UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de un Circuito Integrado con Tecnología de 180 nm Usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificaciones de Antena y Corrección de Errores Obtenidos

Trabajo de graduación presentado por José Adrián Ayala Escobar para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de un Circuito Integrado con Tecnología de 180 nm Usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificaciones de Antena y Corrección de Errores Obtenidos

Trabajo de graduación presentado por José Adrián Ayala Escobar para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

Vo.Bo.:

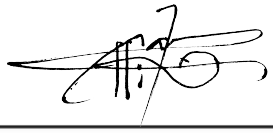
(f) _____

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.

Prefacio

El presente trabajo de graduación detalla los procedimientos y metodología a seguir para llevar a cabo el proceso de síntesis física del flujo de diseño VLSI, así como la posterior verificación de reglas de antena sobre el diseño generado. Al formar parte de un proyecto intergeneracional, este trabajo es un seguimiento a los esfuerzos y objetivos alcanzados en años anteriores. Cabe resaltar que todo el trabajo fue llevado a cabo de forma completamente remota debido a las medidas de bioseguridad implementadas a causa de la pandemia por el COVID-19. Por lo mismo, se agradece los esfuerzos realizados por la Universidad del Valle de Guatemala para buscar e implementar soluciones que permitieran que este trabajo pudiera ser llevado a cabo. También se agradece al MSc. Carlos Esquit por todo el conocimiento transmitido y el esfuerzo realizado para abrir las puertas al campo de la Nanoelectrónica a todos los estudiantes de la Licenciatura en Ingeniería Electrónica. También se agradece al Ing. Jonathan de los Santos por su constante apoyo durante la ejecución de este trabajo, especialmente su ayuda para trabajar con las herramientas de Synopsys y al Ing. Luis Nájera por la asesoría brindada, las recomendaciones provistas y su apoyo incondicional durante la realización del presente.

Índice

Lista de figuras

Lista de cuadros

Resumen

En el siguiente trabajo se detalla la metodología y flujo de trabajo propuesto para poder llevar a cabo la etapa de síntesis física del flujo VLSI en el sistema *IC Compiler II*, utilizando las librerías de 180 nm y 6 capas de metal del *foundry* TSMC. Esto con el objetivo de poder generar un diseño físico para cualquier circuito, sin errores durante la ejecución del flujo y sin errores de diseño. Para la verificación de esto último se detalla cómo se integra la Verificación de Antena al flujo propuesto. De esta manera pudiendo llevar a cabo dicha verificación y validando que los diseños generados, implementando la metodología propuesta, no violan las reglas de Antena del *foundry*.

Este trabajo forma parte de un proyecto que busca la automatización del flujo de diseño VLSI, de forma que se logre generar una herramienta que abra las puertas al campo del diseño de circuitos nanométricos en Guatemala. Al formar parte de un proyecto intergeneracional, se buscó continuar con el trabajo realizado en años anteriores y utilizar todos los avances alcanzados como base para el presente. Para esto último, se inició replicando los resultados presentados en estos trabajos previos con el objetivo de familiarizarse tanto con el proyecto como con las herramientas de trabajo. Posterior a esto, se llevó a cabo la planificación para actualizar el flujo de trabajo utilizado en años anteriores. Esto debido a que el flujo realizaba la síntesis física en el sistema *IC Compiler* y se propuso actualizarlo para permitir su ejecución en la versión más moderna *IC Compiler II*.

Para el proceso de actualización del flujo se inició con la lectura extensiva de los manuales de *IC Compiler II*, los cuales fueron provistos por su desarrollador la empresa *Synopsys*. Estudiando estos manuales se determinó que, debido a que muchos de los comandos utilizados eran obsoletos en el nuevo sistema, la actualización del flujo no sería tan fácil cómo se pensaba. Adicional a esto, se encontró que en el flujo propuesto previamente no se estaban ejecutando ciertos procesos útiles que brindan robustez al flujo y a los diseños generados. Por lo tanto, con los conocimientos adquiridos se decidió generar una nuevo flujo de trabajo, aunque siempre tomando como base los esfuerzos realizados previamente.

En el nuevo flujo aún se incluyen las etapas principales de la síntesis física: *Floorplan*, *Placement* y *Routing*. A estas fueron añadidos procesos clave como la preparación de librerías y el *power planning*. La metodología a seguir para la validación del flujo consistió en iniciar las pruebas utilizando una compuerta NOT, sintetizar esta compuerta de acuerdo al flujo

propuesto y por último verificar errores de ejecución. Luego de corregir estos errores en una nueva versión del flujo, el proceso fue repetido realizando nuevamente la síntesis sobre la compuerta, la validación y corrección de errores.

Una vez se logró que el flujo no generara errores durante su ejecución era necesario validar que el diseño final no tuviera errores de diseño, específicamente violaciones a las reglas de antena impuestas por el *foundry*. Para esto nuevamente se utilizó los manuales y los trabajos previos de forma que se pudiera determinar la mejor manera de integrar esta verificación al flujo. Con esto determinado se procedió a validar el flujo buscando que este fuera capaz de sintetizar una compuerta NOT sin errores. Esto pudo ser comprobado al ejecutar la verificación y validar que diseño no presentaba violaciones a las reglas de antena. Es importante mencionar que en trabajos anteriores esta verificación pudo ser llevada a cabo de forma exitosa, por lo que gran parte del trabajo realizado se encuentra en su integración al nuevo flujo de manera que los diseño generados pudieran ser verificados sin problemas.

Con el flujo validado para una compuerta NOT se pudo proceder a hacer pruebas con circuitos más complejos. Los circuitos utilizados para la validación del flujo fueron un XOR, un *Full Adder*, una ALU, un contador de 4 bits y una RAM de 5 bits. Durante la validación de la compuerta XOR se encontró que se debía cambiar dos configuraciones para obtener un diseño sin errores. La primera fue la de ampliar el tamaño del *die*. Esto debido a que para implementar este diseño se requería un mayor numero de *pads* de entradas y salidas que con la NOT, por lo que ampliando el tamaño del *die* se logró que todos los *pads* pudieran ser acomodados. La segunda configuración fue la de forzar que los *pads* de alimentación siempre se encontrarán en los laterales del diseño. Esto con el objetivo de evitar la formación accidental de cortos durante la creación del *mesh* de poder.

Luego de realizar las validaciones con la compuerta XOR, se encontró que la metodología ya era lo suficientemente robusta para implementar una variedad de circuitos. Esto al lograr que el *Full Adder*, la ALU, el contador y la RAM pudieran ser validados sin necesidad de realizar modificaciones al flujo más que ampliar el tamaño del *die* para ajustar de acuerdo al número de *pads*. Con esto se pudo dar por validado el flujo propuesto al ser capaz de generar diseños sin errores de ejecución en la herramienta *IC Compiler II* y sin violaciones a las reglas de antena del *foundry* TSMC.

Finalmente se realizó la síntesis física sobre el diseño generado en conjunto con todo el grupo de trabajo del proyecto y el cual se pretende enviar a fabricar con TSMC. Este diseño fue nombrado “El Gran Jaguar” ya que al igual que el gran templo maya este busca poner en alto el nombre de Guatemala y abrir las puertas hacia un nuevo mundo. El circuito consiste en una máquina de estados finitos que transmite distintos mensajes en caracteres ASCII, estos mensajes contienen información relevante del proyecto. Esta síntesis fue exitosa al poder generar el diseño, validar que este aprobara la Verificación de Antena y finalmente lograr exportarlo a un archivo GDSII.

Abstract

In the following study a proposal is made for a workflow to carry out the physical synthesis stage of the VLSI design process, along with a detailed explanation of the different steps composing this flow. The workflow is designed to be used in the IC Compiler II tool, utilizing TSMC's 180 nm, 6 layer process libraries. The main goal being able to generate a physical design for any circuit without having tool execution or process errors and without design errors in the final layout. The study also presents how can the antenna rules be integrated and evaluated during the proposed flow in order to verify errors on the design. With the rules specified the flow could be validated to determine if utilizing the proposed methodology a circuit designer would be able to generate an error free pyshical representation of any circuit.

This study is part of an intergenerational project searching to automate the VLSI design flow in order to develop a tool the could be the key for opening the doors of the nanoelectronics design field on Guatemala. Being a part of said project the first objective was to continue the works made in previous years and utilize the advances made as a baseline for this study. With this in mind, the first steps taken were on trying to replicate the obtained results in these previous works in order to get up to date with the state of the project and as an introduction to the design tools. After this process was completed, a roadmap was developed to update the current workflow from the IC Compiler tool to it's more modern version IC Compiler II.

The update process began with extensive research of the IC Comipler II user guides which were provided by Synopsys, the tool developer. With all the aquiered knwoledge and after initial testing it was determined that the update process would be harder than originally thought. This is mainly because many of the IC Compiler commands were reworked or outright non-existen on IC Compiler II. During the research process it was also found that in the previous workflow there were several recommended procedures that were not being executed. After analyzing all the options it was determined that the best course of action would be creating a new workflow, but always keeping in mind the advances and discoveries made on previous works.

In the new worflow the main stages were still being executed, these being the Floorplan, Placement and Routing stages. Several key procedures like library preparation and power

planning were added between stages to increase the effectiveness of the proposed workflow. During the validation process many tests were made on this new workflow. The results were used to identify points of failure and fixing them until the workflow was optimal. The first test was a full physical synthesis for a NOT gate. Any errors that were found were fixed and the process was repeated until a synthesis could be made without any errors.

After a successful physical synthesis the next step was to verify that the generated design did not contain violations to the antenna rules. It was necessary to conduct more research of the user guides and previous works in order determine the best way of performing and integrating this verification in the workflow. Once this was decided, the NOT design was verified and no violations were found in the design. As this verification was already successfully carried out in previous years, most of the work went into achieving an easy integration to the new workflow in order to verify the generated designs without complications.

The first tests being successful allowed the continuation the validation process with increasingly more complex circuits. The circuits utilized in these new tests were a XOR gate, 4 bit Full Adder, 4 bit ALU, 4 bit Counter and 5 bit RAM. During the XOR validation process it was found that in order to get an error free design, two modifications to the workflow were needed. The first one consisted on adjusting the die area since the number of input-ouput pads to implement a XOR gate is greater than the number needed for the NOT. The area was adjusted to allow for all the pads to be positioned in the desing. The second change was forcing the position of the power pads on the right and left sides of the die. This was mainly done to ensure that no shorts could be created during the power mesh creation process.

After the workflow was validated on the XOR gate it was found that no more modifications were needed since the Full Adder, ALU, counter and RAM designs could all be verified without any issues. The only needed changes were the adjustments of the die area in order to accomodate for all the input-output pads in the design. Whith this results the validation process was determined succesful as the proposed workflow could generate a design without any IC Compiler II runtime tool errors and wihtout violations of TSMC's antena rules.

As a last step the proposed physical synthesis flow was utilized to generate a sillicon representation of a circuit designed by all members of the project and destined to fabricated by TSMC. The circuit was named *El Gran Jaguar* after the spanish name for one of the great mayan pyramids. This name was chosen as the chip aims to put in a high place the name of Guatemala and open the doors towards a new world. The main function of the circuit is to send different messages in ASCII characters. This messages contain relevant information about the project. The design synthesis and antenna verification procesess were once again performed and validated succesfully without any issues.

CAPÍTULO 1

Introducción

El campo de la nanoelectrónica surgió como la respuesta a la necesidad, cada vez mayor, de reducir el tamaño de los circuitos eléctricos sin comprometer su desempeño. Gracias a los grandes avances que se han hecho a través de los años este campo se ha vuelto indispensable en el mundo actual. Los circuitos integrados que se producen cumplen una amplia gama de funciones ya sea generales o específicas y, debido a esta flexibilidad, se utilizan en una gran variedad de campos. Esta utilización es tan extensa hoy en día que es difícil imaginarse un mundo donde la nanoelectrónica y los circuitos integrados no existieran.

A pesar de su relevancia a nivel global, estudios y trabajos en el campo de la nanoelectrónica han sido prácticamente inexistentes en Guatemala. En vista de esto la Universidad del Valle de Guatemala ha hecho grandes esfuerzos en los últimos años para abrir las puertas de este campo en nuestro país. Es en estos esfuerzos donde se encuentra este trabajo de graduación, cuyo propósito es el automatizar los distintos procesos que se deben llevar a cabo durante el diseño de un circuito integrado, específicamente los procesos de síntesis física y verificación de antena.

En los siguientes capítulos se presenta información valiosa sobre una propuesta que busca implementar un flujo de diseño para circuitos integrados y el ahínco realizado para definir una metodología de trabajo que permitiera su automatización en la herramienta *IC Compiler II*. Adicional a esto, se válida dicha metodología utilizando las librerías de 180 nm y 6 capas de metal del fabricante TSMC. Finalmente, se expone información importante para que este trabajo pueda ser replicado por futuras generaciones. Para esto se describen los procesos a realizar así como se presenta un *script* que permite la ejecución de los mismos.

CAPÍTULO 2

Antecedentes

El primer avance realizado por la Universidad del Valle de Guatemala en el campo de la nanoelectrónica se dio con la integración del curso *Introducción a Sistemas de Diseño VLSI* al mapa curricular de Ingeniería en Electrónica y el cual fue impartido por primera vez en 2013. En un inicio se utilizó la herramienta *Electric VLSI Design System*, la cual no tenía costo alguno, pero sus capacidades eran limitadas. Un año después, en 2014, la universidad pacta un acuerdo académico con la empresa Synopsys, una de las líderes en el mercado de software para diseño VLSI. En este acuerdo se da acceso a los estudiantes de la carrera a una gran variedad de herramientas que abrieron el paso para la realización de diseños mucho más ambiciosos. Una prueba de esto es el primer trabajo de graduación en el campo de VLSI realizado en 2014 por el Ingeniero Jonathan de los Santos y el cual se presenta en [1]. Desde su realización, este trabajo ha servido como guía para trabajos posteriores al sentar las bases para el uso correcto de las herramientas de Synopsys.

En la readecuación curricular del 2015 se sustituye el curso de *Introducción a Sistemas de Diseño VLSI* por los cursos de *Nanoelectrónica 1 y 2*, buscando expandir el alcance del curso previo y dando la oportunidad para que los estudiantes continúen con las líneas de investigación en el área de VLSI. Algunos de estos primeros trabajos se presentan en [2] y [3] y fueron de gran ayuda durante la realización del trabajo presentado en el presente.

CAPÍTULO 3

Justificación

Elaborar una herramienta que permita llevar a cabo todo un flujo de diseño para circuitos integrados de forma automática es un gran paso para el estudio y desarrollo de proyectos que utilicen tecnología nanométrica en Guatemala. Esto elimina la dependencia que se tiene actualmente de utilizar chips de propósito general al permitir el diseño de chips de propósito específico, con un desempeño mucho mayor debido a su optimización para realizar tareas determinadas. Queda esperar que la existencia de una herramienta de este tipo permita la apertura de nuevos campos laborales dedicados al diseño de circuitos integrados en nuestro país, lo que en un futuro puede resultar como un factor atractivo para la inversión extranjera.

Una etapa fundamental del flujo de diseño VLSI es la síntesis física ya que, es en esta donde se genera la representación física del circuito que se desea fabricar en un chip. Automatizar y optimizar esta etapa para lograr que esta representación cumpla con la verificación de antena es de suma importancia. Esto último ya que el cumplimiento con esta verificación es uno de los requisitos solicitados por los fabricantes para aceptar el trabajo de fabricación del chip. Esta verificación es diseñada por cada fabricante según sus procesos, tomando en cuenta cualquier posible error que pueda surgir durante la fabricación. Por lo mismo, el poder asegurarse que las representaciones físicas generadas por la herramienta cumplan con la verificación nos indica que estas pueden ser fabricadas sin problemas.

Este trabajo forma parte de un proyecto intergeneracional que ha servido como una gran oportunidad de aprendizaje para todos los estudiantes involucrados a lo largo de los años. Con el proyecto se pretende dejar a todos los estudiantes del Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica una herramienta que les sea de gran ayuda para sus propósitos estudiantiles, así como les permita continuar impulsando el desarrollo científico y tecnológico en Guatemala.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Optimizar la etapa de síntesis física del flujo de diseño VLSI tomando en cuenta todos los avances realizados en trabajos anteriores, para lograr que esta etapa pueda ser ejecutada con cualquier circuito, de forma automática y garantizando que los diseños generados cumplan con la verificación de antena. Todo esto manteniendo una comunicación activa y efectiva con todos los miembros del proyecto, buscando que exista un total entendimiento de los avances del proyecto y como forma de apoyo en caso de que se llegase a presentar algún problema durante la realización del mismo.

4.2. Objetivos específicos

- Actualizar los procesos de *floorplaning*, *placement* y enrutamiento, para que estos sean efectuados por la herramienta IC Compiler 2.
- Optimizar la síntesis física para que los *layouts* generados puedan aprobar la verificación de antena.
- Mantener una comunicación activa con todos los miembros del proyecto para solucionar cualquier error o problema que se presente de forma rápida y efectiva.
- Generar los archivos y documentación necesaria para que los procesos diseñados puedan integrarse sin problema con el resto de etapas del flujo de diseño VLSI durante la creación de la herramienta automatizada.

CAPÍTULO 5

Alcance

Este trabajo busca la definición y validación de un flujo de trabajo que permita llevar a cabo una síntesis física sin errores. Para esta validación se limita a comprobar que los diseños puedan ser generados sin errores de ejecución y que estos diseños cumplan con las reglas de antena establecidas por un *foundry*. Con esto último también cabe mencionar que el alcance de este trabajo se limita a la validación del flujo únicamente utilizando las librerías y *runsets* para procesos de 180 nm y 6 capas de metal del *foundry* TSMC.

Dentro del alcance de este trabajo también se encuentra la documentación apropiada del flujo de trabajo propuesto y de todos los procesos que lo componen. Esto buscando que generaciones futuras puedan contar con todo el apoyo posible para expandir sobre los logros alcanzados. Por último, con este trabajo se pretende la generación de *scripts* que puedan ser ejecutados en las herramientas de *Synopsys* y los cuales permitan la fácil automatización de la etapa de síntesis física en el flujo de diseño VLSI.

