UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Posicionamiento e interconexión entre componentes de un circuito sintetizado para el flujo de diseño de un circuito a escala nanométrica utilizando la herramienta de IC Compiler

Trabajo de graduación presentado por Luis Estuardo Abadía López
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

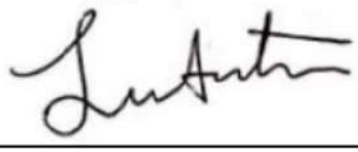
Posicionamiento e interconexión entre componentes de un circuito sintetizado para el flujo de diseño de un circuito a escala nanométrica utilizando la herramienta de IC Compiler

Trabajo de graduación presentado por Luis Estuardo Abadía López
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

Vo.Bo.:



(f) _____

Ing. Luis Nájera

Tribunal Examinador:

(f) _____

Ing. Luis Nájera

(f) _____

MSc. Carlos Esquit

(f) _____

Ing. Jonathan de los Santos

Fecha de aprobación: Guatemala, 5 de enero de 2022.

Agradecimientos

A Dios por haberme permitido gozar a mis abuelitos, por haberme brindado tantas oportunidades y haberme brindado tanta sabiduría y amor.

A mis abuelitos, Ruben Homero López Mijangos † y Blanca Elvira López Pozuelos de López por haberme dado la oportunidad de formarme en esta universidad, por creer en mi potencial y siempre haber apoyarme todos mis sueños. Por ser grandes ejemplos a seguir para mi y toda mi familia.

A mi abuelita, Zoila Fidelia Aquino de Abadía por haberme escuchado y guiado siempre por un buen camino.

A mis papás, Estuardo Alejandro Abadía Aquino y Miriam Leticia López de Abadía, quien gracias a ellos tengo la oportunidad de haberme formado como una persona integra. Agradezco me hayan dado la oportunidad de estudiar en la universidad y siempre ser un apoyo a lo largo de mi vida. Agradezco que nos hayan brindado a mí y a mis hermanas la oportunidad de ser profesionales.

A mis hermanas, María Natalia Abadía López y Ana Lucia Abadía López por haberme dado siempre un motivo por el cual superarme, por acompañarme a lo largo de este tiempo y por siempre ser un apoyo incondicional.

A mi familia, primos y tíos por siempre estar pendiente de mis estudios, guiarme y por tomarme como ejemplo en sus vidas.

A Karol Sophia Cardona Polanco por haber llegado a mi vida y siempre creer en mí. Por haber sido una guía y un gran ejemplo a seguir.

A mis amigos, por compartir cada momento de mi vida y por haber sido siempre un apoyo en mis estudios.

A la Universidad del Valle, a los docentes, en especial a Zaida Urrutia y a mi director de carrera, Ing. Carlos Esquit por promover los estudios tecnológicos en la universidad, por esta investigación y ayudarnos a cumplir esta meta.

Índice

Lista de figuras

Resumen

En los últimos años se ha iniciado el proceso de fabricación de un circuito integrado (*chip*) como trabajo de graduación en la Universidad del Valle de Guatemala. En estos años se ha definido una metodología para llevar a cabo dicha fabricación. En el diseño de circuitos integrados esta metodología se conoce con el nombre de *Design Flow* o flujo de diseño. El flujo de diseño contiene diferentes etapas que combinadas hacen posible la elaboración de un circuito nanométrico con tecnología *CMOS*.

El presente trabajo se enfoca en desarrollar la etapa secundaria del flujo del diseño, la cual abarca los temas de diseño y desarrollo de la síntesis física. El objetivo principal del presente trabajo radica en establecer los pasos a seguir para elaborar de manera exitosa la segunda etapa del flujo de diseño. Estos dan una comprensión de como crear y definir un *Floorplan*, además de como realizar el posicionamiento e interconexión de los componentes que integra un *chip*.

Para el desarrollo de la síntesis física fue necesario estudiar y utilizar la herramienta de *IC Compiler*. Esta es proporcionado por la empresa *Synopsys*, líder en desarrollo de *software* especializado para el diseño de circuitos integrados complejos. Diversas pruebas de síntesis fueron realizadas utilizando distintos circuitos para comprender el funcionamiento de la herramienta. Con base en los resultados se ha determinado que las configuraciones planteadas a lo largo del documento reducen errores de diseño en etapas posteriores.

Abstract

In the last years, the Universidad del Valle de Guatemala has begun with the process of developing an integrated circuit (chip) as a thesis graduation work. In these years they have started with a methodology to implement that. On integrated circuit design this methodology takes the name of Design Flow. The Design Flow implements different ways to elaborate a nano circuit with CMOS technology.

This work focuses on developing the secondary stage of the design flow, which covers the design and development of physical synthesis. The main objective of this work is to establish the steps to follow to get a successful second stage.

For the development of the physical design, it was necessary to study and use the IC Compiler tool. Synopsys provides this tool. To understand the IC Compiler tool different physical synthesis test were made using different circuits. Based on the results, it has been determined that the configurations proposed throughout the document reduce design errors in later stages.

CAPÍTULO 1

Introducción

Actualmente el uso de dispositivos electrónicos es primordial. Cada día se incrementa una dependencia del uso de equipo electrónico o sistemas de computo [1]. Gracias a esta dependencia las personas buscamos que conforme pasa el tiempo los dispositivos mejoren. Estas mejoras implican que los dispositivos tengan mas almacenamiento, sean mas pequeños y que tengan cada vez mas poder computacional para realizar cualquier operación [2][1]. En este momento existe la necesidad de disminuir el *hardware* a la vez que aumenta la velocidad de procesamiento. Esto se ha convertido en un desafió para los diseñadores de circuitos integrados [1][2].

Los dispositivos electrónicos cuentan en su interior con uno o varios *chips* o circuitos integrados. Estos pueden encontrarse de dos tipos: circuitos de propósito general o especifico. El *chip* es el encargado de ejecutar las tareas por medio de la interconexión con otros componentes electrónicos [3]. Los circuitos ASIC, *Application-Specific Integrated Circuit*, son circuitos diseñados para cumplir una función específica [3].

Los *chips* a su vez se definen como componentes electrónicos que en su interior incorporan e interconectan cientos o miles de transistores [4]. Los circuitos integrados son fabricados conjuntamente en una fina oblea de silicio, de un material semiconductor, en un proceso llamado litografía [4].

Existen varias técnicas de diseño para crear los circuitos integrados, una de la mas importante en la actualidad es el diseño de *CMOS*, *Complementary Metal Oxide Semiconductor* [5]. Los diseñadores de circuitos integrados aplican un flujo de diseño que les permiten diseñar un *chip* para que posteriormente se pueda fabricar sin errores [6].

La Universidad del Valle de Guatemala se encuentra estudiando el proceso de desarrollo de un circuito integrado. Para ello se ha llevado a cabo el desarrollo de un flujo de diseño el cual sienta las bases para crear circuitos propios y futuros proyectos [7][8]. El flujo que se ha establecido contiene dos grandes fases las cuales son: *Front End* y *Back End* [7][8].

La fase *Front End* establece el proceso de creación del circuito en fase preliminar. Este

se crea con base en un diseño lógico descrito en lenguaje de programación *Verilog*. Posteriormente el diseño descrito se sintetiza de manera lógica a nivel de compuertas primitivas generando un *netlist*. Este contiene la información del circuito a crear [7][8].

La fase *Back End* determina el diseño físico que tendrá el circuito. En este proceso se define la creación del espacio físico que tendrá el *chip*, este espacio se define como *Floorplan*. Posteriormente se realiza una colocación y la interconexión de los componentes descritos en el archivo *netlist*. En esta fase se realiza la creación de anillos de entradas y salidas, así como validaciones y verificaciones para el cumplimiento de reglas de diseño [7][8].

La Universidad del Valle de Guatemala brinda la posibilidad de utilizar el *software IC Compiler*, el cual se utiliza para llevar a cabo varios procesos en la etapa de *Back End*.

Con motivo de llevar a cabo el proceso de fabricación y realizar una mejora al flujo del diseño de la síntesis física en la etapa de *Back End*, se optó por desarrollar el presente trabajo de graduación. El trabajó abarca la configuración y creación de: *Floorplan*, *Placement* y *Routing*. Además se establece un flujo actualizado para el desarrollo de la síntesis física en la etapa de *Back End*. El presente trabajo desarrolla la metodología en circuitos básicos como: *Not*, *Full Adder* y un *Ripple Carry Adder*. Así mismo establece una base para el desarrollo de circuitos más complejos y más extensos.

CAPÍTULO 2

Antecedentes

Desde hace ya varios años, la Universidad del Valle de Guatemala introdujo la carrera de ingeniería electrónica como una carrera profesional. La carrera de Ingeniería electrónica tiene una gran importancia hoy en día, esto se debe principalmente a que en la actualidad los seres humanos utilizamos la tecnología día a día como una herramienta o como un recurso. La carrera actualmente está dirigida por el Ing. Carlos Esquit. Él es un pionero en el estudio de circuitos nanométricos en Guatemala, y gracias a su conocimiento ha reformado el mapa curricular introduciendo así el curso de diseño de circuitos con tecnología nanométrica.

Actualmente en la Universidad del Valle de Guatemala existe un acuerdo académico con la empresa de Synopsys, la cual es una empresa líder en desarrollo de software especializado en el diseño de circuitos integrados. Por esto último se ha reformado el mapa curricular, pasando del curso Introducción al diseño de sistemas VLSI en el 2013 hasta la reforma actualizada en el año 2015 con los cursos de nanoelectrónica 1 y 2. Esta innovación en la carrera de ingeniería electrónica ha permitido que los estudiantes comprendan de mejor manera la electrónica a nanoescala y ha facilitado que se generen proyectos en el campo de nanoelectrónica, como lo son los trabajos previos realizados por los estudiantes presentados en las referencias [7][8][9].

CAPÍTULO 3

Justificación

El desarrollo de un circuito integrado es muy complejo, este requiere disciplina y tener amplio conocimiento. Para su desarrollo se requiere tener un orden y cumplir procesos y normas que son establecidas por la empresa fabricante. Por esto se desarrolla este trabajo de investigación, el cual pretende establecer la documentación del proceso que conlleva mejorar la estructura del circuito integrado en la síntesis física evitando errores de áreas y conexiones. Esto permite abrir una brecha para que las futuras generaciones puedan desarrollar el primer *chip* para Guatemala.

Por consecuencia, este trabajo sienta las bases para la elaboración o desarrollo de tecnología en Guatemala, siendo un incentivo para empresas o instituciones que quieran apoyar el trabajo de investigación, experimentación e innovación en el país.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Establecer los pasos a seguir para elaborar un posicionamiento e interconexión de componentes de un circuito integrado utilizando la herramienta *IC Compiler*.

4.2. Objetivos específicos

- Delimitar un área que permita el posicionamiento de los componentes que conforman el circuito integrado.
- Realizar un proceso de pruebas de posicionamiento e interconexión entre componentes por medio de comandos, el cual nos permita reducir los errores de diseño en etapas posteriores y poder así llevar a cabo el proceso de fabricación.

CAPÍTULO 5

Alcance

El motivo de esta investigación es la mejora de la síntesis física en un flujo de diseño para el desarrollo de un circuito nanométrico. Se pretende utilizar las librerías y documentos proporcionados por la empresa fabricante (TSMC) para con ello lograr este objetivo. Se busca que al finalizar esta etapa se presente una reducción de errores en esta fase del diseño.

Al finalizar este segmento del flujo de diseño se pretende elaborar una guía que sirva a futuros estudiantes para realizar proyectos en este campo. Esta guía se realizó de la forma más detallada posible para brindar una solución accesible y rápida.

Para el desarrollo de esta propuesta se identificaron fallos de uso en la herramienta, se elaboraron las conexiones pertinentes entre las celdas, así como las interconexiones de estas con los *Pad's* y anillos.

CAPÍTULO 6

Marco teórico

6.1. Circuito Integrado

Los circuitos integrados (CI) o *integrated circuits*, por sus siglas en inglés (IC), pueden ser definidos como componentes electrónicos que en su interior incorporan e interconectan una multitud de dispositivos electrónicos en miniatura, en su mayoría estos dispositivos suelen ser transistores. Muchos de estos circuitos integrados se fabrican conjuntamente en una fina oblea de un material semiconductor que por lo regular suele ser silicio [4].

Existen diferentes técnicas de diseño para circuitos, una de ellas y una de las mas importantes actualmente es el diseño con CMOS, *Complementary Metal Oxide Semiconductor*. Los CMOS están conformados por un par de transistores, un transistor NMOS y un transistor PMOS [5][3]. Esta técnica es ampliamente utilizada para la creación de los circuitos integrados, ya que a través de los años ha demostrado ser una técnica muy funcional y escalable. Esto quiere decir que a pesar de que cambien varios métodos o procesos de fabricación esta técnica se puede seguir utilizando [5].

El proceso de diseño CMOS consiste en: definir entradas y salidas de circuitos, realizar cálculos manuales, diseñar los circuitos en *softwares*, simularlos, realizar extracción de parásitos, reevaluar las entradas y salidas, para finalmente fabricar y realizar pruebas al circuito integrado. Una vez el proceso de fabricación este completo este no podrá ser modificado [5].

Debido a esto último y sabiendo que los circuitos integrados en su interior poseen miles de cientos de interconexiones, los diseñadores realizan un flujo de diseño que les permite hacer simulaciones, comparaciones y mitigar cualquier error posible de manera sistemática. El flujo de diseño tiene el fin de que el circuito integrado cumpla con su propósito una vez que esté fabricado. La Figura #1 muestra cómo es una oblea de silicio y la Figura #2 muestra cómo es un circuito integrado.

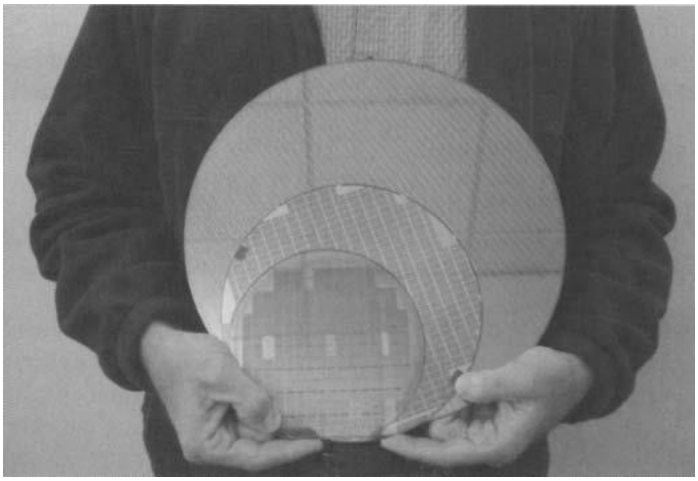


Figura 1: Obleas de silicio [5]

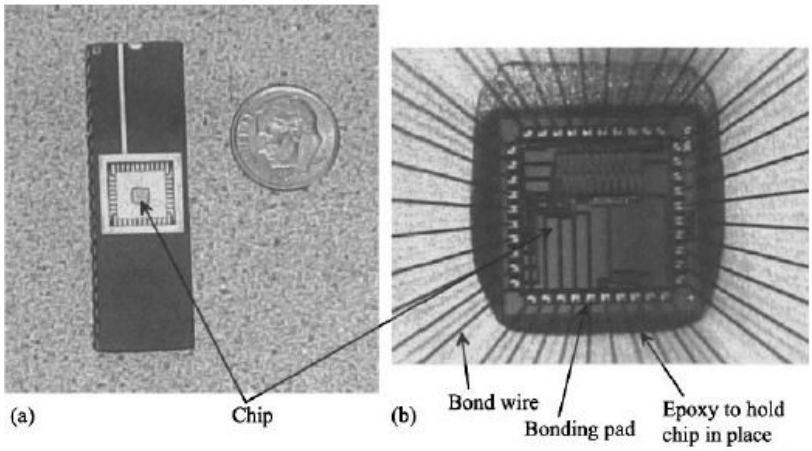


Figura 2: Empaquetado de chip y una vista de cerca [5]

6.2. Design Flow

Como ya se mencionó, para diseñar un circuito integrado el diseñador debe tener una idea sobre qué es exactamente lo que se quiere diseñar. Existen varios tipos de circuitos integrados, estos pueden clasificarse en circuitos de propósito general y circuitos de propósito específico. Los circuitos integrados de propósito general son aquellos a los que se les puede configurar para que realicen cualquier operación o función. Los circuitos integrados de propósito especifico o ASIC, *Application-Specific Integrated Circuit*, son circuitos integrados desarrollados para cumplir una función específica, por ejemplo, una memoria RAM [3].

Un *Design Flow* o Flujo de diseño es un conjunto de procedimientos o instrucciones que le permiten al diseñador cumplir ciertas especificaciones y poder transmitir dichas especificaciones al diseño de un chip para luego poder fabricarlo sin errores. En una estructura de un flujo de diseño existen dos grandes fases, *Front End y Back End* [3][6]. En la Figura #3 se puede observar un flujo de diseño con las dos fases mencionadas con anterioridad.

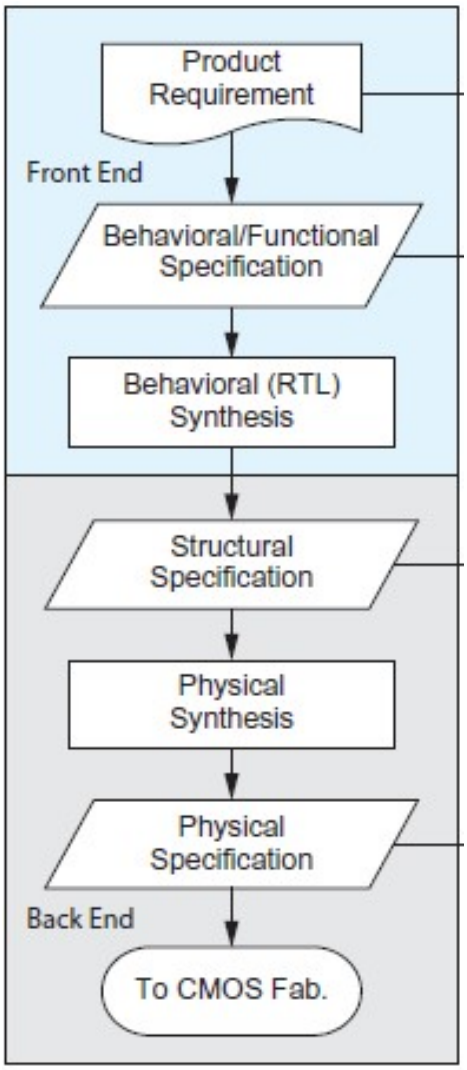


Figura 3: Flujo de diseño, *Front End and Back End* [6]

Para garantizar un diseño ASIC exitoso, los diseñadores deben de seguir un buen flujo de diseño, el cual debe incorporar una buena comprensión de las especificaciones, los requisitos de fabricación, el diseño y rendimiento de baja potencia [6].

6.3. Synopsys

Es una compañía líder en el desarrollo de *software* especializado para diseñar circuitos integrados complejos. La compañía proporciona *software*, propiedad intelectual y servicios. La empresa se encarga de suministrar el *software* de automatización de diseño electrónico (EDA) que los ingenieros usan para diseñar y probar circuitos integrados, también conocidos como chips. Ellos ofrecen a sus clientes productos rediseñados y actualizados de forma frecuente. Synopsys brinda seguridad de *software* y pruebas de calidad que se requieren para diseñar los procesos y modelos que serán utilizados en próximas generaciones de chips.

6.4. Librerías

Las librerías de Synopsys son un conjunto de archivos los cuales permiten elaborar el flujo de diseño. Estos archivos resultan relevantes ya que poseen todas las características de los componentes que conformarán el circuito, de no tener estas librerías no es posible realizar el flujo de diseño. Durante la elaboración del flujo será necesario utilizar distintos archivos de las librerías, cada archivo contiene información relevante sobre cómo funcionará el circuito.

Así mismo las librerías son proporcionadas por la empresa encargada de fabricar los circuitos integrados, ya que son ellos los encargados de brindarnos la información necesaria de los componentes que se pueden utilizar. Un ejemplo de esto es la tecnología que se utiliza, Synopsys utiliza tecnología de 90 nm y 32 nm. De lo mencionado anteriormente, los archivos de librerías más utilizados en las fases *Front End* y *Back End* son: *Target Library*, *Link Library*, *Symbol Library*, *Tluplus*, *Reference Library*, *Technology file*, *Milkyway*, entre otros. Como ya se hizo mención son estas las librerías que contienen la información necesaria para que el circuito opere de manera eficiente [10].

Las librerías de *Target Library* y *Link Library* le permiten al diseñador controlar mejor la asignación de las celdas utilizadas, estas también proporcionan un medio útil para mapear un *netlis* a nivel compuertas de una tecnología a otra. Como su nombre lo indica la librería de enlace o *Link Library*, es la librería que contiene los componentes específicos referenciados en el circuito en verilog. Así bien, la librería objetivo o *Target Library*, es la librería que contiene los componentes que formaran el *netlist* de compuertas lógicas. [10].

La *Symbol Library* o librería de gráficos, contiene la representación gráfica de las celdas empleadas, esta se usa para representar esquemáticamente las compuertas en la herramienta utilizada en su interfaz gráfica. Cabe resaltar que si una de estas librerías no coincide con alguna de las celdas o archivos que se estén usando el circuito permanecerá con fallos y no se podrá lograr su fabricación [10].

6.5. IC Compiler

Integrated Circuit Compiler (*IC Compiler*) por sus siglas en inglés, es una herramienta proporcionada por la empresa de Synopsys. Esta es una herramienta líder en realizar *place and route*, implementaciones físicas, planificación de diseño, planificación jerárquica, *place* de componentes electrónicos, diseño de síntesis de reloj, enrutamiento y optimización, entre otras características.

La tecnología de diseño jerárquico que proporciona *IC Compiler* nos permite una planificación de diseño potente. En esta se pueden realizar particiones de diseños muy complejos y grandes, esta característica nos permitirá realizar varias pruebas de secciones de un circuito sin tener que probar todo el circuito integrado. Esta implementación nos permite ahorrar tiempo de diseño y podemos mitigar de forma mas clara errores existentes [11].

La tecnología de *IC Compiler* nos proporciona un análisis de enrutamiento digital Zroute, el cual utiliza algoritmos avanzados para determinar la mejor ruta posible. El enrutamiento

Zroute optimiza los espacios, las velocidades de envió de datos y mejorar la capacidad de fabricación. La tecnología de *IC Compiler* integra a la perfección diferentes validaciones de análisis, como lo es DRC y brinda soluciones para el relleno de metales. Estas validaciones permiten que el diseñador pueda mitigar los desafíos de cumplimiento de fabricación, antes de mandar a fabricar el circuito integrado [11].

La herramienta de *IC Compiler* es utilizada en la fase de *Back End* en un flujo de diseño, ya que es aquí en donde el circuito se sintetiza de manera física, cumpliendo con los requisitos proporcionados con la empresa de fabricación [11].

6.6. Floorplan

Floorplanning o diseño de piso, es un paso de diseño esencial para abordar la metodología jerárquica que se utiliza para cumplir el proceso de elaboración del circuito integrado en una síntesis física. Este diseño depende del tipo del circuito que se trabaje. *Floorplanning* nos proporciona retroalimentación temprana evaluando las decisiones arquitectónicas del diseño, estimando áreas del chip y congestiones causadas por las conexiones entre componentes [12].

Mientras la tecnología avanza, el tamaño del circuito integrado disminuye. Esto agrega complejidad al diseño y es por esta razón que se ve la necesidad de desarrollar un buen *Floorplan*, que sea mucho más crítico para la calidad y funcionalidad del circuito integrado [12].

Tener un buen *Floorplan* es importante, ya que es el primer paso en el diseño físico del circuito integrado. Así bien, es importante porque el plano de la planta resultante afecta a los pasos posteriores en la síntesis física. La colocación de componentes y las rutas de conexiones que se analizan, conocidas también como *Place and Route*, que se explicarán más adelante son afectadas [12].

El objetivo de tener un diseño de piso es optimizar costo, ya que es en este diseño en donde se delimita el tamaño del plano, dicho de otra forma, el *Floorplan* se correlaciona directamente con el costo del chip en silicio. Cuanto mayor sea el área, mayor es el costo del circuito en silicio. También se consideran otros factores que influyen en el costo, como la longitud del "cable". Una longitud de "cable" mas corta no solo reduce el retraso de la señal, sino que también facilita la interconexión entre componentes en la etapa de *Route* [12]. A continuación se presenta la Figura #4, la cual muestra el *Floorplan* de un circuito.

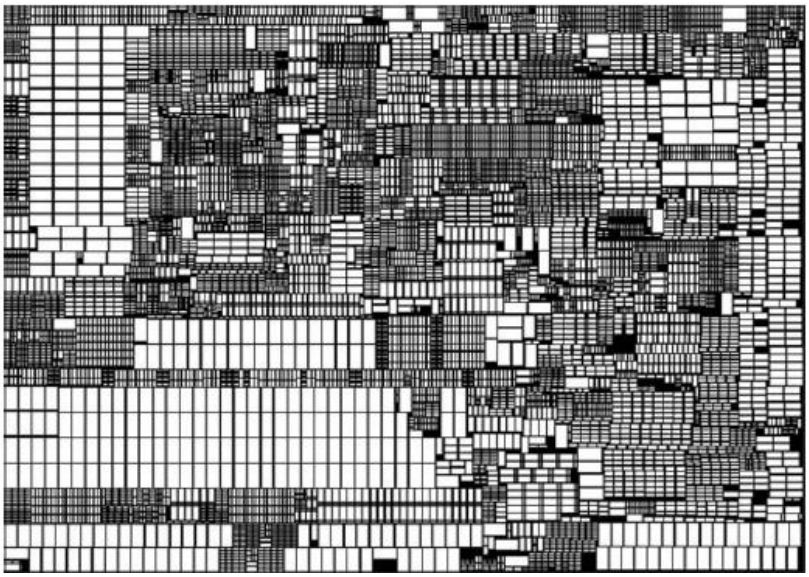


Figura 4: *Floorplan* de un circuito con 9800 modulos, teniendo espacio muerto de 3.44 por ciento en total [12]

La herramienta *IC Compiler* es capaz de soporta diferentes métodos que hacen que la creación del *Floorplan* cumpla con los requisitos de diseño que se estén utilizando. El primero de ellos recibe el nombre de *Channeled Floorplan*, este mantiene un espacio entre bloques para realizar el *Placement* y las celdas a nivel macro. El siguiente *Abutted Floorplan* es una metodología que hace que los bloques dentro del *Floorplan* se mantengan unidos y a su vez la herramienta no asigna espacios para realizar el *Placement* de celdas macro. El *Narrow-Channeled Floorplan* es el tercero de ellos, y es una combinación de los dos antes mencionados [13].

6.7. Place and Route

Placement o colocación es el proceso de determinar la ubicación que tendrán los dispositivos de nuestro circuito en la superficie matriz. El *Placement* es una etapa importante de nuestro diseño ya que esta afecta a ciertas áreas de nuestro circuito, por ejemplo, el rendimiento, la distribución de calor y el consumo de energía. El *Placement* tradicionalmente se aplica después de la etapa de síntesis lógica, *Floorplan* y antes de la etapa de *Route*. Debido al incremento en la tecnología, este proceso ha sido de suma importancia para el desarrollo de los circuitos integrados [12].

Después del *Placement*, el *Route* o ruteo toma lugar, este es un proceso que determina las rutas precisas para las redes que existen en el diseño del chip. Este proceso se encarga de la interconexión entre los componentes del circuito. El ruteo también realiza las conexiones que van del núcleo del circuito a los pines que se unen al empaquetado. Estas rutas deben satisfacer las reglas de diseño proporcionadas por la empresa encargada de su fabricación para garantizar que el chip puede ser fabricado [12].

