ellos, entonces se asume que el *delay* de ambos flancos será igual.

Otro grupo de parámetros para configurar son `-max` y `-min`, a través de ellos se puede especificar si el retardo configurado es un valor máximo o mínimo. A continuación se muestra el ejemplo para un *delay* de entrada con valores máximos y mínimos.

```
set_input_delay 3.0 -max
```

```
set_input_delay 0.1 -min
```

Así como se pueden agregar varios relojes a un solo pin o puerto, también se pueden configurar varios retrasos para realizar pruebas a un mismo pin. Al configurar el parámetro `-add_delay` se está indicando que se desea configurar un nuevo *delay* y que no se causará que información antigua sea eliminada. La estructura del comando se ve de la siguiente manera.

```
set_output_delay 1.1 -min -clock clk1 -add_delay {out1}
set_output_delay 0.2 -min -clock clk1 -add_delay {out1}
```

Por último si se quiere aplicar un mismo valor de retraso de entrada o salida a un grupo de pines o puertos se puede llevar a cabo a través de la siguiente configuración. El parámetro `port_pin_list` especifica el listado de los nombres de los pines o puertos a asignar. Dicha asignación se realiza de la siguiente forma:

```
set_input_delay 2.5 {in1 in2 in3}
```

```
set_output_delay 2.0 {out1 out2 out3}
```

Los últimas dos configuraciones para llevar a cabo las restricciones para el reloj se mencionan a continuación. De igual manera que los dos comandos anteriores, `set_max_delay` y `set_min_delay`, se describirán agrupados, esto se debe a que poseen los mismos parámetros configurables.

Para especificar el valor máximo y mínimo del *delay* de una ruta se utiliza el parámetro `delay_value`. Si el punto de inicio o punto final de la ruta es un pin y este posee un retraso, entonces ese valor es agregado al retraso de la ruta que se asignará manualmente. La configuración de este parámetro posee la siguiente estructura.

```
set_max_delay 12.5
```

```
set_min_delay 4.0
```

Si se quiere restringir el punto final de la ruta respecto a los retrasos de subida o bajada, entonces se utiliza el parámetro `-rise` o `-fall`. Si no se especifica esta configuración, se

tomará por defecto que ambos casos se restringirán.

La configuración de los puntos de inicios de las rutas se puede llevar a cabo a través del parámetro `-from from_list`. En este listado se declaran puertos, pines, relojes o celdas, si se lista un reloj, entonces la restricción del *delay* máximo o mínimo se aplicará a todos las salidas de ese reloj. El listado se referencia a través de llaves o comillas, este listado no puede incluir puertos de salida únicamente de entrada.

De la misma manera se pueden declarar los puntos finales de las rutas a través de `-to to_list`. Hay que tomar en cuenta que aquí no se pueden incluir puertos de entrada. Utilizando la lista de puntos iniciales y puntos finales el comando tendría la siguiente estructura:

```
set_max_delay 16.0 -from {I1 I2} -to {O1 O2}
```

Los parámetros `-rise_from rise_from_list` y `-fall_from fall_from_list` funcionan de la misma manera que `from`, solo que en este caso se listará aquellos puntos de inicio donde la ruta tenga un flanco positivo o negativo respectivamente. No se puede hacer uso de estos tres parámetros juntos en una misma configuración de *delay*, solamente se toma en cuenta uno.

También se pueden declarar puntos por donde la ruta pase, esto se realiza a través de un listado en el parámetro `-through through_list`, en este listado se pueden nombrar varios puntos de paso. El listado puede incluir puertos, pines o celdas que se encuentren en el diseño. Por ejemplo, si se quiere configurar que se tenga un retraso mínimo en todas las rutas que pasen por el pin `n3` y que lleguen a la salida `S`, la línea de comando tendría la siguiente estructura.

```
set_mix_delay 5.3 -through n3 -to S
```

Los parámetros `-rise_through rise_through_list` y `-fall_through fall_through_list` se puede configurar como un listado de objetos. En ellos solo son tomadas las señales que tengan un flanco de subida o de bajada respectivamente. Funcionan de la misma forma que el parámetro anterior.

Al igual que el parámetro `-to`, se pueden configurar los puntos finales de otra manera, ya que si se quiere que los puntos tengan un flanco de subida o de bajada se puede realizar esto a través de `-rise_to rise_to_list` y `-fall_to fall_to_list`. Solamente se puede hacer uso de uno de estos tres parámetros en una declaración, no se pueden configurar juntos.

Si se quiere realizar una nueva configuración para ciertas rutas en específico, se pueden eliminar las configuraciones previas que tenían dichas rutas. Esto se lleva a cabo con el parámetro `-reset_path`, el cual se aplica únicamente a las rutas que hayan sido configuradas, las demás no se verán afectadas.

No es necesario que se lleve a cabo la configuración de todos los parámetros, esto dependerá de qué tan elaborado se desee el proceso.

Area Constraints

Design Vision optimiza el diseño para reducir el área siempre que sea posible, a menos que la optimización afecte negativamente al tiempo y al rendimiento energético [17]. Cabe mencionar que la restricción de área no se refiere al área física que ocupa el circuito, sino se refiere al espacio máximo que se quiere ocupar con las celdas o compuertas. El usuario puede especificarlo a través del comando `set_max_area`. Este proceso se lleva a cabo después de que la optimización del reloj ya se ha realizado [17].

Si se quiere una explicación a más detalle, se puede leer el apartado en el manual de comandos de *Design Vision* del comando a describir.

El valor que se quiera dar al atributo `set_max_area` debe indicarse seguido a la declaración del comando. Al configurar un área de '0' la herramienta realiza la optimización del tamaño a las dimensiones más pequeñas que se puedan. Este comando está relacionado al costo de fabricación del diseño, por lo que configurarlo puede traer resultados positivos.

```
set_max_area 0
```

7.3.5. Revisión de diseño y compilación

Al llegar a esta etapa lo que se realizará es la revisión del estado del diseño, es decir que no existan inconsistencias. Se revisará que todos los pines estén conectados, en caso contrario se obtiene un mensaje de error o *warnings* por parte de la herramienta. Los mensajes de advertencias no deben ser necesariamente arregladas, solo si es necesario o causan problemas serios en el diseño. Esta verificación se realiza antes de la configuración de restricciones [16].

Para poder realizar la tarea de revisión se utiliza el comando `check_design`, descrito en el aparto de **'Análisis de comandos *script* original'**. Existen varios parámetros que pueden hacer el proceso más preciso, como la revisión por separado de pines, puertos, *nets*, celdas y niveles de jerarquía [12]. Pero en este trabajo se decidió solo por utilizar el comando sin ninguna configuración extra. Esto se debe a que la ejecución del proceso se desea completo y no se ve necesario llevar a cabo la verificación por partes.

Si no se resuelven los errores en la etapa de verificación, entonces no se podrá seguir con el siguiente paso que es el proceso de compilación. Este es el paso donde se lleva a cabo el proceso de síntesis lógica y optimización, esta última se elabora solo si se han declarado algún tipo de restricciones [12].

Cuando se ejecuta la compilación la herramienta se encarga de sintetizar el circuito fuente con base a la implementación de celdas que se encuentran en las librerías configuradas. En este proceso se combinan varias celdas para cumplir con los requisitos de diseño [17]. El comando que se utiliza para este paso es `compile`. La síntesis se lleva a cabo a un nivel alto de descripción a nivel compuerta.

Previo a sintetizar el diseño se recomienda lo siguiente para obtener resultados satisfactorios [17]:

- Asegurarse de que la descripción del circuito sea la mejor, seleccionando adecuada-

mente la estructura que este tendrá. Una buena descripción asegura calidad en la optimización y compilación.

- Definir cada una de las restricciones de forma precisa pero sin sobre restringir el diseño.
- Realizar al inicio la selección de las librerías que se acoplen a las características que se desean para el circuito a fabricar.
- Identificar y conocer bien el circuito que se sintetizará.
- Si es necesario, crear las instancias pertinentes de relojes.
- Definir el tipo de compilación, esto dependerá muchas veces de la complejidad que presente el circuito y de la calidad de resultados que se deseen.

La síntesis se puede efectuar utilizando solamente el comando `compile`, sin embargo a este se le pueden configurar diferentes características o parámetros. A continuación se describirá cada uno de ellos, de igual forma se puede apoyar del manual de comandos de la herramienta de *Design Vision* para tener un panorama más claro sobre su uso.