CAPÍTULO 6

Marco teórico

6.1. Transistores

En la actualidad existen dos tipos principales de dispositivos semiconductores de tres terminales, el transistor de unión bipolar y el transistor de efecto de campo metal-óxido-semiconductor. Estos son comúnmente conocidos por sus siglas en inglés como BJT y MOSFET, respectivamente. Siendo inventado en 1948, el BJT fue por tres décadas el dispositivo base en el diseño de circuitos integrados y discretos. El MOSFET fue inventado tiempo después en 1960, pero su uso no se popularizó hasta las décadas de 1970 y 1980. Esto se debió a que conforme la complejidad de los circuitos diseñados iba aumentando, era necesario reducir el tamaño de los componentes. De esta manera sería posible fabricar dispositivos electrónicos cada vez más rápidos y que pudieran cumplir más funciones que dispositivos anteriores, además de tener un tamaño físico igual o incluso menor a estos.

Un MOSFET, comparado a un BJT, puede tener un tamaño mucho más reducido, con un proceso de manufactura más simple y un menor consumo de potencia al operar. Estas características hicieron al MOSFET el dispositivo ideal para atender a las nuevas necesidades de los diseñadores de circuitos. A raíz de esto se empezaron a diseñar circuitos aún más complejos que podían implementar funciones digitales y analógicas utilizando únicamente MOSFETs. Aunque el BJT también posee ciertas ventajas sobre el MOSFET, con el paso de los años su uso ha ido disminuyendo considerablemente. Mientras tanto el MOSFET se ha convertido en el dispositivo electrónico más utilizado hoy en día, especialmente en el diseño de circuitos integrados o ICs por sus siglas en inglés [4].

6.2. Tecnología CMOS

Uno de los usos principales que se le da a los MOSFETs es el de funcionar como *switches* controlados por voltaje. Un MOSFET posee tres terminales de las cuales una es la terminal de control, conocida como *gate*. Cuando se suministra suficiente voltaje a esta terminal se

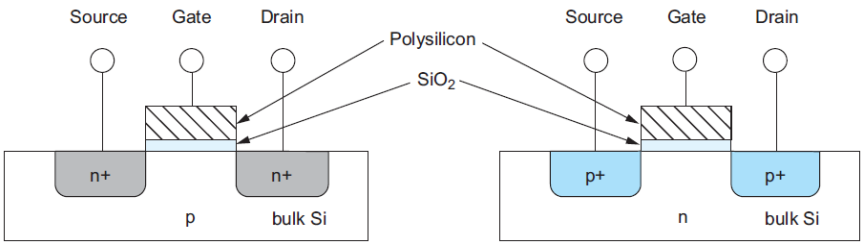


Figura 1: *nMOS* (izquierda) y *pMOS* (derecha) [5]

abre o cierra un canal de conducción entre las otras dos terminales, el cual permite transmitir señales de una terminal a otra. Este par de terminales reciben el nombre de *drain* y *source*. El voltaje al que este fenómeno sucede se conoce como el voltaje de umbral y la apertura o cierre del canal al llegar a este voltaje depende directamente del tipo de MOSFET utilizado, existiendo dos opciones: el transistor tipo n (*nMOS*) y el tipo p (*pMOS*). En un nMOS cuando el voltaje en el *gate* es de 0 V el canal se encuentra cerrado, mientras que al tener el voltaje de umbral este se abre. El funcionamiento del pMOS es el inverso, el canal se encuentra abierto si no hay voltaje en el *gate* y este se cierra al tener el voltaje de umbral [5].

La Figura 1 muestra la estructura física de ambos tipos de MOSFETs y en esta se puede observar tres componentes básicos: las terminales, las difusiones y la capa de óxido. Primero se tiene las terminales, donde el *gate* se encuentra entre el *drain* y *source*. Luego se tiene el cuerpo de silicio (*bulk Si*) y las difusiones que componen al *drain* y *source*. El tipo de difusión que se utilizará será completamente dependiente del tipo de MOSFET que se desea construir, si este es un *nMOS* las difusiones deben ser del tipo n, mientras que si es un *pMOS* del tipo p. Por último se tiene la capa de óxido (SiO_2) que separa al *gate* del cuerpo del transistor y es la que permite la formación del canal de conducción entre *drain* y *source*.

A pesar que ambos tipos de MOSFET son capaces de transmitir las mismas señales por su canal de conducción, existen ciertos casos donde es más conveniente utilizar un nMOS debido a una degradación de la señal si se estuviese utilizando un pMOS. De igual manera, hay ocasiones donde el nMOS degrada la señal por lo que conviene utilizar un pMOS. Esto se debe principalmente a las propiedades físicas de ambos tipos de transistores y resulta ser un problema cuando se trabaja con sistemas digitales, donde es necesario señales definidas que representen los valores lógicos uno o cero. Si existiese un degradamiento en la señal puede que el sistema no fuera capaz de determinar este valor lógico, lo que resultaría en el sistema funcionando de forma incorrecta. La solución de este problema es la implementación de la tecnología CMOS. Esta consiste en utilizar parejas de nMOS y pMOS para transmitir señales, lo que en principio elimina el degradamiento de las mismas ya que, las señales que son degradadas por el nMOS son transmitidas por el pMOS y viceversa. Debido a que eliminan el problema del degradamiento de señales, los circuitos lógicos CMOS son una de las bases fundamentales en el diseño de ICs digitales [4].

Cuadro 1: Clasificación de ICs según su complejidad [6]

Clasificación	GEs
Small-scale integration (SSI)	1-10
Medium-scale integration (MSI)	10-100
Large-scale integration (LSI)	100-10,000
Very-large-scale integration (VLSI)	10,000-1,000,000
Ultra-large-scale integration (ULSI)	1,000,000 ...

6.3. VLSI

Los ICs se clasifican según su complejidad y esto se relaciona directamente con la cantidad de transistores que conforman cada circuito. En el Cuadro 1 se presentan estas clasificaciones y es importante notar que la unidad utilizada no es el número de transistores sino que el GE, o *gate equivalent* por sus siglas en inglés. Un GE equivale a cuatro transistores, lo que es igual al número de transistores que conforman una compuerta NAND implementada con CMOS [6]. Conforme la complejidad de los circuitos fue aumentando se volvió evidente que cada cierto tiempo iba a ser necesaria la creación de una nueva clasificación. Para evitar que se tuviese clasificaciones con nombres cada vez más complejos se decidió utilizar VLSI: para todo circuito integrado con 10,000 o más GEs [5], dejando obsoletos a términos como ULSI. Es por esto que en la actualidad al hablar de VLSI: se refiere al proceso de diseño y fabricación de circuitos integrados que están conformados por miles o millones de transistores, todos dentro de un solo chip.

6.4. Flujo de Diseño

Un Flujo de Diseño es una serie de pasos que le permiten a los diseñadores de ICs avanzar desde las especificaciones de funcionalidad de un chip hasta su fabricación e implementación sin errores [5]. Un flujo de diseño bien definido es esencial para el diseño VLSI: debido a la complejidad de los procesos que se deben llevar a cabo para la fabricación exitosa de un IC. Tener estos procesos divididos en etapas permite tener un mejor orden e idea de todo lo que se necesita para fabricar el chip. En la industria esto resulta muy útil ya que permite subcontratar servicios de otras empresas que tengan mayor experiencia en ciertas etapas del flujo, por lo que el diseño de un mismo chip puede ser llevado a cabo por varias empresas trabajando en conjunto. El flujo de diseño presentado a continuación se divide en dos etapas principales: *Front-End* y *Back-End*

6.4.1. Front-End

El diseño *Front-End* inicia con la elaboración de una descripción *Register Transfer Level* (RTL) [5]. Esto se hace tomando los requerimientos del sistema y describiendo su comportamiento sin necesidad de especificar explícitamente que elementos lo conforman, simplemente basta con definir sus entradas y salidas. Para esta primera parte se utiliza lenguajes descriptos de hardware (HDL), como los son Verilog y VHDL. Para comprobar que la transcripción

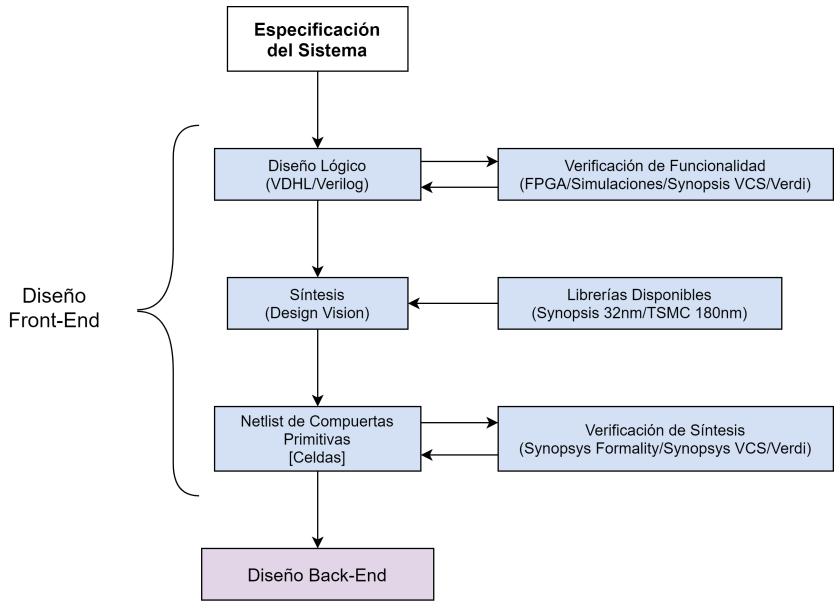


Figura 2: Diseño Front-End [7]

del sistema haya sido exitosa, se realizan simulaciones con el RTL donde se busca verificar que el sistema descrito continué cumpliendo con las especificaciones iniciales.

Luego de una verificación exitosa se procede a sintetizar el sistema, este es el proceso que convierte el RTL a compuertas y registros genéricos o del fabricante en caso de tener la librería correspondiente disponible. Los componentes dentro de estas librerías se conocen como celdas. Con este proceso se logra optimizar la lógica del circuito, lo que significa una mejora en velocidad y área. Al finalizar la síntesis se genera un archivo conocido como el *netlist* [3]. Este proceso es clave ya que disminuye el nivel de abstracción al generar una descripción más detallada del sistema. Esto debido a que el *netlist* no solo describe cómo se comporta el sistema si no que además cómo está estructurado.

Para finalizar el diseño *Front-End* se debe verificar que el sistema descrito en el *netlist* sea equivalente al descrito por el RTL. Aunque en principio esto puede parecer innecesario, puede que ciertas ambigüedades en el RTL generen problemas durante la síntesis, especialmente si el código HDL fue pobremente escrito. Al verificar la equivalencia entre los sistemas descritos por ambos archivos se puede dar por finalizada esta etapa de diseño. El flujo durante esta etapa se puede apreciar de mejor manera en la Figura 2, donde además se especifican algunas de las herramientas que se pueden utilizar para los distintos procesos.

6.4.2. Back-End

La Figura 3 permite visualizar el flujo de diseño para la etapa de *Back-End*. A primera vista se puede notar que el flujo en esta etapa es más complejo que en la etapa anterior y esto se debe a es en esta donde se realiza la síntesis física. Esto significa que se toma el *netlist* y se genera el *layout* del sistema para que este pueda ser fabricado [7]. Para los fines de este trabajo, es necesario enfocarse en los procesos de *floorplan*, *placement*, enrutamiento, y verificaciónes de DRC/ERC/antena, sin embargo, a continuación se presenta una descripción

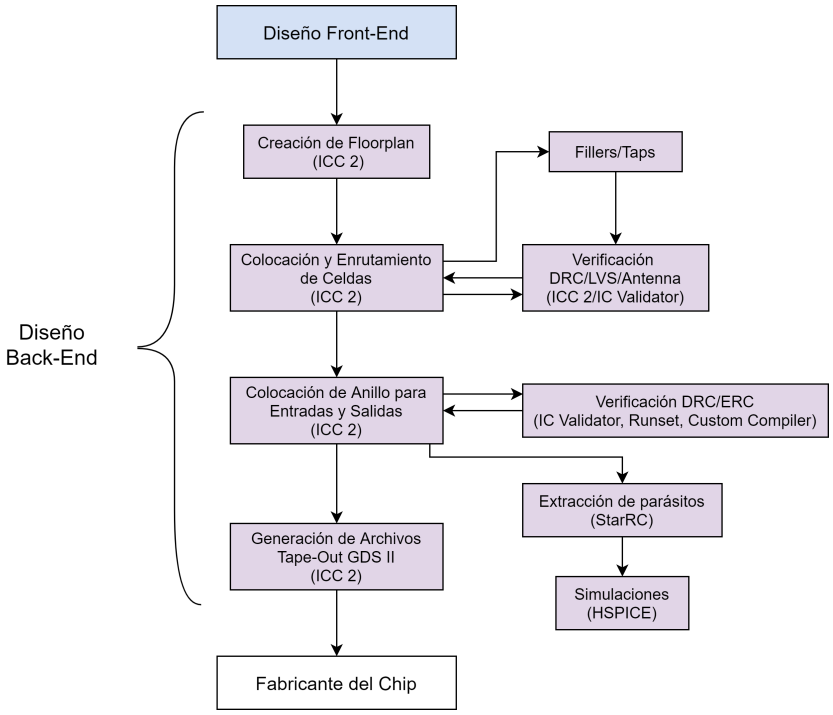
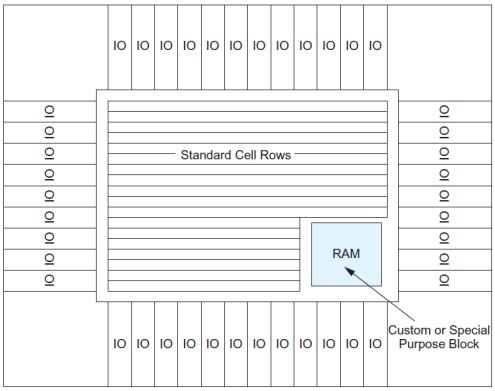


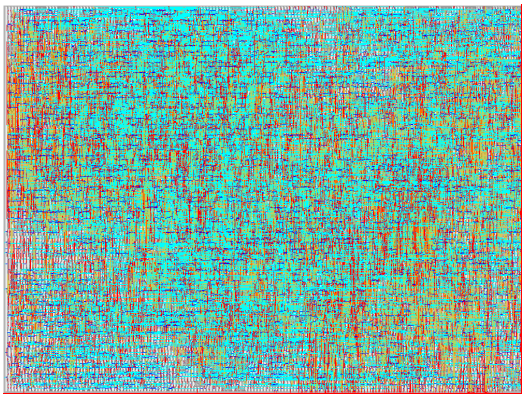
Figura 3: Diseño Back-End [7]

general de los procesos realizados en esta etapa para finalizar de ilustrar el flujo de diseño VLSI:.

- *Floorplan*: Se determina la ubicación física de cada una de las celdas dentro del área del chip y luego estas son colocadas durante el placement.
- Enrutamiento: Se interconecta las distintas celdas entre sí según las conexiones descritas en el *netlist*.
- Anillo de Entradas y Salidas: Se coloca los pines para las entradas y salidas del sistema.
- DRC: Se verifica que el *layout* cumpla con las restricciones del fabricante.
- LVS: Se verifica que haya equivalencia entre el sistema presentado en el *layout* y el descrito por el *netlist*.
- Antena: Proceso para prevenir daños al sistema por el efecto antena.
- ERC: Se verifica que no haya fallas estructurales en el diseño.
- Extracción de parásitos: Se genera un segundo *netlist* que toma en cuenta todos los efectos parásitos en el *layout* para realizar simulaciones más realistas del sistema.
- Generación de GDS II: Se generan los archivos en el formato utilizado por los fabricantes.



(a) *Floorplaning* [5]



(b) Enrutamiento [8]

Figura 4: Síntesis física

6.4.2.1. Floorplan y Placement

Este proceso consiste en el diseño del *layout* buscando que las celdas queden arregladas de forma eficiente para cumplir con la restricción del área máxima que puede tener el chip. Aunque esto se puede realizar de forma manual, este proceso usualmente es llevado a cabo automáticamente por software. Para esto se restringe la altura de las celdas de forma que esta sea fija y dejando que el ancho varíe. Esto permite que las celdas sean arregladas de forma ordenada en filas por el software utilizado, el cual utiliza algoritmos que minimizan el largo de los *tracks* requeridos para las interconexiones. Una parte del proceso que se recomienda hacer previo al arreglo automático es la separación de las celdas en áreas según su necesidad de comunicarse entre sí [5]. Esto permite minimizar aún más el largo total de los *tracks* de metal.

La Figura 4a ejemplifica este proceso. Puede observarse que se tiene creadas las filas donde se estarán organizando las celdas según lo explicado anteriormente. En este caso se tiene un bloque de memoria RAM, el cual tiene un tamaño muy superior al de una celda regular y un funcionamiento más complejo, por lo que es catalogado como un bloque de propósito especial. Debido a sus características, estos bloques no pueden ser organizados de igual manera que las celdas estándar y se les debe asignar una posición previo a la creación de las filas donde se colocará el resto de celdas. Una vez se tiene todas las celdas organizadas estas permanecen fijas en su posición asignada dentro del *layout*.

6.4.2.2. Enrutamiento

Con las celdas ya posicionadas, se puede proceder a interconectarlas. Esto se hace tomando las relaciones descritas en el *netlist* y realizando las conexiones pertinentes. Nuevamente esto se realiza de forma automática por un software especializado, el cual utiliza una serie de criterios como el costo de cada conexión, la geometría del *layout* y los requerimientos del sistema. El proceso de enrutamiento es el más complejo del flujo, computacionalmente hablando, en especial cuando el sistema es complejo y por lo tanto tendrá una mayor cantidad de celdas [3]. La Figura 4b permite ilustrar esto al mostrar el *layout* de un microprocesador

OpenMSP430. En la figura cada línea representa un *track* que conecta un nodo a otro, las líneas de distintos colores permiten diferenciar entre *tracks* ubicados en distintas capas de metal. De la figura se puede intuir el porque este proceso es realizado de forma automática ya que, realizar todas las conexiones de forma manual requeriría una gran inversión de tiempo y recursos humanos. Por otro lado, también se puede ver porque el proceso es demandante computacionalmente hablando. La computadora o el sistema que realiza el enrutamiento debe de tomar en cuenta todas las conexiones que debe realizar, cómo es la relación entre estas y finalmente cuál es la ruta óptima o la de menor costo según su criterio. Todo esto para cada una de las rutas establecidas.

6.4.2.3. DRC

Como todo proceso de fabricación, la manufactura de chips no es perfecta y es necesario tomar medidas para contrarrestar posibles errores de fabricación que son inevitables. Es por esto que, a pesar que durante el diseño del *layout* se tiene completa libertad sobre este, no todos los diseños pueden ser manufacturados por los fabricantes de chips, comúnmente conocidos como *foundries*. Para que la fabricación sea eficiente los *foundries* definen un conjunto de reglas conocidas como *layout rules*. Estas son definidas por cada *foundry* de acuerdo a sus propios procesos de fabricación y tomando en cuenta las imperfecciones en estos procesos. En procesos modernos este conjunto está conformado por cientos de reglas ya que, a medida que se ha aumentado la complejidad de los procesos también se ha vuelto necesario hacer lo mismo con las reglas de diseño [6].