Uno de los objetivos más importantes del *Route* consiste en que el circuito se encuentre

al cien porciento completo en sus conexiones, ya que si hacen falta conexiones este no podrá realizar tarea alguna. Como ya se hizo mención, otro objetivo es reducir los "cables" que interconectan los componentes, esto con el fin de reducir costos y que la red satisfaga tiempos de ejecución [12].

Un chip contiene miles de millones de transistores y millones de redes que los interconectan, por lo tanto, en el diseño moderno de circuitos a gran escala se requiere realizar el proceso de *Route* mediante un algoritmo en el que se adoptan dos enfoques: el enrutamiento global y el detallado. El primero de ellos divide las regiones en mosaicos y decide las rutas para todas las redes analizando mosaico por mosaico, mientras que el enrutamiento detallado determina las pistas y vías exactas para las redes [12]. La Figura #5 muestra el *Place* de los componentes y los dos enfoques de ruteo antes mencionados.

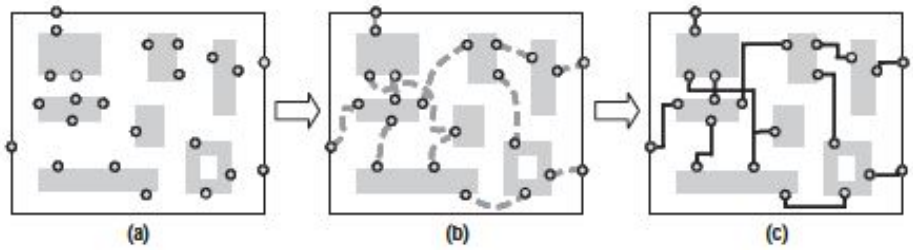


Figura 5: (a) *Place* de componentes, (b)Enrutamiento global, (c) Ruteo detallado [12]

CAPÍTULO 7

Metodología a seguir

Después de una extensa investigación sobre como realizar un buen diseño físico se determino que se debe de conocer a fondo el circuito que se estará trabajando. De igual forma se pudo concluir que este proceso es muy extenso y se requiere de varias personas para llevar a cabo el desarrollo de la síntesis física para cualquier circuito integrado que se desee fabricar.

La metodología planteada trata de exponer el orden jerárquico que se requiere para realizar un circuito básico que no sea muy complejo. Esto no quiere decir que no exista otra forma de hacerlo y en otro orden. Debido a que el proceso es muy extenso este se presenta en un diagrama dividido en cinco partes. La Figura #6 muestra el inicio del proceso, el cual abarca el tema de la preparación del software. La Figura #7 muestra el segundo paso en el flujo del diseño, en esta se muestra el proceso para configurar el *Floorplan* desde cero. El tercer paso que se muestra en la Figura #8, abarca la implementación de los componentes dentro del *Floorplan*. El cuarto paso en el flujo de diseño es elaborar el *Route* entre componentes y las conexiones necesarias de alimentación para el circuito. Este paso se puede observar en la Figura #9. La Figura #10 muestra el quinto paso que explica como guardar el trabajo y como exportarlo para que sea utilizado en la siguiente etapa del flujo de diseño del circuito integrado.

Los circuitos básicos que se tomarán como ejemplo a lo largo del desarrollo del trabajo son: una compuerta *Not*, un *Full Adder* y un *Ripple Carry Adder* (RCA). El fin de utilizar estos tres circuitos es dar una explicación a detalle de las configuraciones utilizadas. Además esto servirá para realizar un proceso modular y jerárquico. Los circuitos no presentan grandes niveles de abstracción, por lo que serán una herramienta para el aprendizaje. La metodología presente pretende establecer las bases que permitirán desarrollar circuitos complejos en el futuro.

La metodología a emplear es la siguiente:

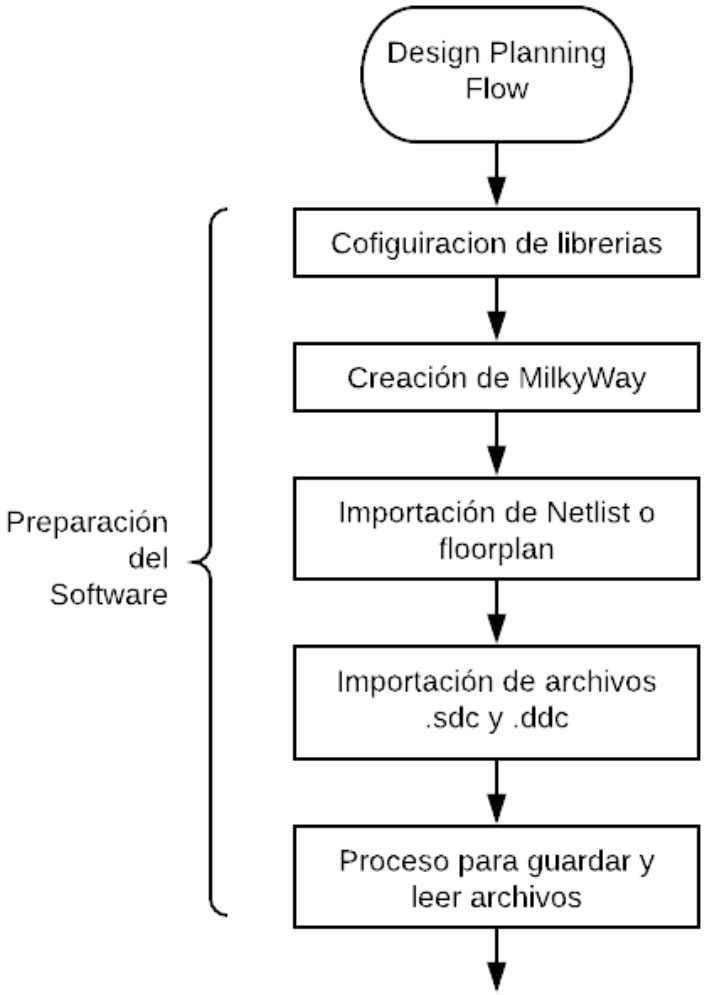


Figura 6: Primera etapa en el flujo de diseño. Preparación del software.

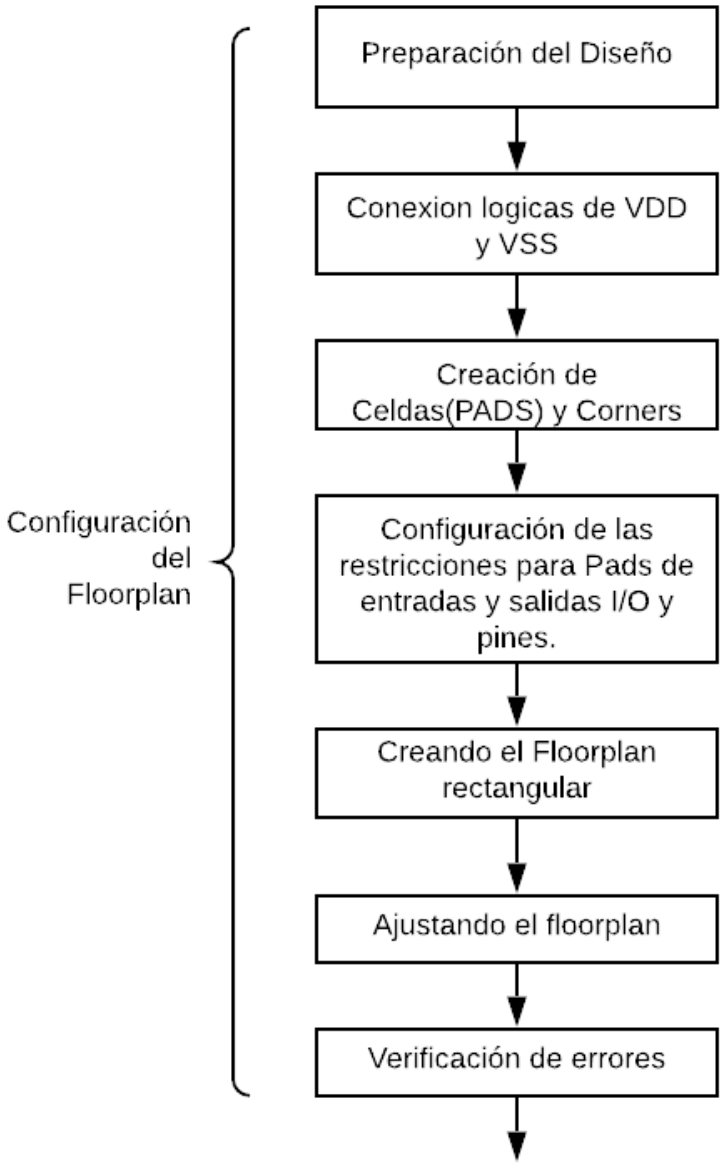


Figura 7: Segunda etapa en el flujo de diseño. Configuración del *Floorplan*.

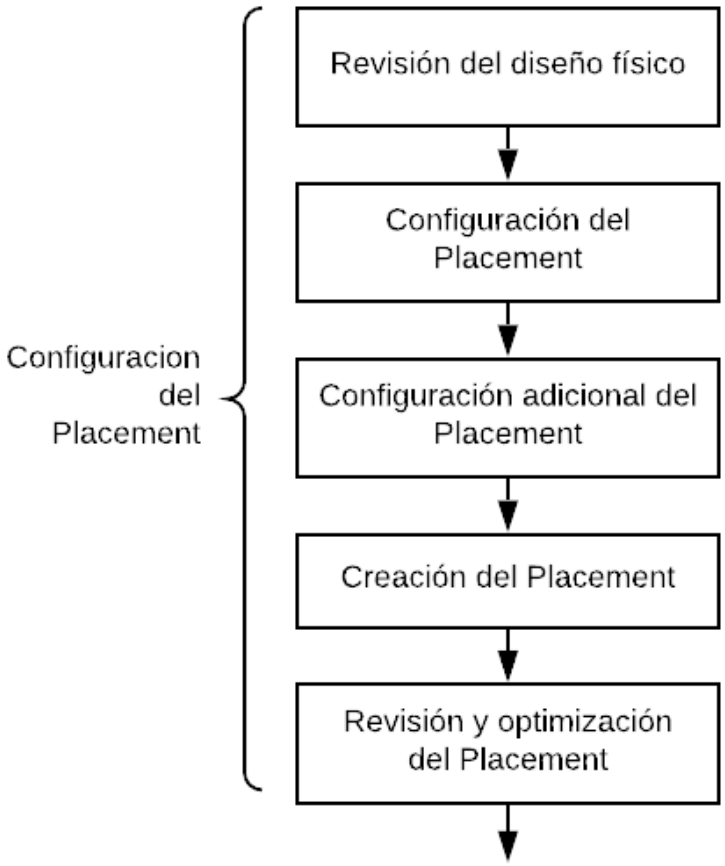


Figura 8: Tercera etapa en el flujo de diseño. Configuración del *Placement*.

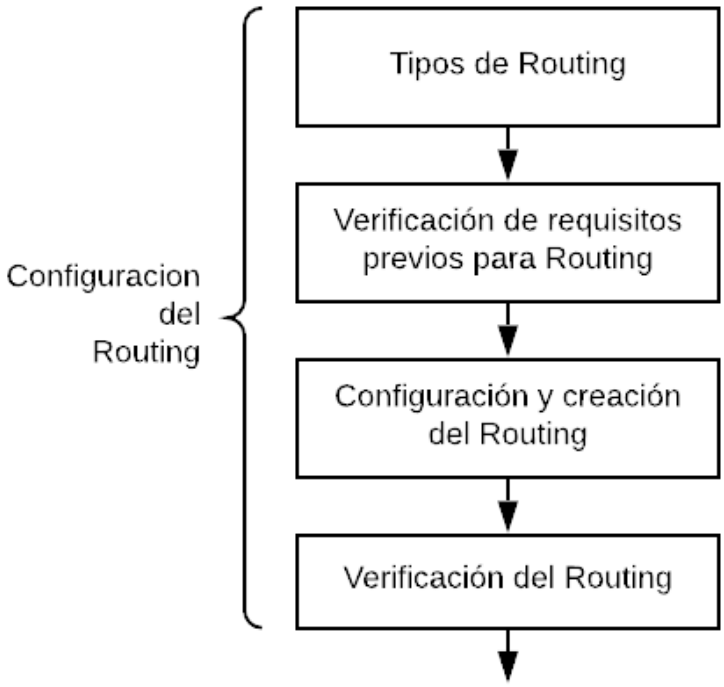


Figura 9: Cuarta etapa en el flujo de diseño. Configuración del *Routing*.

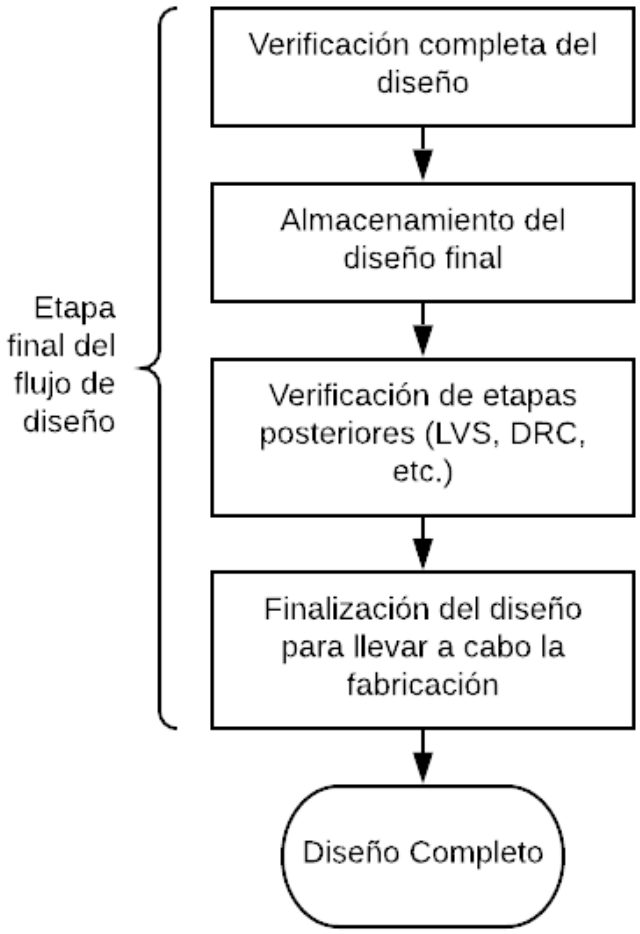


Figura 10: Quinta etapa en el flujo de diseño. Finalización de este.

CAPÍTULO 8

Creación del *Floorplan*

8.1. Preparación del diseño

Después de tener configuradas las librerías con las que se trabajaran, y creada la *Milky-Way*, es posible comenzar con el diseño y configuración del *Floorplan*. Este trabajo no se centrará en la configuración de la primera etapa de la síntesis física, sin embargo se incluirá en la sección de anexos un *script* en donde se presenta como llevar a cabo este procedimiento.

Para realizar el diseño del *Floorplan* se debe saber que existen tres tipos. Estos fueron mencionados anteriormente en la sección de marco teórico, los cuales son: *Channeled Floorplan*, *Abbuted Floorplan* y el *Narrow-Channeled Floorplan*. Existen varias formas de hacer un *Floorplan*. En este trabajo se desarrollara un *Floorplan* desde cero.

Para hacer un buen diseño de *Floorplan* será necesario preparar y configurar varios puntos importantes antes de crearlo. El primer paso es leer el archivo *netlist*, luego será necesario configurar las conexiones lógicas del *netlist* existentes y crear instancias únicas de los módulos de este.

Posteriormente se mostrará cómo crear y configurar los *Pads* de entradas y salidas (I/O), *Corners* y *Pads* de alimentación. Los *Pads* pueden ser creados tanto en *IC Compiler* como en *Design Vision*.

Como se mencionó antes los circuitos a utilizar serán tres: una compuerta *Not*, un *Full Adder* y *Ripple Carry Adder*.

8.2. Conexiones lógicas de VDD y VSS

Teniendo en cuenta la sección anterior, se iniciará la configuración del *Floorplan* con la configuración de las conexiones lógicas de VDD y VSS en el diseño a trabajar. El uso del siguiente comando `derive_pg_connection` es importante, ya que permite agregar a todo el circuito las configuraciones necesarias para establecer las conexiones lógicas dentro del trabajo a desarrollar. Esto quiere decir que el comando `derive_pg_connection` crea una red de alimentación al diseño, conectando cada red creada a los pines del circuito. Físicamente el comando no realiza ninguna modificación, por lo que la síntesis física hasta el momento con la compuerta *Not* se vería como se muestra en la Figura #11.

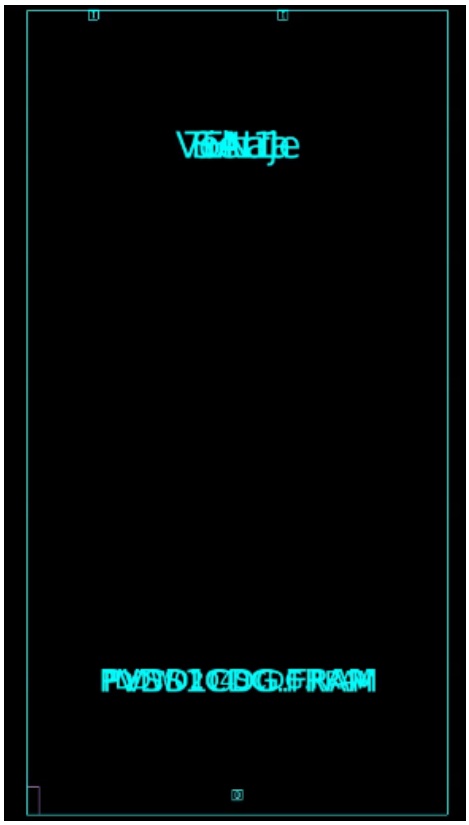


Figura 11: *Floorplan* con el comando "derive_pg_connection"

Para tener una mejor visibilidad de los cambios que se estarán haciendo se deberá mover cada celda a una posición única. De esta manera se obtendrá un mejor orden del diseño que se esté trabajando. Para lograr esto se debe colocar el *mouse* sobre la celda. Luego haciendo *click* derecho sobre ella aparece la opción *mover/dimensionar*. Al seleccionar mover la celda se pondrá de color amarillo y se podrá arrastrar a cualquier lugar del área de trabajo. Una vez desplazadas las celdas se verá de la forma mostrada en la Figura #12.

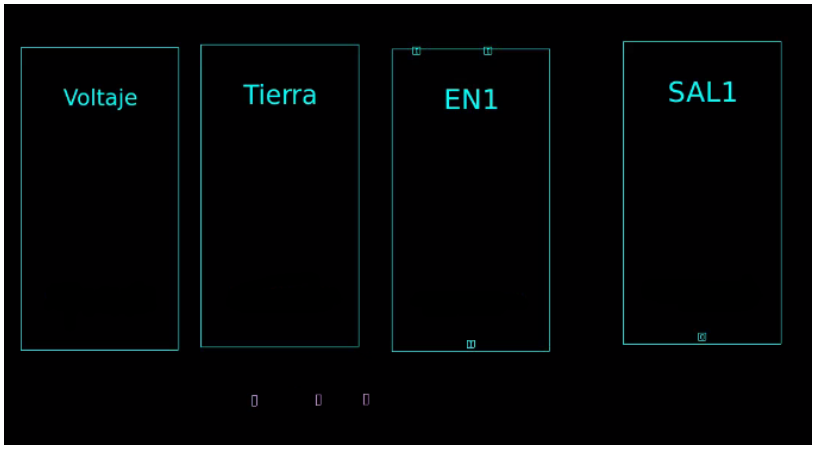


Figura 12: Moviendo celdas en el *Floorplan*.

Para usar el comando `derive_pg_connection` se puede hacer mediante dos formas, el primero es en modo automático y el segundo es en modo manual. En modo automático el comando crea las conexiones basadas en una librería que debe ser definida. Esta librería se conoce como *logic librarie*. Además el modo automático soporta diseños que tengan mas de un solo voltaje. Sin embargo en nuestro caso el modo que se usará será el modo manual. Con el modo manual se tendrá mucho mas control sobre lo que se está trabajando. Así mismo si los circuitos no son muy extensos, se podrá aprender mejor sobre cada detalle.

El realizar las conexiones lógicas de voltaje y tierra es de suma importancia. Es aquí en donde se crea una instancia única de los módulos que tenemos o no en el *netlist*. Debido a esto se debe realizar al inicio de la creación del *Floorplan*. Este comando se debe de ejecutar cada vez que nosotros modifiquemos las conexiones de voltaje y tierra, al igual que antes de guardar el trabajo en un *netlist* de verilog. Esto se realiza con el propósito de que la información sobre las conexiones lógicas no se pierda.

Debido a que el comando `derive_pg_connection` se utilizará de manera manual, existen parámetros que pueden ser modificados. Por ejemplo, se podrá definir cuales son los pines de voltaje y tierra. Se podrá definir cuales son las *nets* de voltaje y tierra, al igual que crear puertos para cada *net* especifica que se realice.

Para configurar cada uno de estos parámetros se deberá ejecutar el comando seguido de sus parámetros. Para lograr esto en el diseño que se esta trabajando se requerirá que las celdas de voltaje y tierra estén conectadas. Estas deben estarlo tanto para pines como para las *nets*. El comando con las conexiones se vería de la siguiente manera, `derive_pg_connection -power_net VDD -power_pin vdd -ground_net VSS -ground_pin vss`. Si se quisieran conectar las *nets* de voltaje y tierra, se deben de agregar los parámetros respectivos, por ejemplo `derive_pg_connection -power_net VDD -ground_net VSS`.

En este ejemplo las conexiones lógicas se llaman VDD y VSS, que hacen referencia a voltaje y tierra respectivamente. Si se quisiera modificar con otros nombres, se deberá de escribir de la siguiente manera: el comando `derive_pg_connection` seguido de los parámetros `-power_net "nombre para voltaje" -ground_net "nombre para tierra"`.

El comando completo para realizar las conexiones lógicas es, `derive_pg_connection`

`-power_net "nombre para voltaje" -ground_net "nombre para tierra"`. A continuación la linea de comando muestra la forma completa de las conexiones lógicas.

```
derive_pg_connection -power_net "nombre para voltaje" -ground_net "
   nombre para tierra"
```

De igual manera es muy útil saber si ya existen conexiones lógicas en un diseño de *Floorplan* que se ha importado, o las conexiones lógicas que se han realizado. Para conocer mas sobre esto se utilizará el siguiente comando, `report_cell_physical`. Al comando se debe agregar el parámetro de conexiones `-connections`. El comando completo se ve de la siguiente manera, `report_cell_physical -connections`. Este comando brinda la información de las conexiones lógicas en el diseño físico en esta instancia. La Figura #13 muestra el reporte que genera la herramienta al pasar dicho comando al diseño de la compuerta *Not*. La linea de comando tendrá la forma siguiente.

```
report_cell_physical -connections
```

```
icc_shell> report_cell_physical -connections
************************************************
Report  : cell_phyiscal
        -connections
Version : O-2018.06-SP5
Cell Name : NOT_IO.CEL;1
Date    : Sat Aug 29 11:15:42 2020

************************************************
--------------------------------------------------------------------
CONNECTION INFORMATION
--------------------------------------------------------------------
Connections for cell 'Compuerta':
    Input Pins          Net
    ---------------     ------------
    A                   A_w

    Output Pins         Net
    ---------------     ------------
    Y                   Y_w

    PG Pins             Net
    ---------------     ------------

    Diode Pins          Net
    ---------------     ------------

--------------------------------------------------------------------
CONNECTION INFORMATION
--------------------------------------------------------------------
Connections for cell 'Tierra':
    Input Pins          Net
    ---------------     ------------

    Output Pins         Net
    ---------------     ------------

    PG Pins             Net
    ---------------     ------------
```

Figura 13: Reporte utilizando el comando report_cell_physical -connections para una compuerta *Not*

8.3. Creación de *Pads* y *Corners*

Una vez realizadas las conexiones lógicas para voltaje y tierra, VDD y VSS respectivamente, para las *nets* y para los pines del diseño, es momento de crear celdas de entradas. Estas celdas de entrada son los *Pads* y las *Corners*. El proceso de crear las celdas de entrada no puede ser opcional, sin embargo habrá que revisar el contenido del *netlist* sintetizado. Puede ser posible que las celdas ya se encuentren definidas en dicho *netlist* generado en el proceso de síntesis lógica.

Después de revisar el *netlist* sintetizado se podrá saber si es necesario crear los *Pads* y los *Corners*. Los *Pads* y los *Corners* permiten hacer las conexiones físicas en el circuito, pasando del mundo exterior hacia el corazón (*core*) del circuito integrado. Las conexiones permitirán alimentar el circuito integrado, así como conectar los valores en las entradas del circuito y conectar la salida del chip.

Los *Pads* y las *Corners* se crean al rededor del *core* del circuito que se esté trabajando, estas celdas serán instanciadas una única vez. Esto quiere decir que si el circuito descrito en el *netlist* no cuenta con una instancia de alguna celda, se debe crear. La celda se define con la estructura con la que se describe el archivo del *netlist*. Es de suma importancia conocer el *netlist* con el que se está trabajando, ya que si por alguna razón existiera una instancia de alguna celda se tendrán problemas a la hora de intentar crearlas nuevamente en esta etapa. Únicamente se podrán crear celdas con nombres que no estén definidos en todo el circuito. Debido a este requisito se decidió colocar la creación de *Pads* y *Corners* como segundo paso en el flujo de diseño para la síntesis física.

Para crear las nuevas celdas es necesario usar el comando `create_cell`. Este comando no puede agregarse sin ningún parámetro, esto se debe a que seria ilógico no definirle algún valor o algún nombre. Por lo tanto, para definir un *Pad* o *Corner* es necesario realizarlo de la siguiente manera. El comando `create_cell`, seguido de una lista con los nombres, por ejemplo `{PAD_1 PAD_2}` o `{CORNER_1 CORNER_2}`. Luego debemos de indicar el nombre de la librería a la que estamos haciendo referencia. La *reference librarie* contiene la información de la celda como tal. Por consiguiente quedaría de la siguiente manera `libreria_PAD's` o `libreria_CORNERS`.

De forma breve el comando quedará como se muestra a continuación en la Figura #14. Redactado se ve de la siguiente manera, `create_cell {PAD_1 PAD_2} libreria_PAD's`, o para *Corners*, `create_cell {CORNER_1 CORNER_2} libreria_CORNERS`.

Los *corners* y *Pad's* usados en los circuitos de ejemplo se muestran a continuación en la linea de comando.

```
create_cell {c1 c2 c3 c4} libreria_Corners
```

```
create_cell {VDD} libreria_Voltaje
```

```
create_cell {VSS} libreria_Tierra
```

Estas crean las cuatro corners que tendrá el *Floorplan*. Además se crean dos *Pad's*, uno para VDD y otro para VSS.

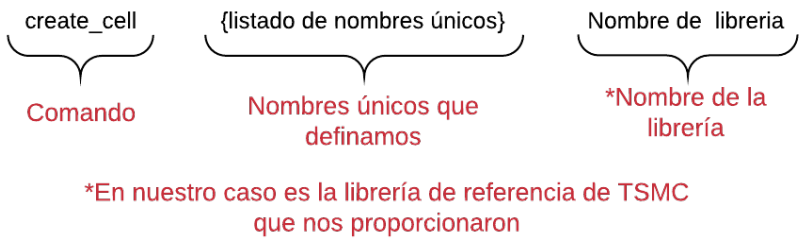


Figura 14: Uso del comando para crear *Pads* y *Corners*

Una vez creadas las *Corners* y *Pads*, no habrá ningún cambio en la vista del *Layout* que se esté trabajando. La posición en donde se crean las celas por defecto es la esquina inferior del circuito. Posteriormente se podrá aplicar transformaciones para mover cada celda instanciada en el paso anterior. La Figura #15 muestra como se vería en *IC Compiler* las celdas creadas. Los *warnings* mostrados en la Figura #15 nos indica que la celda no tiene orientación, y que por defecto el programa le asigna una. El último *Warning* nos indica que no podremos regresar un paso anterior. Los *warnings* mencionados son informativos y no se requiere que hagamos alguna acción, esto significa que no afecta al diseño.

```
icc_shell> create_cell {c1 c2 c3 c4} PCORNER
Warning: Cell PCORNER has no orientation; check FRAM library library
Warning: Cell PCORNER orientation is set to the default. (PSYN-651)
Creating cell 'c1' in design 'NOT_IO'.
Creating cell 'c2' in design 'NOT_IO'.
Creating cell 'c3' in design 'NOT_IO'.
Creating cell 'c4' in design 'NOT_IO'.
Warning: Undo stack cleared by command 'create_cell' (HDUEDIT-104)
1
icc_shell>
```

Figura 15: Celdas creadas para las *Corners*

8.4. Configuración de restricciones para *Pads* y *Pins* de entradas y salidas (I/O)

Terminada la creación de *Corners* y *Pads*, es momento de definir las posiciones y las restricciones (*constraints*). Estos *constraints* se aplican tanto a las celdas creadas en el paso anterior y a las celdas creadas en el *netlist*.