El tipo de esfuerzo que se quiere utilizar durante la fase de mapeo se especifica a través del parámetro `-map_effort medium|high`. Solamente se puede seleccionar una de las dos opciones y por defecto la configuración es media.

```
compile -map_effort high
```

El parámetro anterior es uno de los más importantes a tomar en cuenta. Otro de ellos es `-area_effort none|medium|high`. Con este se especifica el esfuerzo que se desea invertir para llevar a cabo la fase de reducción de área con base a la restricción que se configuró. Si no se configura esta opción entonces el esfuerzo de la herramienta será el mismo que se especificó en `-map_effort`.

```
compile -map_effort medium -area_effort high
```

Aquí también puede especificarse el esfuerzo que se quiere para la fase de optimización de energía. Esto se aplica únicamente si en las restricciones de diseño se incluyó alguna configuración de potencia. El parámetro que realiza esto es `-power_effort none|low|medium|high`.

```
compile -map_effort medium -area_effort high -power_effort low
```

Los siguientes parámetros que se mencionan son de uso secundario. El proceso se puede realizar satisfactoriamente utilizando solo los tres mencionados con anterioridad.

El parámetro `-no_map` lleva a cabo la optimización pero sin ejecutar un mapeo lógico, ahora si se quiere realizar un mapeo incremental entonces se debe hacer uso del parámetro `-incremental_mapping`. Si existe alguna porción del diseño que ya se encuentre mapeada entonces la herramienta la deja de un lado para no realizarle la optimización lógica de nuevo.

Existe un parámetro con el cual se especifica a la herramienta si se desea que los elementos secuenciales en el diseño final coincidan exactamente con las descripciones que se dieron en el *HDL* [12]. El parámetro es `-exact_map`.

En el diseño que se obtiene al final la descripción se da en formato de jerarquía, sin embargo se puede cambiar el formato y dejar la descripción en un solo nivel a través de `-ungroup_all`. Esto se aplica a todo el diseño, exceptuando los objetos a los que se le aplico el atributo de no tocar.

La des agrupación de jerarquías también se puede llevar a cabo de manera automática, afectando específicamente el área y el retraso. El parámetro que se debe utilizar es `-auto_ungroup area|delay`. Si se configura la opción de área, esto se aplica a pequeños arreglos para mejorar el área y esto afectará de manera insignificante en el tiempo de ejecución. Al contrario, si opta por la configuración de `delay` la estrategia se emplea para mejorar el tiempo en general del diseño.

El proceso de optimización se lleva individualmente para cada uno de los elementos, si la opción `-boundary_optimization` se habilita, entonces se fusionan a todos los componentes y se ejecuta una optimización en conjunto. Esto por una parte alargará el tiempo de compilación, sin embargo también puede conllevar a un mejor rendimiento del diseño.

7.3.6. Archivos y reportes generados

Ahora nos encontramos en el último paso el cual es la generación de archivos y reportes. Se comenzará con la descripción del tipo de archivos que se necesitan almacenar. Estos archivos incluyen la descripción del circuito sintetizado a nivel compuerta en formato *netlist*. También se incluyen las restricciones del diseño, formato 'sdc' y la información general del circuito en formato 'ddc'. Para llevar a cabo esto se utilizan los comandos `write_file` y `write_sdc`.

En versiones anteriores el comando `write_file` se representaba solo por `write`, esta segunda opción sigue teniendo hoy en día compatibilidad con la herramienta, sin embargo se sugiere hacer uso del comando nuevo. A continuación se mencionará la descripción de algunos de los parámetros para los dos comandos a utilizar.

Comenzando con `write_file`, esta instrucción almacena el diseño en un archivo físico. Para especificar el formato de salida se utilizará la opción `-format output_format` donde se se permiten los formatos 'ddc', 'verilog', 'svsim' y 'vhdl'. Si se elige un formato 'svsim' solo se estará guardando el listado de conexiones del circuito. Si se quiere guardar el diseño a nivel compuerta se utiliza un formato 'verilog'.

```
write_file -format verilog
```

Debido a que la estructura que se crea al sintetizar el diseño es jerárquica, si se desea almacenar todo el diseño de esta forma entonces se debe configurar la opción `hierarchy`. El incluirla es necesario cuando se quiera crear un formato 'verilolg' y 'ddc' [12].

Si no se desea guardar el diseño jerárquico creado durante la optimización del diseño, entonces se debe indicar a través del parámetro `-no_implicit` esta configuración. Por defecto la estructura con la que se almacena es jerarquía aún así no se especifique con `-hierarchy` [12].

Para especificar el nombre del archivo a guardar, se debe realizar a través del parámetro

`-output output_file_name`. Aquí se debe especificar la ruta donde se almacenará y el nombre del archivo junto a su extensión. Si el formato que se quiere usar es 'ddc' y se habilita la opción `hierarchy`, es necesario que se configure esta opción.

```
write_file -format verilog -output Not_syn.v
```

Al momento de generar un archivo 'ddc', ahí se almacenan todos los escenarios a los que se aplican las restricciones. Con el parámetro `-scenarios scenario_list` puede seleccionarse únicamente los escenarios que se deseen tomar en cuenta para agregar al archivo de salida.

Al tener declarado puertos de VDD y VSS se puede llevar a cabo la inclusión de la información de estos en el archivo obtenido en formato *verilog*. Esto se realiza a través de configurar la opción `-pg`.

Continuando con el comando `write_sdc`, el parámetro más importante que se debe tomar en cuenta es el nombre. Este se configura con la opción `file_name`, en este se debe incluir tanto la ruta donde se quiere guardar el archivo como el nombre con la extensión 'sdc'.

```
write_sdc /Escritorio/Not.sdc
```

Así mismo como pueden almacenarse archivos que describen el circuito a nivel lógico, también pueden generarse reportes que se pueden utilizar para observar los resultados de la síntesis lógica. Estos son una base para analizar y arreglar posibles errores en el diseño [17]. Existe una gran variedad de reportes que se pueden generar,pero los más sobresalientes e importantes son los de área, tiempo y restricciones.

La generación de reportes es recomendable que se lleve a cabo después de la etapa de compilación, ya que si se generan antes de ese paso los valores pueden no ser reales. Generarlos seguido de la síntesis lógica asegura los resultados, ya que en la fase de compilación es cuando se realizan las optimizaciones.

La mayoría de los reportes se pueden obtener a través de la interfaz gráfica. Esto se realiza a través de la pestaña *Design*. Aquí aparecerá un listado con los distintos reportes como se puede observar en la Figura #21. En la Figura #22 se muestra un ejemplo de las ventanas desde donde se puede llevar a cabo la configuración para obtener un reporte de área, reloj o diseño. Todos los reportes se pueden almacenar en un archivo de texto.

Figura 21: Listado de reportes disponibles desde la interfaz gráfica

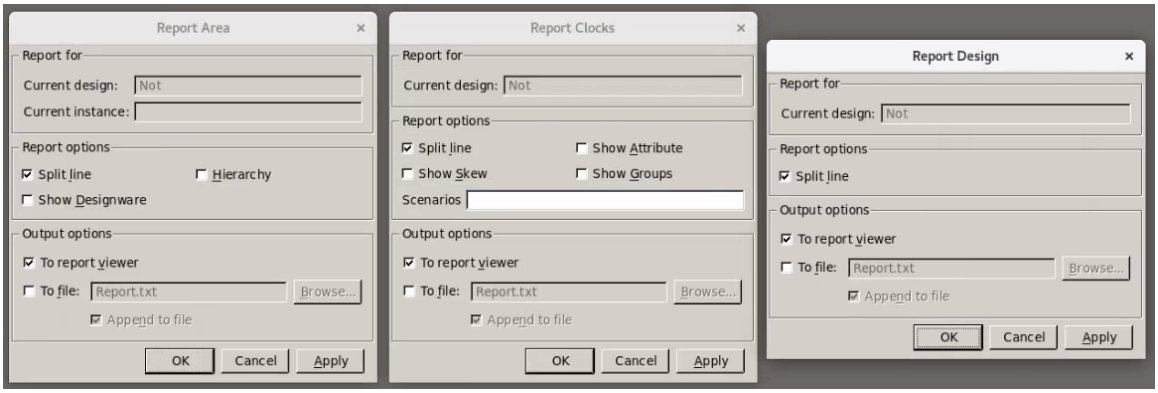


Figura 22: Ventanas para generar reporte de área, reloj y diseño

Sin embargo los reportes también se pueden configurar y crear a través de líneas de comando y parámetros. La siguiente explicación para llevar a cabo estos procesos se soporta del manual de comandos de *Design Vision*.