Para poder fabricar un chip con un *foundry* se debe demostrar que el diseño cumple con sus reglas y esto se hace realizando una verificación DRC (*Design Rule Check*). Esta verificación realiza de forma automática todas las pruebas para determinar si el diseño cumple o no con las reglas de diseño establecidas. Si la verificación es exitosa se cumple con uno de los requisitos impuestos por el *foundry* para poder fabricar el chip. En la Figura 5 se muestran algunos ejemplos de las restricciones impuestas por los *foundries* y los posibles errores de fabricación que pueden surgir si estas no son tomadas en cuenta.

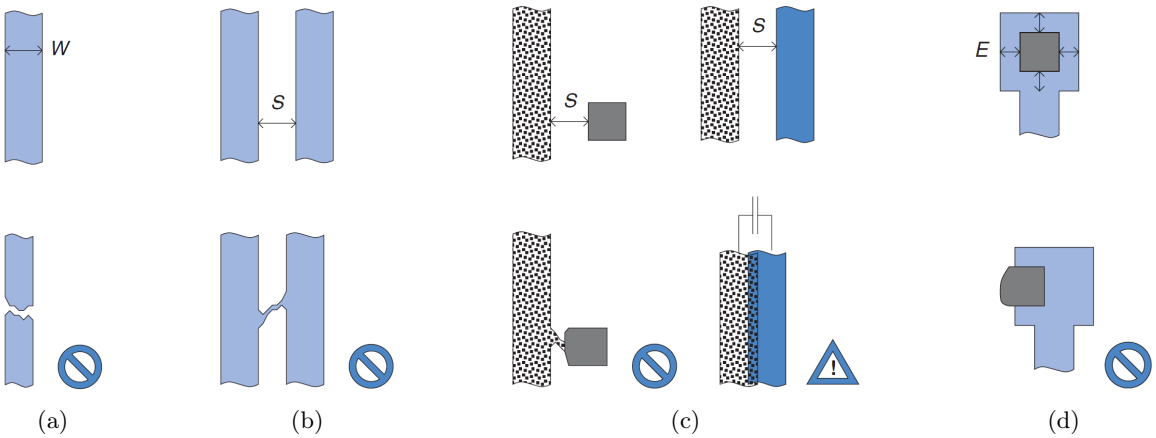


Figura 5: Ejemplos de *layout rules* y las consecuencias de no seguirlas [6]

La Figura 5a indica que los *tracks*, o estructuras, deben tener un ancho mínimo ya que si estas son muy delgadas se corre el riesgo de una fragmentación o ruptura interna. Por otro lado, la Figura 5b indica que debe haber una separación mínima entre las estructuras de una misma capa, debido a que si estas se encuentran muy juntas se corre el riesgo de que se creen conexiones no deseadas. De igual forma, la Figura 5c muestra que hay restricciones similares para estructuras que se encuentran en distintas capas. Si la separación entre las estructuras no es la requerida, nuevamente corremos el riesgo de una posible creación de cortos entre estructuras, y en este caso, la formación de capacitores parásitos que no fueron tomados en cuenta durante el diseño. En la Figura 5d se encuentra que también existen restricciones para las áreas donde se posicionarán los contactos. Estos últimos son los utilizados para conectar las estructuras a las terminales del transistor. Como se indica, las áreas deben ser lo suficientemente grandes para compensar por posibles desfases en la posición final de los contactos. De lo contrario se corre el riesgo de tener conexiones pobres hacia los transistores, lo que afecta de sobremanera el desempeño del sistema.

6.4.2.4. ERC

Luego de años de diseñar y fabricar chips, la industria ha podido diferenciar e identificar entre las estructuras dentro del *layout* que funcionaran de forma segura y las que tienen una posibilidad de presentar fallas. Utilizando su experiencia los ingenieros han creado a través de los años una serie de reglas que determinan cómo se debe ver un buen circuito y qué estructuras se debe evitar. Algunas de estas reglas dependen de una serie de factores como la tecnología utilizada, el circuito que se está intentando implementar en el *layout*, a qué se estará conectando el chip, entre otros [6]. Sin embargo, existen errores que deben ser evitados en todo diseño, como los que se presentan a continuación.

- Nodos de señales distintas conectados directamente.
- Entrada de una celda sin conectar.
- Salida de una celda conectada directamente a alimentación o tierra.
- Terminal de un MOSFET sin conectar.
- Terminales de un MOSFET conectadas directamente.
- Nodos de alimentación y tierra conectados directamente (cortocircuito).
- Falta de pines para alimentación y tierra.

Para comprobar que el diseño no contenga este tipo de errores se realiza una verificación conocida como ERC (*Electrical Rule Check*), la cual se puede llevar a cabo de forma automática por medio de software. La aprobación de esta verificación también forma parte de los requisitos impuestos por el *foundry* para aceptar fabricar el chip.

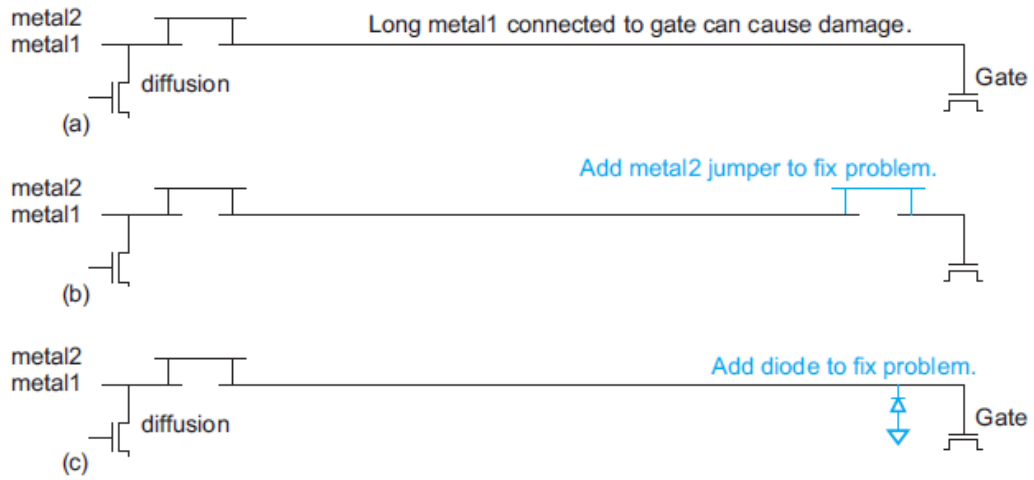


Figura 6: Violaciones a las reglas de antena y formas de solucionarlas [5]

6.4.2.5. Antena

El efecto antena ocurre debido a un proceso durante la fabricación conocido como *plasma etching*. Este proceso es utilizado para la construcción de los *tracks* que conectan a las celdas, lo que se conoce como metalización. Su principio consiste en utilizar plasma para cargar los iones de metal en estado gaseoso que luego son atraídos hacia el silicio por regiones cargadas en la oblea. Estas regiones se cargan selectivamente para que sobre ellas se formen las estructuras metálicas que interconectan el chip. Un problema que surge durante este proceso es que estas estructuras acumulan voltaje debido a la sobreexposición a portadores de carga. Los *tracks* conectados al *gate* del MOSFET pueden llegar a acumular un voltaje suficiente para dañar la capa de óxido que separa la terminal del resto de la estructura del transistor. Este fenómeno se conoce como el efecto antena y como consecuencia del mismo puede que se tenga un incremento en la corriente de fuga en el *gate*, un cambio en el voltaje de umbral, una reducción en la esperanza de vida del transistor o un daño total al transistor dejándolo inutilizable. Todos estos son efectos negativos y entre más largos sean los *tracks* utilizados mayor será la carga acumulada en estos y por lo tanto, mayor será la posibilidad de provocar un daño al transistor.

Conociendo este fenómeno, los *foundries* establecen una serie de reglas conocidas como las reglas de antena. En estas se especifica el área máxima de metal que se puede conectar al *gate* sin necesidad de modificar el *layout* para añadir elementos de descarga que permitan evitar daños en el chip. Una forma de mitigar el efecto es aumentando el tamaño del *gate*, debido a que si este tiene un mayor tamaño puede soportar una mayor acumulación de cargas sin dañarse, sin embargo, esto muchas veces no es una opción ya que significaría un cambio en la tecnología que se está utilizando. Es por esto que en las reglas usualmente se establece la razón máxima entre las áreas del metal y el *gate*, respetando esta razón se asegura que la carga acumulada en el metal no sea dañina para el transistor. Esta razón puede variar desde 100:1 hasta 5000:1 y su valor va a depender directamente del grosor de la capa de óxido en el *gate* [5].

En la Figura 6, se puede ver las dos formas comúnmente utilizadas para solucionar

violaciones a las reglas de antena en el diseño. La Figura 6a muestra un problema, se tiene un *track* muy largo en la capa de metal 1 conectado al *gate* del transistor. Debido al tamaño de este *track* se corre el riesgo de causar daños a este transistor durante la metalización. La Figura 6b muestra una forma de solucionar esto y es cortando este *track* mediante un salto a la capa de metal 2 y luego de vuelta a la capa de metal 1. Esto reduce efectivamente la cantidad de carga acumulada durante la metalización en metal 1 y protege al transistor. Otra forma de solucionar este problema es añadiendo un diodo de antena, tal como se muestra en la Figura 6c. En este caso el diodo proporciona un camino de descarga para las cargas acumuladas sobre el *track*. Debido a encontrarse en reversa, este diodo no afecta la operación del circuito, únicamente sirve como protección durante el proceso de fabricación. Es necesario aprobar la verificación de antena ya que, al igual que el ERC y DRC, es un requisito solicitado por el *foundry* para fabricar el chip.

6.5. Synopsys

A través de los años varias empresas se han dedicado al desarrollo de herramientas que permitan llevar a cabo los distintos procesos que conforman el diseño VLSI:. Actualmente la empresa Synopsys se ha posicionado como la líder en soluciones para el diseño y verificación de chips de cualquier complejidad [9]. Para esto ofrecen una una amplia colección de librerías para distintas aplicaciones, así como herramientas que facilitan el trabajo de los diseñadores de chips. Las herramientas de especial importancia para este trabajo son *IC Compiler 2* y *IC Validator*.

6.5.1. IC Compiler 2

Esta herramienta es la segunda versión de la herramienta *IC Compiler* y permite realizar todos los procesos asociados a la síntesis física. En esta se lleva a cabo el *floorplan*, *placement* y el enrutamiento, así como permite realizar optimizaciones de reloj, rutas y pines [3].

6.5.2. IC Validator

Con esta herramienta se puede realizar las distintas verificaciones necesarias para validar el *layout*. Para esto utiliza una serie de archivos conocidos como *runsets* y sobre estos se realizan las verificaciones de DRC, ERC y *Antenna* [3].

6.6. Resultados de trabajos previos

Como se mencionó anteriormente, este trabajo forma parte de un proyecto intergeneracional en el cual, a través de los años, se ha logrado avances significativos. Estos avances han permitido concentrar el enfoque de este trabajo sobre etapas puntuales del flujo de diseño, lo que es de especial importancia para que el proceso de optimización del flujo sea más eficiente.

6.6.1. Síntesis física

En el trabajo realizado por Abadía (2020) se presenta una recomendación para la metodología a seguir en lo que respecta al proceso de síntesis física y la cual pudo ser implementada y validada por el autor [10]. Esta se presenta en los siguientes diagramas.

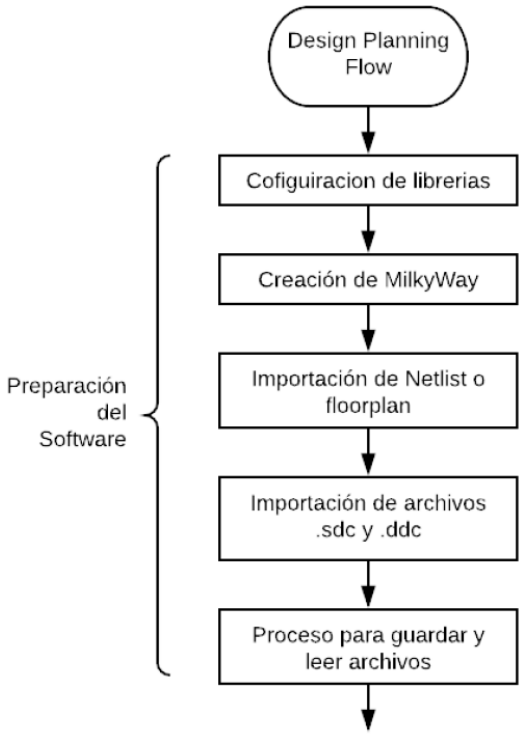


Figura 7: Primera etapa de la síntesis física. Preparación del software. [10]

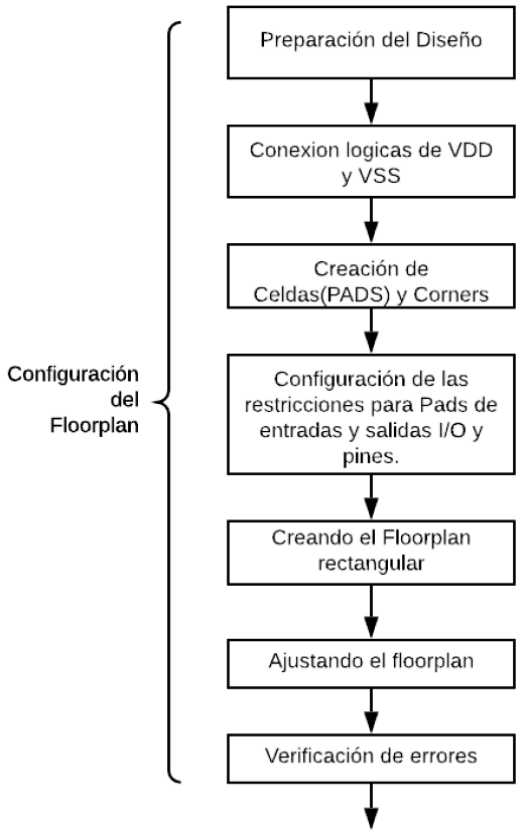


Figura 8: Segunda etapa de la síntesis física. Configuración del *floorplan*. [10]

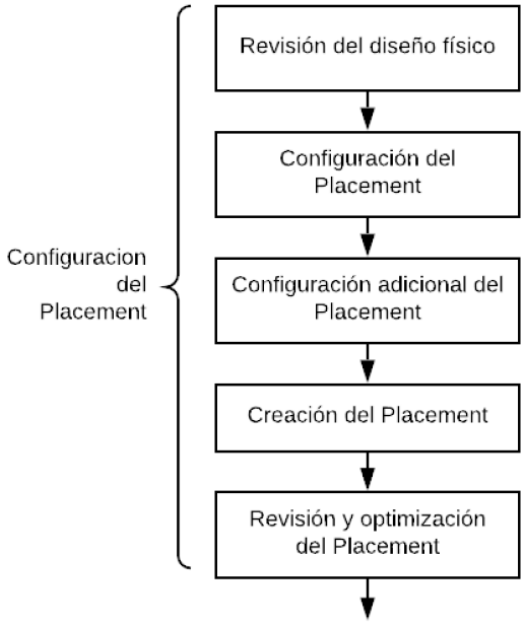


Figura 9: Tercera etapa de la síntesis física. Configuración del *placement*. [10]

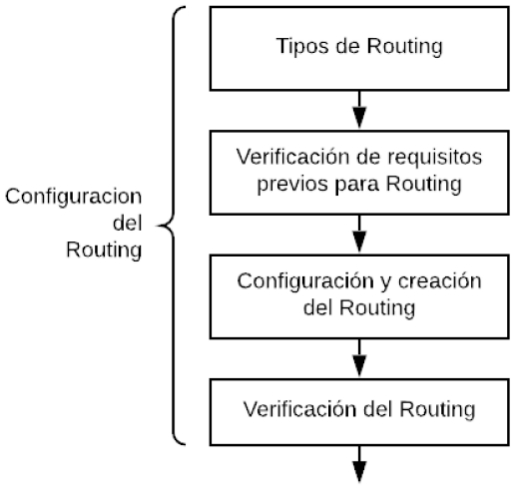


Figura 10: Cuarta etapa de la síntesis física. Configuración del *routing*. [10]

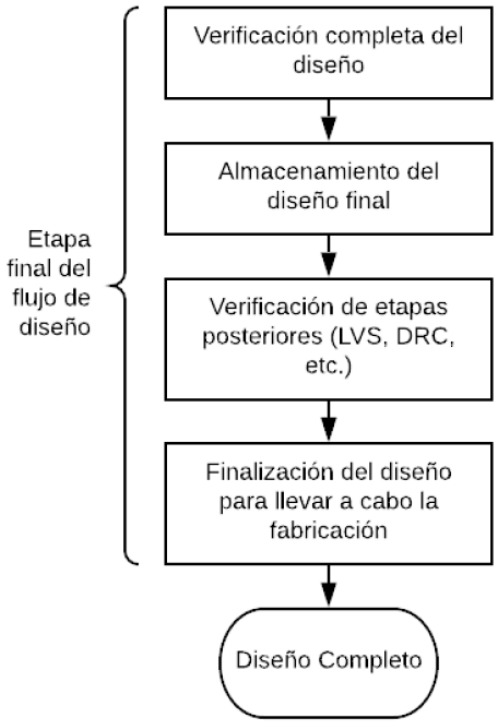


Figura 11: Quinta etapa de la síntesis física. Finalización del proceso. [10]

La validación de esta metodología fue llevada a cabo en la herramienta *IC Compiler*. En el Cuadro 2 se presenta los comandos de consola de esta herramienta que el autor determinó como útiles para llevar a cabo el proceso de síntesis física.

A continuación se presentan los resultados alcanzados por Abadía (2020). Estos fueron obtenidos utilizando la herramienta *IC Compiler* y el flujo de diseño para síntesis física propuesto en su trabajo. Este flujo siendo validado con tres circuitos distintos: compuerta NOT, *Full Adder* y *Ripple Carry Adder* (RCA).