Con los *constraints* se podrá definir la localización de cada celda, su orientación y se podrá definir el orden jerárquico de cada una de estas. De la misma manera se podrá definir el *offset* o la separación que tendrán cada una de estas con respecto a las demás celdas o al *core* del chip.

Para definir los *constraints* se utilizará el comando `set_pad_physical_constraints`. La configuración de los diferentes *constraints* que se pueden modificar se realiza antes de

crear el *Floorplan*. La razón de configurarlo antes es la siguiente. El comando `set_pad_physical_constraints` reserva un espacio en el *Floorplan* próximo a crear, para cada una de las celdas que hayamos definido. Esto quiere decir que si se define la jerarquía y la posición de un *pad* o una *corner*, por ejemplo a la esquina superior derecha y luego se pasa el comando `set_pad_physical_constraints` este espacio quedara reservado en el *Floorplan*. La Figura #16 ilustra este ejemplo.

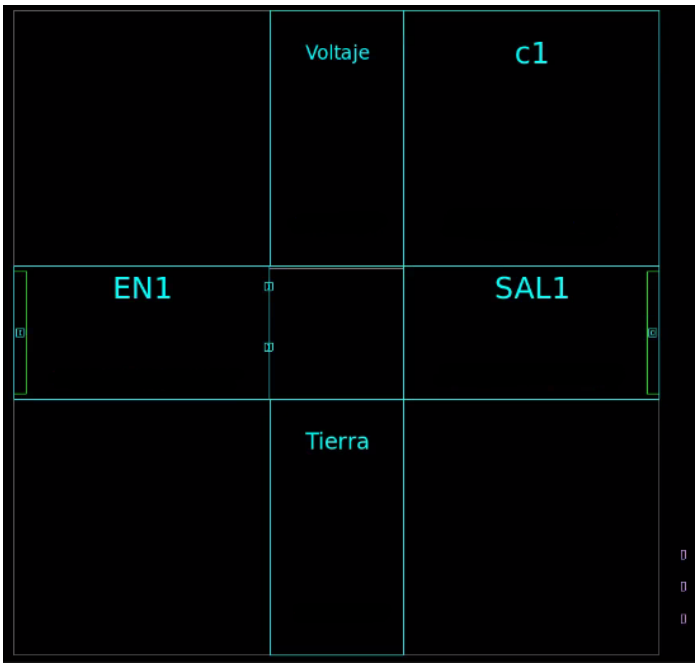


Figura 16: *Floorplan* con *constraints* para la *corner* C1 y el *pad* voltaje en la esquina superior derecha

Por defecto, el programa introduce las celdas y las separa una de otra en sentido antihorario. Por ejemplo en la Figura #16 las celdas con *constraints* eran c1 y Voltaje. Las celdas que no contenían *constraints* fueron colocadas automáticamente por el programa con una separación y rodeando al *core*.

El comando `set_pad_physical_constraints` no realiza modificaciones físicas al *Layout*. Para ver las modificaciones de los *constraints* se debe hacer el *Floorplan*, o revisar el diseño que se tiene a este momento.

De la misma forma que la sección anterior el comando por si solo no realiza ninguna función. Para modificar cada uno de los *constraints* se debe configurarlo de manera manual, para ello se utilizan los siguientes parámetros del comando. Los parámetros que se utilizaron para el circuito son: `-pad_name`, `-side` y `-order`. Aunque existen otros parámetros, para el circuito que se está desarrollando estos son los mas relevantes.

El parámetro `-pad_name` nos permite indicar la celda que queremos modificar, por ejemplo la *corner* c1, `-pad_name "c1"`. Sin embargo como el objetivo es definir las restricciones de las celdas, estas se deben agregar después de seleccionar la celda. Por ejemplo `set_pad_physical_constraints -pad_name "c1" -side 2`. En el ejemplo se seleccionó la

celda llamada c1 y se configuró a la esquina superior derecha. Si se quisiera agregar otra celda en la misma posición de la *corner* c1; esquina superior derecha, habra que pasar el parámetro `-order`. Por defecto la primera celda que se establecerá será la primeramente configurada, luego lleva un orden jerárquico. Por ejemplo si se quisiera modificar la celda Voltaje a la esquina superior derecha, el comando se verá de la siguiente manera, `set_pad_physical_constraints -pad_name "Voltaje" -side 2 -order 2`. Por defecto el valor del orden jerárquico es 0 (cero), esto es por que la celda no posee orden. La Figura #17 ilustra lo explicado.

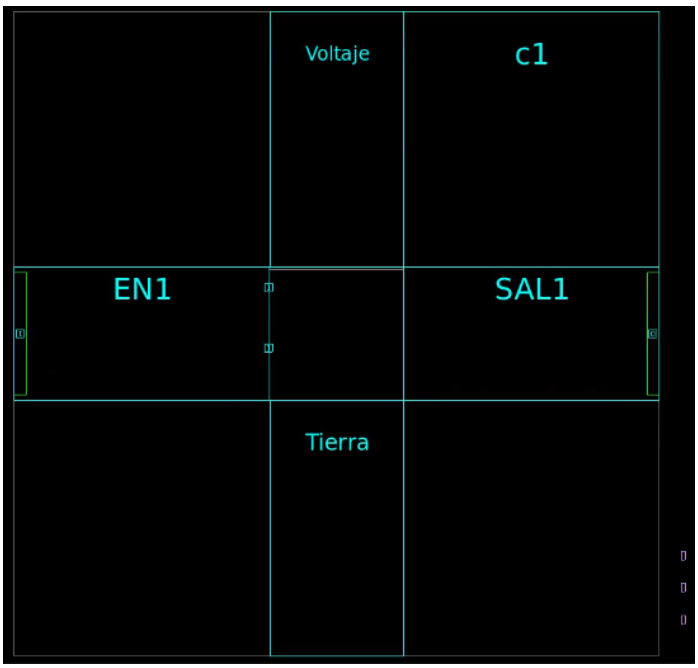


Figura 17: Celda c1 y Voltaje agregadas a esquina superior derecha, siguiendo un orden jerárquico

El orden para colocar las celdas al borde del *core* son las siguientes: 0 (cero), 1 (uno), 2 (dos), 3 (tres), 4 (cuatro). El número 0 (cero), significa que el programa no asigna orden a la celda, esta es la posición por defecto. El número 1 (uno) hace referencia al borde izquierdo del *core*. El número 2 (dos) hace referencia al borde superior del *core*. El número tres (tres) hace referencia al borde derecho del *core*. El número 4 (cuatro) hace referencia al borde inferior del *core*. La siguiente lista muestra lo explicado.

Orden de posición:

- 0, posición por defecto
- 1, borde izquierdo
- 2, borde superior
- 3, borde derecho
- 4, borde inferior

De la misma manera que se puede definir la posición y el orden jerárquico, se pueden definir otros parámetros. A la hora de crear una celda, esta tiene una orientación sobre el plano que se esta trabajando. La orientación por defecto de las celdas de los *pads* y de las *corners* es (N). Esto significa que la celda no esta rotada a ningún grado. Si se quisiera cambiar la orientación de la celda se debería de escribir el parámetro, `-lib_cell` para habilitar la orientación por defecto, seguido de `-lib_cell_orientation` para agregar la orientación.

A continuación se describe cada orientación posible:

- N, cero grados de rotación (0°)
- W, noventa grados de rotación (90°)
- S, ciento ochenta grados de rotacion (180°)
- E, doscientos setenta grados de rotacion (270°)
- FN, cero grados de rotación (0°) y espejeado sobre el eje X.
- FS, cero grados de rotación (0°) y espejeado sobre el eje Y.
- FW, noventa grados de rotación (90°) y espejeado sobre el eje X.
- FE, noventa grados de rotación (90°) y espejeado sobre el eje Y.

Así mismo es de mucha utilidad poder definir el *offset* o el desplazamiento de cada celda con respecto al borde del *die* en donde fue colocado. Para ello nos apoyamos del siguiente parámetro `-offset`, al cual le agregamos la distancia en micrómetros para desplazarlo. Por ejemplo, `-offset 200`. En la siguiente figura, Figura #18 se puede observar el ejemplo.

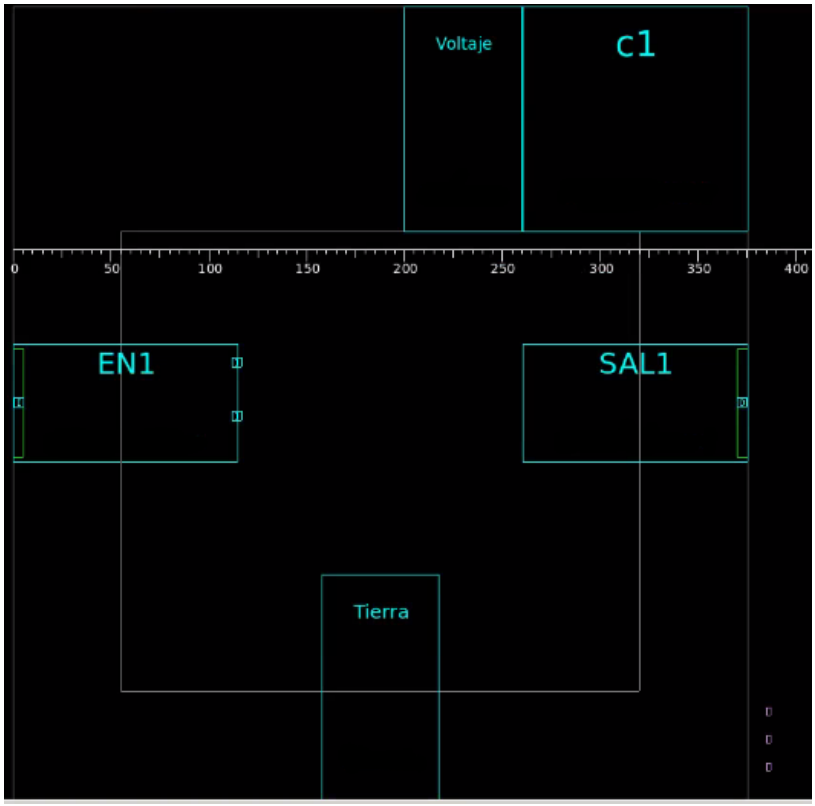


Figura 18: Ejemplo de *offset* de 200 micrómetros para la celda *Voltaje*

Siguiendo con estas modificaciones, es posible ajustar un *offset* entre celdas. El fin de esto es tener un mejor diseño y evitar interrupciones. Sin embargo se desea reducir espacios, dejar pegadas las celdas es una mejor opción. Para lograr este *offset* se tienen dos parámetros, el `-min_right_iospace` y el `-min_left_iospace`. Cada uno de ellos acepta un valor positivo el cual indica la separación en micrómetros respecto a la otra celda. Para ejemplificar esto se usará una separación de 50 micrómetros entre la *corner* c1 y el *pad* de Voltaje y para el lado izquierdo del *pad* Voltaje. La Figura #19 ilustra el ejemplo.

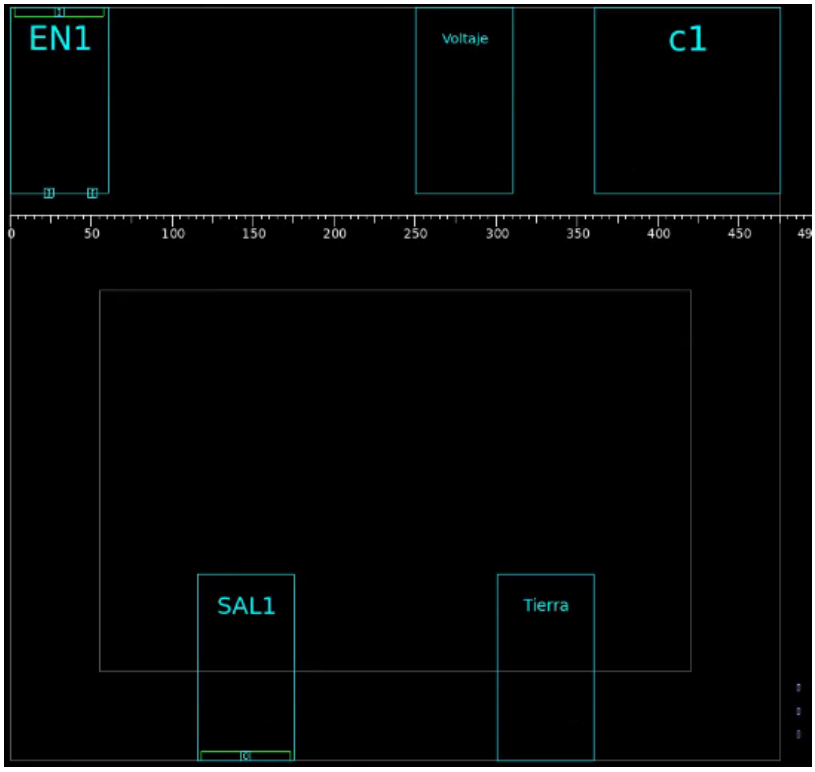


Figura 19: Ejemplo de *offset* de 50 micrómetros entre el *pad Voltaje* y la *corner c1*

Es posible notar que el *offset* que se definió con anterioridad ha cambiado. Esto se debe a que ahora se debe de cumplir otra restricción. Esta restricción hace que el *pad* Voltaje debe mantener una separación de lado a lado de 50 micrómetros.

De la misma manera que se ha trabajado las configuraciones para los *Pads*, se pueden trabajar configuraciones para los pines (*Pins*). Para configurar cada *constraints* para los pines existe otro comando y otros parámetros que podremos modificar. El comando para configurar los *constraints* sobre los pines es el siguiente: `set_pin_physical_constraints`.

Es posible modificar el ancho de los pines, las capas que se desea conectar, el *offset*, la profundidad, etc. Sin embargo para modificar los pines se debe conocer a mas detalle las restricciones que recomienda el fabricante. Para el circuito que se esta desarrollando, esta configuración no se utilizó.

Si por alguna razón, la posición del *pad* colocado o la configuración del *pin* no fuera la ideal que se haya definido, esta se puede cambiar. Después de creado el *Floorplan* se podrá regresar a esta sección y modificar dichos *pads* y dichos *pines*. Para conocer la posición y modificaciones de los *pads* y *pines*, se puede saber mediante el comando `report_pin_pad_physical_constraints`. Al comando podremos definirle si se quiere únicamente el informe de *pines* o de *pads* o de ambos. Para solicitar cada informe se agregarían los siguientes parámetros: `-pin_only` o `-pad_only`. Si se quisiera el informe de ambos, no se debe pasar ningún parámetro. El reporte de la última configuración que se realizó se muestra en la Figura #20.

```
icc_shell> report_pin_pad_physical_constraints
set_pad_physical_constraints -pad_name {c1} -side 2 -min_left_iospace 0 -min_right_iospace 0
set_pad_physical_constraints -pad_name {Voltaje} -side 2 -order 2 -offset 200 -min_left_iospace 50 -min_right_iospace 50
```

Figura 20: Reporte de los *constraints* para los *pads* y pines configurados

Si se presentara alguna modificación errónea o no deseada, se podrá remover los *constraints* de cada *pin* y *pad* creados o ya existentes. Para lograr esto se utiliza el comando `remove_pin_pad_physical_constraints`. Este comando remueve todos los *constraints* previamente definidos. Si se quisiera solo eliminar los *constraints* de los *pads* se debe utilizar el comando seguido del parámetro `-pad_only`. Si fuera el caso de los pines se usaría el comando seguido del parametro `-pin_only`.

Para guardar un archivo con la información de los *constraints* en la *Milkyway* debemos de ejecutar el comando `write_pin_pad_physical_constraints`. A este comando se debe agregar la librería abierta que se esta manejando, además del nombre de la celda. Seguido de esto se debe decir que se quiere guardar, por ejemplo: el orden o la locación. Definido esto se debe indicar si solo se quieren los *pads* o pines. El archivo se guardara como un archivo *.tcl*, con el nombre que se haya definido.

De la misma manera se puede acceder a cualquier archivo que contenga la información de los *constraints* haciendo uso del comando `read_pin_pad_physical_constraints`. Los últimos dos comandos presentados son de utilidad para circuitos mas grandes, con muchos *pads* y muchos pines. Estos comandos guardan y leen la información de los *constraints* en archivos con extensión *.tcl*.

8.5. Creación del *Floorplan* rectangular

El diseño y la realización del *Floorplan* es de suma importancia. Es aquí en donde se crea un espacio físico que contendrá las características de nuestro circuito. El *Floorplan* nos permite visualizar el área que tendrá el *core*. Así mismo nos brinda información de cómo se comportará el circuito que se este diseñando.

Para iniciar se debe conocer el circuito que se estará implementando. El objetivo es tener una orientación de lo que queremos realizar. En este caso se seguirá utilizando como ejemplo una compuerta *Not*, un *FullAdder* y un *RCA*.

De la misma manera debemos conocer que la herramienta soporta tres metodologías para determinar los requisitos del diseño. Los *Channeled*, *Abutted* y los *Narrow-Channeled Floorplans* se describen de mejor manera en la sección de marco teórico.

IC Compiler nos permite realizar diferentes *Floorplans*. Conoceremos sobre la creación del *Floorplan* inicial y la forma que este puede tener. Existen tres formas de *Floorplan* diferentes que se pueden crear, estos son: *Floorplan* rectangular, *Floorplan* rectilíneo y el *Floorplan* complejo. El *Floorplan* mas básico es el rectangular, en este trabajo se estará implementando este diseño.

Cada *Floorplan* se trata como una serie de puntos o vértices para su creación. El crear *Floorplans* a partir de vértices resulta útil ya que se podrá delimitar una única forma y

tamaño.

Conociendo estas características se explicará cómo realizar el *Floorplan* y cómo configurarlo. Como se mencionó, el *Floorplan* que se creará será el rectangular debido a que el circuito no es muy complejo.

8.5.1. *Floorplan* rectangular

Después de configurados los *constraints* para los pines y *pads* se podrá crear el *Floorplan*. A continuación se explicará el comando que se usará para crear el *Floorplan* rectangular. Este comando utiliza varios parámetros para definir mejor un espacio físico del circuito.

Existen dos maneras para crear el *Floorplan*. La primera manera es crearlo por medio de la interfaz gráfica de *IC Compiler*. Para ello nos apoyaremos de la vista del *Layout*. Para crear el *Floorplan* seleccionaremos ***Floorplan*** en la interfaz grafica, seguido de ***create Floorplan...***. Seleccionado esto nos aparecera una ventana como se muestra en la Figura #21.

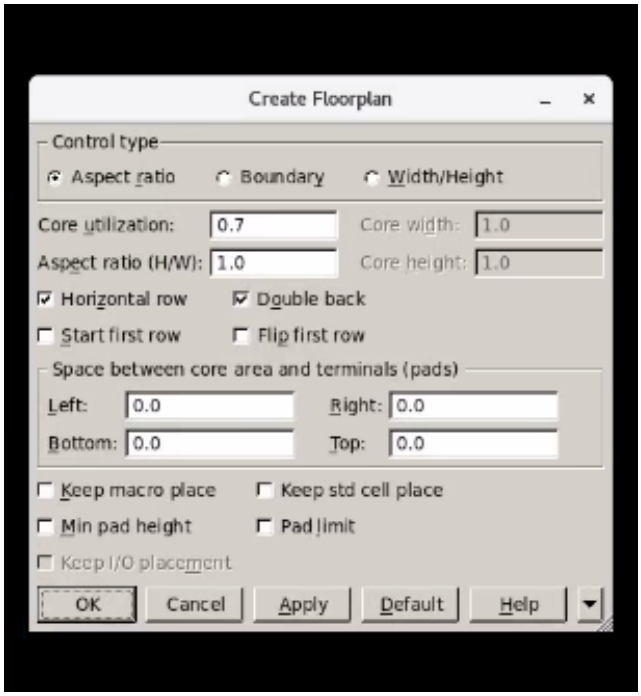


Figura 21: Creación del *Floorplan* por medio de la interfaz gráfica.

La segunda manera es por medio de la línea de comandos. Es de esta forma que se trabajará, pero de las dos formas se obtienen los mismos resultados. Para crear el *Floorplan* nos apoyaremos del siguiente comando `create_floorplan`. Este comando crea el espacio del core, y las dimensiones físicas que requiramos. También coloca el anillo de I/O (entradas y salidas) para los *pads* y *corners* definidos con anterioridad.

El comando `create_floorplan` crea un *Floorplan* con dimensiones y características por

defecto. De manera similar a los otros comandos podremos modificar cada una de estas propiedades. Por ejemplo podremos especificar el tamaño de área del *core* del *Floorplan*. Además se podrá definir parámetros como el uso del área en el *core*.

Esto último es de suma importancia. Ya que, si los requisitos de fabricación indican una cantidad máxima de celdas por área, este parámetro se podrá modificar para cumplirlos. El comando `create_floorplan` además crea un espacio para colocar las celdas dentro del *core*. Estos espacios que se definen reciben el nombre de *Rows*.

A continuación en la Figura #22 se muestra el *Floorplan* creado para la compuerta *Not*. Este ejemplo solo contiene el comando `create_Floorplan` sin ningún otro parámetro. Así mismo, este ejemplo ya contiene las cuatro *corners* configuradas y colocadas. De igual forma ya contiene los *pads* definidos para: entradas, salidas, voltaje y tierra. La Figura #23 muestra el *Floorplan* creado para el circuito (*Full Adder*). La Figura #24 muestra el *Floorplan* creado para el circuito (RCA).

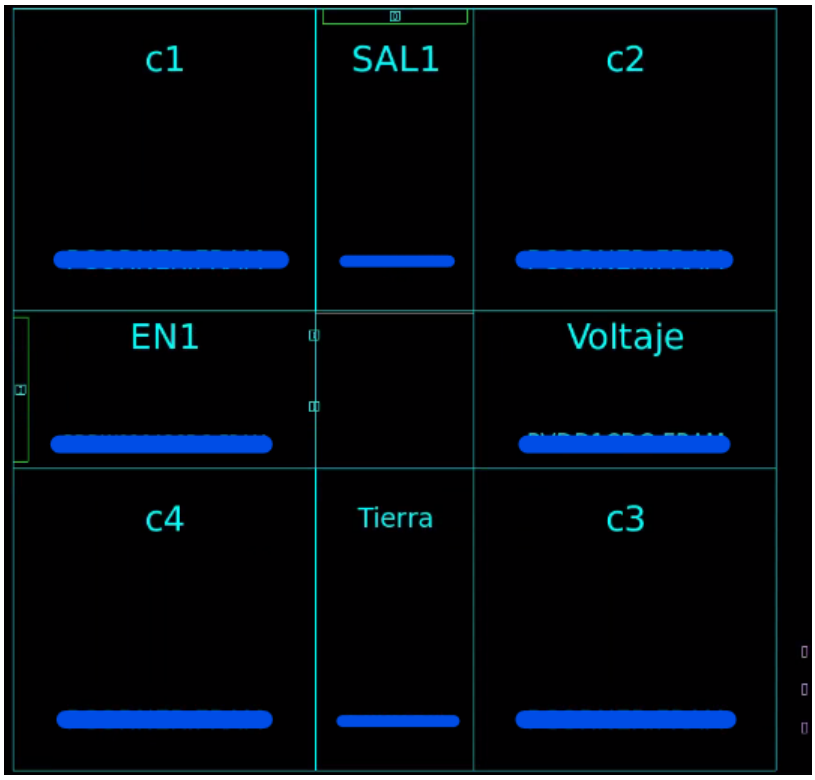


Figura 22: Creación de *Floorplan* por defecto, compuerta *Not*.

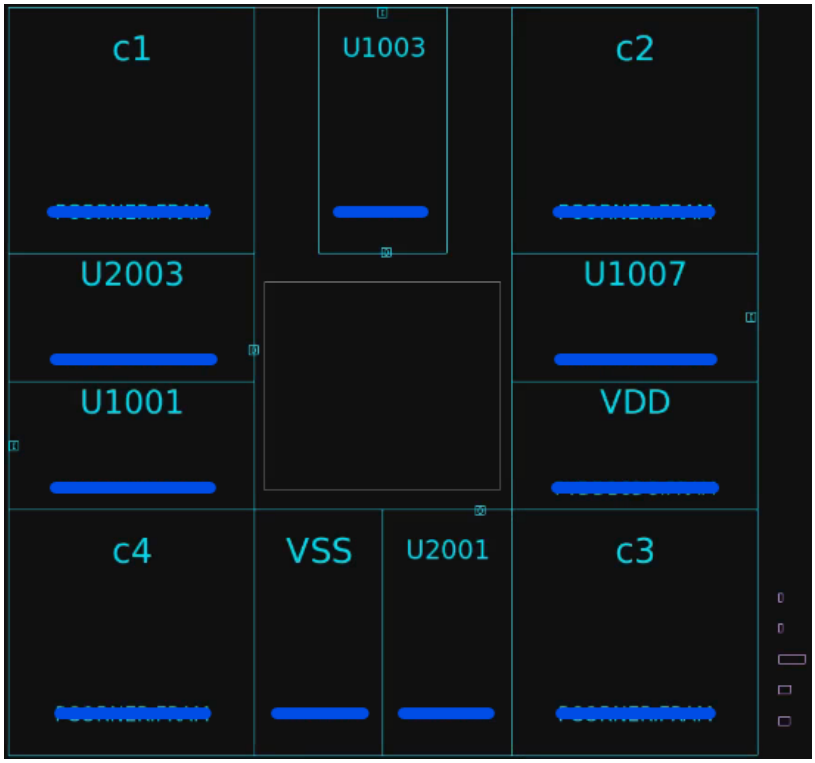


Figura 23: Creación de *Floorplan* por defecto, circuito *Full Adder*.

Figura 24: Creación de *Floorplan* por defecto, circuito *RCA*.

En la Figura #22 se puede observar que existen tres celdas que aun se encuentran fuera del espacio definido. Estas celdas conforman el circuito que se este trabajando. En el ejemplo una de la celdas es la compuerta *Not*. Las dos restantes corresponden a *buffers* para la entrada y la salida de la celda *Not*. La Figura #25 ilustra esto.