Para generar el reporte de área se utiliza el comando `report_area`, al hacer uso de este el texto se muestra en la consola de la interfaz como se puede observar en la Figura #23. Si se quiere más detalle se pueden habilitar los parámetro `-physical`, `-hierarchy` y `-designware`. El primero muestra el reporte del tamaño del núcleo, *core*, y la relación de aspecto que tiene el diseño [12]. El segundo proporciona información del área para cada una de las celdas de la jerarquía en específico, y por último `designware` ejecuta un reporte de

área de las celdas sintéticas si estas existen en el diseño.

```
design_vision> report_area

****************************************
Report : area
Design : Not
Version: O-2018.06-SP5
Date   : Thu Sep 17 17:44:51 2020
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:


Number of ports:                          2
Number of nets:                           2
Number of cells:                          1
Number of combinational cells:            1
Number of sequential cells:               0
Number of macros/black boxes:             0
Number of buf/inv:                        1
Number of references:                     1

Combinational area:                6.585600
Buf/Inv area:                      6.585600
Noncombinational area:             0.000000
Macro/Black Box area:              0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                   6.585600
Total area:                 undefined
1
```

Figura 23: Reporte de área para una compuerta Not

Un reporte de diseño muestra información exclusiva de los atributos del mismo [12]. El comando ha utilizar será `report_design`. En la Figura #24 se observa que el reporte muestra la librería que fue utilizada, los tipos de *Flip-Flops* y *Latchs* que se implementaron, también se muestran las condiciones de operación así como las interconexiones. Dependiendo del tamaño del diseño el reporte será más detallado.

```
****************************************
Report : design
Design : Not
Version: O-2018.06-SP5
Date   : Thu Sep 17 20:40:20 2020
****************************************

Design allows ideal nets on clock nets.

Library(s) Used:


Local Link Library:


Flip-Flop Types:

    No flip-flop types specified.

Latch Types:

    No latch types specified.

Operating Conditions:


    Operating Condition Name : NCCOM
    Library :
    Process :   1.00
    Temperature :  25.00
    Voltage :   1.80
    Interconnect Model : balanced_tree

Wire Loading Model:

    Selected automatically from the total cell area.

Name           :   ZeroWireload
Location       :
Resistance     :   1e-05
Capacitance    :   1
Area           :   0
Slope          :   1e-07
Fanout   Length   Points Average Cap Std Deviation
-----------------------------------------------------
     1     0.00
```

Figura 24: Ejemplo de reporte de Diseño

Otro de los reportes que se pueden obtener es un reporte de jerarquía, en este se muestra el nombre de la celdas utilizadas en la síntesis del diseño actual y la librería a la que pertenecen. El proceso se lleva a cabo a través del comando `report_hierarchy`, en la Figura #25 se observa el ejemplo de reporte del circuito de una compuerta *Not*.

```
design_vision> report_hierarchy

****************************************
Report : hierarchy
Design : Not
Version: O-2018.06-SP5
Date   : Thu Sep 17 20:55:32 2020
****************************************

Not
    CKND0BWP7T         tcb018gbwp7ttc
```

Figura 25: Reporte de jerarquía

Por último también se puede obtener un reporte de potencia, en donde se desglosa la información del voltaje de operación global, capacitancia, potencia interna de celdas y de redes. Además si se están utilizando elementos distintos a las celdas estándar también se reportará la información de las mismas, por ejemplo los *pads* de entrada y salida. El reporte se obtiene con ayuda de `report_power`, Figura #26.

```
---------------------------------------------
Not                   ZeroWireload


Global Operating Voltage = 1.8
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  = 705.3370 nW  (100%)
  Net Switching Power  =   0.0000 mW    (0%)
                         ---------
Total Dynamic Power    = 705.3370 nW  (100%)

Cell Leakage Power     =  18.8771 pW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000              0.0000             0.0000  (   0.00%)
memory             0.0000           0.0000              0.0000             0.0000  (   0.00%)
black_box          0.0000           0.0000              0.0000             0.0000  (   0.00%)
clock_network      0.0000           0.0000              0.0000             0.0000  (   0.00%)
register           0.0000           0.0000              0.0000             0.0000  (   0.00%)
sequential         0.0000           0.0000              0.0000             0.0000  (   0.00%)
combinational  7.0534e-04           0.0000          1.8877e-02         7.0536e-04  ( 100.00%)
--------------------------------------------------------------------------------------------------
Total          7.0534e-04 mW        0.0000 mW       1.8877e-02 nW      7.0536e-04 mW
```

Figura 26: Reporte de potencia

7.4. Diseño de circuito con pines de entrada y salida

En las secciones anteriores en este capítulo se describió el proceso a seguir para poder llevar a cabo la síntesis lógica. Sin embargo, el circuito que se sintetiza no posee pines de entrada y salida físicos aún. En esta sección se brindará la información necesaria para llevar a cabo la configuración de los pines de entrada y salida, también se mostrará de manera gráfica este proceso y el orden que se debe seguir para implementarlo.

Para incluir estos pines lo primero que se debe hacer es seleccionar la librería que se estará utilizando. Esto dependerá de la función que se quiera para ellos así como el voltaje de alimentación que se desee para ellos. Este último punto es importante ya que el voltaje de trabajo de los pines debe ser el mismo que el de las celdas estándar. Entre los documentos de referencia proporcionados por el fabricante se incluyen PDF's que brindan la información de guía para poder elegir de la mejor manera esta librería. En el presente proyecto se estará utilizando la librería `tpd018nvtc.db` brindada por la empresa TSMC.

Para hacer uso de la librería se debe conocer cómo se debe llevar a cabo la instancia de los pines y cuáles se utilizarán. El documento de texto que describe los pines posee información como lo son las tablas de verdad, los voltaje a los cuales trabajan, entre otros aspectos eléctricos. También cabe mencionar la existencia de un archivo descrito en 'verilog', `tpd018nvtc.v`, el cual contiene la descripción interna de estos pines. Este archivo se utiliza como base para llevar a cabo la instancia de los mismos ya que el orden en que se declaran las variables en este caso sí importa.

Previo a declarar los pines de entrada y salida se debe sintetizar el circuito que se estará utilizando, es decir, obtener un *netlist*. En la Figura #27 se puede observar la descripción de una compuerta *Not* después del proceso de síntesis lógica.

```
module Not ( A, Y );
  input A;
  output Y;


  CKND0BWP7T U1 ( .I(A), .ZN(Y) );
endmodule
```

Figura 27: *Netlist* compuerta *Not* sintetizada

Al contar con este archivo se debe crear ahora uno nuevo siempre con extensión de 'verilog', en el cual se realizará la declaración de los pines. El proceso de declaración se realizará a través de un módulo, en el cual se describirá al circuito tal y como se conectaría de manera física. Esto quiere decir que se declararan tanto las entradas y salidas del mismo pero también se declararan la conexión al exterior del circuito integrado a través de *wires*. En las siguientes líneas se muestra esta configuración con el ejemplo de una compuerta Not.

```
1 module Not_IO (A, Y);
2     input A;
3     output Y;
4
5     wire A_w, Y_w;
```

Como se observa en las líneas anteriores solo se muestra el inicio del módulo. Posterior a esta configuración se procede a realizar la instancia al circuito previamente sintetizado. Para crear un enlace primero se debe apuntar al nombre del módulo sintetizado seguido del nombre con el cual se hará referencia al mismo. En esta configuración se utilizan como variables de entrada y salida a los *wires* ya declarados, en este caso `A_w` y `Y_w`.

```
1     Not Compuerta (A_w, Y_w);
```

Una vez realizado lo anterior se puede comenzar a configurar los pines. Para esto se seleccionó el pin `PDDW0204SCDG` perteneciente a la librería `tpd018nvtc`, el cual tiene la función tanto de entrada como de salida, esto dependerá de la configuración que le de el usuario. Para ello se debe abocar a la tabla de verdad correspondiente a dicho pin, en la Figura #28 se muestra la tabla de verdad para el pin que se estará utilizando.

Truth Table

INPUT						OUTPUT	
DS	OEN	I	PAD	PE	IE	PAD	C
0/1	0	0	-	0/1	0	0	0
0/1	0	0	-	0/1	1	0	0
0/1	0	1	-	0/1	0	1	0
0/1	0	1	-	0/1	1	1	1
0/1	1	0/1	0	0/1	0	-	0
0/1	1	0/1	0	0/1	1	-	0
0/1	1	0/1	1	0/1	0	-	0
0/1	1	0/1	1	0/1	1	-	1
0/1	1	0/1	Z	0	0	-	0
0/1	1	0/1	Z	0	1	-	X
0/1	1	0/1	Z	1	0	L	0
0/1	1	0/1	Z	1	1	L	L

Figura 28: Tabla de verdad pin PDDW0204SCDG

Al momento de realizar la configuración de las variables del pin se debe seguir el siguiente orden: `(I,DS,OEN,PAD,C,PE,IE)`.

Para configurar el pin como entrada se debe tomar a `PAD` configurado como *input* y `C` como *output*. En esta configuración el primero se comporta como entrada desde el exterior del pin y el segundo como la entrada hacia el circuito sintetizado. La configuración se puede observar en la Figura #29.