Cuadro 2: Comandos para realizar una síntesis física en la herramienta *IC Compiler* [10]

Comando	Descripción
check_physical_design	Genera un reporte donde se realiza una evaluación al diseño para verificar si este está listo o no para la siguiente etapa. Se debe especificar la etapa en la que se encuentra el diseño.
check_routeability	Verifica que el diseño actual cumpla con los requisitos establecidos para poder realizar las conexiones entre las celdas.
create_cell	Permite crear una celda, se debe indicar la librería donde se encuentra descrito el módulo y se puede especificar la posición deseada dentro del diseño.
create_floorplan	Crea el área del diseño donde se estará colando las celdas y los pines de entrada y salida. Se puede añadir opciones para configurar las filas, colocar las celdas o modificar la forma y tamaño del core.
create_fp_placement	Permite posicionar las celdas dentro del diseño de acuerdo con una serie de opciones que se configuran junto al comando. Entre estas opciones se encuentra el esfuerzo que realizará la herramienta, si se desea optimizar para mejorar la respuesta, entre otras.
create_route_guide	Permite crear rutas dentro del diseño para asegurar que las redes debidas se encuentren completamente conectadas.
derive_pg_connection	Establece las conexiones lógicas de alimentación y tierra dentro del diseño.
focal_opt	Ejecuta un diagnóstico del diseño final y realiza correcciones para solucionar cualquier error de DRC dentro de este.
place_opt	Permite optimizar el proceso de Placement para obtener mejores resultados. Se puede configurar el esfuerzo que se realizará durante la optimización.
preroute_instances	Realiza las conexiones entre el anillo de entradas y salidas con los nets correspondientes dentro del core.
preroute_standard_cells	Realiza las interconexiones especificadas entre las distintas celdas del diseño.
read_pin_pad_physical_constraints	Permite leer las restricciones de la Milkyway desde un archivo generado.
remove_pin_pad_physical_constraints	Remueve todas las restricciones configuradas a los pines.

Comando	Descripción
remove_preferred_routing_direction	Elimina las configuraciones realizadas para indicar la dirección de ruteo preferida para cada capa de metal.
report_cell_physical -connections	Reporta las conexiones físicas que se han realizado en el diseño.
report_error_coordinates	Genera un reporte donde se indica las coordenadas de las características del diseño actual que representan un error de diseño.
report_fp_placement	Genera un reporte conocido como Quality of Results donde se describen varios parámetros asociados al proceso del Placement así como los resultados al ejecutar el proceso.
report_pin_pad_physical_constraints	Genera un reporte indicando la posición de los pines dentro del diseño y las restricciones individuales de cada pin.
report_preferred_routing_direction	Genera un reporte donde se indica la dirección preferida de ruteo para cada capa de metal.
route_opt	Realiza las interconexiones de celda y las conexiones del core con el anillo. De manera simultanea, optimiza estas conexiones para obtener el mejor desempeño posible.
set_app_var	Permite crear variables internas dentro de la sesión actual.
set_fp_placement_strategy	Permite añadir configuraciones adicionales para el comando create_fp_placement. Se tiene un total de 31 parámetros extras que se puede configurar.
set_ignored_layers	Permite indicar que capas de metal se desea utilizar y cuáles deben ser ignoradas.
set_net_routing_layer_constraints	Permite establecer capas de metal para nets específicas.
set_pad_physical_constraints	Reserva un espacio dentro del diseño para todas las celdas creadas.
set_pin_physical_ constraints	Permite establecer restricciones para los pines de entrada y salida.
set_preferred_routing_direction	Permite indicar la dirección preferida de ruteo para cada una de las capas de metal.
set_route_mode_options	Permite seleccionar el tipo de ruteo que se estará utilizando, este puede ser clásico o ZRoute.
verify_route	Genera un reporte donde se indica los resultados del proceso de enrutamiento para determinar si este fue llevado a cabo con éxito.
write_pin_pad_physical_constraints	Genera un archivo que contiene las restricciones en la Milkyway.

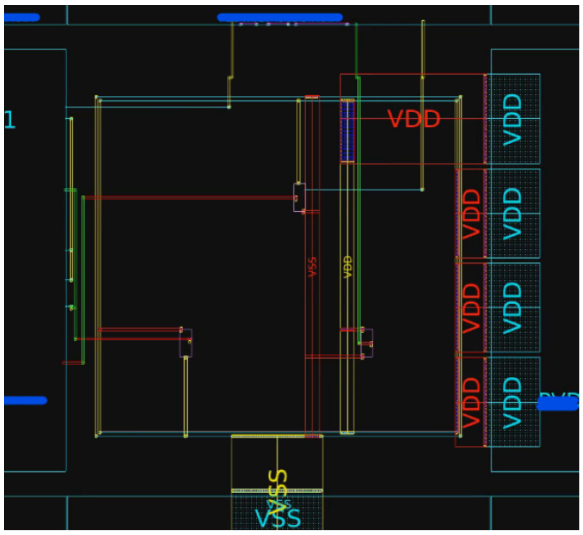


Figura 12: Resultado obtenido en [10] para la síntesis física de una compuerta NOT.

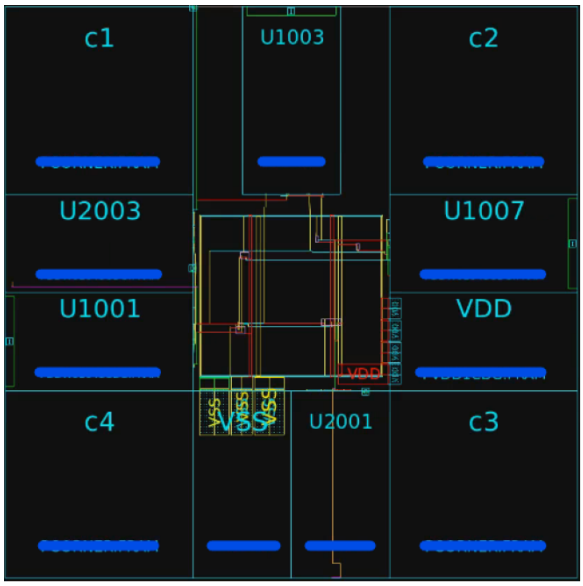


Figura 13: Resultado obtenido en [10] para la síntesis física de un *Full Adder*.

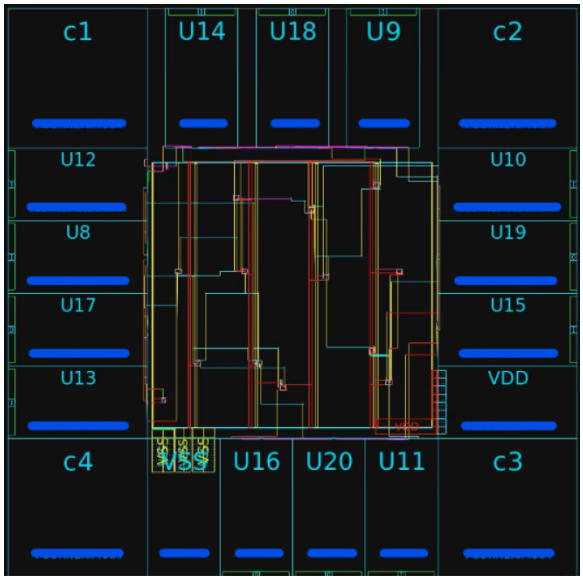


Figura 14: Resultado obtenido en [10] para la síntesis física de un *Ripple Carry Adder*.

6.6.2. Verificación de antena

Flores (2020) trabajó sobre los resultados obtenidos en [10] (presentados en las figuras 12, 13 y 14) y llevó a cabo el proceso de verificación de reglas de antena sobre los tres diseños [11]. En su trabajo, Flores presenta el flujo que utilizó para llevar a cabo esta verificación, presentado en la Figura 15. Adicionalmente, presenta una serie de comandos útiles para poder realizar la verificación en la herramienta *IC Compiler*, estos se presentan en el Cuadro 3.

Figura 15: Flujo propuesto en [11] para llevar a cabo la verificación de antena.

Cuadro 3: Comandos para realizar una verificación de antena en la herramienta *IC Compiler* [11]

Comando	Descripción
define-antenna-layer-rule	Permite definir una regla avanzada para una capa de metal específica del diseño.
define-antenna-rule	Permite definir una regla avanzada para todas las capas de metal.
remove-antenna-rules	Elimina todas las reglas de antena definidas por el usuario con anterioridad.
source	Lee un archivo y lo evalúa como un *script* tipo tcl.
uplevel	Permite acceder a una variable que no se encuentra definida dentro del proceso actual.
verify-zrt-route	Verifica e informa sobre violaciones de reglas de DRC, aperturas de red, violaciones de reglas de antena y violaciones de reglas de área de voltaje.

El archivo que contiene las reglas de antena fue provisto por TSMC, ya que este *foundry* es con el que se tiene el acuerdo para poder fabricar el chip. Este archivo está diseñado para ser ejecutado sobre diseños con tecnología de 180 nm y que utilicen un total de 6 capas de metal. De acuerdo a los resultados presentados en [11], se logró que para los tres diseños la verificación fuera exitosa, es decir que ninguno de estos presentó errores de antena. Esta verificación fue realizada de tres maneras distintas: utilizando comandos de *IC Compiler*, ejecutando el *runset* de TSMC desde *IC Compiler* y utilizando la interfaz *VUE tool* de *IC Validator*.

Para cada uno de los métodos, y para cada diseño, el resultado de la verificación de antena fue exitoso. La Figura 16 muestra el mensaje que Flores obtuvo en la consola de *IC Compiler* luego de ejecutar el *runset* de antena sobre el *Ripple Carry Adder*. Se puede ver que el mensaje indica la cantidad de reglas que fueron evaluadas así como la cantidad encontrada de violaciones a las reglas. Cómo se puede apreciar, se evaluó un total de 44 reglas y se obtuvo un total de 0 violaciones. Por brevedad no se mostrará los resultados obtenidos con los otros dos diseños ya que sería redundante debido a que el formato en el que se presentan es el mismo y, como se mencionó, estos resultados de fueron exitosos.

```
----------------------------------------------------------------------------
Results Summary
----------------------------------------------------------------------------

Rule and DRC Error Summary

44 total rules were run.
0 rules NOT EXECUTED.
0 rules have violations.
There are 0 total violations.
Refer to RCA_IO_CORRUPT.LAYOUT_ERRORS
```

Figura 16: Resultado obtenido en [11] para la verificación de antena sobre un circuito RCA utilizando el *runset* de TSMC.

CAPÍTULO 7

Síntesis física en IC Compiler 2

Igual que todo proceso del flujo de diseño VLSI, la síntesis física cuenta con su propio flujo de trabajo que se recomienda seguir para que este proceso pueda ejecutarse en su totalidad y sin errores. Como se mencionó en secciones anteriores, los avances alcanzados en trabajos previos fueron significativos, sin embargo, no se logró generar diseños que estuvieran libres de errores de diseño. Es por esto que entre los propósitos de este trabajo se encontraba el retomar los esfuerzos realizados anteriormente y optimizar la metodología utilizada para llevar a cabo un flujo de síntesis física capaz de generar un diseño físico sin violaciones a las reglas de diseño del *foundry*. Luego de replicar los avances previos y de explorar las herramientas a disposición se decidió migrar este proceso de la herramienta actual, *IC Compiler*, a su versión más moderna *IC Compiler* 2. Esto buscando que la mayor robustez de está última permitiera una mejor optimización de la síntesis.

Para que la migración pudiera ser llevada cabo de forma efectiva, se investigó detalladamente los manuales de la herramienta provistos por *Synopsys*, en específico los siguientes: *Library Manager User Guide* [12], *Design Planning User Guide* [13] e *Implementation User Guide* [14] [1]. Con los conocimientos adquiridos se generó una propuesta de flujo de trabajo a seguir que permitiera llevar a cabo un proceso de síntesis física sin errores de diseño en la herramienta *IC Compiler* 2, de aquí en adelante referida como ICC2. Los pasos del flujo son los descritos en las siguiente secciones.

Se pretende que este flujo pueda ser ejecutado sin importar el *foundry* o tecnología a utilizar. Sin embargo, es importante notar que actualmente el funcionamiento correcto del mismo únicamente está garantizado para diseños que utilicen las librerías de 180 nm y 6 capas de metal de TSMC. Esto se debe principalmente a las librerías que deben ser provistas por el *foundry* ya que, al momento de la realización de este trabajo, se desconoce si la cantidad y tipo de archivos contenidos en estas librerías varía entre *foundries* o entre tecnologías.

[1] Se debe contar con credenciales de Synopsys | SolvNetPlus para acceder a los manuales

7.1. Preparación de librerías

Debido a que el resultado del proceso de síntesis lógica es un RTL donde se hace referencia a las celdas del fabricante, es necesario que ICC2 tenga referenciadas las librerías que contienen estas celdas para que la traducción de HDL a *layout* pueda ser realizada. Los fabricantes proveen un set de archivos que conforman estas librerías y los mismos pueden ser clasificados en las siguientes categorías según la información que contienen:

- *Technology files*: Describen los parámetros de la tecnología a utilizar, como lo es el nombre y cantidad de capas de metal, las dimensiones de los metales, las propiedades eléctricas de cada capa, las propiedades de las vías, entre otros. Es importante mencionar que estos archivos no contienen información específica de las celdas más que algunos parámetros generales que se mencionarán más adelante.
- Archivos de librerías lógicas: Describen la funcionalidad de las celdas, las entradas y salidas de estas, así como sus parámetros de *timing* y potencia. Estos no describen cómo se componen físicamente las celdas, sino que cómo debe de ser su funcionamiento a nivel lógico.
- Archivos de librería física: Describen los parámetros físicos de las celdas y sus características internas, como lo es el tamaño y tipo de los pines y difusiones. Adicionalmente describen cómo son las conexiones entre estas características. Por lo tanto, estos archivos detallan cómo deben de estar constituidas físicamente las celdas para que estas puedan cumplir su función lógica.

7.1.1. Creación de *Cell Libraries*

Para que toda la información contenida en los archivos pueda ser utilizada por ICC2, estos deben estar contenidos en una o varias librerías conocidas como *Cell Libraries* (CLIBs). Este tipo de librerías también son conocidas como librerías *New Data Model* (NDM) por el tipo de formato que usan (**.ndm**). Para la creación de la CLIB se debe acceder a la herramienta "*Library Manager*". Esta es una extensión de ICC2 que permite al usuario llevar a cabo el flujo de trabajo presentado en la Figura 17. Los pasos a seguir, junto con los comandos útiles que se pueden ejecutar, son los siguientes:

1. Se debe crear un *workspace* sobre el cual se estará trabajando y al cual se le cargarán los archivos pertinentes para la creación de la librería. Al momento de crear el *workspace* es necesario indicar inmediatamente un archivo tipo **.tf**, el cual corresponde al *technology file*. Este paso puede ejecutarse con el siguiente comando:

   create_workspace -flow *tipo-de-flujo* **-technology** *technology-file MiWorkspace*

   Este último parámetro indica el nombre que se desea colocar al *workspace* y puede ser cualquiera que el usuario desee siempre y cuando este no tenga espacios en blanco.

2. Se debe añadir los archivos de las librerías lógicas y física. Para las librerías lógicas se puede añadir archivos tipo **.db** o **.lib** y esto puede realizarse, en el caso de los primeros, con el siguiente comando:

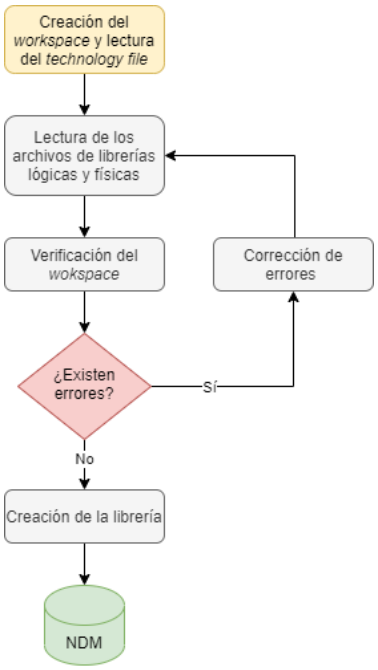


Figura 17: Flujo de trabajo típico para la creación de CLIBs en el *Library Manager*

read_db {*lista de archivos .db*}

El tipo de archivos de librería física que pueden ser agregados son tipo **.ndm**, **.gds**, **.oasis** o **.lef**. Únicamente se debe indicar un archivo y esto puede hacerse, en el caso de los archivos **.lef**, con el comando:

read_lef *archivo.lef*

3. Una vez todos los archivos necesarios hayan sido añadidos estos deben ser verificados. Para esto la herramienta identifica y corrige automáticamente ciertos errores y, de ser posible, crea una nueva vista conocida como *frame views*. Estos son archivos que contienen tanto información física como lógica, todo a un nivel alto de abstracción. Para ejecutar este proceso se debe indicar el *workspace* que se desea verificar con el comando:

   current_workspace *nombre-del-workspace*

   Se puede no indicar el nombre del *workspace* y la herramienta utilizará el último *workspace* que fue editado. Seguido de esto se utiliza el comando:

   check_workspace

   Este último es el que realiza la verificación del *workspace*.

4. Existen errores que la herramienta no es capaz de corregir automáticamente, para obtener una lista de los errores restantes luego de la verificación se puede utilizar el comando:

gui_create_window -type MessageBrowserWindow

Estos errores pueden ser de distintos tipos, por ejemplo, si el usuario no indica archivos de librerías lógicas o física, se mostrará un error indicando que la información necesaria para la creación de la librería está incompleta. En cualquier caso estos errores deben ser corregidos por el usuario directamente en los archivos que fueron agregados.

5. Cuando la verificación se ejecute sin errores es posible proceder con la creación de la librería. Primero debe indicarse el *workspace* que se desea utilizar para la creación de la librería, esto con el comando:

   current_workspace *nombre-del-workspace*

   Seguido de esto se debe ejecutar el comando:

   commit_workspace -output *MiLibreria.ndm*

   El nombre de la librería puede ser cualquiera que el usuario desee siempre y cuando este no contenga espacios en blanco. Es importante indicar el formato **.ndm** luego del nombre. Si se ejecuta el comando **commit_workspace** sin indicar un nombre la herramienta guardará la librería utilizando el nombre del *workspace*.