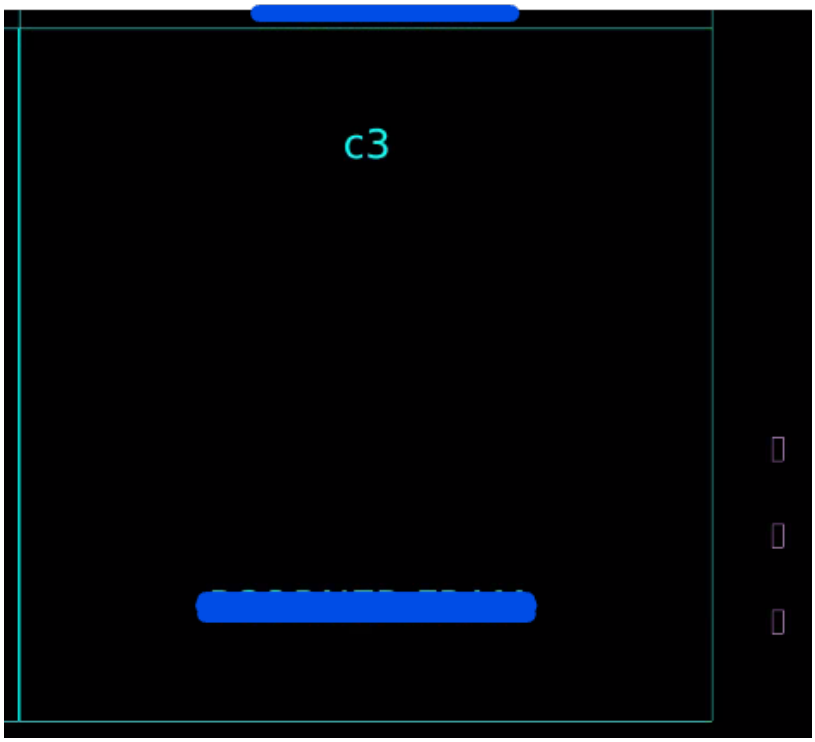


Figura 25: Celdas del circuito afuera del *Floorplan*

Cuando se crea el *Floorplan* solo se crea un espacio físico. Las celdas no se colocan adentro del *core*. Para colocar las celdas dentro del *core* se utilizan otros métodos. Esto se vera a detalle en el capitulo siguiente.

Existen varios parámetros que podremos configurar, a continuación se explicarán los mas relevantes para el circuito que se esta implementando. El comando `create_floorplan` seguido del parámetro `-control_type` permite dimensionar el área del *core*. Para esto se podrán usar tres métodos diferentes: `aspect_ratio`, `width_and_height` o `boundary`. El primer método `aspect_ratio`, dimensiona el área del *core* mediante una división. La altura del *core* se divide dentro del ancho del *core*. El alto y el ancho del *core* se modifica después. La Figura #26 ilustra el primer método.

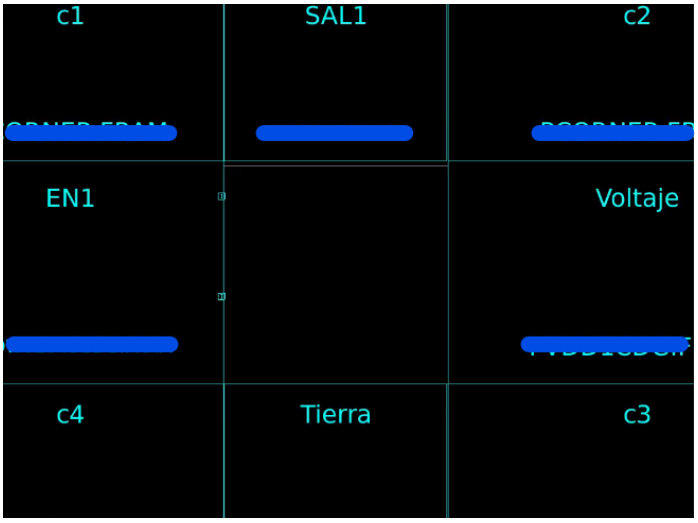


Figura 26: *Floorplan* con *aspect_ ratio*

El segundo método `width_and_height`, dimensiona el área del *core* basando su estrategia con la configuración real del alto y ancho del *core*. Es decir, si se definiera un alto de diez micrómetros y un ancho de seis micrómetros, el área del *core* mediría exactamente eso. El segundo método puede observarse en la Figura #27.

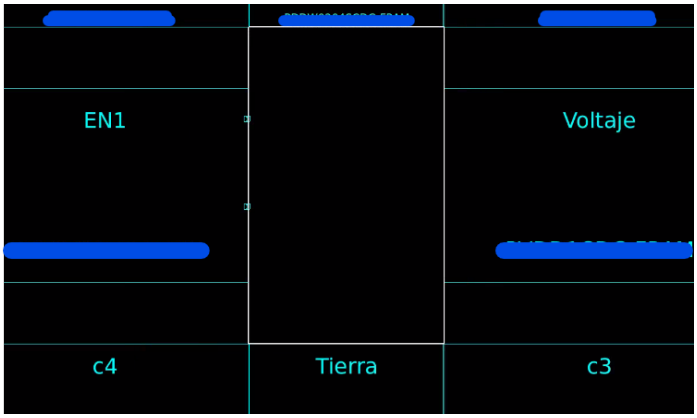


Figura 27: *Floorplan* con *width_ and_ height*

El tercer método `boundary` especifica el área del *core* mediante los limites del *Place and Rouete* en el diseño actual. Este metodo es mas complejo, y no es de tanta utilidad para el circuito que se esta desarrollando. El método por defecto que es configurado por el parámetro `-control_type` es el de `aspect_ratio`. Terminado el primer parámetro, el comando completo se escribe de la siguiente manera: `create_floorplan -control_type width_and_height -core_width 3 -core_height 30`.

Seguido de esto, podremos modificar el aspecto y las dimensiones del *Floorplan*. De la misma manera que se genera el `aspect_ratio` en el *core*, podremos definirlo en el *Floorplan* con el parámetro `core_aspect_ratio`. Este parámetro recibe cualquier valor numérico positivo, por ejemplo: 0.7, 1, 5 o 6.3. Esto quiere decir que si el valor fuera 0.5 el ancho es mas

grande que el alto por un factor de dos. Si fuera el caso del valor 5, quiere decir que el alto es cinco veces mas grande que el ancho.

Como se mencionó antes se puede establecer el porcentaje que se utiliza en el área del *core* para las celdas. Esto lo podremos lograr con el parámetro `-core_utilization`. El parámetro recibe cualquier valor positivo entre 0 y 1. El valor 0 representa el cero por ciento (0 %) del valor usado en el área por las celdas. El valor de 1 representa el cien por ciento (100 %) del área usada por las celdas. Esto quiere decir que si se define un valor de 0.75, se utilizará en el área del *core* el setenta y cinco por ciento (75 %) para colocar las celdas. El comando completo tendra la siguiente forma: `create_floorplan -core_utilization 0.75`. El porcentaje que no se usa para reservar el espacio de las celdas, se usara para reservar el espacio de las conexiones. En el ejemplo anterior se tiene un espacio reservado para las celdas del setenta y cinco por ciento (75 %) y el veinticinco por ciento (25 %) extra se reserva para el *route*. El comando `-core_utilization` solo se puede aplicar cuando se usa el comando `-control_type`.

El primer parámetro `-control_type` se rige en las dos formas de uso por el ancho y el alto del *core*. Para establecer el ancho del *core* a utilizar se usa el parámetro `-core_width`. Siguiendo la analogía, el alto del *core* a utilizar se define con el parámetro `-core_height`. Ambos parámetros modifican el ancho y el alto respectivamente. Estos parámetros solo se podrán usar si esta habilitado el comando `-control_type` con el parámetro `width_and_height`. Los comandos aceptan un valor positivo para la creación del core y esta unidad se mide en micrómetros. El comando completo para modificar el ancho y/o alto se escribe a continuación: `create_floorplan -control_type width_and_height -core_width 60.48 -core_height 98`. La Figura #28 muestra el uso del comando completo.

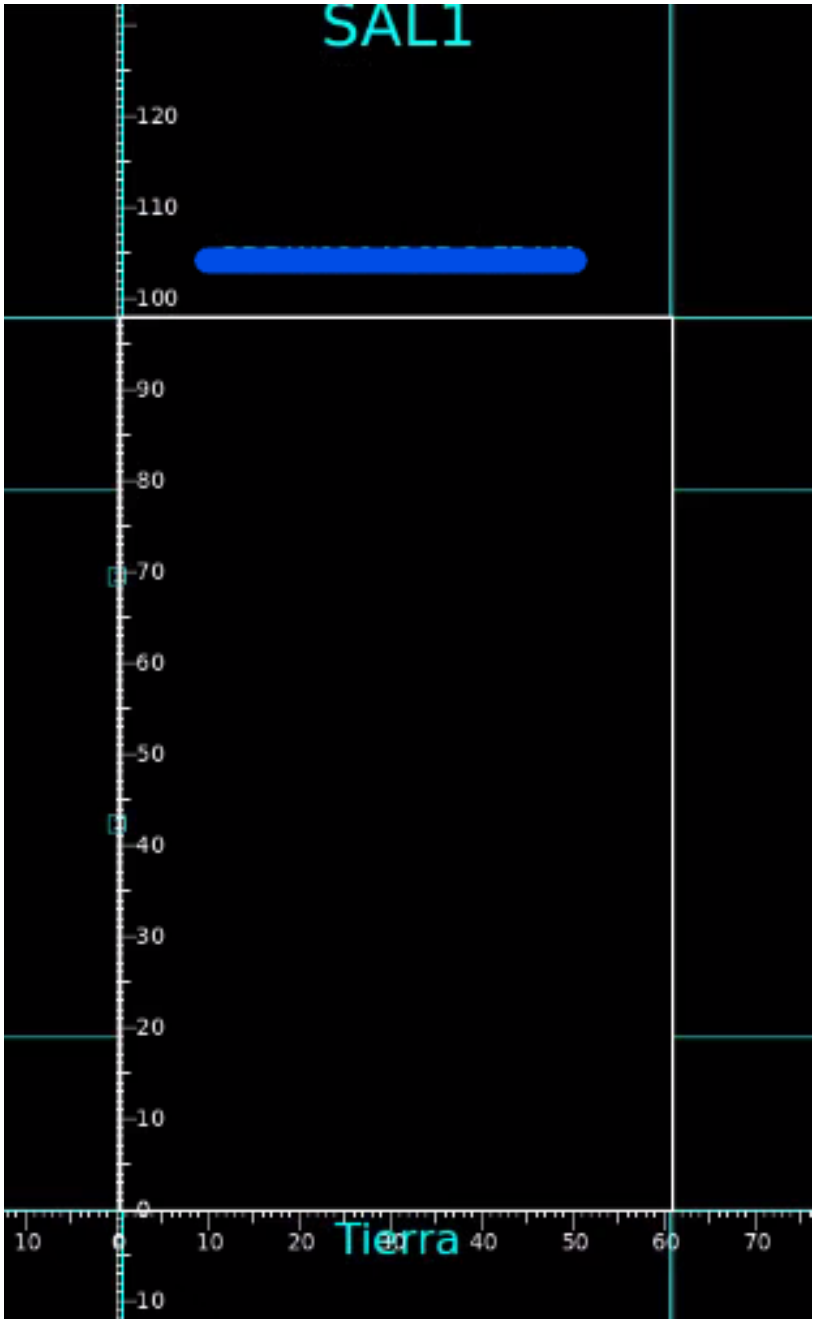


Figura 28: *Floorplan* con *width* de 60.48 y *height* de 98 micrones.

Definido la utilización del *core* y establecido su ancho y su largo, se puede iniciar a delimitar los *Rows*. Los *Rows* mencionados anteriormente crean espacios para colocar las celdas dentro del *core*. Los *Rows* o `"filas"` por su traducción a español, están establecidos en el interior del *core* de extremo a extremo. La Figura #29 ilustra los *Rows*.

Por defecto el programa crea en todo el interior del *core* los *Rows*. Esto lo hace de manera automática. El programa los coloca en orientación normal `N` y espejeados `FW`. Es decir que el primer *Row* visto en la parte inferior del *core* tiene orientación normal `N`. El

siguiente *Row* arriba del anterior se encuentra espejeado o con orientación `FW`. Esto se repite sucesivamente hasta llegar a la parte superior del *core*. La Figura #29 muestra la colección de *Rows* alternando en orientación `N` y orientación `FW`.

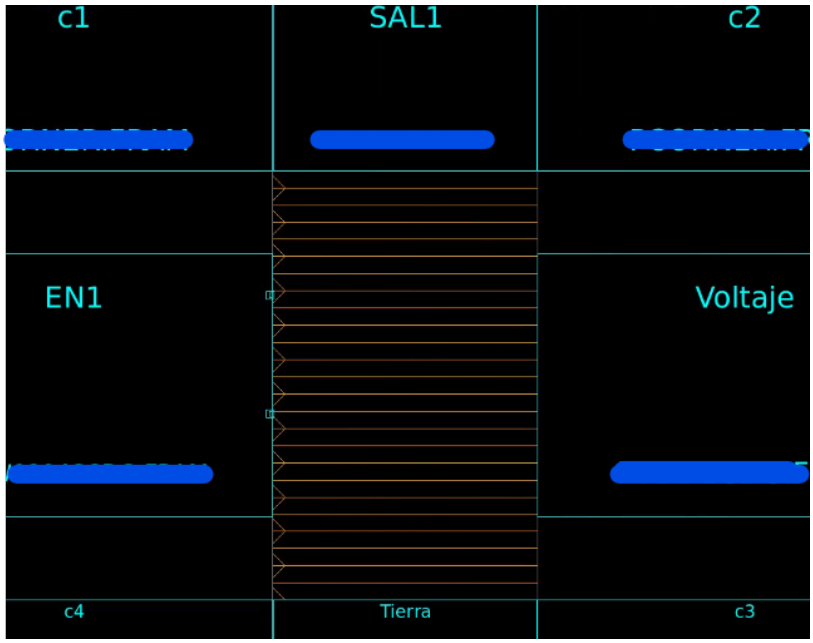


Figura 29: *Floorplan* con *Rows* por defecto

Para los circuitos que se están tomando de ejemplo, la orientación y la cantidad de los *Rows* no afecta. Sin embargo para circuitos mas complejos esto será importante. El espacio de cada *Row*, es el espacio en donde se colocarán las celdas posteriormente. La Figura #30 muestra esto.

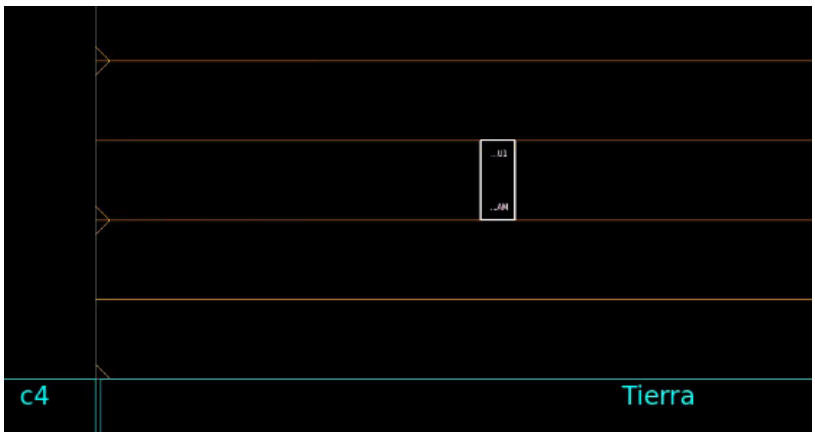


Figura 30: *Floorplan* con *Rows* por defecto, con celda *Not* colocada.

Por consiguiente es posible modificar cada *Row* establecida. Para ello debemos de seguir ciertos criterios. Se debe conocer la cantidad de celdas que contiene el circuito. También se debe saber si será necesario generar todo el *core* con *Rows* o no. Dependiendo del circuito

será necesario hacer los *Rows* de forma vertical u horizontal. Para lograr modificar los *Rows* cuando se cree el *Floorplan* serán necesarios varios parámetros. Los parámetros son los siguientes: `-use_vertical_row`, `-no_double_back`, `-start_first_row`, `-flip_first_row`. A continuación se explicará cada uno de ellos.

El parámetro `-use_vertical_row` crea los *Rows* de forma vertical, como si fueran columnas. A este parámetro no se le debe especificar algún valor o condición. La Figura #31 muestra como se vería el *Floorplan* con la configuración por defecto pero cambiando las filas a columnas. Si no se especifica que se quiere hacer el cambio, el programa establece *Rows* horizontales por defecto.

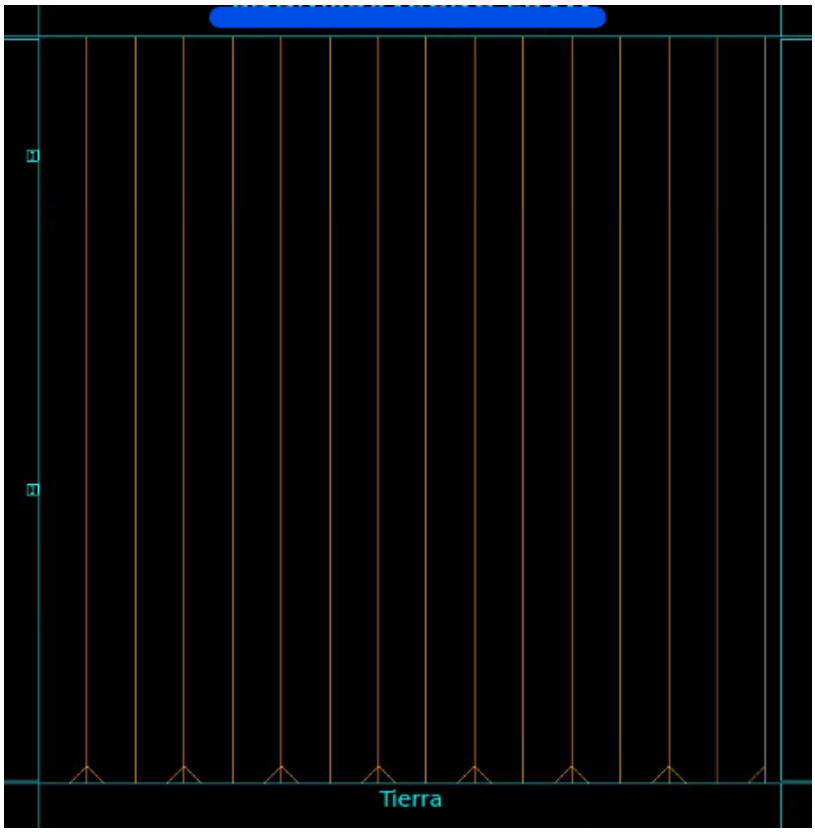


Figura 31: *Floorplan* con *Rows* verticales.

La configuración está por defecto exceptuando el cambio de los *Rows*. La cantidad de estos siguen llenando el espacio del *core*. Los *Rows* siguen el patrón de orientación definidos con anterioridad, orientación `N` y `FW`. Debido a que la posición de los *Rows* cambio de horizontal a vertical, las celdas también lo hacen. Es decir las celdas que se establecerán dentro de los *Rows* cambian su orientación de vertical a horizontales. La Figura #32 muestra la celda *Not* colocada en el *core* del *Floorplan* creado con *Rows* verticales.

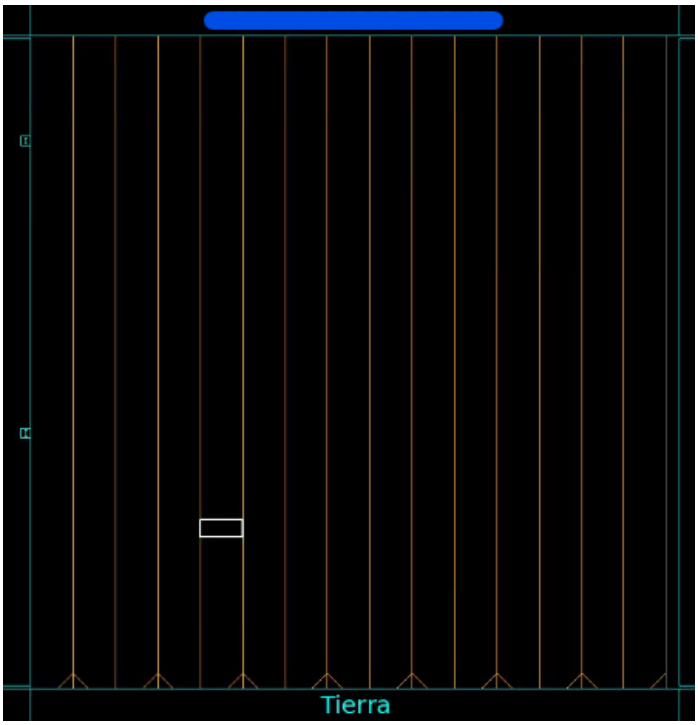


Figura 32: *Floorplan* con *Rows* verticales y la compuerta *Not* colocada.

La posición horizontal y vertical de los *Rows* dependerá del circuito y el diseño que se este trabajando. Este podrá cambiar la posición y conexiones en la etapa de *Route*. El motivo de esto se conocerá más a detalle en los siguientes capítulos. El comando completo para cambiar la posición horizontal de los *Row* a vertical es el siguiente: `create_floorplan -use_vertical_row`. En linea de comando se vera de la siguiente forma.

```
create_floorplan -use_vertical_row
```

Para lograr esto usando la interfaz gráfica habrá que de-seleccionar la opción `Horizontal row`. La Figura #33 muestra el ejemplo.

Figura 33: *Floorplan* con *Rows* verticales por medio de interfaz gráfica.

Los *Rows* siempre están en parejas. Un *Row* con orientación `N` y otra con orientación `FW`. El parámetro `-no_double_back` crea parejas de *Rows* cambiando la de orientación del *Row* `FW`, es decir, los dos *Rows* que conforman la pareja tienen orientación `N`. El parámetro `-no_double_back` también genera un espacio entre parejas. Este espacio que deja entre parejas servirá para conexiones futuras. La Figura #34 muestra la configuración por defecto agregando únicamente el parámetro `-no_double_back`.

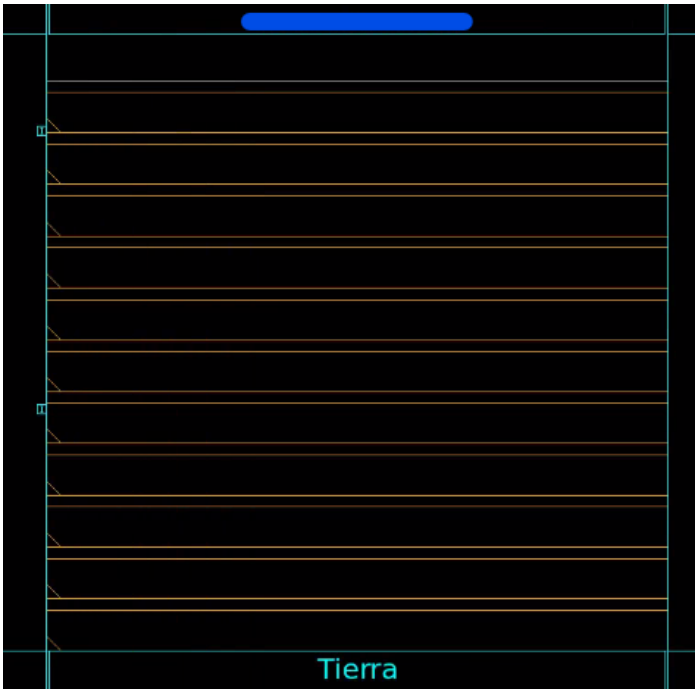


Figura 34: *Floorplan* con *-no_ double_ back Rows.*

Todos y cada uno de estos parámetros se pueden usar individualmente o en conjunto. Si se usan de manera individual el programa automáticamente asigna valores por defecto en los otros campos. Para tener un mejor control en el diseño se deben conocer y configurar todos al mismo tiempo. A continuación se muestra un ejemplo con los *Rows* de manera vertical y con el parámetro `-no_double_back`. La Figura #35 ilustra el ejemplo.

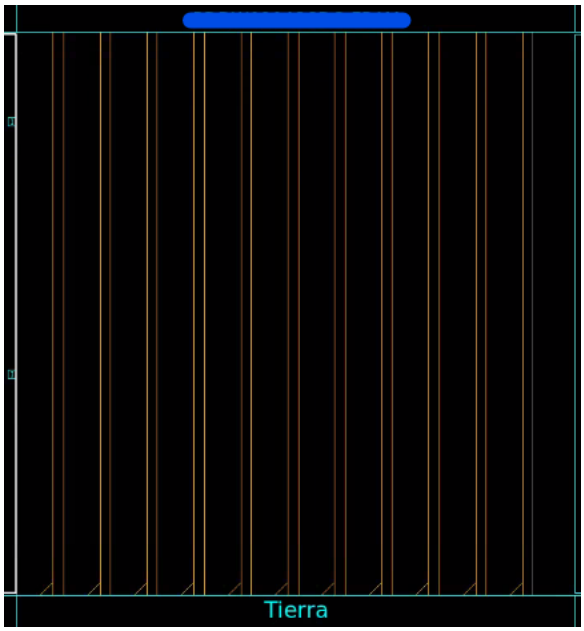


Figura 35: *Floorplan* combinando dos parametros *-no_ double_ back -use_ vertical_ row.*

Al igual que antes las celdas se colocarán dentro de los *Rows*, la única diferencia es que se debe respetar la separación entre cada uno de estos. Esta separación corresponde a un espacio para las conexiones. Un ejemplo de esto se muestra en la Figura #36

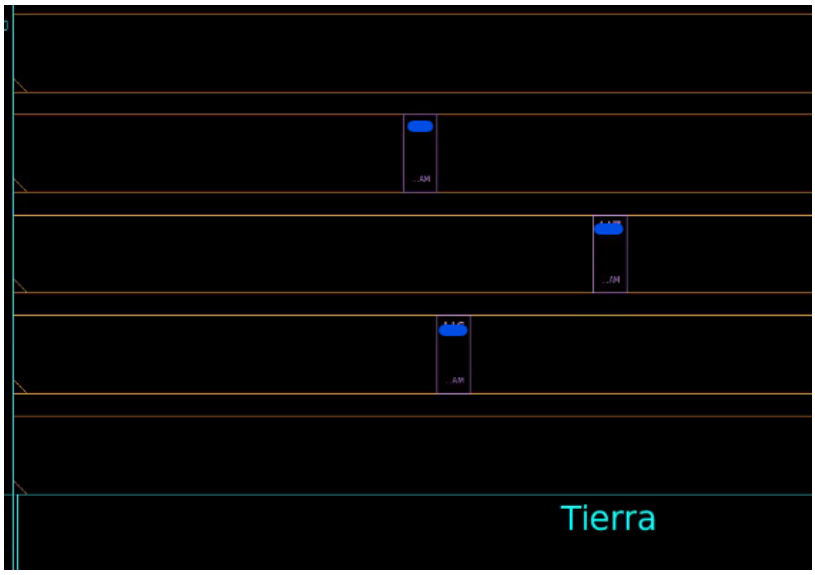


Figura 36: *Floorplan* con *-no_ double_ back Rows* y celdas colocadas.

Esta opción se puede seleccionar o de-seleccionar por medio de la interfaz gráfica. Para ello se debe hacer *click* en esa opción. Si el objetivo es configurarlo sin crear la pareja espejeada, la casilla se debe de-seleccionar. Para observarlo de mejor manera refiérase a la Figura #37. El comando completo con `-no_double_back` *Rows* y posicionados verticalmente se configura de la siguiente forma: `create_floorplan -use_vertical_row -no_double_back`. Al parámetro `-no_double_back` al igual que `-use_vertical_row` no se debe establecer ningún valor o característica. A continuación se presenta la configuración en linea de comando.

```
create_floorplan -use_vertical_row -no_double_back
```

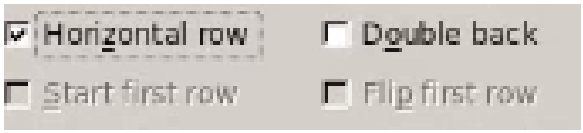


Figura 37: *Floorplan* con *-no_ double_ back Rows* por medio de la interfaz gráfica.

Los otros dos parámetros `-flip_first_row` y `-start_first_row` modifican como empezaran los *Rows*. El parámetro `-flip_first_row` voltea el primer *Rows* del *core*. Este puede encontrarse en la parte superior del *core* para *Rows* horizontales. Para *Rows* verticales se encuentran pegados al borde izquierdo del *core*. El parámetro `-start_first_row` configura el primer *Row* presentado en el *core* con orientación `N`. Si el parámetro `-no_double_back` esta activado, no se podrán agregar los parámetros `-flip_first_row` y `-start_first_row`.