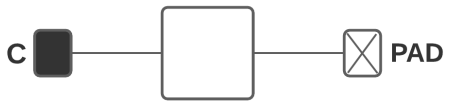


Figura 29: Representación de pin PDDW0204SCDG configurado como entrada

Lo que se desea es que cuando se ingrese un uno lógico en el pin, este pase como un uno hacia la compuerta. En el otro caso también se desea que al ingresar un cero lógico en el pin este pase como cero hacia el circuito, tomando esto en cuenta se procede a analizar los casos listados en la tabla de verdad. Se debe seleccionar un caso donde tanto `PAD` como `C` sean uno y otro caso donde ambos sean cero. Los demás valores de las variables deben ser iguales en ambos casos, siendo unos o ceros.

Para la configuración del pin de entrada se seleccionó el caso seis y ocho de la tabla de verdad. Como se observa en la Figura #30 todos los valores en ambas selecciones son iguales, a excepción de los valores de `PAD` y `C` que son lo que cambian entre 1 y 0.

0/1	1	0/1	0	0/1	0	-	0
0/1	1	0/1	0	0/1	1	-	0
0/1	1	0/1	1	0/1	0	-	0
0/1	1	0/1	1	0/1	1	-	1

Figura 30: Configuración para pines de entrada

Para llevar a cabo la configuración del pin como entrada se debe hacer de la siguiente forma:

```
1     PDDW0204SCDG U10 (1'b0,1'b0,1'b1,A,A_w,1'b01'b1);
```

En donde `A` es el valor que se le da a `PAD` y `A_w` se utiliza en `C`.

La asignación del pin de salida se realiza siguiendo los mismo pasos, lo diferente en este caso son las terminales que se utilizan, aquí se configurará a `PAD` como *output* y a `I` como *input*. Para esta configuración `PAD` es la salida del circuito hacia el exterior, e `I` la salida lógica del circuito. En la Figura #31 se puede observar lo mencionado.

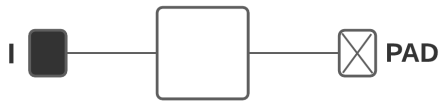


Figura 31: Representación de pin PDDW0204SCDG configurado como salida

Para la configuración del pin como salida se seleccionaron los casos dos y cuatro de la tabla de verdad como se muestra en la Figura #32. Aquí las variables que cambian su valor entre cero y uno de un caso a otro son `PAD` e `I`. Los valores de `C` no serán tomados en cuenta, debido a esto se le asignará un valor de *don't care* ya que no se hace uso de ellos y no afectan en el diseño.

0/1	0	0	-	0/1	0	0	0
0/1	0	0	-	0/1	1	0	0
0/1	0	1	-	0/1	0	1	0
0/1	0	1	-	0/1	1	1	1

Figura 32: Configuración para pines de salida

La configuración final del pin de salida se ve de la siguiente forma:

```
1     PDDW0204SCDG U11 (Y_w,1'b0,1'b0,Y,1'bx,1'b0,1'b0);
```

En donde `Y_w` es el valor que se le da a `I` y `Y` se le asigna a `PAD`.

Se puede configurar más de un pin para cada función, esto dependerá de la cantidad de entradas y salidas que posea el circuito.

Con esto se termina de crear el módulo, lo único que hace falta es copiar la descripción del circuito inicial sintetizado y colocarlo justo después del módulo que se acaba de crear. El resultado final uniendo ambas partes se puede observar en la Figura #33.

```
module Not_IO (A, Y);
 input A;
 output Y;

 wire A_w, Y_w;

 Not Compuerta(A_w,Y_w);

                //Configuracion de PIN I/O

 //Entradas (C):
 PDDW0204SCDG U10(1'b0,1'b0,1'b1,A,A_w,1'b0,1'b1);

 //Salidas (I):
 PDDW0204SCDG U11(Y_w,1'b0,1'b0,Y,1'bx,1'b0,1'b0);

endmodule

module Not ( A, Y );
  input A;
  output Y;

  CKND0BWP7T U2 ( .I(A), .ZN(Y) );
endmodule
```

Figura 33: Descripción de circuito *Not* con pines de entrada y salida

Ahora con esta nueva configuración se procede a ejecutar de nuevo el proceso de síntesis, pero utilizando el archivo que se acaba de crear. A través de este paso ya se obtiene un *netlist* del circuito completo. Como configuración extra cuando ya se tenga el archivo final resultado de la síntesis final, en este se puede crear la instancia a los *pads* de alimentación para el circuito. Existe una terminal dedicada para `vdd` y una para `vss`, estos pertenecen a la misma librería de los pines de entrada y salida configurados con anterioridad. El pin de voltaje se encuentra como `PVDD1CDG` y el de tierra como `PVSS1CDG`. Su configuración se lleva a cabo de la siguiente forma:

```
1     PVDD1CDG VDD ();
2     PVSS1CDG VSS ();
```

Esta es la manera correcta de configurarlos, estos se agregan en el módulo de IO del *netlist* final. Este paso no es necesario llevarlo a cabo, ya que estos se pueden crear también en el proceso de síntesis física. En la siguiente sección se verá el *netlist* final con la inclusión de estos *pads*.

7.5. Nuevo *script* y *netlist* finales

Para llevar a cabo la presente investigación, se utilizó como base la información, pruebas y resultados brindadas por el Ingeniero Luis Arturo Nájera, uno de los predecesores del proyecto del año 2019. El archivo principal fue un *script* con el que se realiza de manera satisfactoria el proceso de síntesis lógica. Debido a los resultados, el archivo no se modificó significativamente, lo único que se agregó fue el parámetro de esfuerzo para ejecutar la compilación y optimización. También se agregaron las líneas de comando que permiten poder observar algunos reportes que pueden ser de ayuda para analizar el circuito.

El *script* se divide en dos partes, en una se realiza la síntesis al circuito descrito inicialmente en 'hdl' y en la otra se lleva a cabo la síntesis al *netlist* que se obtuvo en la primer etapa, pero con la respectiva instancia a los pines de entrada y salida físicos. La creación de este nuevo archivo con pines se detalló en la sección **'Diseño de circuito con pines de entrada y salida'**

A continuación se muestra el *script* final utilizado y el *netlist* final que se obtuvo al sintetizar una compuerta *Not*, una compuerta *Nand* de dos entradas, una compuerta *Nor* de tres entradas, un *Full Adder* y un sumador de ocho *bits*. Con este último archivo en verilog y el archivo 'sdc' respectivo a cada caso se puede dar inicio a la siguiente etapa, la síntesis física.

```
#        SINTETIZACION DE CIRCUITO EN HDL         #

#Abrir la herramienta
./design_vision

#Importacion de liberias
lappend search_path /home/administrador/Escritorio/DV_S/Librerias/
    tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM
set link_library " * tcb018gbwp7ttc.db tcb018gbwp7twc.db
    tcb018gbwp7tbc.db tpd018nvtc.db"
set target_library "tcb018gbwp7ttc.db"

#Inclusion de archivo en verilog
read_file -format verilog {/home/administrador/Escritorio/DV_S/NOT.v}

#Revision de disenio
check_design

#Asignacion de restricciones
reset_design
set_max_area 0

#Compilacion
compile -map_effort high -area_effort high

# Generacion de archivos
write -format ddc -h -o /home/administrador/Escritorio/DV_S/Outs/
    NOT_Syn_1.ddc
write -hierarchy -format verilog -output /home/administrador/Escritorio/
    DV_S/Outs/NOT_Syn_1.v
write_sdc /home/administrador/Escritorio/DV_S/Outs/NOT_Syn_1.sdc