En el primer paso del flujo anterior se mencionó que al momento de ejecutar el comando de **create_workspace** se debe indicar el tipo de flujo a utilizar. Esto se debe a que dependiendo del tipo de flujo indicado la herramienta solicitará distintos tipos de archivos, por lo que la información contenida en la CLIB será distinta. Los flujos que pueden ser elegidos son: *Normal*, *Technology-only*, *Frame-only*, *Extraction timing model* (ETM), *Physical-only*, *Exploration*, *Edit* y *Agreggate*. Los flujos que resultaron útiles para este trabajo fueron el *Normal*, *Physical-only* y *Agreggate*, estos se explican a continuación:

- *Normal*: Este flujo sigue los pasos indicados en la Figura 17 y descritos anteriormente. La CLIB generada contiene las celdas que existen tanto en los archivos de librerías lógicas como en el de las físicas. Por lo mismo, la librería contiene información lógica y física de cada celda. Se recomienda utilizar este flujo para crear librerías que contengan las celdas estándar y/o celdas de pads IO.
- *Physical-only*: Existen celdas cómo los *corners*, *filler cells* o *tap cells* que se encuentran descritas únicamente de forma física. Cuando este es el caso no se puede utilizar un flujo normal, ya que en este último toda celda que no tiene descripción lógica y física es descartada. Por lo mismo se utiliza un flujo *physical-only* el cual, al igual que el flujo normal, sigue los pasos indicados en la Figura 17, sin embargo, la CLIB resultante contiene aquellas celdas que únicamente se encontraban descritas en los archivos de librerías físicas.
- *Aggregate*: Este flujo permite combinar distintas CLIBs en una sola. Es de especial importancia debido a que se puede dar el caso donde se tiene librerías obtenidas luego de utilizar un flujo normal y uno *physical-only*. En este caso las celdas contenidas en cada librería son distintas y por lo tanto se puede utilizar este flujo para lograr

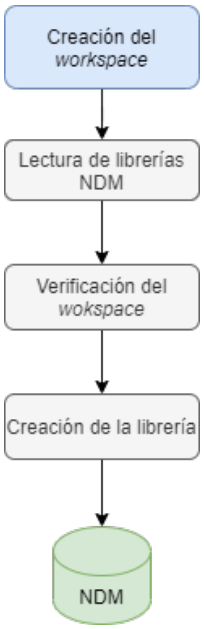


Figura 18: Pasos para ejecución de un flujo *Aggregate* en el *Library Manager*

que todas las celdas se encuentren en una misma librería. Los pasos de este flujo son distintos a los de flujos anteriores ya que este es más simple, cómo se puede observar en la Figura 18. En este caso no se debe indicar el *technology file* al momento de crear el *workspace*. Para indicar las librerías que se desea agregar se puede utilizar el comando:

read_ndm {*lista de archivos .ndm*}

Para la verificación y creación de la librería se puede utilizar los mismos comandos que en los flujos anteriores.

Cuadro 4: Archivos de TSMC para la creación de CLIBs con las celdas estándar

Tipo de archivo	**Nombre**
Technology file	tsmc018_6lm.tf
Librerías lógicas	tcb018gbwp7tbc.db tcb018gbwp7tlt.db tcb018gbwp7tml.db tcb018gbwp7ttc.db tcb018gbwp7twc.db tcb018gbwp7twcl.db
Librería física	tcb018gbwp7t_6lm.lef

Cuadro 5: Archivos de TSMC para la creación de CLIBs con los *pad cells*

Tipo de archivo	**Nombre**
Technology file	tsmc018_6lm.tf
Librería lógica	tpd018nvtc.db
Librería física	tpd018nv_6lm.lef

Las librerías de 180 nm de TSMC contienen una amplia cantidad de archivos y muchas veces se debe navegar por varias carpetas para encontrar la que contiene los archivos que se debe utilizar. Para facilitar este proceso se puede referir al Cuadro 4 donde se detalla el nombre de los archivos útiles para la creación de una librería NDM que contiene las celdas estándar y destinadas a ser utilizadas en un proceso con tecnología de 180 nm y 6 capas de metal. Estos archivos contienen información tanto de las celdas estándar cómo de las *filler cells* estándar.

A pesar que una CLIB con esta información ya permite iniciar la síntesis física, esta no es suficiente para completar la síntesis en su totalidad. También se debe contar con las celdas para los *pads* de entradas y salidas (IO), así como los *corners* y *fillers* IO. Para la creación de la librería que contiene estos componentes se puede utilizar los archivos detallados en el Cuadro 5, los cuales nuevamente son específicos para un proceso de 180 nm y 6 capas de metal. La información sobre estas librerías se encuentra detallada en el documento "*AN_-001_miniasic_information_20170125.pdf*", el cual fue provisto por TSMC.

Cómo se puede ver hasta el momento, existen distintos tipos de archivos según el tipo de celda, adicionalmente, dentro de estos archivos existen celdas que no tienen descripción lógica, únicamente física. Para realizar el proceso de síntesis física en ICC2 es necesario indicar la o las librerías donde se encuentran las celdas a utilizar. Por lo tanto, se pueden crear las CLIBs necesarias para tener una descripción completa de las celdas del *foundry* y posteriormente indicarle estas librerías a ICC2. Sin embargo, por motivos de orden se recomienda unificar estas librerías en una sola, lo cual es posible de acuerdo a los flujos mencionados anteriormente.

Luego de varias pruebas se pudo determinar que este proceso puede llevarse a cabo utilizando el flujo presentado en la Figura 19. Es importante mencionar que este proceso solo debe ejecutarse una vez ya que, al tener la o las librerías necesarias creadas estas simplemente deben ser referenciadas al crear un nuevo diseño. Esto último es válido siempre que se tome en cuenta que una CLIB es específica para una tecnología y *foundry*. Se puede encontrar el *script* que ejecuta todos los procesos descritos anteriormente en los anexos de este trabajo.

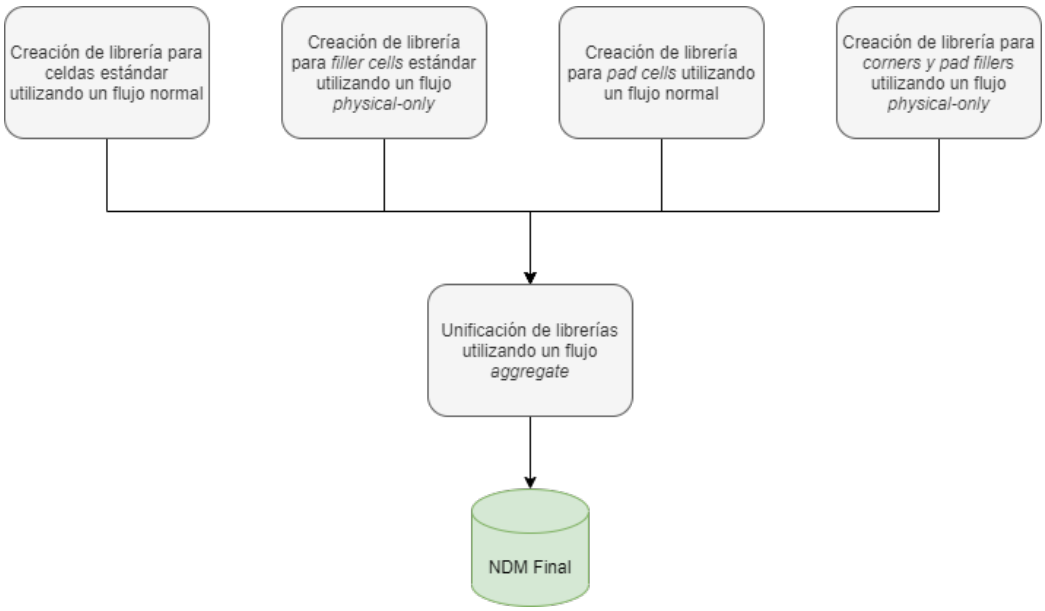


Figura 19: Pasos para la creación de una CLIB utilizando las librerías de TSMC en el *Library Manager*

7.1.2. Interfaz gráfica del *Library Manager*

Al momento de abrir el *Library Manager* se puede indicar que este se abra junto con su interfaz gráfica de usuario (gui). Esto sirve de gran apoyo para guiar al usuario en la creación de las CLIBs que requiera. Al inicializar la interfaz se muestra una ventana cómo la que se encuentra en la Figura 20, en esta se cuenta con la opción de crear una nueva librería, editar una librería o simplemente visualizar los contenidos de una librería ya existente.

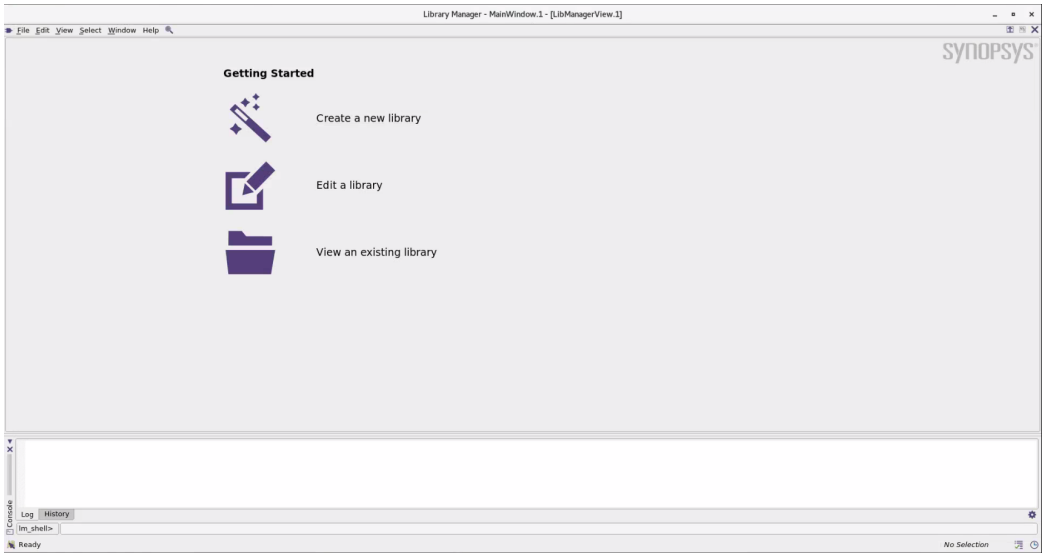
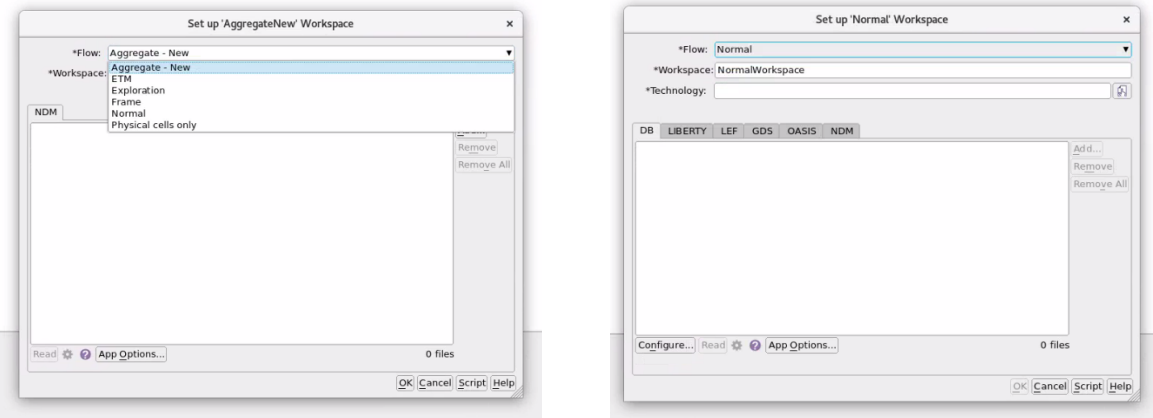


Figura 20: Ventana de inicio del *Library Manager*

Si se desea crear una nueva librería se muestra una ventana como la de la Figura 21, donde primero se debe elegir el tipo de flujo a utilizar. Luego se debe ingresar los archivos necesarios para ejecutar dicho flujo, como se observa en la Figura 21b para un flujo normal. Una vez los archivos han sido añadidos, la ventana de inicio de la herramienta cambia a la mostrada en la Figura 22. Cómo se observa en la figura, se muestran los archivos añadidos y se da la opción de verificarlos con el botón de "*Check Workspace*". Posterior a la verificación se habilita la opción de "*Commit Workspace*", con la cual se finaliza la creación de la librería.



(a) Ventana de selección de flujo (b) Ventana para ejecución de un flujo normal

Figura 21: Ventana para creación de una nueva librería en el *Library Manager*

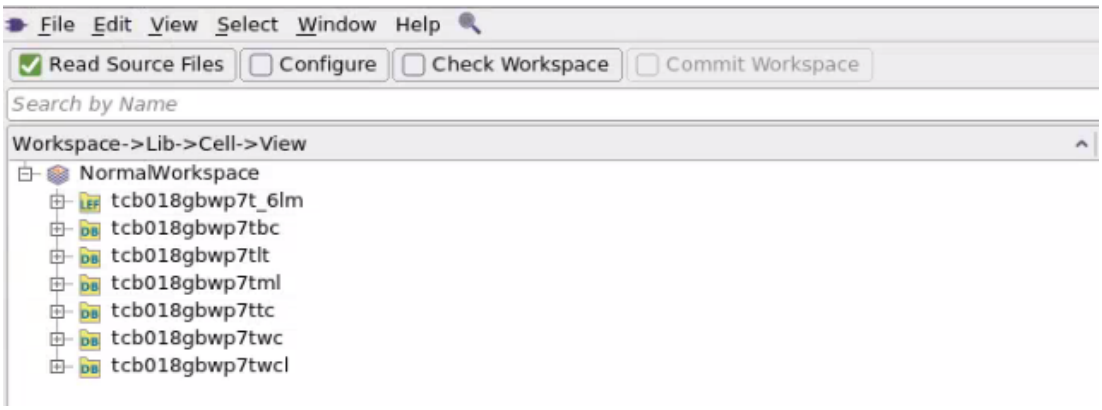


Figura 22: Ventana de visualización de la librería en el *Library Manager*

7.2. *Floorplan*

Luego de familiarizarse con la herramienta ICC2 se pudo determinar que el flujo de trabajo propuesto en trabajos anteriores para la etapa de *floorplaning* debía ser modificado. Esto debido principalmente a la migración de ICC a ICC2, la cual hizo necesario actualizar ciertos procesos, así como agregar algunos pasos adicionales con el fin de optimizar esta etapa. Con los conocimientos adquiridos y las pruebas realizadas se pudo generar una propuesta para llevar a cabo un flujo de trabajo que implemente la etapa del *floorplan* en ICC2, este se muestra en la Figura 23. Los pasos a ejecutar se detallan a continuación.

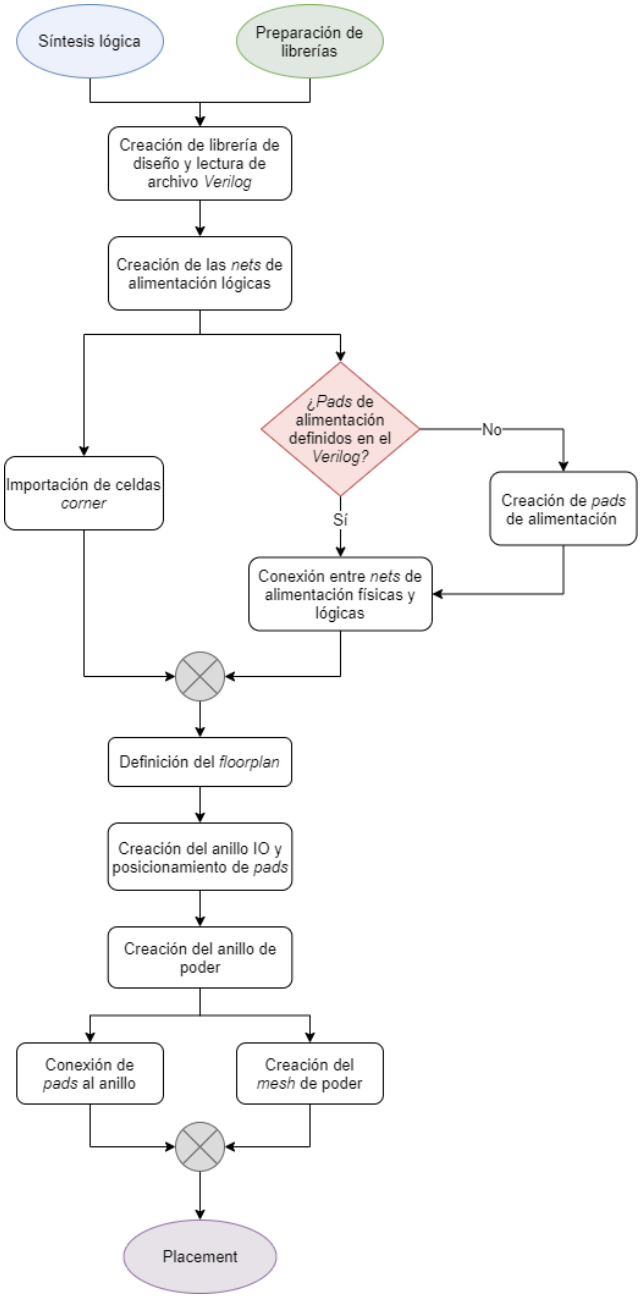


Figura 23: Flujo para ejecución del *floorplan* en ICC2

7.2.1. Lectura de Verilog y creación de *power nets*

Luego de tener una CLIB que permita referenciar todas las celdas del fabricante se puede iniciar formalmente con la síntesis física en ICC2. Para iniciar la etapa del *floorplan* en ICC2 se debe crear una librería conocida como librería de diseño y cuyo propósito es el de almacenar toda la información del diseño generado por el usuario. Es importante mencionar que en las librerías de diseño de ICC2 cada diseño es conocido como un bloque y una librería puede almacenar varios bloques si así se desea. Se puede crear una librería de diseño utilizando el comando:

create_lib *LibDiseño.ndm* **-technology** *technology-file.tf* **-ref_libs** {*lista de librerías de referencia*}

El nombre de la librería puede ser cualquiera que el usuario desee siempre y cuando este no contenga espacios en blanco o el carácter de doble punto (:). Para la lista de librerías de referencia puede indicarse únicamente la CLIB unificada o alternativamente la distintas librería de celdas si estas se encontraran separadas. Como se puede notar, el formato de las librerías de diseño es el mismo que el de las CLIBs (**.ndm**). Cuando se crea la librería de diseño se hace con una copia de las librerías de referencia indicadas, esto de forma que se pueda tener todas las CLIBs y los diseños realizados en una sola librería. Adicional a este comando, se debe importar los modelos parasíticos que permitirán ejecutar el proceso de extracción de parásitos más adelante en el flujo. Estos modelos se encuentran en un archivo con formato TLUPlus y pueden ser añadidos a la librería de diseño con el siguiente comando:

read_parasitic_tech -tlup *modelos_parásitos.tluplus* **-layermap** *archivo-de-layermap*

Cómo se observa en el último comando, también es necesario indicar un archivo tipo *layermap*. Este archivo es el que indica cómo deben codificarse los nombres de las *layers*, definidos en el *technology file*, una vez el diseño se exporta al formato GDSII. Para el caso de un diseño con librerías de TSMC de 180 nm y utilizando 6 capas de metal, se puede utilizar los archivos del Cuadro 6. En esta etapa del flujo este último paso es completamente opcional, sin embargo, se recomienda añadir estos archivos al inicio cuando la librería aún no contiene información del circuito a sintetizar.