Los siguientes cuatro parámetros modifican la separación que contiene el *core* con los *pads* y con las *corners*. Estos parámetros reciben valores positivos y estos son: `-left_io2core`,

`-right_io2core -bottom_io2core` y `-top_io2core`. Esta separación se puede configurar agregando la unidad en micrones que se quiera espaciar. Esto quiere decir que si se requiere que el *core* este separado en sus cuatro extremos una unidad de 13.7 micrones, el programa lo creará. Otro caso podría ser que solo se requiera que el extremo derecho e izquierdo estén separados por 5 micrómetros y que la superficie del *core* con el fondo mantengan una separación de 15 micrones. Esta configuración se muestra en la Figura #38.

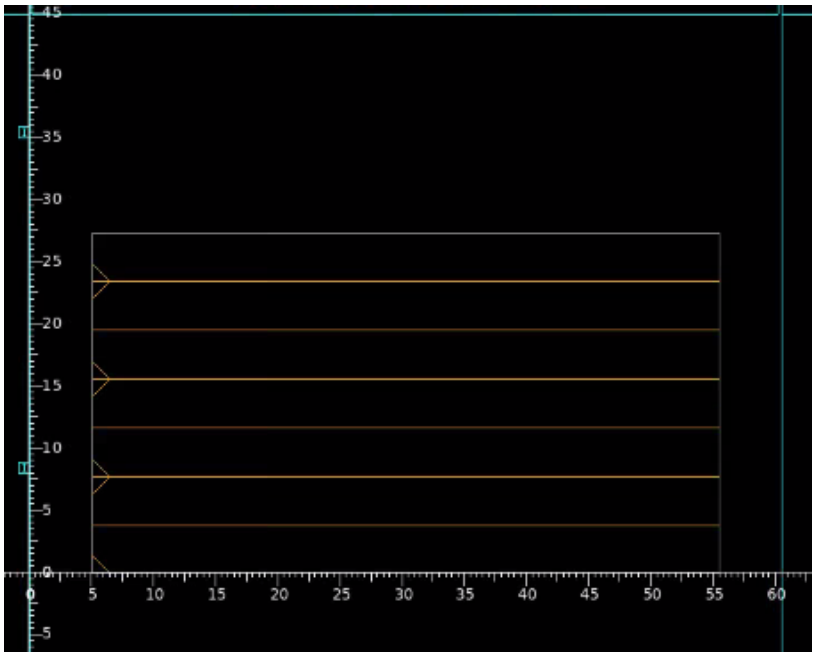


Figura 38: *Core* separado 5 micrones a los costados y 15 en la parte superior e inferior.

Como ejemplo se utilizarán todos los parámetros junto al comando para crear un *Floorplan*. La configuración es: `create_floorplan -control_type width_and_height -core_utilization 0.7 -core_width 5 -core_height 50 -no_double_back -left_io2core 5 -right_io2core 5 -bottom_io2core 10 -top_io2core 10`. Esto crea un *Floorplan* con un *die* de ancho y alto de cinco y cincuenta micrómetros respectivamente. Además se agrega un uso del *core* del setenta por ciento (70 %). Se agregan *Rows* sin orientación `FW` y una separación de los extremos de cinco micrómetros. Así mismo se configura una separación de 10 micrones para los bordes superiores e inferiores del *core*. A continuación se expresa la configuración en linea de comando. La Figura #39 muestra el ejemplo y la Figura #40 un *zoom* del *core*.

```
create_floorplan -control_type width_and_height -core_utilization
   0.7 -core_width 5 -core_height 50 -no_double_back -left_io2core 5
    -right_io2core 5 -bottom_io2core 10 -top_io2core 10
```

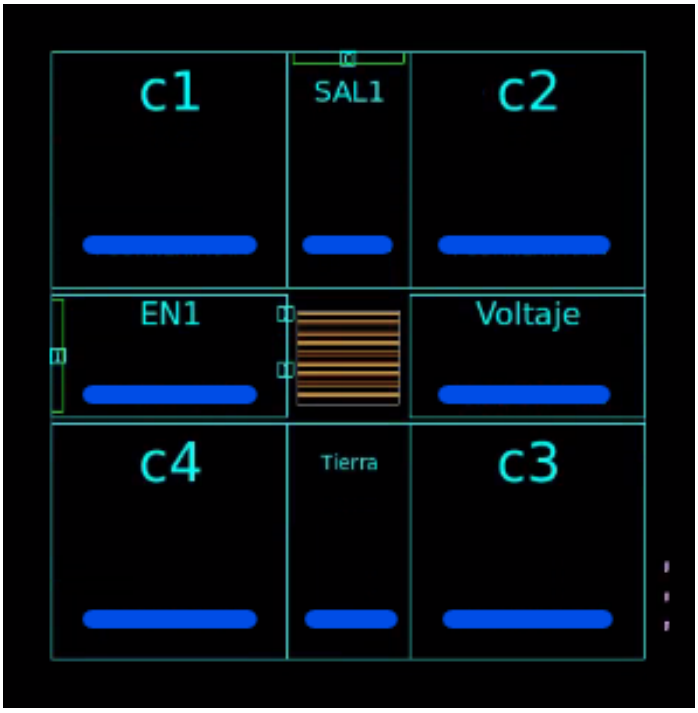


Figura 39: Ejemplo de *Floorplan* con parámetros configurados.

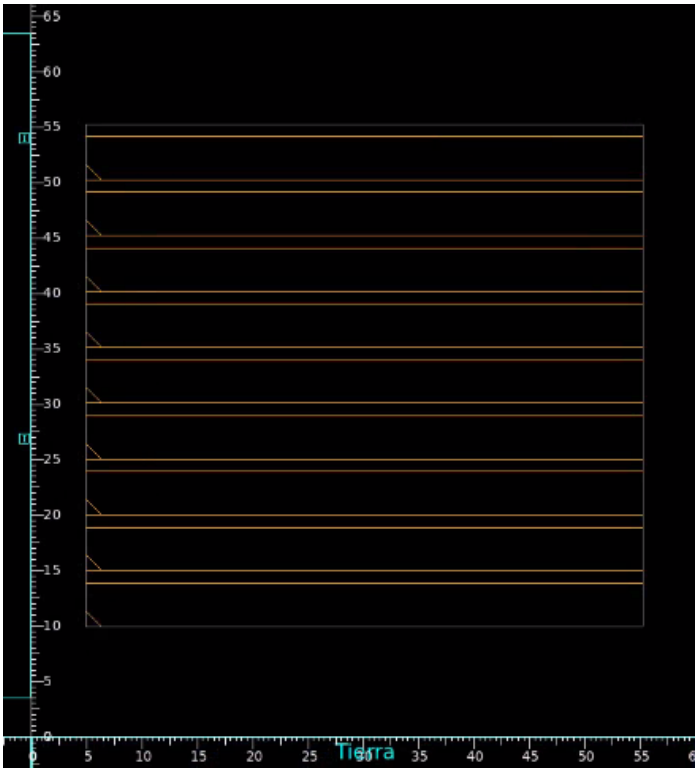


Figura 40: *Zoom* al ejemplo de *Floorplan* con parámetros configurados.

CAPÍTULO 9

Placement

Creado un ambiente apto para el circuito a fabricar, es momento de introducir al *core* las celdas que conforman el circuito. Para realizar este proceso se creará una estrategia a seguir. La acción que requiere agregar las celdas al *core* del *chip* recibe el nombre de *Placement*. El *Placement* al igual que el *Floorplan* configura una colocación de celdas por defecto.

Antes de definir el *Placement* se debe conocer y establecer los anillos de poder y los *power Straps*. Los anillos de poder y *power straps* sirven para alimentar el *core* y a las celdas en el. Para ello se deben de establecer áreas del *chip* que contendrán los anillos en un futuro. Si por alguna razón no se conociera las áreas que requieran estos anillos de poder, estos se podrán crear después del *Placement*. Para los circuitos que se están implementando no es necesario realizar los anillos de poder ni los *power straps*. Sin embargo para circuitos mas complejos podrá ser útil crearlos.

La estrategia que se establecerá da una mejor comprensión del *Placement*. Esta no es única, existen muchas maneras y vías de realizar el *Placement*. La estrategia es la siguiente. Primero se revisará el diseño creado anteriormente. Luego se podrán establecer *constraints* para optimizar el *Placement* o llevar a cabo directamente la creación del *Placement*. Por último se revisará que no existan errores en todo el diseño. La Figura #41 muestra la estrategia a seguir.

El camino de la estrategia que se seguirá será el de crear el *Placement* de manera directa sin configurar parámetros adicionales. Esto se debe a que el circuito que se esta implementando no es muy complejo o extenso. Además se desea dar una mejor comprensión del panorama de como llevar a cabo el proceso descrito anteriormente.

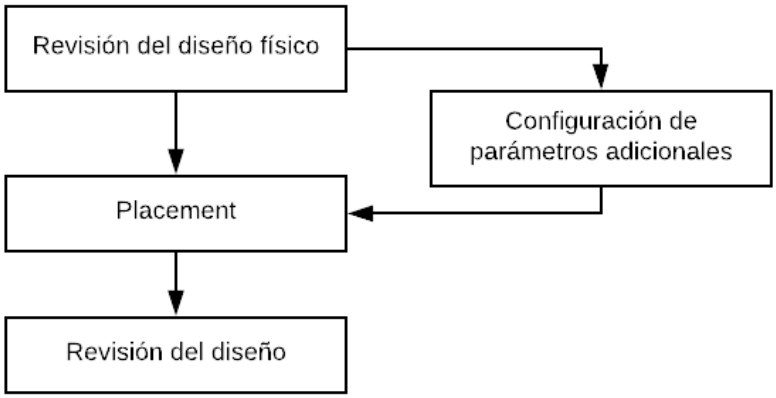


Figura 41: Estrategia propuesta para realizar un *Placement* funcional.

9.1. Configuración del *Placement*

Como se explico anteriormente, esta sección abarcará la configuración preliminar para crear el *Placement*. Esta configuración será básica. Sin embargo se analizará la mejor manera de configurar el *Placement*. Para ello se hará uso de la linea de comandos que IC Compiler incorpora. De igual forma, se analizará la realización del *Placement* por medio de interfaz gráfica. La manera a la que se le dará mas enfoque será a la primera: linea de comandos.

El comando a utilizar es: `create_fp_placement` [13][14]. Al igual que en los capítulos anteriores se deberán configurar ciertos parámetros para tener un mejor control sobre el comando. La configuración se estará desarrollando sobre los circuitos antes descritos. La compuerta *Not*, el *Full Adder* y el *RCA*. A continuación se explicará cómo utilizar el comando con los parámetros.

9.1.1. Revisión del diseño físico

Siguiendo la estrategia a utilizar, el primer paso es la revisión del diseño físico. Esto servirá para comprobar que no existan errores antes de crear el *Placement*. El comando que realiza esto es el siguiente: `check_physical_design` [14]. Sin embargo el comando solo da un reporte general del diseño físico. Este muestra un mensaje en el cual podremos conocer la aceptación que tiene el circuito. También dará a conocer si el diseño estará preparado para la siguiente etapa. La Figura #42 muestra el reporte que da la herramienta de la aprobación del diseño.

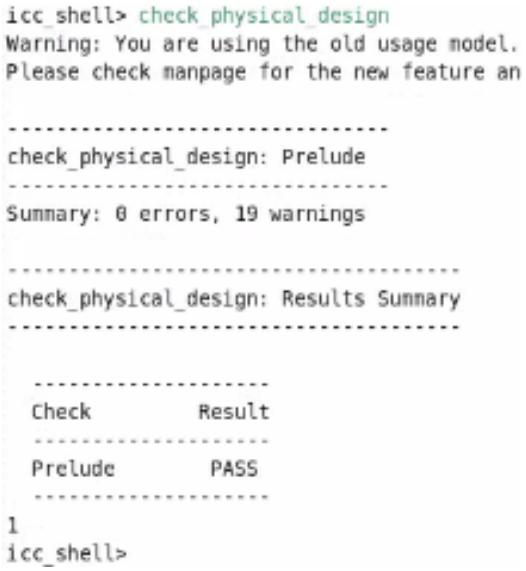


Figura 42: Reporte al usar el comando *check_ physical_ design*

Para tener una comprobación del diseño físico en la etapa después del *Floorplan* y antes del *Placement* se ejecuta el siguiente parámetro `-stage pre_place_opt`. El parámetro muestra un resultado que indica si los datos del *Floorplan* y los datos del *netlist* están preparados para seguir. Para ello los *constraints* deberán estar ya configurados y establecidos [14].

Mostrado el resumen de los errores y los datos en el diseño físico, habrá que verificar que la cantidad de errores sea igual a cero. Caso contrario habrá que mitigar los errores antes de proseguir. Adicionalmente se mostrará un listado de *warnings*, los cuales habrá que revisar y corregir si es necesario. El comando completo a utilizar será el siguiente.

```
check_physical_design -stage pre_place_opt
```

9.1.2. Configuración de *Placement*

Al crear el *Placement* de componentes, se deberán configurar ciertas características. Estas configuraciones están basadas en lo que se requiera para el circuito. Por ejemplo, se podrá configurar el espacio que tendrá cada celda. Se podrán evitar aglomeraciones de celdas dentro del *core*. Se podrá optimizar el tiempo de señales, además de las ubicaciones de los pines para evitar grandes conexiones en procesos futuros [13]. Muchas de estas características no se encuentran presentes en los circuitos a emplear. Sin embargo se explicará por la importancia que conlleva para circuitos mas grandes y complejos.

Para crear las características que rigen al circuito se deberá de usar el siguiente comando `create_fp_placement` seguido de los parámetros que se consideren necesarios. El primer parámetro que se utilizará es: `-effort`. Este parámetro establece el nivel del esfuerzo con el que se llevará a cabo el *Placement*. Para ello existen dos opciones, `-effort low` y `-effort high`.

La primera opción establece un nivel bajo con resultados básicos de colocación para las

celdas, sin superposición. También hace que el tiempo en el que se ejecuta el *Placement* sea corto. La segunda opción da una mejor calidad al momento de realizar el *Placement*. El inconveniente de esta es el tiempo con el que se ejecuta, ya que puede llegar a ser demasiado largo [14]. Para los circuitos que se están implementando el nivel de *effort* que se elija no afectará en el diseño, debido a que las celdas en estos son pocas. Para circuitos con mayor numero de celdas habrá que tomar en consideración el *effort* debido al tiempo de ejecución y la calidad de resultados que se quieran.

El comando a utilizar tiene la siguiente estructura.

```
create_fp_placement -effort low
```

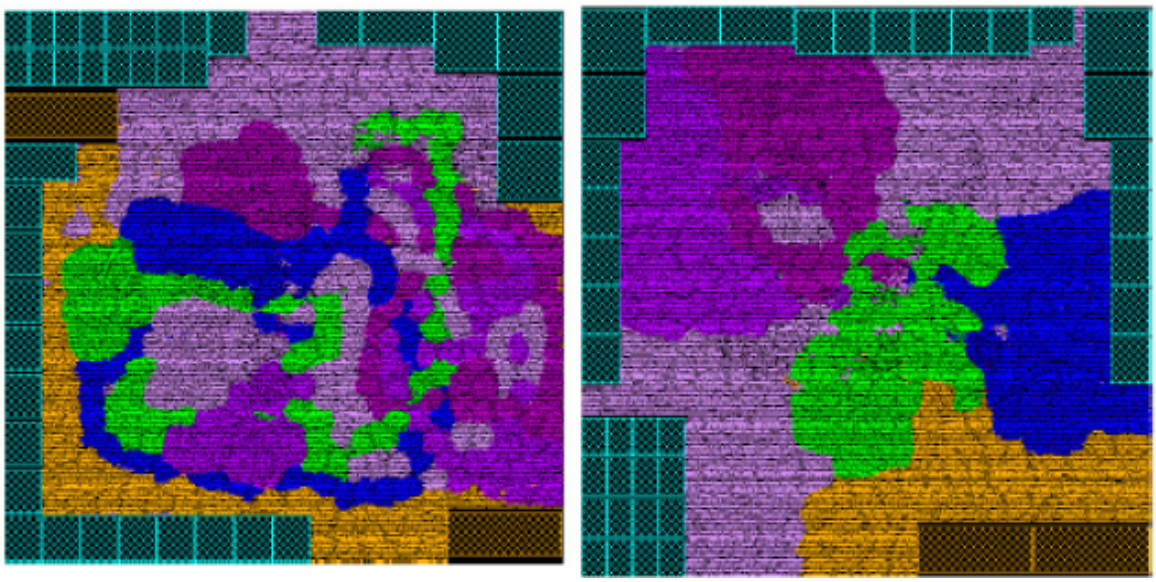
```
create_fp_placement -effort high
```

Para configurar la cantidad máxima de conexiones posibles que puede tener una celda a las *nets* se deberá usar el siguiente parámetro: `-max_fanout`. Este parámetro hace que el *Placement* ignore a las redes que tengan un cantidad de conexiones mayor a la que se establece. Para configurar el numero máximo de *fanout* en el diseño se debe hacer mediante un valor entero positivo. El valor por defecto que se establece es de `512` [14]. La manera de configurarlo es la siguiente.

```
create_fp_placement -max_fanout 126
```

El siguiente parámetro `-no_hierarchy_gravity` evita la agrupación de celdas con el mismo bloque jerárquico [14]. Este parámetro se utiliza con circuitos con mayor cantidad de celdas y con circuitos más complejos. Por defecto al pasar el comando `create_fp_placement`, este parámetro se encuentra deshabilitado con la finalidad de brindar mejores resultados. La Figura #43 ilustra este parámetro. La manera de configurarlo es la siguiente.

```
create_fp_placement -no_hierarchy_gravity
```



(a) con parámetro *-no_ hierarchy_ gravity.*

(b) Sin parámetro, por defecto.

Figura 43: Ejemplo de un circuito con y sin el parámetro [13].

Para evitar que las celdas colisionen al momento de realizar el *Placement*, existe una función por defecto que evita la superposición al momento de pasar el comando `create_fp_placement`, este recibe el nombre de `legalize`. Su función es verificar que no existan violaciones de metal con metal (*metal-to-metal*) entre celdas y *nets* de poder [13][14]. Si no se quisiera utilizar esa configuración por defecto se podrá utilizar el parámetro `-no_legalize` como se muestra a continuación.

```
create_fp_placement -no_legalize
```

Para incrementar el desempeño del *Placement* actual se podrá utilizar una variante conocida como *incremental*. Para acceder a esta opción utilizaremos el parámetro `-incremental`. Este parámetro se podrá configurar con cuatro opciones según sean necesarias, las cuales son: `all`, `top_level_cells`, `plan_groups` y `voltage_areas`. La opción `all` da un mejor rendimiento al *Placement* en todo el diseño del circuito. La opción `top_level_cells` mejora el *Placement* de las celdas superiores que no pertenezcan a un área de voltaje. Las opciones `plan_groups` y `voltage_areas` mejoran la colocación de celdas dentro de los grupos y las áreas de voltaje que se establezcan [13][14]. El comando se verá de la siguiente forma.

```
create_fp_placement -incremental all
```

```
create_fp_placement -incremental top_level_cells
```

```
create_fp_placement -incremental plan_groups
```

```
create_fp_placement -incremental voltage_areas
```

Para obtener mejores tiempos de respuesta de las señales entre las conexiones de las celdas, se puede incorporar un parámetro que realice esto. El parámetro es el siguiente `-timing_driven`. Este parámetro hace que el *software* haga un mayor esfuerzo para mejorar la colocación de las celdas. Esto lo hace con el fin de evitar que existan rutas críticas. Sin embargo el tiempo de ejecución se vera afectado haciéndolo mas tardado, ya que lo elabora en paralelo al *Placement* [13]. Por defecto el *Placement* se realiza con este parámetro desactivado. El comando a utilizar es el siguiente.

```
create_fp_placement -timing_driven
```

En todos los procesos realizados anteriormente no se requiere mucha capacidad de procesamiento para llevarlos a cabo. Sin embargo, para llevar a cabo el *Placemente* el poder computacional que se requiere deberá ser mucho más intenso. Esto se debe a que el programa frente a circuitos mas complejos analizará cada espacio establecido en el *core* y todo el *Floorplan*. Llevando a cabo varios análisis como por ejemplo: posiciones, mejores rutas, superposiciones de celdas, etc.

Al realizar el *Placement* se podrá definir la cantidad de *CPUs* que serán utilizados. Esto se define haciendo uso del parámetro `-num_cpus`. El valor a definir dependerá de dos factores. El primer factor es conocer la cantidad de *CPUs* que posee la computadora en donde se esté llevando a cabo el proceso. El segundo factor es conocer sobre el nivel de complejidad del circuito que se este sintetizando.

Tomando en cuenta los dos factores anteriores se podrá proceder a definir la cantidad de *CPUs* que se usarán. El valor numérico deberá ser menor o igual al numero de *CPUs*

libres en la computadora. El valor mínimo que es aceptado para configurarlo es uno. Si no se define ningún valor la herramienta asignará automáticamente el uso de un *CPU* [14]. El comando se puede observar a continuación.

```
create_fp_placement -num_cpus 2
```

Los siguientes dos comandos: `-plan_groups` y `voltage_areas` depuran el *Placement* de grupos y de áreas de voltaje respectivamente. El tipo de dato que reciben es de tipo colección. Estos parámetros cumplen su función solo si se configuró previamente la opción de `plan_groups` o `voltage_areas` en el parámetro `-incremental` [14]. Las dos lineas de comando se escriben de la siguiente forma.

```
create_fp_placement -plan_groups coleccion_de_grupos
```

```
create_fp_placement -voltage_areas coleccion_de_areas
```

Existe un parámetro que mejora en paralelo el desempeño para el *Placement* y pines a nivel de bloques de circuito en el diseño. Para mas información sobre bloques de circuito o *block level design* refiérase al manual *IC Compiler Design Planning User Guide*. El parámetro a utilizar es el siguiente: `-optimize_pins`. Cuando se especifica esta opción IC Compiler busca el mejor *Placement* y asignación de pines para el diseño [14]. La estructura del comando es la siguiente.

```
create_fp_placement -optimize_pins
```

El comando `create_fp_placement` contiene un parámetro que sirve para realizar las pruebas de conexiones mientras se ejecuta el *Placement*. Para habilitar esta prueba se utiliza el parámetro `-consider_scan` [14]. El comando adquiere la siguiente forma.

```
create_fp_placement -consider_scan
```

Al momento de trabajar con bloques de circuitos en el *Placement* se podrá almacenar la información de estos mediante el uso del parámetro `-write_placement_blockages`. El archivo se guarda con una extensión *.tcl*. Si ya existiera un archivo con el mismo nombre, este se sobre escribe. El comando que se utiliza para guardar esta información es el siguiente.

```
create_fp_placement -write_placement_blockages
```

Existe un modo a la hora de crear el *Placement* que sirve para realizar una exploración rápida. Esto se realiza por medio del parámetro `-exploration`. La finalidad de este parámetro es brindar una idea acerca del posible posicionamiento inicial de las celdas y bloques desde una vista superior, *top level design*. Este parámetro no funciona o es incompatible si se configuró anteriormente los siguientes parámetros: `-timing_driven`, `-congestion_driven` e `-incremental`. Esto se debe a que los otros parámetros realizan un análisis detallado en la búsqueda de la mejor ubicación de las celdas [14].

Para realizar un análisis rápido sobre un posible *Placement*, el comando se configura de esta manera:

```
create_fp_placement -exploration
```

En la sección: **Creación del *Placement*** se menciona e ilustra la configuración del

Placement que se utilizó.

9.1.3. Configuración adicional del *Placement*

El comando a emplear para realizar configuraciones adicionales en el *Placement* es: `set_fp_placement_strategy`. Este comando configura ciertos parámetros especiales. Además configura parámetros que no configuran los comandos `create_fp_placement` o el comando `legalize_fp_placement` [13][14].

Existen treinta y un (**31**) parámetros disponibles que se pueden configurar con el comando `set_fp_placement_strategy`. Esta sección no se profundizará demasiado ya que las configuraciones de los parámetros son especiales y/o específicos para circuitos mas extensos.

Estas configuraciones especiales se deben de implementar bajo ciertas características. Para implementarlas habrá que saber qué parámetros son los que mejor se ajustan al diseño a implementar.

El comando `set_fp_placement_strategy` solo funciona si luego de configurarlo se pasa el comando `create_fp_placement` [13]. Es decir que existe un orden para realizar un *Placement* mas detallado. El orden es el siguiente:

1. Configuración preliminar del *Placement* con la configuración adicional de este. Usando el comando `set_fp_placement_strategy` con sus respectivos parámetros.
2. Configuración del *Placement* y la creación de este. Haciendo uso del comando `create_fp_placement` configurado con sus parámetros respectivos.

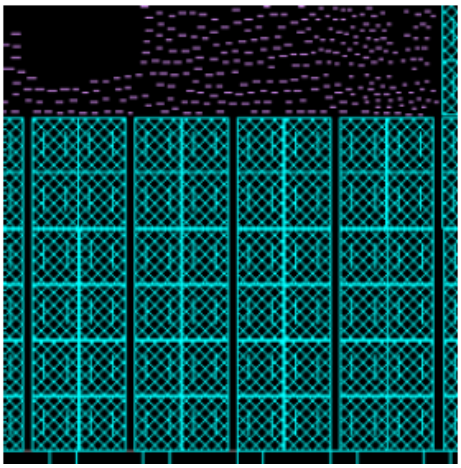
La razón por la que el comando `set_fp_placement_strategy` se usa después del comando `create_fp_placement` es la siguiente. El comando **no** almacena la información de los parámetros configurados dentro de la librería *Milkyway*. Es decir que el comando `set_fp_placement_strategy` es "volátil". Por lo tanto el comando deberá configurarse cada vez que quiera usarse para realizar un *Placement* especifico [13].

A continuación se muestran algunos de los parámetros que se pueden configurar con el comando. Algunos de ellos serán explicados mas adelante.

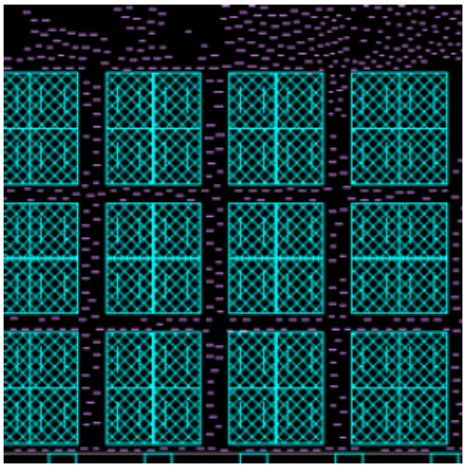
1. `-adjust_shapes`
2. `-congestion_effort`
3. `-hierarchy_gravity_blocks`
4. `-honor_mv_cells`
5. `-macro_orientation`
6. `-pin_routing_aware`
7. `-sliver_size`

Algunos de los parámetros son utilizados para configurar bloques de circuitos a nivel *macro*. Es decir que la configuración de estos parámetros no afectará a celdas estándares. A continuación se explicará el parámetro `-min_distance_to_auto_array` y cómo es que este afecta solo a grupos macros dentro del *chip*.

El parámetro `-min_distance_to_auto_array` crea un espacio adicional alrededor de los grupos macros y los bordes del *core* [13][14]. Para lograr esto se debe agregar el espacio definido en micrómetros. El parámetro es de tipo flotante, por ejemplo: 31.85 o 7.921. La Figura #44 ilustra un ejemplo del uso del parámetro. En el ejemplo se usa el valor por defecto del programa con 0 micrones de separación y se compara con otro ejemplo con 20 micrones de separación.



(a) Parámetro por defecto con 0 micrones de separación



(b) Separación de 20 micrómetros entre los bloques macros y el *core*.