```

```
#Salir de la herramienta
exit
```

```
#       SINTETIZACION DE CIRCUITO CON I/O       #

#Abrir la herramienta
./design_vision

#Importacion de librerias
lappend search_path /home/administrador/Escritorio/DV_S/Librerias/
    tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM
set link_library " * tcb018gbwp7ttc.db tcb018gbwp7twc.db
    tcb018gbwp7tbc.db tpd018nvtc.db"
set target_library "tcb018gbwp7ttc.db"

#Importacion de archivo con pines de entradas y salidas incluidos
read_file -format verilog {/home/administrador/Escritorio/DV_S/IO/
    NOT_IO.v}

#Revision de disenio
check_design

#Compilacion
compile -map_effort high -area_effort high

#Generacion de archivos
write -format ddc -h -o /home/administrador/Escritorio/DV_S/Outs/
    NOT_IO.ddc
write -hierarchy -format verilog -output /home/administrador/Escritorio/
    DV_S/Outs/NOT_IO_Syn.v
write_sdc /home/administrador/Escritorio/DV_S/Outs/NOT_IO.sdc

#Generacion de reportes
report_area
report_design
report_power

exit
```

```
module Not ( A, Y );
  input A;
  output Y;

  CKND0BWP7T U1 ( .I(A), .ZN(Y) );
endmodule


module Not_IO ( A, Y );
  input A;
  output Y;
  wire   A_w, Y_w, n1, n2;
  tri   A;
  tri   Y;

  Not Compuerta ( .A(A_w), .Y(Y_w) );
  PDDW0204SCDG U10 ( .I(n1), .OEN(n2), .IE(n2), .PAD(A), .DS(n1), .PE(n1), .C(A_w) );
  PDDW0204SCDG U11 ( .I(Y_w), .OEN(n1), .IE(n1), .PAD(Y), .DS(n1), .PE(n1) );
  TIELBWP7T U6 ( .ZN(n1) );
  TIEHBWP7T U7 ( .Z(n2) );
  PVDD1CDG VDD ();
  PVSS1CDG VSS ();
endmodule
```

Figura 34: *Netlist* final producto de la síntesis lógica a una compuerta *Not*

```
module Nand ( A, B, Y );
  input A, B;
  output Y;


  CKND2D0BWP7T U1 ( .A1(B), .A2(A), .ZN(Y) );
endmodule


module Nand_IO ( A, B, Y );
  input A, B;
  output Y;
  wire   A_w, B_w, Y_w, n1, n2;
  tri   A;
  tri   B;
  tri   Y;

  Nand Compuerta ( .A(A_w), .B(B_w), .Y(Y_w) );
  PDDW0204SCDG U10 ( .I(n1), .OEN(n2), .IE(n2), .PAD(A), .DS(n1), .PE(n1), .C(A_w) );
  PDDW0204SCDG U11 ( .I(n1), .OEN(n2), .IE(n2), .PAD(B), .DS(n1), .PE(n1), .C(B_w) );
  PDDW0204SCDG U12 ( .I(Y_w), .OEN(n1), .IE(n1), .PAD(Y), .DS(n1), .PE(n1) );
  TIELBWP7T U6 ( .ZN(n1) );
  TIEHBWP7T U7 ( .Z(n2) );
endmodule
```

Figura 35: *Netlist* final producto de la síntesis lógica a una compuerta *Nand* de dos entradas

```
module Nor ( A, B, C, Y );
  input A, B, C;
  output Y;


  NR3D0BWP7T U1 ( .A1(A), .A2(C), .A3(B), .ZN(Y) );
endmodule


module Nor_IO ( A, B, C, Y );
  input A, B, C;
  output Y;
  wire   A_w, B_w, C_w, Y_w, n1, n2;
  tri   A;
  tri   B;
  tri   C;
  tri   Y;

  Nor Compuerta ( .A(A_w), .B(B_w), .C(C_w), .Y(Y_w) );
  PDDW0204SCDG U10 ( .I(n1), .OEN(n2), .IE(n2), .PAD(A), .DS(n1), .PE(n1), .C(A_w) );
  PDDW0204SCDG U11 ( .I(n1), .OEN(n2), .IE(n2), .PAD(B), .DS(n1), .PE(n1), .C(B_w) );
  PDDW0204SCDG U12 ( .I(n1), .OEN(n2), .IE(n2), .PAD(C), .DS(n1), .PE(n1), .C(C_w) );
  PDDW0204SCDG U13 ( .I(Y_w), .OEN(n1), .IE(n1), .PAD(Y), .DS(n1), .PE(n1) );
  TIELBWP7T U6 ( .ZN(n1) );
  TIEHBWP7T U7 ( .Z(n2) );
endmodule
```

Figura 36: *Netlist* final producto de la síntesis lógica a una compuerta *Nor* de tres entradas

```
module Full_Adder_Structural_Verilog ( X1, X2, Cin, S, Cout );
  input X1, X2, Cin;
  output S, Cout;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(X2), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X1), .A2(Cin), .B1(n1), .B2(X2), .Z(Cout) );
  CKXOR2D0BWP7T U3 ( .A1(X1), .A2(Cin), .Z(n1) );
endmodule


module FA_IO ( X1, X2, Cin, S, Cout );
  input X1, X2, Cin;
  output S, Cout;
  wire   X1_w, X2_w, Cin_w, S_w, Cout_w, n1, n2;
  tri   X1;
  tri   X2;
  tri   Cin;
  tri   S;
  tri   Cout;

  Full_Adder_Structural_Verilog Compuerta ( .X1(X1_w), .X2(X2_w), .Cin(Cin_w), .S(S_w), .Cout(Cout_w) );
  PDDW0204SCDG U8 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X1), .DS(n1), .PE(n1), .C(X1_w) );
  PDDW0204SCDG U9 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X2), .DS(n1), .PE(n1), .C(X2_w) );
  PDDW0204SCDG U10 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Cin), .DS(n1), .PE(n1), .C(Cin_w) );
  PDDW0204SCDG U11 ( .I(S_w), .OEN(n1), .IE(n1), .PAD(S), .DS(n1), .PE(n1) );
  PDDW0204SCDG U12 ( .I(Cout_w), .OEN(n1), .IE(n1), .PAD(Cout), .DS(n1), .PE(n1) );
  TIELBWP7T U6 ( .ZN(n1) );
  TIEHBWP7T U7 ( .Z(n2) );
  PVDD1CDG VDD ();
  PVSS1CDG VSS ();

endmodule
```