Cuadro 6: Archivos de TSMC para importación de modelos parasíticos

Tipo de archivo	**Nombre**
TLUPlus	t018lo_1p6m_typical.tluplus
Layermap	star.map_6M

Luego de crear la librería de diseño es posible cargar los archivos generados durante la síntesis lógica y los cuales describen el circuito a sintetizar ya utilizando las celdas del *foundry*. Para hacer esto se debe leer el Verilog sintetizado junto con sus respectivas restricciones. Estas últimas se encuentran en un archivo cuyo formato es *Synopsys Design Constraints* o SDC. Para importar estos archivos se puede utilizar los siguientes comandos:

read_verilog *netlist.v*
read_sdc *restricciones.sdc*

Cuando ICC2 realiza la lectura del *netlist* automáticamente guarda los módulos instanciados y los asigna al diseño actual. Adicionalmente, aplica al diseño las restricciones especificadas en el SDC.

Es importante mencionar que las celdas que se encuentran instanciadas en el *netlist* son aquellas que tienen descripción física y lógica, como lo son las celdas que describen compuertas lógicas (celdas estándar) o *pads* de entrada y salida (celdas IO). Sin embargo, para la síntesis física es necesario contar con celdas de *corners* o *fillers*, celdas que únicamente tienen descripción física. Ya que el propósito principal de estas es el de tener un *layout* más uniforme, no tiene sentido que se encuentren descritas en el *netlist*. A consecuencia de esto al leer el *netlist* no se insertan automáticamente estas celdas *physical-only*, por lo que este proceso debe hacerse de forma manual. En este punto del flujo únicamente es necesario insertar los *corners* y esto se puede hacer con el comando:

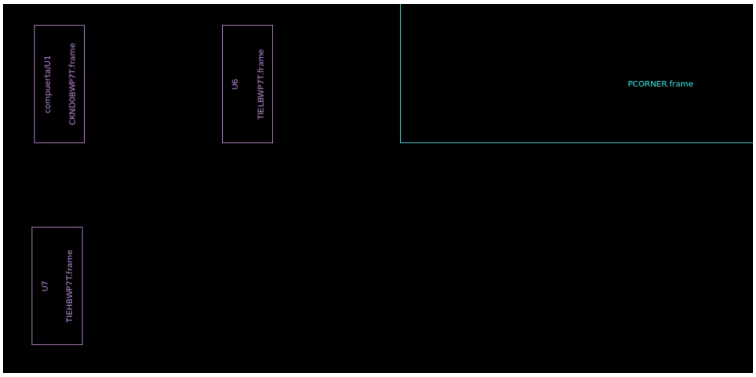
create_cell {*Corner1 Corner2 ... CornerN*} *celda-de-la-librería*

Este último comando no es de uso exclusivo para celdas *corner*, puede utilizarse para insertar cualquier tipo de celda que se desee, siempre y cuando esta se encuentre definida en la librería de diseño. En las librerías de TSMC utilizadas el nombre de la celda de *corners* es "PCORNER" y, en el caso de un *layout* rectangular, se debe insertar un total de 4, una por cada esquina.

Adicional a los *corners* y *fillers*, puede que los *pads* de alimentación y tierra tampoco se encuentren definidos en el *netlist*. Esto puesto que en un lenguaje HDL como Verilog no es necesario definir explícitamente cómo se estará alimentando a los módulos. Si este fuera el caso nuevamente se puede utilizar el comando de **create_cell** para insertar estos *pads*. En el caso de las librerías de TSMC de 180 nm, las celdas que se puede utilizar son "PVDD1CDG" para el *pad* de VDD y "PVSS1CDG" para el de VSS. Con todo lo realizado hasta este punto ya se debe de poder visualizar todas celdas en el diseño, tal como se muestra en la Figura 24. Este es el caso para una compuerta NOT donde las celdas de *pads* y *corners* se muestran de color celeste, mientras que las celdas estándar de color rosa.

Cómo se puede observar en la Figura 24a, existe una diferencia significativa entre el tamaño de las celdas de *pads* o *corners* y el de las celdas estándar. Es de especial importancia prestar atención a la Figura 24b que muestra las celdas estándar necesarias para implementar una compuerta NOT. Las librerías de TSMC cuentan con celdas dedicadas para implementar funciones lógicas como NOT, AND, OR, XOR, etc. Por lo tanto, es de esperar que al realizar la lectura del Verilog solo debería mostrarse una celda, sin embargo, en el Verilog puede que existan conexiones como las de los pines de los *pads* que deben ir a un 1 o 0 lógico, no

(a) Celdas de *pads* y *corners*



(b) Celdas estándar

Figura 24: Celdas importadas en ICC2

necesariamente a VDD o VSS. En estos casos durante la síntesis lógica se incluyen dos celdas adicionales, la "TIEHBWP7T" y la "TIELBWP7T", cuyo propósito es hacer en el diseño esta traducción de VDD y VSS a un 1 ó 0 lógico, respectivamente. Es por esto que estas celdas, a pesar de no impactar de ninguna manera la función a implementar, son de especial importancia para que ICC2 pueda conectar correctamente los pines de los *pads* y así evitar posibles errores de LVS más adelante.

Con todas las celdas importadas se debe proceder a crear las *power nets*. A pesar que las celdas de VDD y VSS fueron definidas en el Verilog o creadas manualmente, ICC2 aún no sabe el nombre de las *nets* que estarán brindando alimentación al circuito y a que *pads* IO deben ir conectadas. Para crear las nets de VDD y VSS se puede utilizar los siguientes comandos:

create_net -power *VDD*
create_net -ground *VSS*

Para asociar estas *power nets* a los *power pins* de las celdas, se puede utilizar el siguiente comando:

```
Power data for cell 'U6':
Corner: default


Power pins:

    Pin Name       Type      PG Type   Power Rail  Netlist Net    UPF Supply Net Early Voltage  Late Voltage
-------------------------------------------------------------------------------------------------------------
    VDD            power     primary   <default_rail>             VDD            3.30 *         3.30 *
    VSS            ground    primary   <default_gnd>              VSS            -- *           -- *


Power data for cell 'U7':
Corner: default


Power pins:

    Pin Name       Type      PG Type   Power Rail  Netlist Net    UPF Supply Net Early Voltage  Late Voltage
-------------------------------------------------------------------------------------------------------------
    VDD            power     primary   <default_rail>             VDD            3.30 *         3.30 *
    VSS            ground    primary   <default_gnd>              VSS            -- *           -- *
```

(a) Previo a creación y conexión de *nets* para VDD y VSS

```
Power data for cell 'U6':
Corner: default


Power pins:

    Pin Name       Type      PG Type   Power Rail  Netlist Net    UPF Supply Net Early Voltage  Late Voltage
-------------------------------------------------------------------------------------------------------------
    VDD            power     primary   <default_rail>
                                                    VDD            VDD            3.30 *         3.30 *
    VSS            ground    primary   <default_gnd>
                                                    VSS            VSS            -- *           -- *


Power data for cell 'U7':
Corner: default


Power pins:

    Pin Name       Type      PG Type   Power Rail  Netlist Net    UPF Supply Net Early Voltage  Late Voltage
-------------------------------------------------------------------------------------------------------------
    VDD            power     primary   <default_rail>
                                                    VDD            VDD            3.30 *         3.30 *
    VSS            ground    primary   <default_gnd>
                                                    VSS            VSS            -- *           -- *
```

(b) Luego de creación y conexión de *nets* para VDD y VSS

Figura 25: Resultado del comando **report_cells -power**

connect_pg_net -net *power-net-logica* [**get_pins -physical_context** *power-net-fisica*]

Para el nombre de la *power net* física basta con utilizar "*VDD" o "*VSS" según sea el caso. Una vez realizado este proceso se puede generar un reporte donde se describe los *power pins* de cada celda que se encuentra en el diseño. Esto último con el comando **report_cells -power**. Este reporte contiene información de cada celda, sus pines de alimentación, el tipo de pin, la *net* asociada, etc. Cómo se puede ver en la Figura 25, los resultados del reporte varían si se realizó o no el proceso de creación y conexión de las *nets* para VDD y VSS. En la figura se muestra la misma parte del reporte donde se describe las conexiones en dos celdas "U6" y "U7". Previo a la creación de las *nets* se describe los pines de alimentación y tierra, pero no se tiene asociada una *Netlist Net*. Si esto se deja de esta manera al momento de realizar el proceso de enrutamiento estos pines no se conectarían a nada y por lo tanto, ninguna celda del diseño tendría alimentación. Es por eso que el proceso de creación y conexión de estas *power nets* es de suma importancia para la síntesis física.

7.2.2. Creación de *floorplan* y anillo IO

Con todas las celdas importadas y las *nets* asignadas se puede proceder a definir las áreas donde estarán posicionadas las celdas, su orientación, distancia entre sí y en general su organización dentro del diseño. Para esto se debe definir el área total del *die* y de su *core*,

siendo este último el espacio en el centro del *die* donde se colocan las celdas estándar. Los parámetros del *floorplan* pueden ser definidos con el siguiente comando:

initialize_floorplan -control_type ***die/core*** **-site_def** *nombre-del-site* **-use_site_row -keep_all -side_length** {*X Y*} **-core_offset** {*número*}

Este último comando es muy importante ya que define varios puntos fundamentales para el resto del flujo. Primero se debe indicar si los parámetros que se especificarán son para el *die* o el *core*, siendo este último el valor por defecto. Las diferencias de seleccionar uno u otro se explican más adelante. Luego se especifica un *site definition*, el cual es un parámetro que indica el espaciado que deben tener las filas donde se colocarán las celdas estándar dentro del *core*. Este valor se encuentra especificado dentro del *technology file* y es el utilizado por el *foundry* para definir las dimensiones de las celdas estándar.

Las celdas estándar de un *foundry* pueden tener todas un mismo tamaño o distintos tamaños y cada uno de estos se encuentra definido como un *site definition*. Con esta opción indicamos cuál de todos los *sites* se desea utilizar en el *core*. En el caso del *technology file* provisto por TSMC se tiene dos *site definitions* mostrados en la Figura 26 y donde se puede ver que ambos definen una misma altura, cambiando únicamente el ancho. Debido a que la altura es la misma no importa cual se indique en el comando, el espaciado entre filas será el mismo. Para este trabajo la definición utilizada fue la de "*unit*".

```
Tile            "gaunit" {
                width                           = 2.24
                height                          = 3.92
}

Tile            "unit" {
                width                           = 0.56
                height                          = 3.92
```

Figura 26: *Site definitions* en el *technology file* de 6 capas y 180 nm de TSMC

Luego de definir el *site* a utilizar se especifica la opción de **use_site_row**, la cual es completamente opcional y su efecto es que las filas dentro del *core* se generan individualmente en vez de ser generadas como un único arreglo de filas. Cuando se define un único *site* este efecto no es perceptible, pero puede resultar útil si se desea usar varios *sites* y tener filas con espaciado variable. La opción de **keep_all** únicamente es para garantizar que luego de crear el *floorplan* no se eliminen los elementos que ya se encontraban en el diseño. Las opciones de **side_length** y **core_offset** dependen directamente de lo especificado en el **control_type**. Si el valor de este último parámetro es *die*, con **side_length** definimos el área total del diseño y a estas dimensiones se les resta el valor de **core_offset** para determinar el área del *core*. Por otro lado, si el valor del parámetro fuera *core*, el **side_length** define el área únicamente del core y el valor de **core_offset** se suma a estas dimensiones para calcular el área total del *die*.

Se debe tomar en cuenta que el tamaño del *core* rara vez será exactamente el que especificamos y esto se debe a que ICC2 ajusta el tamaño para que la división de la altura del *core* sobre el espaciado de las filas (el *height* del *site definition*) sea un número entero. Con esto se evita que haya filas que tengan un espaciado distinto al resto, especialmente para

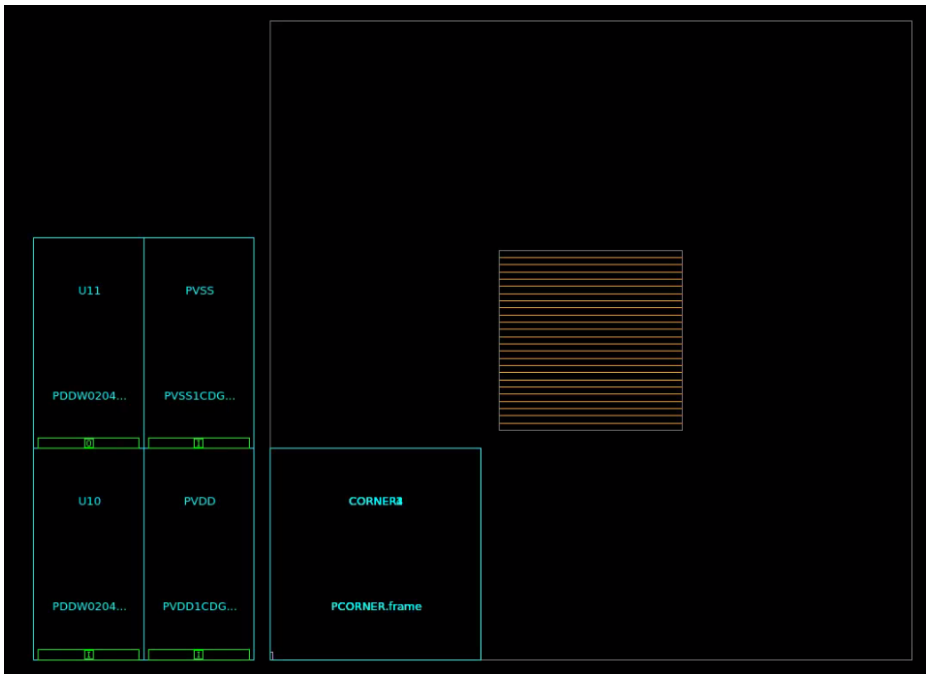


Figura 27: Inicialización del *floorplan* para una NOT en ICC2

las filas que se encuentran cerca del borde superior o inferior. Por ejemplo, si se especifica una altura del *die* de 275 la altura real sería de 274.4, de forma que se puedan crear 70 filas con espaciado uniforme. La Figura 27 muestra un ejemplo del diseño de una NOT luego de haber ejecutado el comando de inicialización del *floorplan*. Es importante mencionar que por defecto ICC2 no muestra las filas dentro del *core*, para habilitar esta vista se debe activar el objeto *Site Array* o *Site Row* en la pestaña *Objects* de la ventana *View Settings*. Esta ventana por defecto se encuentra en la barra lateral derecha de la interfaz gráfica.

Luego de la creación del *floorplan* es necesario crear el anillo IO (*input-output)*. Este anillo debe ser generado dentro del *die*, pero afuera del *core* y permite definir cómo se posicionarán los *pads* IO. Estos *pads* serán organizados de forma que su distribución sea uniforme en cada lado del *die*, de esta manera formando un anillo. Para ejecutar este proceso y luego colocar los *pads* se puede usar los siguientes comandos:

create_io_ring -name *nombre-del-anillo* **-corner_height** *altura-corner*
place_io

El primer comando requiere que se especifique el nombre que se desea poner al anillo y es completamente opcional, si no se coloca un nombre ICC2 le asigna uno por defecto, aunque se recomienda asignarle un nombre ya que puede resultar útil como se verá más adelante. Adicional a esto, se debe definir la altura de los *corners* para que ICC2 pueda determinar el espaciado que debe haber entre el anillo interior y el exterior. Para visualizar esto de forma más fácil se puede dirigir a la Figura 28. Esta muestra el diseño luego de haber creado el anillo y colocado los *pads* de entradas y salidas con el comando **place_io**. Cómo se puede ver, el anillo exterior se crea en el borde del *die* y el anillo interior se crea utilizando las dimensiones de los *corners* como referencia. Con esto se puede ver que, en conjunto, el anillo

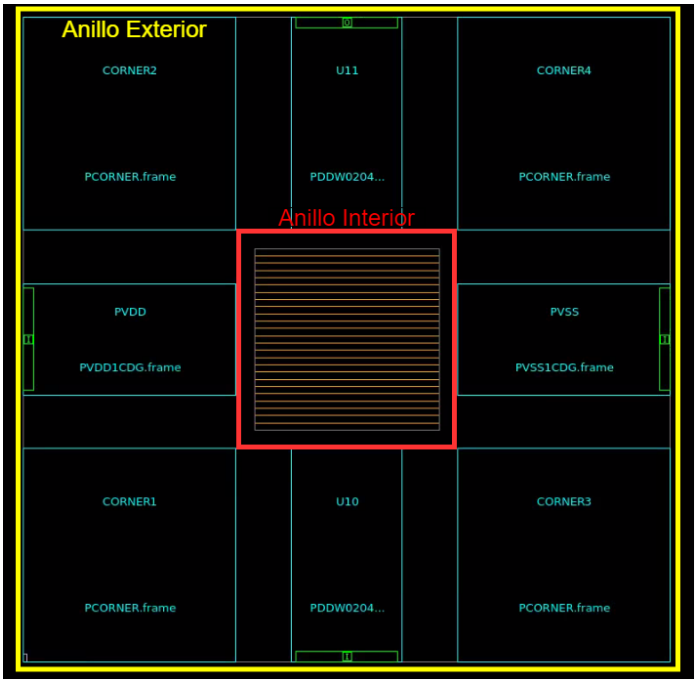


Figura 28: Anillo de entradas y salidas en ICC2

interior y exterior forman el anillo de entradas y salidas. Se debe tomar en cuenta que ICC2 no despliega los anillos en el diseño como se muestran en la figura y tampoco hay una opción para hacerlo, estos fueron dibujados para ilustrar estos conceptos.