Figura 44: Ejemplo del parámetro *-min_ distance_ to_ auto_ array* [13].

El parámetro `-congestion_effort` controla la cantidad de esfuerzo que realizará la herramienta para crear el *Placement*. Este puede recibir tres valores: *low*, *medium* o *high*. Estos valores indicarán que tanto esfuerzo deberá aplicar la herramienta para evitar que existan aglomeraciones de celdas y grupos de circuitos.

Para los dos parámetros ejemplificados se mostrará su uso en linea de comando.

```
set_fp_placement_strategy -min_distance_to_auto_array 20
```

```
set_fp_placement_strategy -congestion_effort high
```

9.2. Creación del *Placement*

Como se menciono anteriormente, el comando `create_fp_placement` se utiliza para crear el *Placement*. En esta sección se estará mostrando los resultados que se obtienen al configurar algunos de los parámetros. Algunos de estos parámetros no se podrán mostrar, ya que algunos solo indican a la herramienta el nivel de *effort* que debe realizar. Además

algunos de estos parámetros se verán restringidos por el tamaño de los circuitos que se están implementando.

A continuación se mostrará como crear el *Placement* por defecto. Para ello utilizaremos únicamente el comando `create_fp_placement`. Al pasar el comando la herramienta brinda información sobre: el nivel de esfuerzo que se utilizo, la cantidad de *CPUs* utilizados, si el chip se roto, así como el tiempo que le tomo al *CPU* realizar la operación, etc.

El comando agrega también varios reportes en su estructura para tener una idea de lo que cambio en el diseño. Los reportes generados son cuatro y son los siguientes: *virtual Flat Placement*, *chip summary*, *pnet options* y *Legalized Displacement*. En la Figura #45 se muestra el *Placement* de componentes sin configurar ningún parámetro.

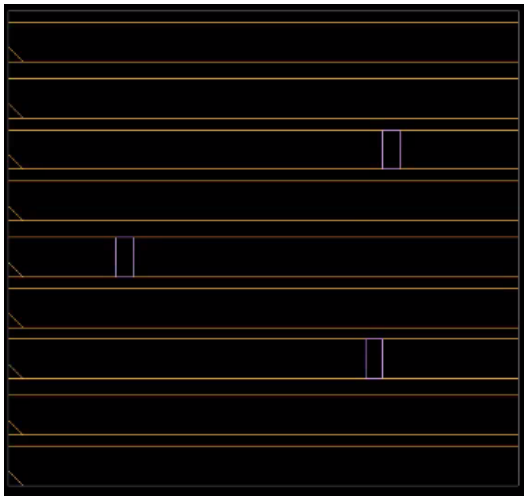


Figura 45: *Placement* sin configuración de ningún parámetro.

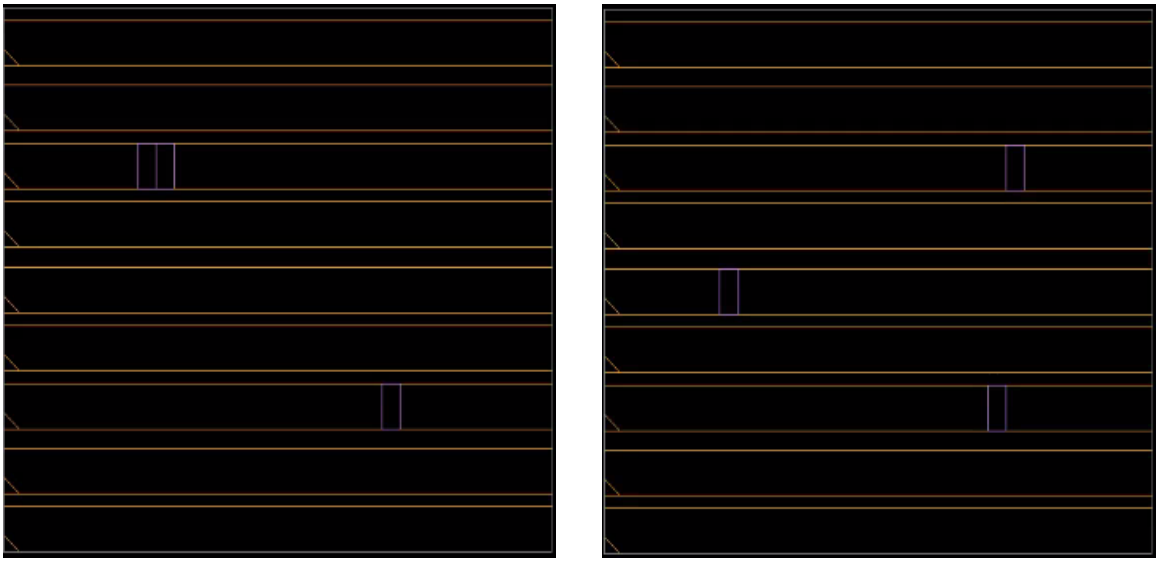
El *Placement* realizado anteriormente esta definido con las configuraciones por defecto que realiza la herramienta. A continuación se estará armando un comando que servirá para configurar mas adelante circuitos mas complejos. Esto con el fin de dar una documentación de un *script* que sea funcional para el desarrollo del chip que se estará implementando.

El primer parámetro que se modificará será el del esfuerzo. La herramienta realiza un esfuerzo bajo por defecto al momento de realizar el *Placement* como se explicó previamente. Para fines de desarrollo de un circuito, el nivel de esfuerzo que deberá realizar la herramienta será el mayor nivel. Esto es porque existen muchas compuertas, conexiones y celdas a nivel estándar y macro que deberán de ser colocadas. Para el caso de los circuitos de *Not*, *Full Adder* y *Ripple Carry Adder* con un esfuerzo leve bastará. El comando toma la siguiente forma:

```
create_fp_placement -effort low
```

Al pasar el comando anterior físicamente no presenta ningún cambio, pero funcionalmente si hay un cambio. Este parámetro es muy opcional, ya que se podrá pasar con esfuerzo *high* para los circuitos que se estarán utilizando. El resultado será el mismo por el nivel de complejidad y el tamaño de estos.

El siguiente parámetro a configurar es el de `-max_fanout`. Como se explicó anteriormente este parámetro hace que la herramienta ignore las redes que tengan una cantidad mayor de la que se establece. Esto es útil para saber si una compuerta será capaz de soportar varias salidas conectadas a otras de ellas. La Figura #46 muestra una comparación con un `-max_fanout` de **6** y otro de **800**.



(a) *Placement* con *-max_fanout* de 6

(b) *Placement* con *-max_fanout* de 800.

Figura 46: Ejemplo de *Placement* con dos *fanout* diferentes.

Al comparar la Figura #45 y Figura #46 (b), no se vería ninguna diferencia, ya que el parámetro por defecto establece un *fanout* de **512**. El comando actualizado tiene la siguiente forma:

```
create_fp_placement -effort low -max_fanout 800
```

Estos últimos dos parámetros deberán de ser utilizados con base al criterio de quien se encuentre realizando la síntesis física. Ya que dependerá del circuito que se este implementando. El parámetro `-no_hierarchy_gravity` al igual que el parametro `-no_legalize` no se configurarán para los circuitos que se están usando.

El parámetro `-incremental` se configurará, pero con la opción de `all`. Ya que vamos a requerir que exista un mejor rendimiento en todo el *chip*. Si se utiliza este parámetro no podrá configurarse posteriormente el parámetro `optimize_pins` o viceversa. Así mismo se configurará el comando `-timing_driven`. Este permitirá que la herramienta optimice a las celdas para que sean colocadas para que no existan rutas críticas. Actualmente el circuito no presenta ningún cambio y se mantiene como en la Figura #45. El comando a usar es el siguiente:

```
create_fp_placement -effort low -max_fanout 800 -incremental all -
   timing_driven
```

Los últimos tres parámetros que se configurarán son los siguientes: `-optimize_pins`, `-num_cpus` y `-consider_scan`. Estos parámetros aunque no son necesarios para los circuitos

que se están trabajando podrán ser de mucha utilidad para circuitos más complejos. El primero de ellos mejora el desempeño para los pines a nivel de bloques de circuito mientras se realiza el *Placement*. El segundo de ellos se podrá modificar si el *chip* tendrá mucha más información y celdas. Con este se podrá hacer que el programa no se estrese demasiado. El tercero de ellos será de suma importancia para circuitos que tengan un nivel de lógica mas complejo. Ya que permitirá que se realicen pruebas cuando se colocan las celdas y los bloques. Esto con el fin de comprobar el rendimiento del diseño.

El parámetro `-congestion_driven` se configurará de igual forma. Esto para evitar que exista sobre-posiciones de celdas en un mismo espacio. Si no se define para usarlo la herramienta lo establece en modo *off* o apagado. El comando a utilizar es el siguiente.

```
create_fp_placement -effort low -max_fanout 800 -timing_driven -
   num_cpus 1 -optimize_pins -consider_scan -congestion_driven
```

El *Placement* realizado se muestra a continuación en las Figura #47 para la compuerta *Not*, en la Figura #48 para el *Full Adder* y en la Figura #49 para el *Ripple Carry Adder*.

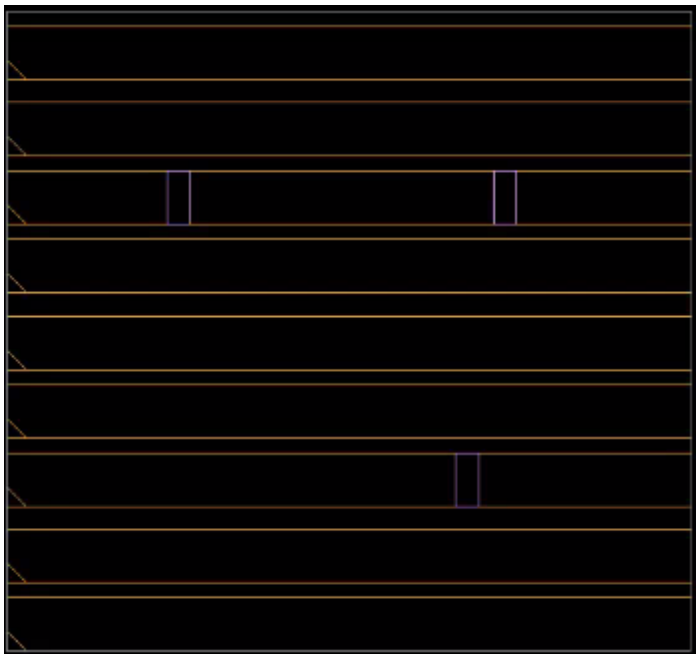


Figura 47: *Placement* configurado y realizado para la compuerta *Not*.

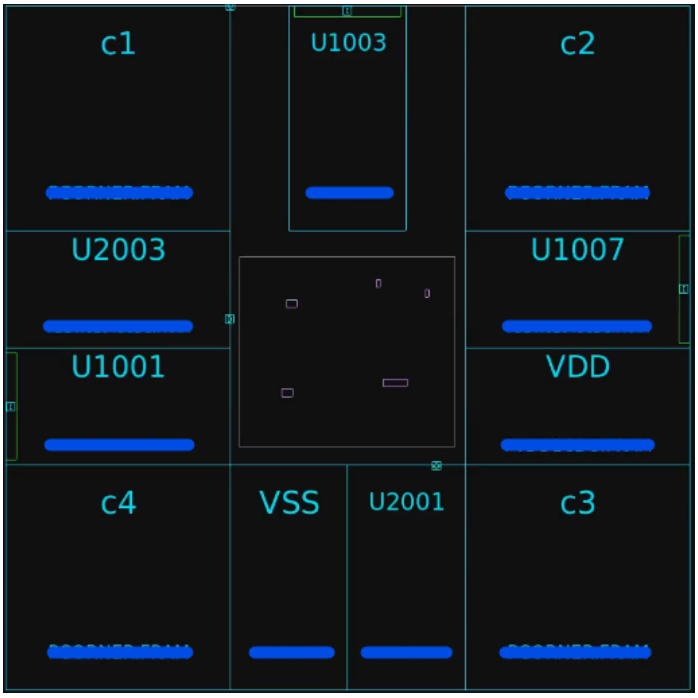


Figura 48: *Placement* configurado y realizado para el circuito *Full Adder*.

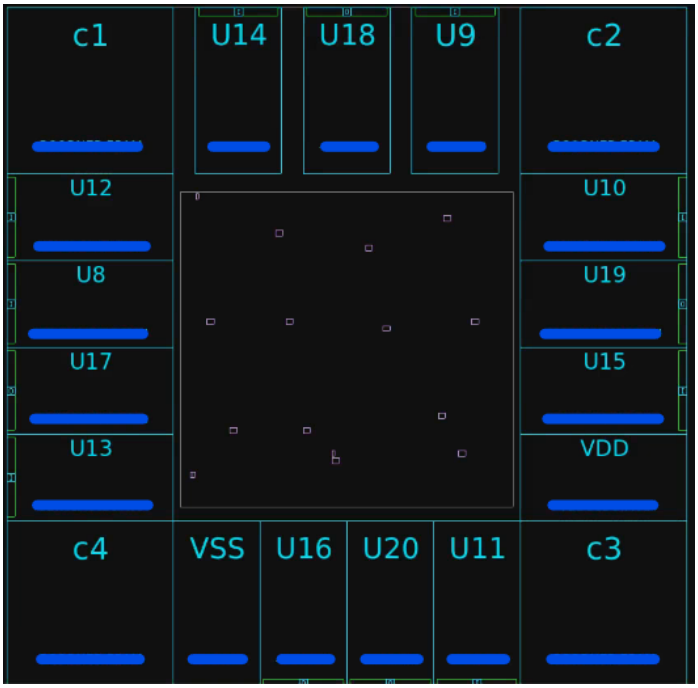


Figura 49: *Placement* configurado y realizado para el circuito *RCA*.

Una vez realizado el *placement* de los componentes se ejecutará el comando `legalize _placement` para evitar superposición de celdas. Con esto nos aseguramos que el posicionamiento de las celdas esté sin violaciones de colisión [14].

```
legalize_placement
```

9.3. Revisión y optimización del *Placement*

En esta sección se abarcará sobre la revisión posterior que se debe hacer al *Placement*. También se realizará un método para corregir posibles desperfectos y poder así optimizar el *Placement* previamente realizado.

La revisión del diseño y la optimización de este no se ven afectados si uno se realiza antes que el otro. El método más convencional será realizar una verificación del circuito en la fase actual, seguido de realizar una optimización adecuada. Posteriormente se podrá realizar otra verificación para terminar de comprobar si los resultados son los deseados.

Esto último será más utilizado dependiendo del circuito que se esté realizando y la complejidad. Esto se debe a que el diseñador deberá de estar seguro si el diseño en la fase actual es funcional. Es decir que la revisión y la optimización del *Placement* quedan a criterio del diseñador.

A continuación se mostrarán los comandos que realizan estas operaciones. El proceso que se realizará será el de verificación, optimización y verificación final. La Figura #50 muestra el proceso a realizar. Este proceso es opcional, ya que puede ser que el *Placement* se encuentre sin errores y optimizado.

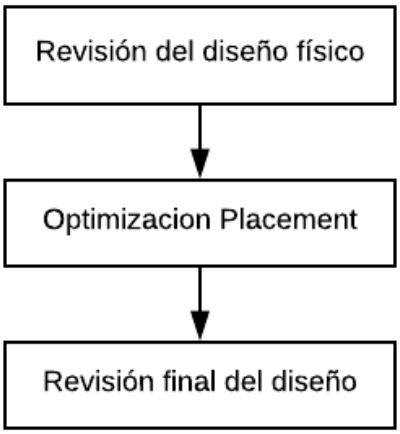


Figura 50: Proceso de verificación del *Placement*.

Antes de revisar el *Placement* será de mucha utilidad conocer sobre él. Con el comando `report_fp_placement` se podrá generar un reporte conocido como *quality of results* (QoR) [13]. Este reporte nos brinda la información de varios parámetros establecidos en el *Placement*. Algunos de los parámetros son: la longitud total de las conexiones, la cantidad de

regiones con densidad de celdas altas y bajas, la cantidad de sobre-posiciones de celdas y el numero de celdas que violan el área del *core* [13]. El comando se muestra a continuación. La Figura #51 muestra un reporte de *QoR*.

```
report_fp_placement
```

```
icc_shell> report_fp_placement
Reference Point: Lower Left-hand corner of Core Base Array
Number of plan group pins = 0
  0 blocks freed
  0 bytes freed
****************************************
Report     : Virtual Flat Placement
Design     : NOT_IO
Version    : O-2018.06-SP5
Date       : Wed Sep 16 10:28:06 2020
****************************************

Total wirelength: 208.26
Number of 100x100 tracks cell density regions: 1
Number of low (< 10%) cell density regions: 4 (4.000%)
Number of high (> 200%) cell density regions: 0 (0.000%)
Maximum cell density: 1.11% (at 120 147 145 171)
Checking hard macro to hard macro overlaps...
Number of hard macro to hard macro overlaps: 0
Checking hard macro to std cell overlaps...
Number of hard macro to std cell overlaps: 0
Checking plan group to plan group overlaps...
Number of plan group to plan group overlaps: 0
Number of TL cells overlapping PG: 0
Number of cells violating core area: 0
Total number of cells violating plan group or core area: 0
1
icc_shell>
```

Figura 51: Reporte de *quality of results* (QoR).

Conocidos los datos antes descritos se podrá generar la verificación del *Placement*. El comando a utilizar para realizar la verificación es el siguiente: `check_physical_design`. Este comando ya fue utilizado previamente antes de realizar *Placement* de componentes. El comando se utilizará con el parámetro `-stage post_initial_placement`.

Al igual que antes el comando con el parámetro generará una serie de reportes que se podrán revisar. Es aquí en donde se verifica que el diseño no contenga errores y que la cantidad de *warnings* sean bajos. Si por alguna razón existieran errores en el diseño actual se recomienda revisar las configuraciones del *Placement*. El comando se redacta de la siguiente manera:

```
check_physical_design -post_initial_placement
```

Teniendo claro los errores posibles y el resultado de la realización del *Placement* se podrá optar por realizar la optimización. Para realizar esto se deberá ejecutar el comando `place_opt`. Seguido de esto se podrá configurar varios parámetros que darán mejores resultados. Si no se desea configurar ninguno de ellos la herramienta configura una optimización general del diseño. Algunos de los parámetros que estarán disponibles para configurar son los siguientes: `-effort`, `-area_recovery`, `-congestion`, `-power`, entre otros [14].

El parámetro `-effort` hace que la herramienta incorpore niveles mayores o menores de esfuerzos para obtener mejores resultados en el reporte (QoR). Para ello se pueden configurar tres niveles, los cuales son: `low`, `medium` y `high`. La herramienta por defecto establece un

nivel medio si no se configura nada. El nivel bajo (*low*) hace que la herramienta no invierta tanto tiempo para mejorar el resultado del reporte (QoR). El nivel alto (*high*) realiza la operación contraria al nivel *low*, haciendo que la herramienta se exceda en el tiempo para mejorar el reporte (QoR) [14].

El parámetro `-congestion` habilita algoritmos para perfeccionar la eliminación de congestión dentro del área del *core*. Esto hará posible que se refinen las capacidades para realizar el *Route*. Por defecto esta opción se encuentra deshabilitada [14].

Además de estos existen muchos otros comandos que se podrán configurar. Estos comandos perfeccionan el *Placement* y el *Routing* en el diseño actual. Si se prefiere podrá configurarse también la red de reloj (*clock*) en el diseño.

Los comandos a utilizar son los siguiente:

```
place_opt -effort high
```

```
place_opt -congestion
```

Una vez realizada la optimización pertinente del *Placement*, se podrá realizar la verificación del diseño. Para ello se deberá generar nuevamente el comando descrito al inicio de la sección.

Creada la verificación y la optimización del circuito en la fase actual se podrá seguir con el tema de *Routing*. Este punto será de suma importancia para cualquier circuito que se esté implementando. En el siguiente capítulo se abarcará sobre este tema.

CAPÍTULO 10

Routing

Este capítulo toma lugar con la interconexión de componentes dentro del *chip*. En este capítulo se utilizará el termino *Routing* de referencia para las interconexiones. El proceso de *Routing* es de suma importancia para interconectar el circuito y así lograr que este realice su propósito.

El sistema utilizado anteriormente por los estudiantes predecesores de este proyecto para llevar a cabo el proceso de síntesis física, no posee un orden. El presente trabajo establece una base más detallada del proceso que se debe seguir para lograr un buen diseño. Así bien, el proceso a seguir no es único y existen diferentes maneras de realizarlo.

Para el desarrollo de una buena interconexión de componentes se ha establecido un proceso el cual cumple con la función de realizar un *Routing* funcional. Este proceso que se establece al igual que los procesos anteriores no es único. Dependerá de la tecnología que se esté utilizando y los circuitos que se estén implementando.

En este capítulo se estudiará la manera de realizar una buena configuración para el *Routing* de celdas. Además se explorará la creación de interconexiones a los anillos de Voltaje y Tierra, VDD y VSS respectivamente. Se realizarán análisis para verificar que todo este conectado y así poder lograr reducir errores en etapas posteriores del *Flujo* del *chip*.

El proceso o la metodología a seguir para realizar un buen *Routing* de componentes puede observarse en la Figura #52. Para este proceso se deberá tener configurado las conexiones de las *nets* de VDD y VSS. Además de poseer los archivos que contienen las reglas de diseño.

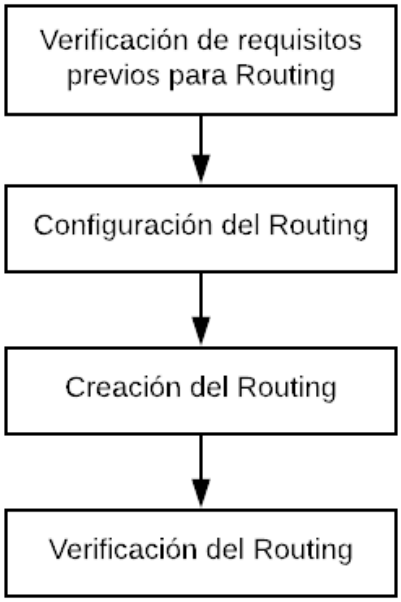


Figura 52: Metodología a seguir para el desarrollo de un *Routing*

Al igual que en los capítulos anteriores, se tomarán circuitos básicos para el desarrollo del tema. Esto con el fin de dar una explicación al lector de los pasos importantes que se requieren para elaborar un buen diseño. Los circuitos a utilizar son los siguientes: compuerta *Not*, *Full Adder* y *Ripple Carry Adder*.

Es importante llevar a cabo el proceso del *Routing* después de haber finalizado el *Placement* de componentes y las creaciones de anillos VDD y VSS [15].

10.1. Tipos de *Routing*

Actualmente en la herramienta de *IC Compiler* existen dos formas comunes de realizar el *Routing* entre componentes. El uso de estas depende de la tecnología que se esté utilizando. Los dos métodos que incorpora la herramienta son: *Classic Route* y *ZRoute*.

Las configuraciones de *Routing* existentes permiten la conexión entre celdas. Una de las diferencias que se debe tomar en cuenta a la hora de utilizarlas es el tamaño de la tecnología que se está implementando. Al utilizar el *Route* clásico se limita la tecnología que se puede usar. Esto es debido a que esta forma solo acepta tecnologías arriba de 45 nm [15].

El *Routing* clásico es la forma original que se implementaba en *IC Compiler*. Es la manera en la que actualmente se utiliza en el paquete del software de *IC Compiler-XP*. En la actualidad para los paquetes *IC Compiler*, *IC Compiler-DP* y *IC Compiler-PC* es una manera opcional que se puede llevar a cabo [15].

Es recomendable no utilizar las dos formas de *Routing* en el flujo del diseño. Esto puede ocasionar un error. Si se utiliza, se deberá ejecutar de primero el *Route* clásico y posteriormente el *ZRoute*. Ejecutando de primero el *ZRoute* y luego la manera clásica el programa mostrará un error [15].

La manera en la que operan estas dos formas es muy similar. Las dos se rigen por un flujo para cumplir con el *Route* del diseño actual. Ambas formas deben tener tres requisitos antes de operar. Estos requisitos son los siguientes:

1. En el diseño antes del *Routing* deben existir las conexiones de voltaje y tierra [15][16].
2. Las conexiones de *clocks* no deben estar presentes en esta instancia [15][16].
3. Tiene que estar establecida la *Technology file* con las reglas de diseño [15][16].

Ambas maneras de crear los *Routing* utilizan reglas de diseño para ello. Esta información esta detallada en la *Technology file*. Esta contiene la información sobre las características físicas y eléctricas de cada capa de metal, así como las reglas de las interconexiones [17].

La configuración que se utilizará para el *Routing* del *chip* que se esta desarrollando será la de: *Route* clásico. Debido a que este contiene todas las características que se requieren para el desarrollo del *chip*. Esto no quiere decir que no se puedan utilizar otras maneras. Al contrario, existe la posibilidad de utilizar las dos pero haciendo uso del *Route* clásico antes que el *ZRoute*.

10.2. Verificación de requisitos previos para Routing

El proceso de realizar el *Routing* es complejo. La metodología que se implementa es básica, ya que actualmente no hay tantos componentes en el circuito para hacer pruebas, por ejemplo: *flip-flops*, *clock*, etc.

Es importante tener en cuenta antes de realizar el *Routing* los requisitos antes mencionados. Ya que si no se cumplen con estos el sistema no sabrá que realizar y podrá generar conexiones aleatorias y de tamaños diversos o en casos extremos dar error.

El primer requisito para desarrollar las interconexiones en el circuito es: conexiones de poder. Las conexiones de poder (voltaje y tierra) se llevan a cabo con el uso del comando `derive_pg_connection`. Este comando se explico previamente en el capítulo del *Floorplan*.

Para conocer sobre las conexiones de poder con las que cuenta nuestro diseño actual se debe utilizar el comando `report_pg_net`. Este mostrará un reporte detallado de todas las conexiones de voltaje y tierra existentes en el circuito. Si se requiere conocer la información acerca de una red, por ejemplo voltaje se debe de indicar con el parámetro: `-net`. Si por el contrario se quisiera conocer sobre la información de las conexiones de los pines con las *nets* se debe de agregar el parámetro `-connections`. Por defecto la herramienta al pasar solo el comando muestra todas las conexiones de las redes (*nets*), pero no muestra la conexión de los pines [14].

La Figura #53 muestra el reporte de conexiones al pasar el comando sin ningún parámetro. En este se puede notar la cantidad de *nets* totales, el tipo de cada una de ellas (voltaje o tierra) y las conexiones que posee cada una.

```
icc_shell> report_pg_net

****************************************
Report : pg net
Option :
Design: Not_IO
Version: O-2018.06-SP5
Date : Tue Sep 22 12:24:09 2020
****************************************

Net                                                    Type    Power Domain          Num Connections
-------------------------------------------------------------------------------------------------------
Compuerta/VSS                                          ground                        2
VSS                                                    ground                        3
Compuerta/VDD                                          power                         2
VDD                                                    power                         3
-------------------------------------------------------------------------------------------------------
Total 4 nets
```

Figura 53: Reporte generado con el comando *report_pg_net* para el circuito *Not*

Al agregar el parámetro de `-connections` el reporte adjunta los datos presentados en la Figura #53, seguido de un listado con las conexiones de los pines. Este reporte puede llegar a ser grande, y habrá que revisar si todo esta en orden. La Figura #54 muestra una porción del reporte generado con el parámetro `-connections`.

```
Connections

  net Compuerta/VSS connected with:

     Compuerta/VSS
     Compuerta/U1/VSS    (pg)

  net VSS connected with:

     Compuerta/VSS
     U7/VSS    (pg)
     U6/VSS    (pg)

  net Compuerta/VDD connected with:

     Compuerta/VDD
     Compuerta/U1/VDD    (pg)

  net VDD connected with:

     Compuerta/VDD
     U7/VDD    (pg)
     U6/VDD    (pg)
```

Figura 54: Reporte generado con el parámetro *-connections* para el circuito *Not*

El segundo requisito que se debe verificar es el siguiente: no deben existir conexiones para *clock*. Ya que si existieran estas podrán ocasionar errores posteriores para *Design Rule Check*

(DRC). Así mismo podrá generar errores en conexiones futuras debido a superposición de estas. Para comprobar esto se podrá hacer uso de los siguientes dos comandos: `report_clock` ó `report_clock_tree`.