Figura 37: *Netlist* final producto de la síntesis lógica a un sumador de dos *bits*

```
module fulladder_0 ( X, Y, Ci, S, Co );
  input X, Y, Ci;
  output S, Co;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(Y), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X), .A2(Ci), .B1(n1), .B2(Y), .Z(Co) );
  CKXOR2D0BWP7T U3 ( .A1(X), .A2(Ci), .Z(n1) );
endmodule

module fulladder_7 ( X, Y, Ci, S, Co );
  input X, Y, Ci;
  output S, Co;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(Y), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X), .A2(Ci), .B1(n1), .B2(Y), .Z(Co) );
  CKXOR2D0BWP7T U3 ( .A1(X), .A2(Ci), .Z(n1) );
endmodule

module fulladder_6 ( X, Y, Ci, S, Co );
  input X, Y, Ci;
  output S, Co;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(Y), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X), .A2(Ci), .B1(n1), .B2(Y), .Z(Co) );
  CKXOR2D0BWP7T U3 ( .A1(X), .A2(Ci), .Z(n1) );
endmodule

module fulladder_5 ( X, Y, Ci, S, Co );
  input X, Y, Ci;
  output S, Co;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(Y), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X), .A2(Ci), .B1(n1), .B2(Y), .Z(Co) );
  CKXOR2D0BWP7T U3 ( .A1(X), .A2(Ci), .Z(n1) );
endmodule

module fulladder_4 ( X, Y, Ci, S, Co );
  input X, Y, Ci;
  output S, Co;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(Y), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X), .A2(Ci), .B1(n1), .B2(Y), .Z(Co) );
  CKXOR2D0BWP7T U3 ( .A1(X), .A2(Ci), .Z(n1) );
endmodule
```

```
module fulladder_3 ( X, Y, Ci, S, Co );
  input X, Y, Ci;
  output S, Co;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(Y), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X), .A2(Ci), .B1(n1), .B2(Y), .Z(Co) );
  CKXOR2D0BWP7T U3 ( .A1(X), .A2(Ci), .Z(n1) );
endmodule

module fulladder_2 ( X, Y, Ci, S, Co );
  input X, Y, Ci;
  output S, Co;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(Y), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X), .A2(Ci), .B1(n1), .B2(Y), .Z(Co) );
  CKXOR2D0BWP7T U3 ( .A1(X), .A2(Ci), .Z(n1) );
endmodule

module fulladder_1 ( X, Y, Ci, S, Co );
  input X, Y, Ci;
  output S, Co;
  wire   n1;

  CKXOR2D0BWP7T U1 ( .A1(Y), .A2(n1), .Z(S) );
  AO22D0BWP7T U2 ( .A1(X), .A2(Ci), .B1(n1), .B2(Y), .Z(Co) );
  CKXOR2D0BWP7T U3 ( .A1(X), .A2(Ci), .Z(n1) );
endmodule

module sumador ( X, Y, S, Co );
  input [7:0] X;
  input [7:0] Y;
  output [7:0] S;
  output Co;
  wire   w1, w2, w3, w4, w5, w6, w7;

  fulladder_0 u1 ( .X(X[0]), .Y(Y[0]), .Ci(1'b0), .S(S[0]), .Co(w1) );
  fulladder_7 u2 ( .X(X[1]), .Y(Y[1]), .Ci(w1), .S(S[1]), .Co(w2) );
  fulladder_6 u3 ( .X(X[2]), .Y(Y[2]), .Ci(w2), .S(S[2]), .Co(w3) );
  fulladder_5 u4 ( .X(X[3]), .Y(Y[3]), .Ci(w3), .S(S[3]), .Co(w4) );
  fulladder_4 u5 ( .X(X[4]), .Y(Y[4]), .Ci(w4), .S(S[4]), .Co(w5) );
  fulladder_3 u6 ( .X(X[5]), .Y(Y[5]), .Ci(w5), .S(S[5]), .Co(w6) );
  fulladder_2 u7 ( .X(X[6]), .Y(Y[6]), .Ci(w6), .S(S[6]), .Co(w7) );
  fulladder_1 u8 ( .X(X[7]), .Y(Y[7]), .Ci(w7), .S(S[7]), .Co(Co) );
  TIELBWP7T U2 ( .ZN(1'b0) );
  TIELBWP7T U3 ( .ZN(1'b0) );
endmodule
```

```
module SUMADOR_IO ( X, Y, S, Co );
  input [7:0] X;
  input [7:0] Y;
  output [7:0] S;
  output Co;
  wire   Co_w, n1, n2;
  wire   [0:7] X_w;
  wire   [0:7] Y_w;
  wire   [0:7] S_w;
  tri   [7:0] X;
  tri   [7:0] Y;
  tri   [7:0] S;
  tri   Co;

  sumador SUM ( .X(X_w), .Y(Y_w), .S(S_w), .Co(Co_w) );
  PDDW0204SCDG U9 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X[0]), .DS(n1), .PE(n1), .C(X_w[0]) );
  PDDW0204SCDG U10 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X[1]), .DS(n1), .PE(n1), .C(X_w[1]) );
  PDDW0204SCDG U11 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X[2]), .DS(n1), .PE(n1), .C(X_w[2]) );
  PDDW0204SCDG U12 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X[3]), .DS(n1), .PE(n1), .C(X_w[3]) );
  PDDW0204SCDG U13 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X[4]), .DS(n1), .PE(n1), .C(X_w[4]) );
  PDDW0204SCDG U14 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X[5]), .DS(n1), .PE(n1), .C(X_w[5]) );
  PDDW0204SCDG U15 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X[6]), .DS(n1), .PE(n1), .C(X_w[6]) );
  PDDW0204SCDG U16 ( .I(n1), .OEN(n2), .IE(n2), .PAD(X[7]), .DS(n1), .PE(n1), .C(X_w[7]) );
  PDDW0204SCDG U17 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Y[0]), .DS(n1), .PE(n1), .C(Y_w[0]) );
  PDDW0204SCDG U18 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Y[1]), .DS(n1), .PE(n1), .C(Y_w[1]) );
  PDDW0204SCDG U19 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Y[2]), .DS(n1), .PE(n1), .C(Y_w[2]) );
  PDDW0204SCDG U20 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Y[3]), .DS(n1), .PE(n1), .C(Y_w[3]) );
  PDDW0204SCDG U21 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Y[4]), .DS(n1), .PE(n1), .C(Y_w[4]) );
  PDDW0204SCDG U22 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Y[5]), .DS(n1), .PE(n1), .C(Y_w[5]) );
  PDDW0204SCDG U23 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Y[6]), .DS(n1), .PE(n1), .C(Y_w[6]) );
  PDDW0204SCDG U24 ( .I(n1), .OEN(n2), .IE(n2), .PAD(Y[7]), .DS(n1), .PE(n1), .C(Y_w[7]) );
  PDDW0204SCDG U25 ( .I(S_w[0]), .OEN(n1), .IE(n1), .PAD(S[0]), .DS(n1), .PE(n1) );
  PDDW0204SCDG U26 ( .I(S_w[1]), .OEN(n1), .IE(n1), .PAD(S[1]), .DS(n1), .PE(n1) );
  PDDW0204SCDG U27 ( .I(S_w[2]), .OEN(n1), .IE(n1), .PAD(S[2]), .DS(n1), .PE(n1) );
  PDDW0204SCDG U28 ( .I(S_w[3]), .OEN(n1), .IE(n1), .PAD(S[3]), .DS(n1), .PE(n1) );
  PDDW0204SCDG U29 ( .I(S_w[4]), .OEN(n1), .IE(n1), .PAD(S[4]), .DS(n1), .PE(n1) );
  PDDW0204SCDG U30 ( .I(S_w[5]), .OEN(n1), .IE(n1), .PAD(S[5]), .DS(n1), .PE(n1) );
  PDDW0204SCDG U31 ( .I(S_w[6]), .OEN(n1), .IE(n1), .PAD(S[6]), .DS(n1), .PE(n1) );
  PDDW0204SCDG U32 ( .I(S_w[7]), .OEN(n1), .IE(n1), .PAD(S[7]), .DS(n1), .PE(n1) );
  PDDW0204SCDG U33 ( .I(Co_w), .OEN(n1), .IE(n1), .PAD(Co), .DS(n1), .PE(n1) );
  TIELBWP7T U6 ( .ZN(n1) );
  TIEHBWP7T U7 ( .Z(n2) );
endmodule
```

Figura 38: *Netlist* final producto de la síntesis lógica a un sumador de ocho *bits*

CAPÍTULO 8

Traducción de netlist

La descripción del *netlist* generado como producto de la síntesis lógica se encuentra a nivel compuerta. Cada una de las celdas y componentes empleados se referencian en las librerías incluidas al inicio del proceso. Todo el proceso posterior a la síntesis descrita en el capítulo anterior pueden llevarse a cabo a nivel celdas. Sin embargo, si se desea analizar con más detalle y llevar a cabo una comparación más profunda más adelante, entonces se puede llegar a necesitar un *netlist* a nivel transistor.

Este archivo puede ser de ayuda para saber la cantidad de transistores que se utilizarán para la fabricación, así como las características de estos componentes, el ancho, largo, la conexión de sus pines, etc. Para poder realizar la traducción del *netlist* se debe contar con los siguientes archivos y herramientas:

- *Netlist* a nivel compuerta generado del proceso de síntesis lógica.
- Archivo descriptivo a nivel transistor de las celdas, correspondiente a la librería implementada en el proceso. Normalmente se encuentra en formato *spice* o 'cdl'. En la Figura #39 se muestra solo una parte del archivo que se utilizará, ya que este es extenso.
- La herramienta para llevar a cabo la traducción. Para este caso se utilizará *NetTran*, perteneciente al *software* de *IC Validator*.

Cumpliendo con los requisitos anteriores se procede a llevar a cabo la traducción, cabe mencionar que los archivos que se usan deben estar en una misma ruta para evitar errores. Si no se cuenta con el archivo descriptivo de las celdas a nivel transistor, se debe verificar si el fabricante lo proporciona. Las pruebas realizadas en esta sección serán efectuadas con las librerías e información brindada por *Synopsys*, esto se debe a que el fabricante, TSMC, no provee dichos archivos por temas de confidencialidad.

```
.subckt AND2X1 IN1 IN2 Q VDD VSS
mmp2 net1 IN2 VDD VDD p12 l = 0.1u w = 0.41u m = 1
mmp1 net1 IN1 VDD VDD p12 l = 0.1u w = 0.41u m = 1
mmp3 Q net1 VDD VDD p12 l = 0.1u w = 1.12u m = 1
mmn2 net2 IN2 VSS VSS n12 l = 0.1u w = 0.21u m = 1
mmn1 net1 IN1 net2 VSS n12 l = 0.1u w = 0.21u m = 1
mmn3 Q net1 VSS VSS n12 l = 0.1u w = 0.4u m = 1


.ends AND2X1



.subckt AND2X2 IN1 IN2 Q VDD VSS
mmp4 Q net1 VDD VDD p12 l = 0.1u w = 1.12u m = 2
mmp5 net1 IN1 VDD VDD p12 l = 0.1u w = 0.41u m = 1
mmp6 net1 IN2 VDD VDD p12 l = 0.1u w = 0.41u m = 1
mmn4 net2 IN2 VSS VSS n12 l = 0.1u w = 0.21u m = 1
mmn5 net1 IN1 net2 VSS n12 l = 0.1u w = 0.21u m = 1
mmn6 Q net1 VSS VSS n12 l = 0.1u w = 0.4u m = 2


.ends AND2X2



.subckt AND2X4 IN1 IN2 Q VDD VSS
mmp4 Q net1 VDD VDD p12 l = 0.1u w = 1.12u m = 4
mmp5 net1 IN1 VDD VDD p12 l = 0.1u w = 0.41u m = 1
mmp6 net1 IN2 VDD VDD p12 l = 0.1u w = 0.41u m = 1
mmn4 net2 IN2 VSS VSS n12 l = 0.1u w = 0.21u m = 1
mmn5 net1 IN1 net2 VSS n12 l = 0.1u w = 0.21u m = 1
mmn6 Q net1 VSS VSS n12 l = 0.1u w = 0.4u m = 4


.ends AND2X4
```

Figura 39: Archivo descriptivo de las celdas para llevar a cabo la traducción de *netlist*

8.1. Proceso

La herramienta de *NetTran* realiza por defecto una traducción que cumple con el formato de entrada para *IC Validator*. Si no se desea este formato, entonces se puede elegir cualquier otro que sea aceptado: *verilog*,*spice* y *spice_flat*. Cualquiera de los archivos generados seguirá conteniendo la descripción del circuito sintetizado y sus interconexiones [14], pero ahora también tendrá la instancia a la definición de las celdas en un mismo archivo.