El comando **place_io** posiciona las celdas IO de forma automática de acuerdo a criterios internos de ICC2. Si se desea indicar explícitamente la posición deseada de un *pad* se puede utilizar el siguiente comando:

add_to_io_guide [get_io_guides *nombre-del-anillo.lado*] *nombre-pad*

Para este comando se debe indicar el nombre del anillo de referencia y el lado donde se prefiere que sea colocada la celda. En un *die* rectangular se tiene cuatro lados y estos son identificados con los nombres en inglés *left*, *right*, *top* y *bottom*. El nombre del *pad* debe ser el nombre de la celda en el diseño, no su nombre en la CLIB. Por ejemplo, en la Figura 28 el *pad* de VDD tiene dos nombres, el primero es "PVDD" y es el nombre de la celda en el diseño actual, el cual fue asignado por el usuario. El segundo nombre es "PVDD1CDG" y es el nombre de referencia en la CLIB. Esta misma distinción aplica para todas las celdas en el diseño. Aunque este comando es opcional es conveniente especificar que se desea colocar los *pads* de VDD y VSS en uno de los lados izquierdo o derecho, tal como se demostrará en las siguientes secciones.

7.2.3. *Power planning*

Con el anillo IO creado se puede proceder a la etapa del *power planning*. Esta etapa del *floorplaning* es muy importante ya que permite garantizar que todas las celdas puedan

conectarse correctamente a las *nets* de poder (VDD y VSS), lo que es esencial para que el circuito funcione correctamente. La necesidad de esta etapa surge del hecho que no es apropiado tratar las *nets* de poder cómo si fuesen una *net* de señalización más. Una de las principales razones de esto es que su cobertura es mucho más extensa al tener conexión con la gran mayoría de elementos del *layout*. Por lo mismo, no conviene enrutar estas *nets* junto y de la misma manera que las de señalización, al correr el riesgo de que se creen rutas innecesariamente complejas y que sean difíciles de optimizar, lo que trae una gran cantidad de efectos negativos sobre el desempeño del circuito. Además, el impacto de estos efectos incrementa conforme la complejidad del diseño es aumentada.

Se debe tener cuidado de categorizar el *power planning* simplemente como el enrutamiento de las *nets* de poder. Este proceso no es un enrutamiento como tal ya que, no crea las conexiones de las celdas hacia toda la red de alimentación de circuito. En lugar de esto define cómo se encontrarán distribuidas VDD y VSS en todo el *layout* para que posteriormente las celdas puedan ser conectadas eficientemente durante el *placement* y enrutamiento. Para lograr esto se busca acercar lo más posible tanto VDD como VSS a las áreas donde se encontrarán ubicadas las celdas, de forma que todos estos elementos tengan un acceso fácil a las *nets* de poder.

7.2.3.1. Anillo de poder

El primer paso del *power planning* es la creación de un anillo de poder o PG (*power-ground*). Este debe encerrar el *core* con dos anillos, uno transportando VDD y el otro VSS. Para lograr esto en ICC2 se debe iniciar definiendo un patrón de metales. Ya que el patrón que se desea es un anillo, se puede utilizar el siguiente comando:

create_pg_ring_pattern *nombre-del-patrón* **-horizontal_layer** *capa-de-metal-N* **-horizontal_width** {*número*} **-horizontal_spacing** {*número*} **-vertical_layer** *capa-de-metal-M* **-vertical_width** {*número*} **-vertical_spacing** {*número*}

Este último comando define un arreglo de metales que deben formar un anillo utilizando metal de la capa N para los *tracks* horizontales y metal de la capa M para los tracks verticales. Además de esto, define cuanto debe ser el ancho y espaciado de los *tracks* con orientación horizontal y lo mismo para los *tracks* con orientación vertical. Por otro lado, requiere que se asigne un nombre al patrón y este puede ser cualquiera que el usuario desee siempre y cuando no tenga espacios. Con el patrón de anillo definido este debe ser asignado a lo que se conoce como una estrategia. Las estrategias permiten definir las *nets* que deben ser asignadas al patrón, así como varios parámetros importantes para su creación. Una estrategia que implemente este patrón de anillo puede definirse con el siguiente comando:

set_pg_strategy *nombre-de-la-estrategia* **-pattern** {{**name:** *nombre-del-patrón*} {**nets:** {*VDD VSS*}} {**offset:** {$offset_x$ $offset_y$}}} **-core**

Para este comando se debe definir el nombre que se desea colocar a la estrategia, nuevamente este puede ser cualquiera que el usuario desee siempre que no tenga espacios en blanco. Luego se le debe asignar un patrón y para esto se debe indicar el nombre de un

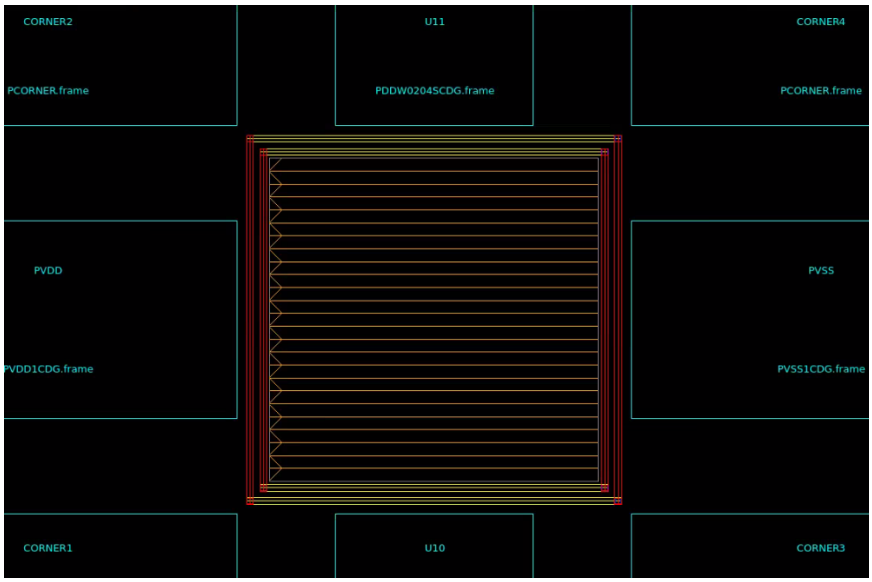


Figura 29: Anillo de poder en ICC2

patrón previamente creado, indicar el nombre de las *nets* de alimentación y los valores de *offset*. Por último, se debe indicar el área dentro del *layout* donde se ubicará el patrón. Al ser un patrón de anillo esto debe ser alrededor del *core* con el *offset* indicado. Para que todo lo definido anteriormente sea creado y pueda ser visualizado en el diseño se debe compilar la estrategia. Esto se realiza utilizando el comando:

compile_pg -strategies *nombre-de-la-estrategia*

En la Figura 29 se muestra el *layout* para una compuerta NOT luego de haber creado el anillo PG. Para la creación del anillo se utilizó metal capa 2 (color amarillo) para los tracks horizontales y capa 3 (color rojo) para los verticales (METAL2 y METAL3 en las librerías de TSMC), así como un valor de 2 para el ancho y el espaciado en ambas direcciones. La figura muestra dos anillos, el más cercano al *core* contiene la *net* de VDD ya que esta fue la primera que se indicó en la estrategia. Adicional a esto, el anillo se encuentra a un *offset* de 1 en ambas direcciones. El segundo anillo es para VSS y su distancia del primer anillo es la indicada por el valor del espaciado entre *tracks* de metal.

7.2.3.2. Conexión de anillos

Luego de haber creado el anillo de poder, este debe ser acoplado al anillo IO, específicamente a los *pads* de VDD y VSS. Para realizar esto se recomienda indicar a ICC2 que debe reconocer las celdas de *pads* como celdas tipo macro. A pesar que puede parecer confuso categorizar un *pad* como una celda del mismo tipo que una memoria RAM, ROM u otra que contenga un circuito de alta complejidad, hacer esto es recomendado ya que puede ayudar a evitar errores en el acople de los *pads* al anillo. Para esto se debe cambiar la opción de la aplicación que indica cómo deben ser tratadas las celdas *pad* en ICC2. Se debe tomar en cuenta que todas las opciones de aplicación relacionadas al *floorplan* se encuentran en la categoría *plan*. En este caso, la opción que nos interesa puede ser modificada con el comando:

set_app_options -name plan.pgroute.treat_pad_as_macro -true

Seguido de esto se debe crear un nuevo patrón de metales cuyo objetivo sea el de conectar los *pads* al anillo. Esto puede realizarse con el siguiente comando:

create_pg_macro_conn_pattern *nombre-del-patrón* **-pin_conn_type** *long_pin/ring_pin/scatterd_pin* **-layers** {*capa-horizontal capa-vertical*} **nets** {*VDD VSS*} **-pin_layers** {*capa-pines*}

En este último comando se debe definir primero el nombre del patrón, luego de esto se debe indicar el tipo de pines a los que se hará conexión. Se puede elegir tres tipos según la distribución interna de las *nets* de poder en el *pad*. Para los *pads* de VDD y VSS de TSMC el tipo que debe ser indicado es el de *scattered_pin*. Luego de esto se debe indicar las capas de metal que se utilizarán para realizar las conexiones, indicando un metal para los *tracks* horizontales y otro para los verticales. También se debe indicar el nombre de las *nets* de VDD y VSS. Por último, se puede indicar la capa de metal donde se encuentran los pines que se desea conectar. Esto último es opcional y puede ser utilizado si se tiene pines en ciertas capas que no se desea conectar con el patrón creado. De no ser utilizada todos los pines en todas las capas son conectados.

Para identificar fácilmente en que capa de metal se encuentran los pines se puede activar su visualización en la pestaña de *Objects* de la ventana *View Settings*. Luego de hacer esto se debería poder observar los pines de cada celda y la capa de metal a la que pertenecen como se muestra en el ejemplo de la Figura 30. Alternativamente se puede abrir la CLIB correspondiente en el *Library Manager*, buscar la celda que se desea y abrir su vista de *frame*.

Con el patrón creado se debe proceder a crear una nueva estrategia que permita implementarlo. Para esto se puede utilizar nuevamente el comando **set_pg_strategy** con las siguientes configuraciones:

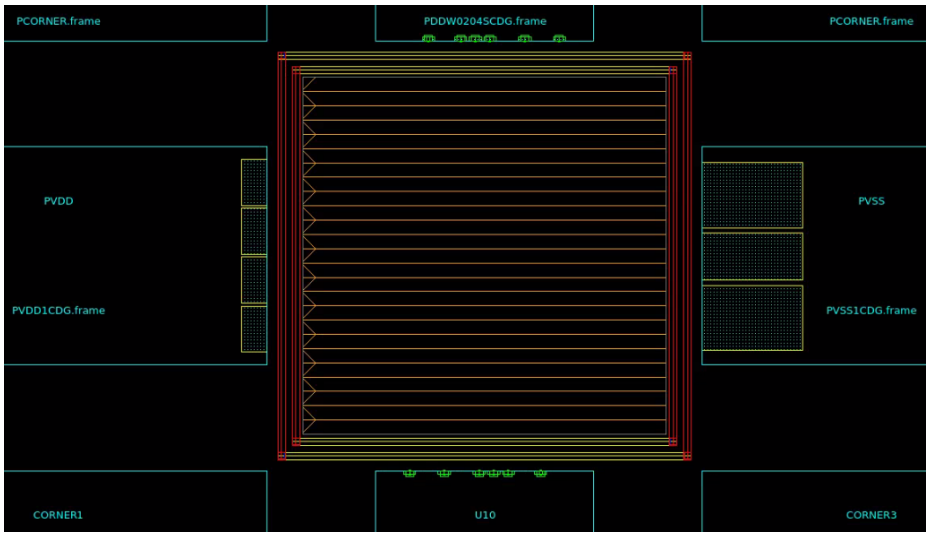


Figura 30: Visualización de pines de las celdas en ICC2

set_pg_strategy *nombre-de-la-estrategia* **-macros [get_cells** {*Pad-VDD Pad-VSS*}**]**
-pattern {{**name:** *nombre-del-patrón*} {**nets:** {*VDD VSS*}}}

Utilizando el comando se debe indicar el nombre deseado para la estrategia y seguido de esto indicar las celdas a las que se desea aplicar el patrón. Ya que lo que se busca es acoplar los *pads* de VDD y VSS al anillo de poder, únicamente deben ser indicadas las dos celdas correspondientes. El nombre a utilizar nuevamente debe ser el nombre en el diseño, no en la CLIB. Cómo se puede ver en la Figura 30, los pines de los *pads* IO se encuentran en capa 2 (metal amarillo) y estos deben conectarse a los *tracks* verticales del anillo que se encuentran en capa 3 (metal rojo). Esto será el caso siempre que se tenga los *pads* de VDD y VSS en los lados laterales del *die*. Con todo definido se debe compilar la estrategia, nuevamente utilizando el comando:

compile_pg -strategies *nombre-de-la-estrategia*

En la Figura 31 se muestra el resultado de implementar lo discutido anteriormente en el diseño de una compuerta NOT. Cómo se puede observar, las conexiones de los *pads* al anillo fueron creadas exitosamente. Es importante identificar a qué anillo fue conectado cada *pad*. El *pad* de la derecha corresponde al de VDD y el de la izquierda a VSS, tal como se muestra en la Figura 30. Se puede ver que el *pad* de VDD fue conectado al anillo de poder interno el cual, como se comentó anteriormente, tiene asignado el transporte de la *net* VDD. Por otro lado, el *pad* de VSS fue conectado al anillo externo que tiene designado el transporte de la *net* VSS. Con estas observaciones se puede comprobar que el acople de los *pads* de alimentación con el anillo de poder fue el correcto.

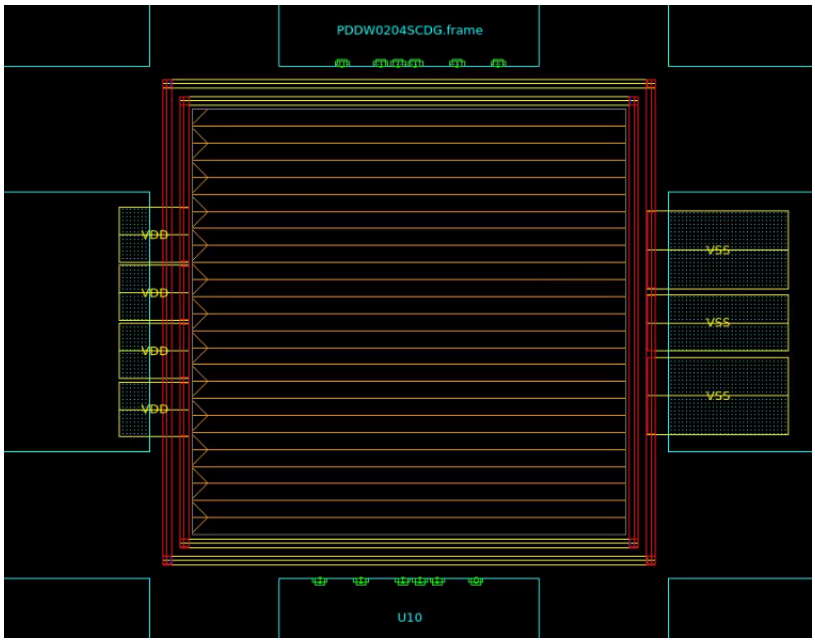


Figura 31: Conexión de *pads* de alimentación al anillo de poder en ICC2

7.2.3.3. *Mesh* de poder

Con estos resultados únicamente queda un paso para finalizar el *power planning*. Este es la creación de la estructura que transporta VDD y VSS desde el anillo de poder hacia el interior del *core* y distribuye estas *nets* de forma que cualquier celda tenga acceso ellas. Al analizar el *layout* que se tiene hasta el momento se puede ver que parte del esqueleto de esta estructura ya existe y son las filas que se encuentran dentro del *core*. Estas se crean de forma que su espaciado permita que en medio de dos filas puedan ser posicionadas las celdas estándar. Por lo tanto, parece lógico pensar que estas filas puedan ser sustituidas por *tracks* de VDD y VSS, de forma que en cualquier punto que se coloque la celda esta va a estar directamente conectada a la red de alimentación.

Se debe hacer una aclaración con esto último y es que las filas realmente no deben ser sustituidas como tal ya que estas físicamente no existen. Son colocadas por la herramienta con el fin de tener un *layout* más ordenado y por lo tanto sobre ellas se puede colocar estos *tracks* físicos sin problemas. Esto puede ser llevado a cabo siempre que se tomen en cuenta ciertas consideraciones. La primera es que en una fila no pueden coexistir dos *tracks* a la vez y esto significa que una fila puede tener solamente un *track* que transporta VDD o VSS. Esto se debe a que cuando se tiene varios *tracks* en paralelo se debe dejar un espaciado mínimo entre cada *track* para evitar la creación de cortos. El problema es que si se deja este espaciado reducimos efectivamente el espaciado real entre las filas y este está calculado para el tamaño exacto de una celda estándar. Esto se corrige limitando el número de *tracks* por fila a uno.

La segunda consideración es que la *net* de poder siendo transportada en cada *track* debe alternar entre filas, es decir, si la primera fila contiene un *track* de VDD, la segunda debe tener uno de VSS, la tercera uno de VDD, cuarta VSS y así sucesivamente. Esto es necesario ya que, si se pretende que entre *tracks* se coloquen celdas, estas últimas deben de poder acceder tanto a VDD como a VSS. Por último, se debe tomar en cuenta que a diferencia de las filas los *tracks* tendrán un ancho y esto significa que cuando se coloquen las celdas parte se colocará encima de los *tracks*. Para que esto no sea un problema se debe de poder garantizar que la parte de la celda que se coloque sobre el *track* de VDD sea donde se encuentra el pin de VDD y lo mismo para VSS.

Todo esto puede parecer un proceso bastante complejo, por suerte, ICC2 y las librerías de TSMC se encargan de tomar en cuenta estas consideraciones. Al crear los *tracks* ICC2 únicamente crea uno por fila y alterna las *nets* de forma automática. Por otro lado, todas las celdas estándar en las librerías tienen el pin de VDD en la parte superior y el pin de VSS en la parte inferior, por lo que cuando estas son colocadas entre los *tracks* automáticamente se tiene los pines posicionados sobre el *track* con la *net* correcta. Cuando se da el caso donde el *track* de VSS corresponde a la fila superior y el de VDD a la inferior, ICC2 automáticamente voltea la celda al momento de colocarla, de forma que los pines siempre estén posicionados sobre el *track* apropiado.