Ambas generan un reporte del *clock* en el diseño. Sin embargo el segundo comando, `report_clock_tree` da un reporte mas detallado de las características. Estas son la estructura y el tiempo del *clock tree* [14].

Por último se debe cumplir con el requisito de: *Technology file* establecido. Para este ultimo requisito la única manera de comprobarlo es revisando las configuraciones que se establecieron al empezar la síntesis física. Este archivo debe configurarse desde antes del *Floorplan*, al configurar la herramienta. La importancia de este se mencionó en la sección anterior. Los comandos se presentan a continuación.

```
report_pg_net -connections
```

```
report_clock_tree
```

Los tres requisitos previos son los más importantes para el desarrollo del *Routing*. Existen otros tres requisitos que se pueden estar verificando constantemente, estos son: la congestión, que el tiempo sea aceptable y verificar que no existan capacitancias máximas [15]. Si se quisiera revisar una por una se debería de pasar un comando que realice esa operación. Por ejemplo para estar verificando la congestión se puede pasar el comando `report_congestion`.

Los últimos tres requisitos mencionados se pueden comprobar de una manera rápida. Esto se realiza mediante la verificación del diseño para comprobar si este cumple para que se realice el *Routing*. Esto debe de verificarse luego de tener finalizado el *Placement* [15]. Para ello se utiliza el comando `check_routeability`.

El comando `check_routeability` hace que la herramienta verifique el punto de acceso de los pines, las instancias de los espacios para los cables de las celdas, las reglas de diseño para los pines, etc [14]. Este comando genera un reporte que contiene todos los errores y una lista de violaciones que se tienen en el diseño actual.

Si este reporte sale sin ninguna violación se podrá proseguir para realizar el *Routing*. El comando genera un archivo que contiene estos informes, el cual es nombrado a partir de la celda superior del diseño. Este contiene la extensión *.err* que significa que el archivo mantiene todos los errores que hemos generado [15][14].

Si el reporte generado se quisiera guardar con un nombre propio este se puede generar con el parámetro `-error_cell`. Este solo se podrá usar si existiera un error [14]. Por lo regular este parámetro no se configura ya que es mas fácil acceder con el nombre de la celda superior.

Por lo tanto, el comando para verificar si el diseño es enrutable se configura de la siguiente manera:

```
check_routeability
```

Para diseños más complejos es muy recomendable corregir cada error presente en este. Luego de haber ejecutado el comando anterior se podrá conocer la ubicación de cada error.

Para esto se debe de ejecutar el comando `report_error_coordinates`. Este comando por si solo no ejecuta ninguna acción y si se ejecuta solo así muestra un error. El error indica que no se agrego ningún archivo que contenga el listado de violaciones en el diseño. Es por esto que se debe agregar el nombre del archivo generado anteriormente. Este archivo tiene que contener el mismo nombre con la extensión *.err* [15][14]. El comando se verá de la siguiente manera:

```
report_error_coordinates Not.err
```

En la siguiente sección se abarcará la configuración básica para el *Routing* y se llevará a cabo el proceso de creación de este.

10.3. Configuración y creación del *Routing*

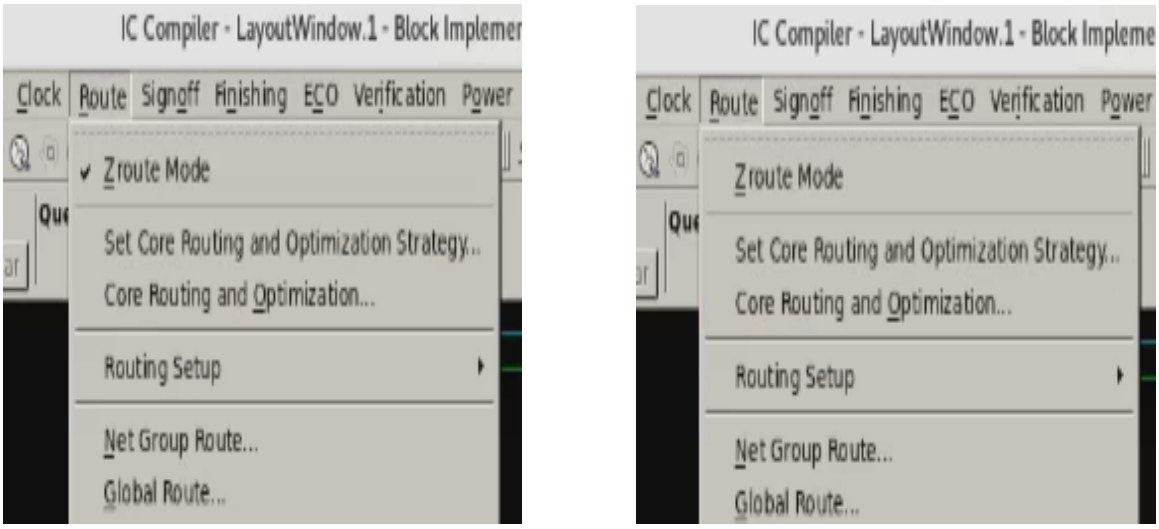
Realizada la verificación de los requisitos previos para llevar a cabo el *Routing* se prosigue a la configuración de este. Este proceso es singular, ya que se puede realizar de varias maneras. La configuración dependerá demasiado del diseño que se este elaborando, ya que se pueden establecer configuraciones únicas y especiales para circuitos específicos.

El proceso que se detalla en esta sección es básico. Pretende establecer los pasos a seguir para elaborar la interconexión de componentes en un circuito integrado. Así mismo el proceso de pruebas que conlleva realizar una interconexión exitosa permitirá reducir los errores de diseño en etapas posteriores y poder así llevar a cabo el proceso de fabricación.

Para realizar la configuración de la herramienta se deberá especificar el tipo de *Routing* que se utilizará. Para el *ZRoute* no se deberá pasar ningún comando. Esto es por que la herramienta por defecto realiza las interconexiones con este como se explicó con anterioridad. Para configurar la herramienta y establecer el *Routing* clásico se deberá deshabilitar el *ZRoute*. Para ello se utiliza el comando `set_route_mode_options`. Este comando permitirá configurar el tipo de ruteo que se quiere establecer. Este comando solo acepta operadores booleanos: *true* o *false* (verdadero y falso respectivamente). Por defecto la herramienta está configurada con el valor de *true*. Por ende, para realizar la configuración del *Routing* clásico el comando deberá de establecerse con el valor *false* [14][15]. A continuación se muestra la forma del comando completo.

```
set_route_mode_options false
```

La configuración del *Routing* también se puede realizar por medio de la interfaz gráfica. Para habilitar o deshabilitar la opción se deberá seguir la siguiente ruta. En la pantalla de *Layout* habrá que seleccionar la pestaña de *Route* y dentro de ella se deberá hacer *click* en la opción *Zroute Mode*. La Figura #55 muestra la opción de *ZRoute* en *true* o en *false* (habilitado o deshabilitado).



(a) Opción en *true* (b) Opción en *false*

Figura 55: Método para habilitar el *Routing* clásico.

Habilitada la opción de *Routing* clásico se podrá continuar con la configuración de este. La configuración posterior que se llevará a cabo dependerá mucho del circuito utilizado. Esto se debe a que se podrán definir varios aspectos de carácter único o especiales. Así mismo dependerá del diseñador encargado en esta área, ya que él tomará las decisiones finales.

Las configuraciones posteriores no se tratarán a detalle en el presente trabajo. Ya que los circuitos con los que se esta trabajando no cumplen con requisitos a estos. Con las configuraciones se podrá especificar una ruta que permita ser una guía para las conexiones. Además se podrán definir reglas propias que no se encuentren por defecto en el programa o en las reglas de diseño. Para esto se debe estar consciente de que el proceso pueda fallar o no cumplir con los criterios que pide el fabricante.

Para crear las rutas se utilizaría el siguiente comando `create_route_guide`. Las rutas permiten que las redes de señales o de poder se mantengan completamente conectada. Con este se podrán corregir violaciones y se podrá tener un control amplio de la densidad y la dirección de los cables que realizan las interconexiones [15].

De la misma manera existen configuraciones que permitirán modificar la dirección de las interconexiones. Estas modifican las direcciones de las capas de metales que están establecidas por defecto. Es decir, con estas configuraciones se puede establecer que la capas de metal **"M2"**, **"M3"**, etc., se utilicen solo para realizar el *Routing* de manera vertical u horizontal. Para modificar esto se utiliza el comando `set_preferred_routing_direction` seguido de los parámetros: `-layers` y `-direction`. Al parámetro `-layers` se deberá especificar la capa o las capas que se quieran modificar. En el parámetro `-direction` se especifica la dirección de estas, horizontal o vertical. Al pasar este comando se sobre escriben las direcciones establecidas en las librerías, y esto puede generar una cantidad alta de *Warnings* [14][15]. El ejemplo de las dos capas antes mencionadas quedaría de la siguiente forma:

```
set_preferred_routing_direction -layers {m2 m3} -direction
   horizontal
```

Si se desea conocer las direcciones establecidas antes o después de modificarse habrá que usar el comando `report_preferred_routing_direction`. Este comando generará una tabla con las direcciones de cada capa. la Figura #56 muestra el reporte de las direcciones establecidas. A continuación se presenta el comando:

```
report_preferred_routing_direction
```

```
****************************************
Report : Layers
Design : Not_IO
Version: O-2018.06-SP5
Date   : Thu Sep 24 15:50:06 2020
****************************************

Layer Name                  Library           Design            Tool understands
METAL1                      Horizontal        Horizontal        Horizontal
METAL2                      Vertical          Vertical          Vertical
METAL3                      Horizontal        Horizontal        Horizontal
METAL4                      Vertical          Vertical          Vertical
METAL5                      Horizontal        Horizontal        Horizontal
METAL6                      Vertical          Vertical          Vertical

1
icc_shell>
```

Figura 56: Reporte generado para el circuito *Not*

Si al modificar la dirección de la capa para realizar el *Routing* estuviese incorrecta o simplemente no se está satisfecho, esta se puede remover. Para remover la configuración se debe ejecutar el comando `remove_preferred_routing_direction`. Este comando únicamente removerá las direcciones que se han establecido por el diseñador [14][15].

El *Routing* clásico soporta dos formas de establecer reglas para llevar a cabo la interconexión. La primera de ellas es de manera automática por el programa. El programa decidirá las mejores rutas y direcciones que mejor se acomoden al circuito. La segunda manera es estableciendo reglas que no estén por defecto en la herramienta. Para ello se debe definir que se quieren utilizar reglas propias y posteriormente crearlas [15]. Para circuitos mas específicos con tamaños y áreas limitadas, esta opción resulta útil, sin embargo para el desarrollo del circuito integrado bastará con una configuración básica.

De la misma manera que se puede seleccionar la dirección que tendrán las conexiones se pueden definir las capas que se quieran conectar. Este tema es complejo, ya que establecer una, dos o mas capas para realizar las conexiones dependerá de los requisitos del fabricante. El fabricante en este caso TSMC, estableció que para el desarrollo del chip de la Universidad del Valle de Guatemala se pueden utilizar seis capas, M6. En dado caso se puede trabajar en las diferentes capas. Si por alguna razón la empresa de TSMC indicará que de las seis capas solo se puedan usar dos, habrá que respetar dicha decisión y trabajar bajo esas restricciones.

Por lo tanto la herramienta de IC Compiler permite seleccionar qué capas se quieren trabajar y qué capas se quieren ignorar [15]. De la misma manera se pueden establecer capas para redes específicas, por ejemplo una capa solo para voltaje y otra solo para tierra.

Al decidir qué capas utilizar habrán que definirlas posteriormente en la herramienta. Para ello se utiliza el comando `set_ignored_layers` [14][15]. Este comando acepta un rango que establecerá qué capas utilizar. Para definir el rango se utilizan los parámetros `-min_routing_layer` y `-max_routing_layer`. El parámetro `-min_routing_layer` define el límite inferior de las capas a utilizar, este valor se definirá con el nombre de la capa que se quiera establecer como tal. El parámetro `-max_routing_layer` establece el límite superior de las capas, al igual que el otro este se define con el nombre de la capa a usar [14][15]. Por ejemplo, si para realizar las conexiones se requieren utilizar las capas M3, M4 y M5 estas se podrán definir de la siguiente forma:

```
set_ignored_layers -min_routing_layer M3 -max_routing_layer M5
```

Por ende todas las capas que no se encuentren dentro de este rango serán ignoradas [14][15]. Este ejemplo se ilustra en la Figura #57 con el fin de dar una explicación gráfica. Por defecto la herramienta también establece qué capas serán ignoradas para las estimaciones de *Rule Check* (RC) y el análisis de congestión. Estos análisis por defecto se configuran en las capas en las cuales se crean las conexiones. Al definir las capas M3 hasta M5 para llevar a cabo el *Routing* automáticamente se establecieron esas mismas capas para realizar los análisis. Las capas para llevar a cabo esos análisis puede cambiarse haciendo uso del parámetro `-rc_congestion_ignored_layers`. Este recibe una lista de las capas en donde se deseen realizar los análisis, por ejemplo M2, M3 y M6. Al modificarlo se rompe el vínculo de las capas definidas automáticamente y se pueden establecer capas en las que no se creen conexiones [14][15]. El comando obtiene la siguiente forma:

```
set_ignored_layers -rc_congestion_ignored_layers {M2 M3 M6}
```

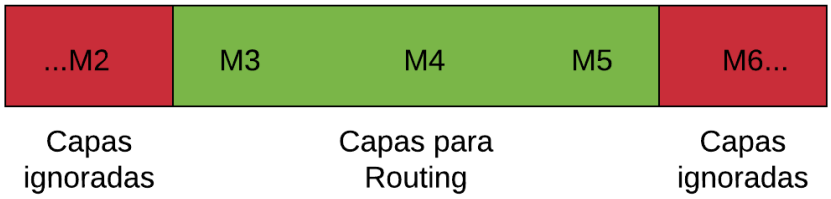


Figura 57: Rango de capas definidas para realizar el *Routing*

De igual forma se pueden establecer capas para *nets* especificas, esto se logra con el comando `set_net_routing_layer_constraints`. Al igual que el comando anterior se podrán establecer un rango de capas que sean de utilidad para una *net* en especifica, o diferentes *nets* pero con diferentes rangos. Para lograr esto se utilizan los parámetros `-min_layer_name` y `-max_layer_name`. Antes de definir las capas que se usarán para una *net* en especifico habrá que indicar el nombre de esta que se quiere configurar. Para ello se agrega una lista con el nombre de la *net* que se usará, por ejemplo: `set_net_routing_layer_constraints {net VDD}` [14] [15]. Este comando sobrescribe cualquier restricción que se encuentre establecida. Por ejemplo, si ya se encuentra definida una condición sobre la *net* a utilizar la herramienta despliega un *warning* para avisar que los datos se modificarán [15]. Además el nombre de la *net* deberá estar contenida en la librería física del diseño, o esta se tuvo que crear por el usuario. Si no fuera este el caso el programa despliega un error ya que no

hay correlación con los datos del diseño [15]. El comando completo se ve a continuación, así mismo se muestra un ejemplo para definir un rango de capas para VDD y otro para VSS.

```
set_net_routing_layer_constraints {netA} -min_layer_name M1 -
   max_layer_name M3
```

```
set_net_routing_layer_constraints {VDD} -min_layer_name M1 -
   max_layer_name M2
```

```
set_net_routing_layer_constraints {VSS} -min_layer_name M3 -
   max_layer_name M4
```

Configuradas las capas para establecer conexiones únicas en el circuito, se pueden establecer ciertas configuraciones especificas que dan un mayor desempeño al realizar el *Routing* clásico. Estas permiten tener un control sobre el *Routing*: global, de las pistas y el detallado [15]. El comando a utilizar es: `set_route_options`. Este comando modifica las conexiones globales en todo el circuito, así como modifica las conexiones de las pistas y permite configurar opciones diferentes para el *Routing* detallado. El uso de este comando no se explicará en el presente trabajo, ya que por defecto la herramienta brinda una configuración básica y funcional. El uso de este comando podrá realizarse de igual forma por medio de la interfaz gráfica. Para ello se debe seleccionar la siguiente ruta: **Route> Routing Setup> Set Route Options**. Luego de configurar las opciones se podrá realizar una afinación a cada uno de los diferentes *Routing*: global y detallado. Para lograr esto se utilizan los comandos `set_groute_options` y `set_droute_options` [15].

Previamente a realizar el *Routing* entre componentes se deberán de realizar las siguientes interconexiones:

- Las interconexiones entre los *pads* de voltaje y tierra con los anillos de poder.
- Las interconexiones de las celdas con los anillos de poder.

Estas interconexiones se podrán realizar antes o después de la creación de la interconexión entre celdas. En este caso se optó por agregarlos al principio, ya que al realizar pruebas se verificó que el resultado es el mismo.

Para realizar las interconexiones de los puntos antes mencionados se deberán establecer variables en el programa. Estas variables pueden ser de tipo *var* o de valor. Para realizar ello se utiliza el comando `set_app_var` [14]. En la configuración utilizada para el diseño del circuito se definieron dos variables; una para voltaje y otra para tierra, estas se definieron de la siguiente manera:

```
set_app_var logical mw_logic1_net "VDD"
```

```
set_app_var logical mw_logic0_net "VSS"
```

Posteriormente a esto se realizaron nuevamente las conexiones lógicas de voltaje y tierra. En este caso se agregaron nuevamente las *nets* de voltaje y tierra, así como los pines de estos mismos. Esto como se explicó en el capítulo de *Floorplan*, debe realizarse cada vez que se modifiquen las redes de alimentación. El comando usado para realizar esto es el siguiente:

```
derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS -
    ground_pin VSS
```

Posterior a este se deberá nuevamente realizar estas conexiones solo que esta vez solo se deberán agregar las *nets* usadas y el parámetro `-tie`. El comando a utilizar es el siguiente:

```
derive_pg_connection -power_net VDD -ground_net VSS -tie
```

Estos comandos preparán las redes que se utilizarán y hacen posible que las interconexiones se puedan llevar a cabo. Una vez establecidos estos comandos se podrá realizar la interconexión de *pads* con los anillos de poder. Para realizar esto se utiliza el comando `preroute_instances`. Este realiza las conexiones entre los pines de los *pads* de voltaje y tierra con los anillos con los respectivos nombres. Este comando se guía por las reglas de diseño establecidas. Este puede generar las conexiones por tamaños de pines, distancias entre los pines y los anillos y por distintas *nets* usadas como objetivos [14]. Para establecer las reglas de diseño para esta fase se podrá utilizar el comando `set_preroute_drc_strategy` [14].

Debido a que los circuitos presentados no poseen tanta complejidad, el comando no se utilizara con las modificaciones que posee. Se recomienda utilizar estas modificaciones para circuitos mas complejos. Estos permitirán que se reduzcan más los errores de diseño en el circuito en desarrollo. La Figura #58 ilustra el comando `preroute_instances` utilizado. En este se puede observar que fueron conectados los *pads* de voltaje y tierra con los anillos de poder. La Figura #59 muestra a detalle las conexiones con los anillos. El comando utilizado se muestra a continuación:

```
preroute_instances
```

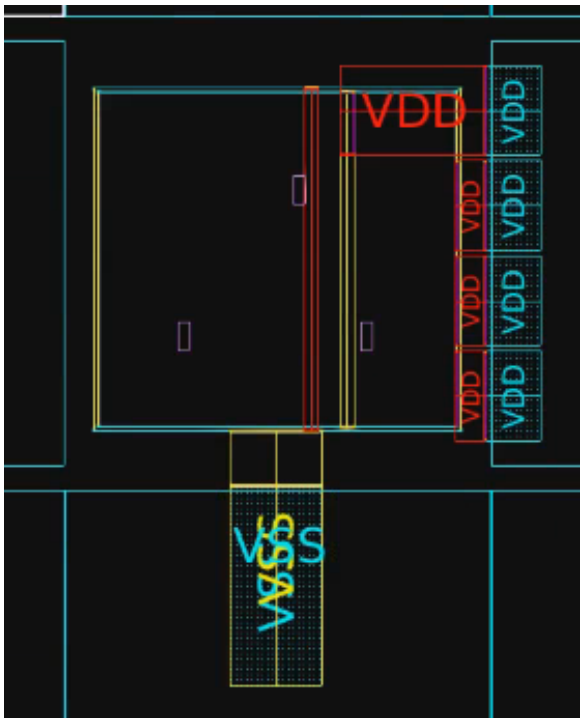


Figura 58: Conexiones de VDD y VSS entre *pads* y anillos; circuito *Not*.

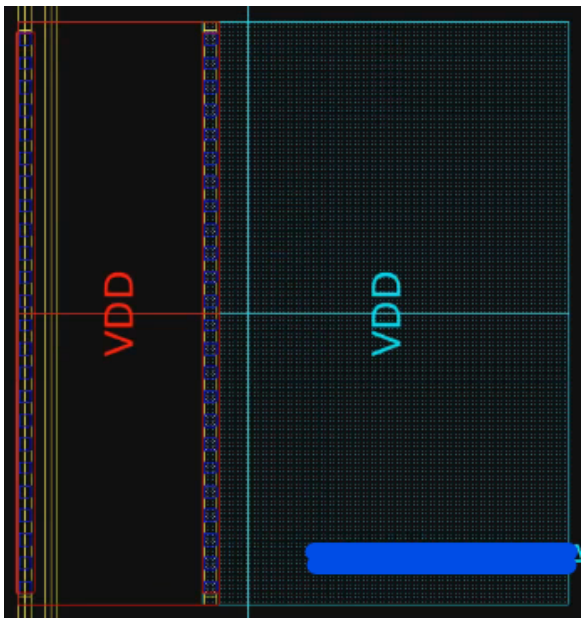


Figura 59: Detalle de los anillos conectados.

Para realizar las conexiones entre las celdas y los anillos, o así bien las celdas y los *power straps* se deberá ejecutar el comando `preroute_standard_cells` [14]. Este realiza las conexiones que alimentan a cada celda dentro del circuito integrado. Este comando permite realizar varias modificaciones de las interconexiones así como: el modo de las conexiones,

de que forma realizar las conexiones (horizontal o vertical), las redes que se conectarán, conexiones dentro del área de trabajo, etc. Estas modificaciones dependerán del diseño que se este realizando [14].

En los circuitos utilizados en el presente trabajo se modificaron únicamente con tres parámetros, estos son: `-mode`, `-connect` y `-nets`. El parámetro `-mode` conecta las celdas por tres factores diferentes: `rail`, `tie` y `net`. Es decir al pasar el parámetro `-mode net` la herramienta conectará a las celdas únicamente con las *nets* que estén definidas. De la misma manera se lleva a cabo el proceso con *rail* y *tie*. El parámetro `-connect` realiza las conexiones de tres formas posibles: horizontal, vertical o ambas. Para circuitos específicos estas conexiones podrán ser horizontales o verticales. En el presente trabajo se utilizaron ambas opciones. El parámetro `-nets` especifica la *net* a la que se quiere realizar las conexiones. En este caso se usaron VDD y VSS. Debido a que esto se utiliza para conectar las celdas con *nets* respectivas de voltaje y tierra, habrá que definir cada una de ellas por separado [14]. Los comandos utilizados para interconectar las celdas con las anillos o con los *power staps* se muestran a continuación.

```
preroute_standard_cells  -mode  net  -connect  both  -nets {VDD}
```

```
preroute_standard_cells  -mode  net  -connect  both  -nets {VSS}
```

La Figura #60 muestra las conexiones de las celdas con ambas fuentes de poder, anillos y *power straps*. La Figura #61 muestra una vista mas cercana de las conexiones.

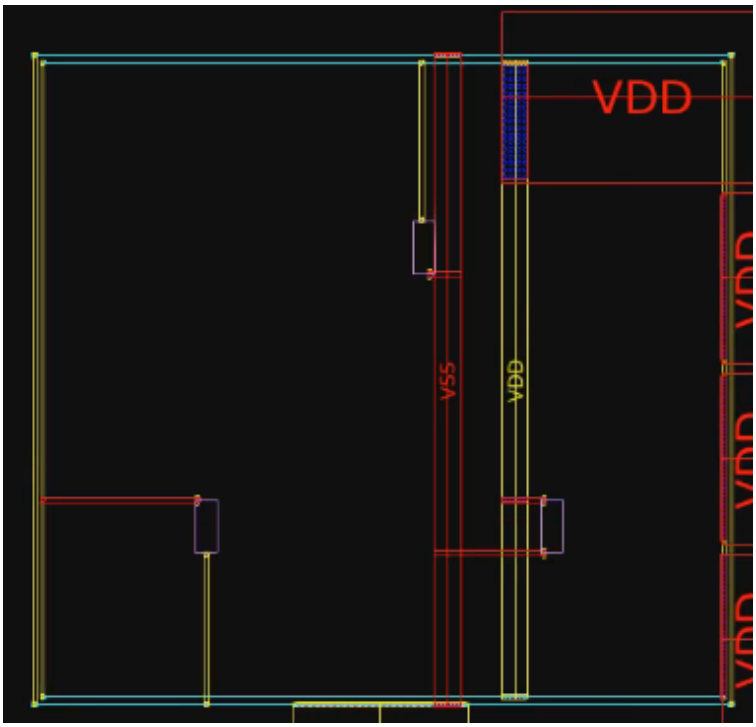


Figura 60: Conexiones de las celdas con ambas fuentes de poder en el circuito *Not*.

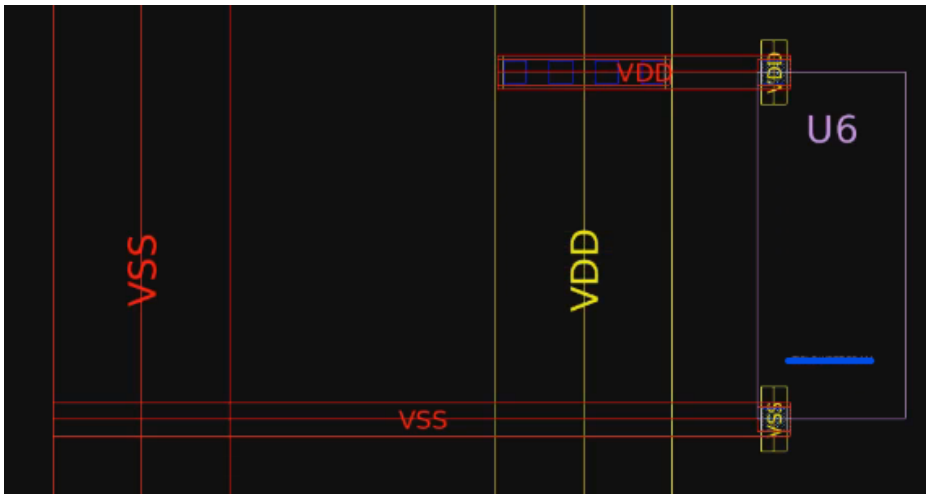


Figura 61: Vista cercana de las conexiones de celdas a fuentes de poder.

Como se explicó antes, el resultado de realizar estas operaciones después o antes de las interconexiones entre celdas es el mismo. Por motivos de orden se decidió agregarlos antes. Es importante no mantener celdas debajo de los *power straps*, esto genera error ya que se interfieren con las conexiones. Si el proceso de creación de estos *power straps* interfiere, habrá que regresar a la instancia de creación y corregirlo.