El primer paso para realizar la traducción es convertir a formato *spice* el *netlist* obtenido de la síntesis. La traducción se lleva a cabo a través de la línea de comando `icv_nettran -verilog netlist.v -outName netlist.sp -outType spice`.

Esto se realiza ya que se necesita crear la instancia al archivo con la descripción de las celdas. La inclusión se ejecuta a través de la instrucción `.include "filename.cdl"`. Ya con esta configuración se puede proceder al paso final, traducción a nivel transistor. Aquí se utiliza de nuevo la línea de comando `icv_nettran -sp netlist.sp -outName netlist_transitores.sp -outType spice`, con la diferencia que el formato de entrada ahora es *spice*, y el parámetro que sustituye el formato *verilog* es `-sp`.

En el archivo obtenido se incluye la descripción a nivel transistor de todas las celdas y componentes que incluye la librería utilizada como sub-circuitos. Al final de este se encuentran los sub-circuitos de todas las celdas creadas y utilizadas en la síntesis. En la siguiente sección se muestran los resultados obtenidos al ejecutar el proceso descrito. Estas pruebas fueron realizadas para una compuerta *Not* y un *Full Adder* de dos *bits*.

8.2. Pruebas

Para empezar se debe tomar en cuenta que todos los pines necesarios se encuentren conectados en la descripción del circuito. Si esto no se cumple se tendrá entonces el error de `icv_floatnet` aunque la traducción se haya realizado correctamente. La mayoría de veces ocurre porque no se instanció la alimentación de la celda que se utilizó [14]. Esta configuración se puede realizar en el circuito ya sintetizado.

Las pruebas con los comandos se llevarán a cabo desde la terminal de la herramienta. Para hacer uso de ella se debe dirigir al siguiente directorio `usr/synopsys/icvalidator`. Una vez allí, entonces se puede abrir la terminal desde la opción *Open in terminal* que se encuentra al presionar *click* derecho en la ubicación.

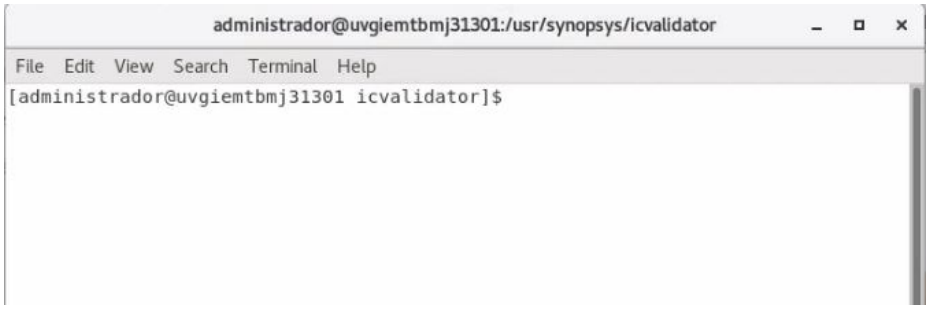


Figura 40: Terminal de la herramienta *IC Validator*

8.2.1. Not

Del proceso de síntesis se genera lo mostrado en la Figura #41a. Como se observa aquí hacen falta las instancias a VDD y VSS. Si se quiere llevar a cabo este proceso, entonces se debe asignar `.VDD(VDD)` y `.VSS(VSS)` en la celda #41b. Este proceso no lo ejecuta la herramienta automáticamente.

```
module Not ( A, Y );
  input A;
  output Y;

  INVX0 U2 ( .INP(A), .ZN(Y) );
endmodule
```

(a) Síntesis de circuito sin VDD y VSS

```
module Not ( A, Y, VDD, VSS );
  input A, VDD, VSS;
  output Y;

  INVX0 U2 ( .INP(A), .ZN(Y), .VDD(VDD), .VSS(VSS));
endmodule
```

(b) Síntesis de circuito con VDD y VSS

Figura 41: *Netlist* generado de la síntesis circuito *Not*

Con el *netlist* incluyendo pines de alimentación ahora se procede a la traducción de formato *verilog* a formato *spice*. En el encabezado se incluye la instancia hacia el archivo que contiene la descripción de las celdas, como se muestra en la Figura #42. El comando a utilizar es:

```
icv_nettran -verilog Not_Syn.v -outType spice -outName Not_Syn.sp
```

```
.include "saed90nm.cdl"

.SUBCKT Not A Y
XU2 A Y INVX0
.ENDS
```

(a) Traducción *netlist* circuito sin VDD y VSS

```
.include "saed90nm.cdl"

.SUBCKT Not A Y VDD VSS
XU2 A Y VDD VSS INVX0
.ENDS
```

(b) Traducción *netlist* circuito con VDD y VSS

Figura 42: Traducción *netlist* circuito *Not*

Por último ya se podrá generar el *netlist* a nivel transistores con la siguiente instrucción.

```
icv_nettran -sp Not_Syn.sp -outType spice -outName Not.sp
```

```
.SUBCKT INVX0 INP VDD VSS ZN
mmp1 ZN INP VDD VDD p12 L=0.1u W=0.55u M=1
mmn1 ZN INP VSS VSS n12 L=0.1u W=0.24u M=1
.ENDS

.SUBCKT Not A Y
XXU2 A Y icv_floatnet_1 icv_floatnet_2 INVX0
.ENDS
```

(a) Traducción a nivel transistores de circuito sin VDD y VSS

```
.SUBCKT INVX0 INP VDD VSS ZN
mmp1 ZN INP VDD VDD p12 L=0.1u W=0.55u M=1
mmn1 ZN INP VSS VSS n12 L=0.1u W=0.24u M=1
.ENDS

.SUBCKT Not A Y VDD VSS
XXU2 A Y VDD VSS INVX0
.ENDS
```

(b) Traducción a nivel transistores de circuito con VDD y VSS

Figura 43: Traducción a nivel transistores de circuito *Not*

8.2.2. Full Adder

El ejemplo con el circuito de un *Full Adder* que se mostrará se realiza con la asignación de VDD y VSS para evitar *nets* flotantes, con el netlist a nivel transistor mostrado en la Figura #46.

```
module Full_Adder_Structural_Verilog ( X1, X2, Cin, S, Cout, VDD, VSS );
  input X1, X2, Cin, VDD, VSS;
  output S, Cout;
  wire   n2;

  XOR2X1 U4 ( .IN1(X2), .IN2(n2), .Q(S), .VDD(VDD), .VSS(VSS) );
  AO22X1 U5 ( .IN1(Cin), .IN2(X1), .IN3(X2), .IN4(n2), .Q(Cout), .VDD(VDD), .VSS(VSS) );
  XOR2X1 U6 ( .IN1(Cin), .IN2(X1), .Q(n2), .VDD(VDD), .VSS(VSS) );
endmodule
```

Figura 44: Síntesis de circuito *Full Adder* con instancias de VDD y VSS

```
.include "saed90nm.cdl"

.SUBCKT Full_Adder_Structural_Verilog X1 X2 Cin S Cout VDD VSS
XU4 X2 n2 S VDD VSS XOR2X1
XU5 Cin X1 X2 n2 Cout VDD VSS AO22X1
XU6 Cin X1 n2 VDD VSS XOR2X1
.ENDS
```