Como se mencionó antes, el comando `set_preroute_drc_strategy` establece las configuraciones en las reglas de diseño para optimizar el *Routing*. Este configura las reglas de diseño (DRC) para: los anillos, los *power straps*, las configuraciones de `preroute_instances` y `preroute_standard_cells` [14][15]. El uso de este comando estará establecido por el diseñador, ya que con este la herramienta provee una estrategia para optimizar el *Routing* y el *Post Routing* [15]. El comando es muy específico al usarse, este queda a criterio del diseño y del diseñador, este se muestra a continuación:

```
set_preroute_drc_strategy
```

Realizadas las conexiones de los *pads* con los anillos de poder, las conexiones de las celdas a las fuentes de poder y realizadas todas las configuraciones, es posible crear las interconexiones entre celdas. Para ello se utiliza el comando `route_opt`. Este genera y mejora de manera simultánea las conexiones entre las celdas y las conexiones posteriores a este [14]. Por defecto el comando genera un *Routing* amplio con las configuraciones del *Routing* global y detallado antes establecidas, si estos no son especificados antes la herramienta los configura de manera automática [15].

Con el comando se puede especificar el área que se quiere cubrir al realizar las conexiones. Además se puede establecer una conexión por fases, es decir se puede iniciar por un *Routing* global, *tracks* y detallado. Este permite ejecutar una opción de *crosstalk* para reducir errores y optimizar las conexiones y la integridad de la señal: *signal integrity* (SI) [14]. Así mismo se puede definir el nivel de esfuerzo realizado por la herramienta para que esta lleve a cabo la operación de crear las interconexiones [15]. Este comando también mejora y reduce la fuga de potencia en las conexiones [15].

Todos las modificaciones mencionadas se llevan a cabo al usar el comando junto a sus parámetros respectivos. Para los circuitos que se están desarrollando la reducción de algunos de estos parámetros no es necesaria. Esto se debe a que la cantidad de celdas es de baja densidad y no existen interferencias de celdas ni de conexiones. Para llevar a cabo la interconexión de las celdas en los circuitos utilizados el comando se implemento en su forma estándar. La Figura #62, la Figura #63 y la Figura #64 ilustran las conexiones realizadas en todo el circuito. El comando utilizado se muestra a continuación:

```
route_opt
```

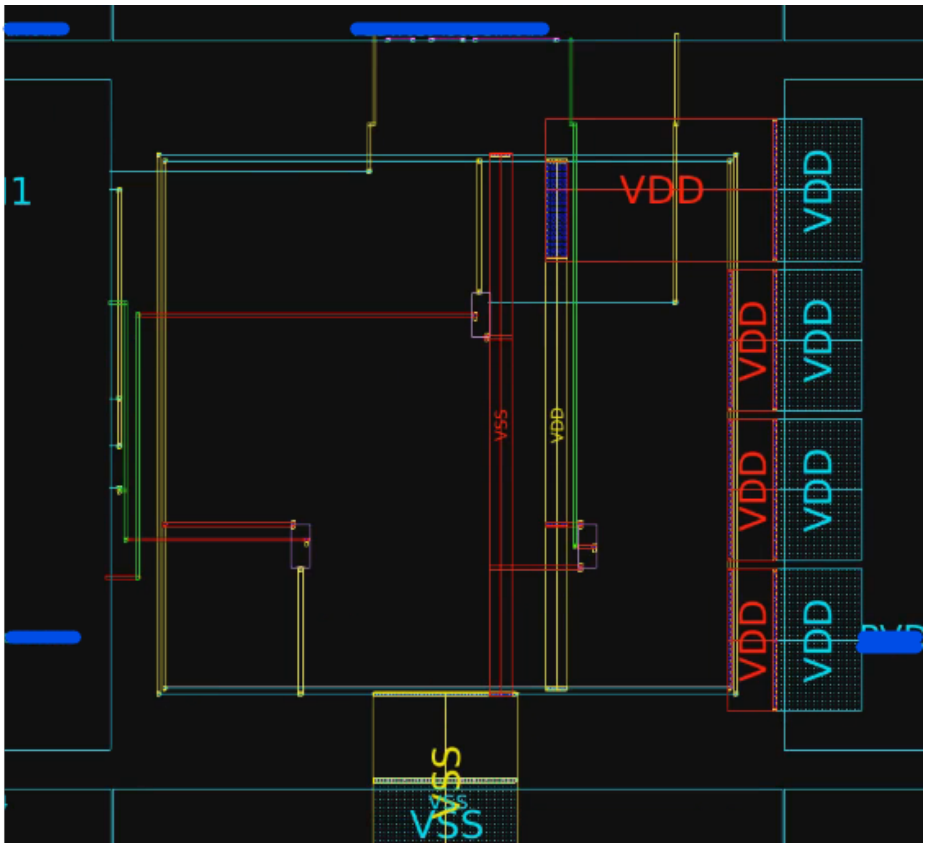


Figura 62: Conexión total de la compuerta *Not*.

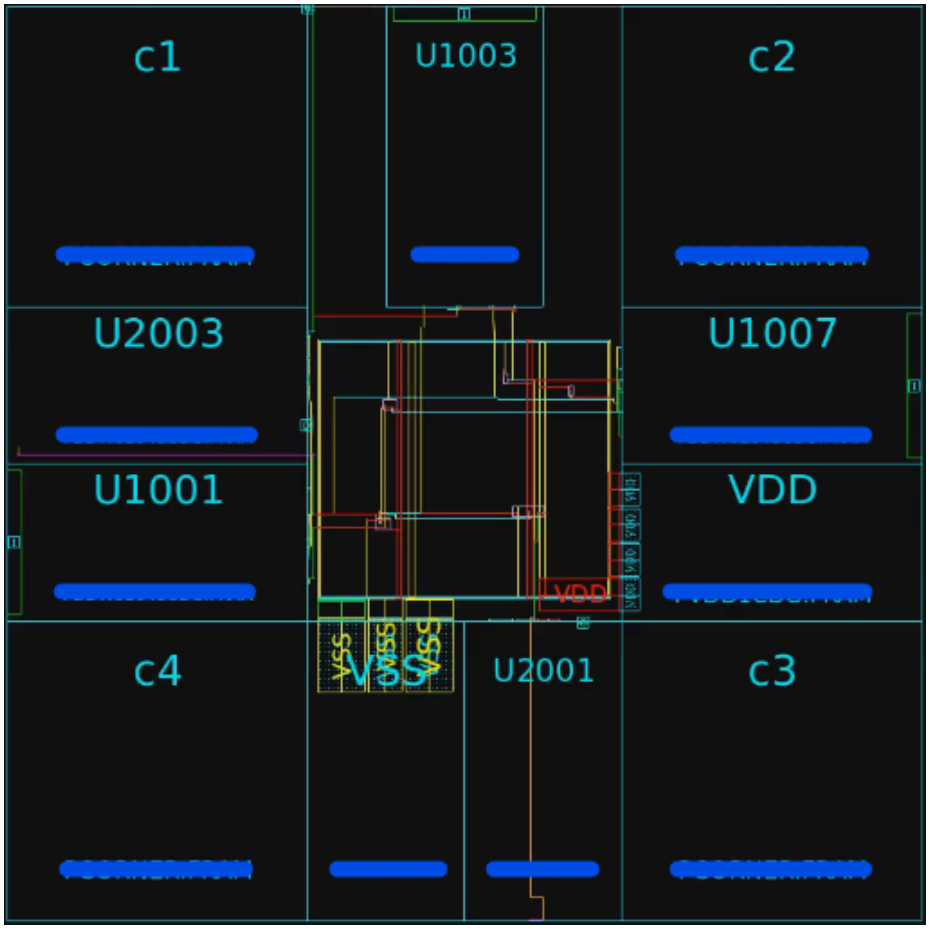


Figura 63: Conexión total del circuito *Full Adder*.

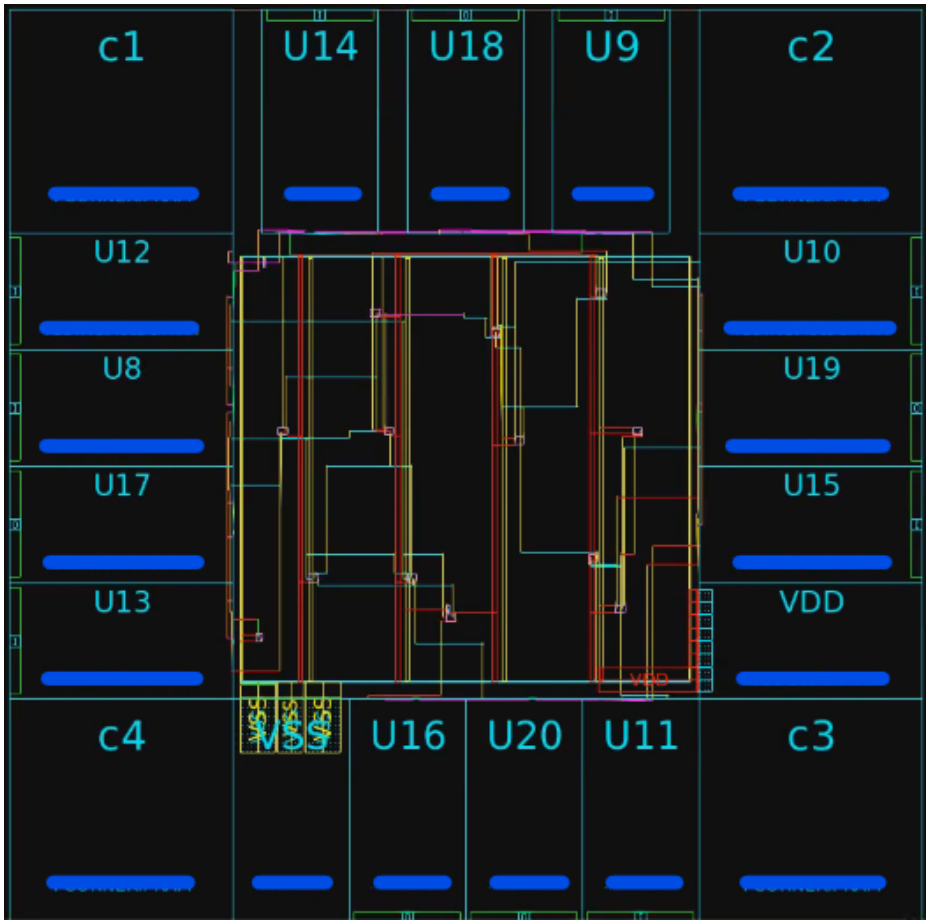


Figura 64: Conexión total del circuito *RCA*.

Finalizada las conexiones en el circuito se recomienda revisar cada una de ellas si fuese posible. Para circuitos pequeños este paso es recomendable ya que puede verificarse que todo este correcto y que cumpla con el diseño establecido en la fase de síntesis lógica. Al acercarse a una conexión podrá observarse el nombre que recibe el "cable".

Luego de llevar a cabo el proceso de *Routing* y realizar una verificación rápida de conexiones, se deberá establecer una verificación mas detallada. En la siguiente sección se muestra como llevar a cabo una verificación del cumplimiento de las reglas de diseño (DRC).

10.4. Verificación de *Routing*

En esta sección se abarcará una verificación y un reporte generado por la herramienta para verificar que las conexiones cumplan con los requisitos del diseño (DRC).

Al tener el circuito con las conexiones establecidas en ella se deberá de ejecutar un comando que se especializa en arreglar cualquier violación de DRC existente en el diseño. Este paso es opcional a realizar. Sin embargo se recomienda utilizar para circuitos en los que la verificación rápida desarrollada en la sección anterior no se pueda llevar a cabo. El

comando que realiza las correcciones de (DRC) es el siguiente: `focal_opt` [15].

El comando `focal_opt` se puede configurar para corregir errores de diseño en *nets* especificas o realizar las correcciones en todas ellas. Así mismo se puede configurar para llevar a cabo la corrección de pines específicos o corregir todos los presentes en el diseño. Para llevar a cabo la corrección de *nets* en el diseño se debe ejecutar el parámetro `-drc_nets`. Para llevarlo a cabo en los pines se ejecuta el parámetro `-drc_pins`.

Al utilizar el parámetro `-drc_nets`, este podrá configurarse de las dos maneras antes mencionadas. Para seleccionar una corrección especifica en cada *net*, deberá pasarse el nombre de la *net* que se quiera corregir. Si se quisieran corregir todas las *nets* establecidas se deberá agregar la opción de `all`. De manera análoga se deberá establecer estas dos variantes para la corrección de pines. Los siguientes ejemplos muestran las dos opciones posibles para corrección de las *nets*.

```
focal_opt -drc_nets {VDD}
```

```
focal_opt -drc_nets all
```

Este comando además de realizar correcciones para el cumplimiento de las reglas de diseño en *nets* y pines, también reduce el *crosstalk* y la fuga de potencia [14].

Para verificar que el proceso de creación del *Routing* se haya realizado de forma correcta se utiliza el comando `verify_route`. Este genera un reporte detallado de cada parte de las conexiones. A este comando se puede especificar si se quiere recibir ciertas redes en específico o que no genere reporte de DRC. Así mismo genera un reporte general de las violaciones de antena. Por defecto el comando genera un reporte de todos los parámetros [14][15]. El comando utilizado para el desarrollo del reporte de los circuitos es en su forma estándar, este se muestra a continuación.

```
verify_route
```

CAPÍTULO 11

Conclusiones

- A través de las pruebas realizadas se comprobó que la metodología establecida cumple con el desarrollo y creación de un *Floorplan* funcional, además de colocar e interconectar de manera satisfactoria los componentes de los circuitos planteados.
- Por medio del análisis expuesto, fue posible observar que se debe cumplir un orden para desarrollar de manera exitosa las fases de *Floorplan*, *Placement* y *Routing*. En donde se toma en cuenta las fases de preconfiguración, creación y verificación.
- Se logró delimitar un área para el posicionamiento de los componentes que conformaron los circuitos integrados utilizados al momento de realizar el *Floorplan*.
- Se determinó a través de pruebas que al realizar una buena colocación de los componentes e interconexión, se logró reducir errores en etapas posteriores en el flujo de diseño.
- A través de pruebas realizadas se comprobó que varias de las instancias creadas en la etapa de síntesis física como: *Pad's* y *corners*, se pueden llevar a cabo en la etapa de síntesis lógica.
- Se debe crear un plan que ayude a la colocación de los *Power Straps* para evitar que estos se posicionen sobre las celdas creando errores de conexión.
- Se mejoró el *script* proporcionado con el que ahora se puede llegar a concluir exitosamente el proceso de *Floorplan*, *Placement* y *Routing*.

CAPÍTULO 12

Recomendaciones

Para familiarizarse con la herramienta de *IC Compiler* se recomienda desarrollar el flujo de diseño base presentado por los Ingenieros Electrónicos Luis Nájera y Steven Rubio. Asimismo es recomendable leer las tesis [7] y [8] además de apoyarse con los vídeos presentados por ellos.

Para seguir en el proceso de aprendizaje y antes de desarrollar cualquier proceso del flujo del diseño se recomienda que se dedique tiempo a navegar por las diferentes formas de utilizar linux. Con esto el estudiante adquirirá una experiencia en cualquier parte de los programas utilizados.

Se recomienda leer y estudiar los manuales que brinda la empresa de *Synopsys*. Específicamente para mejorar el proceso de la síntesis física se recomiendan los siguientes manuales: *IC Compiler Design Planning User Guide, IC Compiler Tool Commands, IC Compiler Clasic Route User Guide, IC Compiler Implementation User Guide* y *Synopsys Technology File and Routing Rules Reference Manual*.

Se recomienda hacer uso de la plataforma de *Solvnet* para leer preguntas de profesionales en el tema y respuestas brindadas por la empresa *Synopsys*.

Se recomienda verificar detalladamente los archivos presentados por el fabricante, leer y analizar si los parámetros establecidos en los documentos son incluidos al momento de importar las librerías con las que se lleva a cabo la síntesis física y poder así reducir errores de diseño.

Se recomienda estudiar la posibilidad de realizar una exploración automática para definir el tamaño del área del *chip* como tal.

Se recomienda revisar cada una de las conexiones realizadas al momento de crear el *Routing*, ya que con esto se podrá revisar que cada *net* y cada conexión se encuentren bien establecidas.

Por último se recomienda migrar y estudiar la herramienta de *IC Compiler II*. Esto se

debe a que la herramienta de *IC Compiler* quedará descontinuada u obsoleta.

CAPÍTULO 13

Bibliografía

[1] C. Wang, G. D. Hachtel y F. Somenzi, *Abstraction refinement for large scale model checking*. Springer Science & Business Media, 2006.

[2] A. Tonk y S. Goyal, "A Literature Review on Leakage and Power Reduction Techniques in CMOS VLSI Design", *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 3, n.º 2, págs. 554-558, 2015.

[3] 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.

[4] H. Kaeslin, *Digital integrated circuit design: from VLSI architectures to CMOS fabrication*. Cambridge University Press, 2008.

[5] R. J. Baker, *CMOS: circuit design, layout, and simulation*. John Wiley & Sons, 2019.

[6] N. H. Weste y D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[7] 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.

[8] 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.

[9] 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.

[10] H. Bhatnagar, *Advanced ASIC chip synthesis*. Springer, 2002.

[11] *IC Compiler Comprehensive Place and Route System*, Synopsys.

[12] L.-T. Wang, Y.-W. Chang y K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[13] *IC Compiler™ Design Planning User Guide*, Synopsys.

[14] *IC Compiler™ Tool Commands*, Synopsys.

[15] *IC Compiler™ Clasic Route User Guide*, Synopsys.

[16] *IC Compiler™ Implementation User Guide*, Synopsys.

[17] *Synopsys Technology File and Routing Rules Reference Manual*, Synopsys.

CAPÍTULO 14

Anexos

14.1. *Script* para implementar la síntesis física

```
1  ##################################################################
2  ###
3  ###              Design Planning; Floorplan y Place and Route
4  ###
5  ###
6  ###              Trabajo presentado por Luis Abadia
7  ###
8  ###
9  ###              Este Script fue modificado por Luis Abadia,
10 ###               sobre el trabajo presentado por:
11 ###
12 ###                       Ingeniero Luis Najera y
13 ###                       Ingeniero Steven Rubio
14 ###
15 ### Sobre las modificaciones:
16 ###              Detalles sobre los comandos utilizados,
17 ###              el orden jerarquigo planteado,
18 ###              creacion del Floorplan, Placement de los
19 ###              componentes y la interconexion entre ellos
20 ##################################################################
21
22
23 #----------------------------Configuramos las librerias
24 #Le decimos la direccion a donde ir a buscar de las librerias de TSMC de
       sintesis logica
25
26 lappend search_path /home/administrador/Escritorio/FA_TSMC_DRC/Libs/
       tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM
27 set link_library " * tcb018gbwp7ttc.db tcb018gbwp7twc.db
       tcb018gbwp7tbc.db tpd018nvtc.db"
28 set target_library "tcb018gbwp7ttc.db"
29
30 #----------------------------Configuramos las TLUPLUS
31
```

```
32  set_tlu_plus_files -max_tluplus /home/administrador/Escritorio/
        FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE/tluplus/
        t018lo_1p6m_typical.tluplus -min_tluplus /home/administrador/
        Escritorio/FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE/tluplus/
        t018lo_1p6m_typical.tluplus -tech2itf_map  /home/administrador/
        Escritorio/FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE/tluplus/star.map_6M
33
34
35  #----------------------------Configuracion de la MilkyWay
36
37  #Revisar los warnings y datos faltantes.
38
39  #creamos la milkyway
40
41  create_mw_lib  -technology /home/administrador/Escritorio/FA_TSMC_DRC/
        Libs/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf   -mw_reference_library
        {/home/administrador/Escritorio/FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE
        /tcb018gbwp7t /home/administrador/Escritorio/FA_TSMC_DRC/Libs/iolib/
        tpd018nv} -bus_naming_style {[%d]}  -open  /home/administrador/
        Escritorio/LUIS_ABADIA/Milky_Aba
42
43  ##################################################################
44
45
46  #               3.3 V.
47
48  #----------------------------Configuramos las librerias
49  #Le decimos la direccion a donde ir a buscar de las librerias de TSMC de
        sintesis logica
50
51  #lappend search_path /usr/synopsys/TSMC/180/CMOS/G/IO3.3V/stclib/9-track/
        tcb018g3d3_280a_FE/TSMCHOME/digital/Back_End/milkyway/tcb018g3d3_280a
        /frame_only/tcb018g3d3/LM
52  #set link_library " * tcb018g3d3tc.db tcb018g3d3wc.db tcb018g3d3bc.db
        tpd018nvtc.db"
53  #set target_library "tcb018g3d3tc.db"
54
55  #----------------------------Configuramos las TLUPLUS
56
57  #set_tlu_plus_files -max_tluplus /home/administrador/Escritorio/
        FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE/tluplus/
        t018lo_1p6m_typical.tluplus -min_tluplus /home/administrador/
        Escritorio/FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE/tluplus/
        t018lo_1p6m_typical.tluplus -tech2itf_map  /home/administrador/
        Escritorio/FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE/tluplus/star.map_6M
58
59
60  #----------------------------Configuracion de la MilkyWay
61
62  #Revisar los warnings y datos faltantes.
63
64  #creamos la milkyway
65
66  #create_mw_lib  -technology /home/administrador/Escritorio/FA_TSMC_DRC/
        Libs/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf   -mw_reference_library
        {/home/administrador/Escritorio/FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE
        /tcb018gbwp7t /home/administrador/Escritorio/FA_TSMC_DRC/Libs/iolib/
        tpd018nv} -bus_naming_style {[%d]}  -open  /home/administrador/
        Escritorio/LUIS_ABADIA/Milky_Aba
```

```
67
68  ##################################################################
69
70
71
72  #----------------------------Importamos el Netlist
73
74  ## Full Adder:
75
76  read_verilog -dirty_netlist -allow_black_box -verbose {/home/
        administrador/Escritorio/FA_TSMC_DRC/Out/FA_syn_p_m.v}
77
78  ## NOT: read_verilog -dirty_netlist -allow_black_box -verbose {/home/
        administrador/Escritorio/LUIS_ABADIA/PruebaNot/NOT_SOPHIE.v}
79
80  ## RCA: read_verilog -dirty_netlist -allow_black_box -verbose {/home/
        administrador/Escritorio/LUIS_ABADIA/RCA/RCA_SOPHIE.v}
81
82
83
84  #----------------------------Importamos el SDC
85
86
87  ## Full Adder:
88
89  read_sdc -echo -syntax_only -version Latest "/home/administrador/
        Escritorio/FA_TSMC_DRC/Out/FA_sdc_p.sdc"
90
91  ## NOT: read_sdc -echo -syntax_only -version Latest /home/administrador/
        Escritorio/LUIS_ABADIA/PruebaNot/NOT_sdc_p.sdc
92
93  ## RCA: read_sdc -echo -syntax_only -version Latest /home/administrador/
        Escritorio/LUIS_ABADIA/RCA/RCA_sdc_p.sdc
94
95
96  #----------------------------Power and Ground Connections
97
98  set_app_var mw_logic1_net "VDD"
99  set_app_var mw_logic0_net "VSS"
100
101 derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS
        -ground_pin VSS
102 derive_pg_connection -power_net VDD -ground_net VSS -tie
103
104 report_cell_physical -connections
105
106 #----------------------------Creacion de Corners
107
108 create_cell {c1 c2 c3 c4} PCORNER
109 create_cell {VDD} PVDD1CDG
110 create_cell {VSS} PVSS1CDG
111
112 #----------------------------Agregando Constraints a los pads de I/O
113 #----------------------------Esto se hace antes de hacer el floorplan
114
115 set_pad_physical_constraints -pad_name "c1" -side 1
116 set_pad_physical_constraints -pad_name "c2" -side 2
117 set_pad_physical_constraints -pad_name "c3" -side 3
118 set_pad_physical_constraints -pad_name "c4" -side 4
```

```
set_pad_physical_constraints -pad_name "VDD" -side 3 -order 2
set_pad_physical_constraints -pad_name "VSS" -side 4 -order 2

report_pin_pad_physical_constraints

remove_pin_pad_physical_constraints


#----------------------------------Creamos el FloorPlan

create_floorplan -control_type width_and_height -core_utilization 0.7
    -core_width 5 -core_height 50 -no_double_back -top_io2core 10
    -bottom_io2core 10 -left_io2core 5 -right_io2core 5

#----------------------------------Ajustamos el floorplan
adjust_fp_floorplan

#----------------------------------Placement

#para revisar que no hayan errores, en el floorplan
#check_physical_design -help
check_physical_design

set_fp_placement_strategy -adjust_shapes on
report_fp_placement_strategy

create_fp_placement -effort High -max_fanout 800 -optimize_pins
    -congestion_driven -timing_driven -consider_scan

legalize_placement

#----------------------------------Anillos VDD VSS

create_rectangular_rings -nets {VDD VSS} -around core


#----------------------------------Power Straps
create_power_straps     \
          -direction vertical   \
          -start_at 155         \
          -num_placement_strap 4        \
          -increment_x_or_y 50  \
          -nets  {VDD}  \
          -layer METAL2 \
          -width 2      \
          -do_not_route_over_macros

create_power_straps     \
        -direction vertical     \
        -start_at 150   \
        -num_placement_strap 4  \
        -increment_x_or_y 50    \
        -nets  {VSS}    \
        -layer METAL3   \
        -width 2        \
        -do_not_route_over_macros

#para revisar que no hayan errores, en el floorplan
check_physical_design
```

```
175
176 #----------------------------------Routing
177
178 #-----------------------------Power and Ground Connections
179
180
181 set_app_var mw_logic1_net "VDD"
182 set_app_var mw_logic0_net "VSS"
183
184 derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS
    -ground_pin VSS
185 derive_pg_connection -power_net VDD -ground_net VSS -tie
186
187 preroute_instances
188
189 preroute_standard_cells -mode net -connect both -nets {VDD}
190 preroute_standard_cells -mode net -connect both -nets {VSS}
191
192 set_preroute_drc_strategy
193
194 #--------------Verificar el disenio para determinar si esta listo para el
     ruteo
195 check_routeability
196
197 #--------------Se tiene que pasar la error_cell que se creo en la
    verificacion para generar el reporte
198 # report_error_coordinates
199
200 #--------------Configuramos que no este habilitado el zroute para
201 #--------------hacer el clasic route para configuracion inicial
202 set_route_mode_options -zroute false
203
204 #--------------Creamos una guia para las rutas
205 # create_route_guide -coordinate {0 0 390 310}
206 #--------------Obtenemos la informacion de todas las guias de rutas
207 # get_route_guides
208
209 #--------------Obtenemos las opciones de las rutas, se puede configurar
    la ruta global
210 #--------------la ruta detallada y una variante de la global
211 report_route_options
212
213 #----------Global
214 #set_route_options
215 #----------variante de la global
216 #set_groute_options
217 #----------Detallada
218 #set_droute_options
219
220 #--------------Configurando las conexiones y parametros para ejecutar el
    ruteo
221 #--------------Este paso nos hace revisar errores de conexion, asi como
    problemas de tiempos
222 #--------------loops para corregir errores, asi como corregir y evitar
    que no exista X-talk
223 set_route_opt_strategy
224
225 #--------------obtenemos la informacion sobre los parametros que estan
    configurados, si este
```

```
#--------------comando se pasa antes, la configuracion que se mostrara
    sera la default.
report_route_opt_strategy

#--------------Una vez establecidos los parametros, podemos pasar el
    siguiente comando
#--------------el cual nos servira para generar el ruteo, si se pasa el
    comando solo asi,
#--------------se genera un ruteo con caracteristicas basicas aparte del
    las que ya se
#--------------han definido.
route_opt


verify_route
################################################################


#---------------------------Guardamos el disenio en Celdas
save_mw_cel  -design "Full_Adder.CEL;1"

#Cerramos las celdas
close_mw_cel

#Cerramos la milkyway
close_mw_lib

#------------------------Abrimos el trabajo
open_mw_lib /usr/synopsys/icc/bin/mw_TSMC_DRC_FA_lib_PAVA
```