Figura 45: Traducción a *spice* de *netlist* e inclusión de librería

```
.SUBCKT XOR2X1 IN1 IN2 Q VDD VSS
mmp2 net2 IN2 VDD VDD p12 L=0.1u W=0.32u M=1
mmp1 net1 IN1 VDD VDD p12 L=0.1u W=0.32u M=1
mmp3 net3 IN2 VDD VDD p12 L=0.1u W=0.6u M=1
mmp4 net7 IN1 net3 VDD p12 L=0.1u W=0.6u M=1
mmp5 net5 net2 VDD VDD p12 L=0.1u W=0.6u M=1
mmp6 net7 net1 net5 VDD p12 L=0.1u W=0.6u M=1
mmp7 Q net7 VDD VDD p12 L=0.1u W=1.12u M=1
mmn2 net2 IN2 VSS VSS n12 L=0.1u W=0.21u M=1
mmn3 net7 IN1 net4 VSS n12 L=0.1u W=0.21u M=1
mmn1 net1 IN1 VSS VSS n12 L=0.1u W=0.21u M=1
mmn4 net4 net2 VSS VSS n12 L=0.1u W=0.21u M=1
mmn5 net7 net1 net6 VSS n12 L=0.1u W=0.21u M=1
mmn6 net6 IN2 VSS VSS n12 L=0.1u W=0.21u M=1
mmn7 Q net7 VSS VSS n12 L=0.1u W=0.4u M=1
.ENDS

.SUBCKT AO22X1 IN1 IN2 IN3 IN4 Q VDD VSS
mmn2 net3 IN1 net2 VSS n12 L=0.1u W=0.21u M=1
mmn4 net3 IN3 net4 VSS n12 L=0.1u W=0.21u M=1
mmn5 Q net3 VSS VSS n12 L=0.1u W=0.43u M=1
mmn3 net4 IN4 VSS VSS n12 L=0.1u W=0.21u M=1
mmn1 net2 IN2 VSS VSS n12 L=0.1u W=0.21u M=1
mmp3 net3 IN4 net1 VDD p12 L=0.1u W=0.47u M=1
mmp5 Q net3 VDD VDD p12 L=0.1u W=1.12u M=1
mmp2 net1 IN1 VDD VDD p12 L=0.1u W=0.47u M=1
mmp1 net1 IN2 VDD VDD p12 L=0.1u W=0.47u M=1
mmp4 net3 IN3 net1 VDD p12 L=0.1u W=0.47u M=1
.ENDS

.SUBCKT Full_Adder_Structural_Verilog X1 X2 Cin S Cout VDD VSS
XXU4 X2 n2 S VDD VSS XOR2X1
XXU5 Cin X1 X2 n2 Cout VDD VSS AO22X1
XXU6 Cin X1 n2 VDD VSS XOR2X1
.ENDS
```

Figura 46: *Netlist* a nivel transistor de circuito *Full Adder*

CAPÍTULO 9

Conclusiones

- Se completó la reestructuración de los comandos y la configuración para llevar a cabo la síntesis lógica.
- Los cambios realizados al *script* antiguo brindan un proceso de compilación y optimización más dedicado así como la generación de reportes.
- El proceso de síntesis lógica depende tanto de la descripción del circuito a trabajar como la correcta asignación de los pines de entrada y salida.
- Con base en las pruebas realizadas en el presente proyecto, se llegó a obtener la mejor configuración para la instancia de los pines de entrada y salida.
- La herramienta que se utilizó para llevar a cabo la traducción fue *NetTran* perteneciente al grupo de *Synopsys*.
- Para llevar a cabo la traducción del *netlist* a nivel compuerta hacia una descripción a nivel transistores, se determinó que es necesario contar con el archivo descriptivo de las celdas y componentes implementados a nivel transistor.
- El proceso de traducción de *netlist* se logró concluir a nivel transistores con ayuda de la documentación y librerías académicas proporcionadas por Synopsys. Todo el proceso y paso a paso está documentado en el presente trabajo, sin embargo la traducción con las librerías brindadas por TSMC no se logró realizar, ya que el archivo descriptivo de las celdas únicamente se encontraba en un nivel de *black box* indicando sus entradas y salidas.

CAPÍTULO 10

Recomendaciones

Al llevar a cabo una investigación se requiere siempre de la revisión de bibliografía, ya que normalmente son herramientas que sirven para dar inicio o integrarse al ámbito de la temática. Debido a esto se sugiere que paralelo a la lectura del presente proyecto, se haga una revisión de las guías y manuales utilizados. Específicamente *Design Compiler User Guide* [17], *Design Vision User Guide* [10], *Synthesis Tool Commands* [12] y *IC Validator LVS User Guide* [14].

Así mismo se sugiere la lectura de los trabajos de graduación que le dieron inicio al proyecto [6][4][3]. Esto servirá para que el estudiante logre un mejor entendimiento de las herramientas, configuraciones, comandos implementados en el proceso, uso de las herramienta y familiarización con el entorno.

Estos y otros documentos son proporcionados por la empresa *Synopsys*, a través de la plataforma de *SolvNet*. En esta plataforma también existen foros y una sección de preguntas y respuestas en donde la persona que esté revisando el presente proyecto podrá consultar la mayoría de dudas que surjan. Así como posibles errores si se realizara alguna modificación o implementación de nuevos comandos en la configuración proporcionada.

Se recomienda al estudiante también la revisión de la descripción del circuito a implementar, así como realizar distintas pruebas al mismo a través de una herramienta externa al grupo de *Synopsys*. Esto se debe llevar a cabo para verificar el correcto funcionamiento del diseño y así poder evitar futuros errores.

La declaración de *pads* de alimentación no es necesaria en esta etapa, sin embargo puede ser de gran ayuda llevarlo a cabo para facilitar la ejecución de la siguiente etapa, el proceso de síntesis física.

Debido a que la síntesis lógica es la primer etapa del flujo de diseño para la fabricación de un *chip*, los errores que se obtengan en esta fase así como las mejoras obtenidas migrarán a las siguientes etapas. Debido a esto se recomienda que el estudiante o estudiantes que hagan revisión de los resultados de esta sección, tengan una comunicación constante con los

integrantes encargados de las siguientes fases. Con esto se podrá llegar a obtener un proceso limpio y así mismo se sabrá si los errores que los otros estudiantes obtengan son producto de la configuración de síntesis lógica.

Por último, al momento de realizar la traducción del *netlist* se documentó el proceso con base a los archivos brindados por la empresa de *Synopsys*. Esto se debe a que el fabricante, TSMC, no brinda el archivo que se necesita para completar el proceso debido a temas de confidencialidad. Este archivo es de suma importancia, ya que contiene la descripción de las celdas a nivel transistor, sin este documento la traducción no se puede ejecutar con las celdas de TSMC. Sin embargo, se recomienda llevar a cabo la solicitud de dicho archivo al fabricante para poder completar satisfactoriamente esta tarea.

CAPÍTULO 11

Bibliografía

[1] (). Circuito integrado, dirección: `https://www.ecured.cu/Circuito_integrado`.

[2] H. Kaeslin, *Digital Integrated Circuit Design from VLSI Arrchitectures to CMOS Fabrication*. Cambrige, 2008.

[3] S. H. Rubio Vásquez, "Definición del Flujo de Diseño para Fabricación de un Chip con Tecnologia VLSI CMOS", Tesis de licenciatura Ingeniería Electrónica, Universidad del Valle de Guatemala, 2019.

[4] L. A. Nájera Vásquez, "Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado", Tesis de licenciatura Ingeniería Electrónica, Universidad del Valle de Guatemala, 2019.

[5] P. Kurup y T. Abbasi, *Logic synthesis using Synopsys®*. Springer Science & Business Media, 2012.

[6] J. A. Santos Chonay, "Diseño de un sumador/restador de 32 bits con tecnología CMOS en un proceso de 28 nanómetros usando aplicaciones de diseño de la empresa Synopsys", Tesis de licenciatura Ingeniería Electrónica, Universidad del Valle de Guatemala, 2014.

[7] M. M. Mano y M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013.

[8] D. K. Tala, "Verilog tutorial", *http://www. asic-world. com/verilog/veritut. html.(in English)*, 2003.

[9] C. R. Romano, R. C. Bracamontes y M. B. Sánchez, "Síntesis de un procesador digital elemental",

[10] Synopsys, *Design Vision User Guide Q-2019.12*. Solvnet, 2020.

[11] W. F. Lee, *VHDL coding and logic synthesis with synopsys*. Elsevier, 2000.

[12] Synopsys, *Synthesis Tool Commands Q-2019.12*. Solvnet, 2020.

[13] S. Sakthikumaran, S. Salivahanan y V. K. Bhaaskaran, *16-Bit RISC processor design for convolution application*. 2011, págs. 394-397.

[14] Synopsys, *IC Validator LVS User Guide Q-2019.12*. Solvnet, 2020.

[15] *IC Validator Physical Verification*, Synopsys.

[16] H. B. Kommuru y H. Mahmoodi, "ASIC design flow tutorial using synopsys tools", *Nano-Electronics & Computing Research Lab, School of Engineering, San Francisco State University San Francisco, CA, Spring*, 2009.

[17] Synopsys, *Design Compiler User Guide Q-2019.12*. Solvnet, 2019.