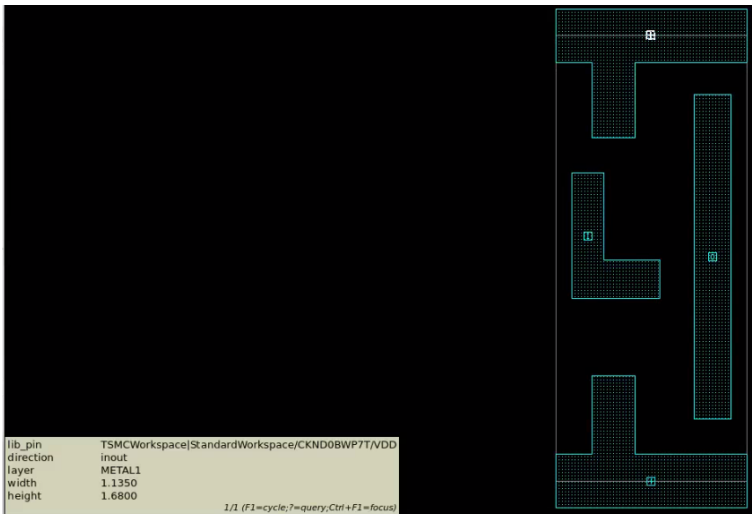


Figura 32: Vista *frame* de una celda estándar en el *Library Manager*

El posicionamiento de estos pines se puede observar abriendo nuevamente la librería en el *Library Manager*, buscando cualquier celda que se desee y accediendo a su vista *frame*. En el caso de una celda estándar se puede esperar algo similar a la Figura 32. En esta se muestra la vista *frame* de la celda "CKND0BWP7T" de las librerías de TSMC y cuyo propósito es el implementar la función de un inversor lógico o NOT. En la figura se encuentra seleccionado el pin para VDD y se puede ver que efecto este se encuentra en la parte superior. Por otro lado, en la parte inferior se tiene el pin de VSS.

Para iniciar con la creación de los *tracks* dentro del *core* se debe crear un nuevo patrón específico para la conexión de celdas estándar. Esto se puede realizar con el siguiente comando:

create_pg_std_cell_conn_pattern *nombre-del-patrón*

El nombre especificado no debe contener espacios y esto es lo único que necesita el comando para crear el patrón. Opcionalmente se puede especificar explícitamente ciertos parámetros como el metal a utilizar o el ancho de los *tracks*, pero estos pueden ser calculados automáticamente por ICC2 utilizando los parámetros de las celdas estándar de la librería. Luego de esto el patrón debe ser asociado a una estrategia, nuevamente utilizando el comando:

set_pg_strategy *nombre-de-la-estrategia* **-core -pattern** {{**pattern:** *nombre-del-patrón*} {**nets:** {*VDD VSS*}}}

Se debe especificar el nombre de la estrategia sin espacios en blanco. Luego se indica el área donde se desea crear el patrón, en este caso es dentro del *core*. Por último, se indica el nombre del patrón a crear y el nombre de las *nets* de poder. La estrategia creada debe ser compilada con el comando **compile_pg** y con esto ya se debería poder ver un patrón en el *layout* como el que se muestra en la Figura 33. En la figura se puede ver que los *tracks* son creados en la capa de metal 1 (metal celeste). Esta coincide con la capa donde se encuentran los pines de las celdas estándar, como se pudo ver en la Figura 32.

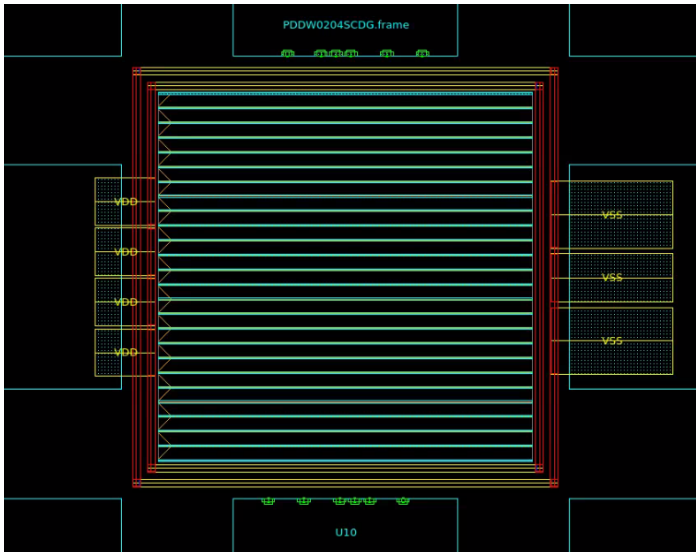


Figura 33: *Tacks* de VDD y VSS en el interior del *core* en ICC2

Como se mencionó al inicio de esta sección, la creación de estos *tracks* era únicamente una parte de la estructura para la alimentación interna del *core*. La segunda parte consiste en la conexión hacia el resto de la red de alimentación del circuito, específicamente con el anillo de poder. Esto ya que el anillo ya se encuentra conectado a los *pads* de VDD y VSS, por lo que al conectar el interior del *core* al anillo se completa la red de alimentación del circuito. Para crear esta conexión se debe generar una vez más un patrón y para este tipo de conexiones el comando que se puede utilizar es el siguiente:

create_pg_mesh_pattern *nombre-del-patrón* **-layers** { {{**horizontal_layer:** *capa-horizontal*} {**width:** *minimum/value_list*} {**pitch:** *número*} {**spacing:** *interleaving/minimum/value_list*}} {{**vertical_layer:** *capa-vertical*} {**width:** *minimum/value_list*} {**pitch:** *número*} {**spacing:** *interleaving/minimum/value_list*}} }

En este comando se debe indicar el nombre del patrón, nuevamente sin que este tenga espacios en blanco. Luego se debe definir la información de los metales a utilizar. Primero se debe dar las especificaciones para el metal horizontal y después para el vertical. El objetivo es que existan varios *tracks* de metal horizontal y vertical de manera que estos formen una malla o *mesh*. Sin embargo, en la Figura 33 se observa que este arreglo de *tracks* horizontales ya fue creado, por lo que únicamente falta el conjunto de *tracks* verticales. Debido a que solamente se desea crear *tracks* en esa dirección, en el comando se puede omitir toda la parte donde se específica los parámetros para el metal horizontal. Por lo tanto, únicamente se debe definir los siguientes parámetros para los *tracks* verticales: capa de metal a utilizar, ancho, *pitch* y espaciado.

La capa de metal puede ser cualquiera que se desee, pero se debe basar esta decisión tomando en cuenta si los nuevos *tracks* pasarán sobre otros ya existentes y en que capa se encuentran estos. El ancho de los *tracks* puede definirse de forma general como "*minimum*" para indicar que se desea utilizar el valor definido en el *tech-file* o puede indicarse una lista de valores numéricos donde cada valor corresponde al ancho individual de cada *track*. Si la lista únicamente tiene un valor, este es utilizado para todos los *tracks*. Esta definición es

opcional y por defecto se utiliza "*minimum*". El valor del *pitch* es obligatorio, este debe ser numérico y es general para todos los *tracks*. Para el espaciado las opciones de "minimum" y lista de valores funcionan de igual manera que con el ancho. Adicionalmente, se tiene la opción de "*interleaving*", donde el espaciado se calcula de manera que todas las distancias entre *tracks* sean iguales. La opción por defecto nuevamente es "*minimum*".

Para crear una estrategia que implemente este patrón exitosamente se puede utilizar el siguiente comando:

set_pg_strategy *nombre-de-la-estrategia* **-polygon** $\{\{X_o\ Y_o\}\ \{X_f\ Y_f\}\}$ **-pattern** {{**pattern:** *nombre-del-patrón*} {**nets:** {*VDD VSS*}}} **-bockage** {*elemento*}

Se debe indicar el nombre (sin espacios) de la nueva estrategia y luego el área donde se creará el patrón. En este caso se tiene una situación un poco distinta a la de las otras estrategias creadas durante el *power planning* y es que esta área no puede ser una ya definida en el diseño. Si se utiliza el área del *core*, los *tracks* verticales serán muy cortos y no se podrá hacer la conexión hacia el anillo. Si se eligiera el área del *die*, estos serían muy largos, lo que puede generar errores en el diseño. Por lo tanto, se debe generar una nueva área que encierre el *core* y cierta parte de los *tracks* horizontales del anillo de poder, un área como la resaltada en la Figura 34. Esto de forma que se pueda hacer la conexión entre *core* y anillo. Para ejecutar esto se puede utilizar la opción de polígono, la cual requiere que se especifiquen dos puntos del área, la posición de la esquina inferior izquierda y la de la esquina superior derecha. Utilizando ambas coordenadas se genera un área rectangular.

Para que el polígono tenga el área que se necesita, los puntos X_o y X_f deben coincidir con los del *core*, mientras que para el cálculo de Y_o y Y_f se puede utilizar las siguientes fórmulas:

$$Y_o = Y_{o_{core}} - (offset_y\ del\ anillo) - 2*(ancho\ de\ los\ tracks\ horizontales\ del\ anillo) - (espaciado\ entre\ tracks\ horizontales\ del\ anillo)$$
$$Y_f = Y_{f_{core}} + (offset_y\ del\ anillo) + 2*(ancho\ de\ los\ tracks\ horizontales\ del\ anillo) + (espaciado\ entre\ tracks\ horizontales\ del\ anillo)$$

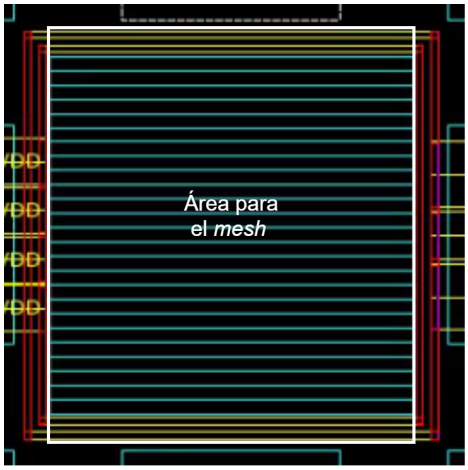


Figura 34: Área que debe encerrar el *mesh* de poder en ICC2

El comando para la creación de la estrategia finaliza cuando se indica el nombre del patrón a utilizar y el nombre de las *nets* de alimentación. Aunque esto no es obligatorio se puede indicar si se debe tomar en cuenta algún tipo de bloqueo. Un bloqueo le indica a la herramienta que no puede generar las estructuras del patrón sobre un área ocupada por un elemento específico al momento de implementar la estrategia. Este elemento puede ser cualquiera que ya se encuentre definido en el *layout* como una *net*, macro, un área definida por un polígono, etc. Como se mencionó, esto es opcional, pero da un mayor control sobre la estrategia y puede ser utilizado para corregir posibles errores en el diseño.

Con todo definido queda utilizar nuevamente el comando **compile_pg** para compilar la estrategia y crear el patrón en el diseño. En la Figura 35 se muestra un ejemplo luego de ejecutar todos los procesos descritos para el diseño de una compuerta NOT. En este caso se utilizó metal 3 (color rojo) para los *tracks* verticales y se definió un ancho de 4.2, *pitch* de 42 y la opción de "*interleaving*" para el espaciado. Cómo se observa en la figura, la creación de estos *tracks* verticales fue exitosa y se puede apreciar cómo, junto a los *tracks* horizontales dentro del *core*, forman la malla de poder.

Es importante mencionar que al momento de compilar una estrategia ICC2 conecta automáticamente dos o más metales que se encuentran sobrepuestos si detecta que se encuentran en distintas capas y pertenecen a la misma *net*. Si por alguna razón esta conexión no se realizará de forma automática esto se puede ejecutar de forma manual con el siguiente comando:

merge_pg_mesh -nets {*VDD VSS*} **-types** {*lista-de-elementos*} **-layers** {*lista-de-capas*}

Como se observa, el comando requiere que se indique el nombre de las *nets* de alimentación y luego el tipo de elementos que se desea buscar para hacer el *merge*. En el caso de las estrategias creadas durante el flujo, la lista de elementos a conectar sería la siguiente: {*stripe ring macro_pin_ connect*}. Por último, se puede indicar en que capas se debe buscar metales

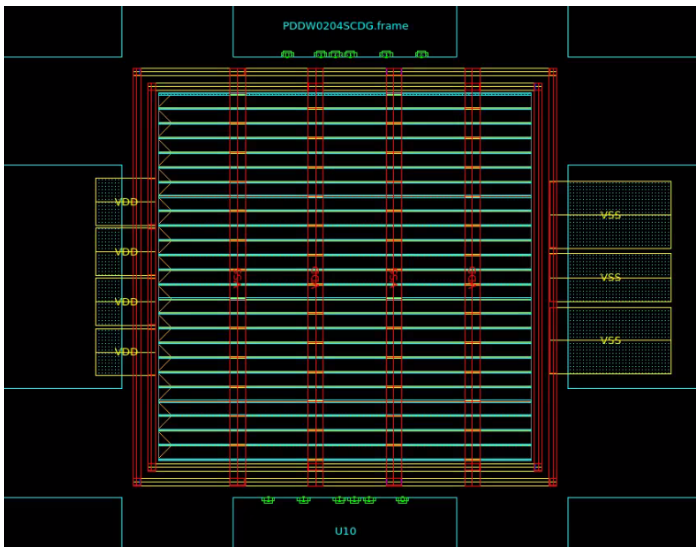


Figura 35: *Mesh* de poder en ICC2

sobrepuestos. En el caso de la Figura 35 se puede ver que las capas que deberían ir indicadas son metal 1 (color celeste), metal 2 (color amarillo) y metal 3 (color rojo), ya que estas fueron las utilizadas para crear los distintos patrones. Si no se desea especificar nada de esto, simplemente se puede indicar el comando sin ninguna opción y se conectarán todos los metales sobrepuestos que pertenecen a una misma *net* de alimentación.

Observando los resultados obtenidos en la Figura 35 se puede dar por finalizado el *power planning* y por lo tanto, la etapa de *floorplan* en el flujo de la síntesis física. Utilizando los resultados obtenidos se puede validar la estructura del flujo propuesto para realizar esta etapa en ICC2. La efectividad del mismo únicamente podrá ser validada si el resto de la síntesis física se ejecuta sin problemas y si los resultados de las distintas verificaciones son exitosos.

7.2.3.4. Definición de *Via-Rules*

Con los resultados obtenidos en las secciones anteriores ya es posible iniciar la siguiente etapa del flujo de síntesis física. Sin embargo, se considera importante mencionar un concepto que no es obligatorio para el proceso de *power planning*, pero se considera útil para tener un mayor control sobre las estructuras creadas durante el mismo. Este concepto son las *via-rules* y es útil ya que como se mencionó anteriormente, al compilar una estrategia ICC2 intenta conectar automáticamente los metales sobrepuestos que pertenecen a una misma *net*. Sin embargo, puede que la forma en la que ICC2 conecta estas estructuras genere problemas en el diseño o simplemente no sea la ideal. En este caso se pueden crear reglas para indicar como deben crearse las vías en los puntos donde estos metales se encuentran sobrepuestos. Se puede definir estas reglas con el siguiente comando:

set_pg_strategy_via_rule *nombre-de-la-regla* **-via_rule** {{{{**strategies:** *nombre-de-la-estrategia*}} {{**existing:** *tipo-de-elemento*} {**layers:** {*lista-de-capas*}}} {**via_master:** *nombre-de-vía*}} {{**intersection:** *all/adjacent/undfined*} {**via_master:** *nombre-de-vía*}}}

En este último comando primero se debe especificar el nombre que se desea asignar a la regla. Luego de esto empieza la definición de la misma. En la configuración utilizada del comando la regla se encuentra dividida en dos partes que se pueden pensar como sub-reglas. La primera sub-regla indica qué debe pasar cuando se tenga una intersección entre elementos que se crean al compilar la estrategia (definida con la opción **strategies**) y los elementos ya existentes en el *layout* (definidos con la opción **existing**). Esta última opción permite especificar una gran variedad de elementos, por ejemplo, uno de los casos discutidos es cuando se desea conectar el anillo de poder a los *pads* IO, entonces se puede colocar el nombre de la estrategia para esta conexión en **strategies** y "ring" en **existing**. Con esto únicamente se crearán vías en las intersecciones entre elementos de los anillos ya existentes y elementos de la estrategia creada para los *pads*.

El grupo de elementos ya existentes se puede filtrar aún más con la opción de **layers**. Con esta se puede especificar que, solamente sean tomados en cuenta para la regla los elementos que formen parte del grupo definido en **existing** y se encuentren en las capas de metal

indicadas en la lista. Si no se desea especificar un grupo de elementos ya existentes se puede utilizar "all" para que todos los elementos del *layout* sean tomados en cuenta, sin embargo, se puede seguir usando la opción de **layers** para filtar todos los elementos por capa.

La primera sub-regla finaliza indicando el nombre de la vía a utilizar cuando una intersección de elementos cumpla con los criterios especificados, esto con la opción de **via_master**. Las vías se encuentran definidas en las librerías del *foundry* y si no se desea utilizar una vía en específico se puede indicar que la vía a utilizar será la "default". Con esto ICC2 automáticamente utilizará la vía que determine como la más apropiada para conectar los elementos. Esto último es lo que hace la herramienta cuando no se especifican *via-rules*.

La segunda sub-regla inicia con la opción de **intesection**. En esta se pueden definir tres métodos, el primero es "all" y se puede utilizar cuando se desea que la regla creada aplique para todas las intersecciones de metal en el *layout*. El segundo es "adjacent" y permite hacer lo mismo pero únicamente se aplicará la regla entre intersecciones de capas de metal adyacentes, por ejemplo, metal 1 y 2, metal 2 y 3, metal 3 y 4, etc. Se debe tomar en cuenta que, cuando se utiliza cualquiera de estos dos métodos, realmente esta sería la única regla ya que cualquier otra sub-regla sería redundante. El método de "undefined" se utiliza cuando se han definido una o varias sub-reglas antes. Esto se puede ver como la regla final o el que debe pasar con las intersecciones del *layout* que no entren en alguno de los criterios definidos en las sub-reglas.

Luego de haber indicado el método de **intesection** se debe indicar la vía a utilizar en la regla. Cuando no se desea utilizar ninguna vía se puede indicar "NIL", una palabra clave para indicar que no se debe crear vías. Este es el caso más común cuando se utiliza el método de "undefined" ya que la idea es que con las sub-reglas previas se haya definido todas las intersecciones que son de interés, por lo que con "NIL" se indica a ICC2 que cualquier otra intersección se debe quedar igual, sin conexión.

7.3. Placement

Una etapa de *floorplan* bien definida y ejecutada proporciona al diseño una gran robustez y permite que los procesos ejecutados en las etapas posteriores del flujo de síntesis física sean puntuales. La primera de estas siguientes etapas es el *placement* de las celdas estándar. Para este punto gran parte de lo que se ha realizado en el diseño gira en torno a cómo se puede transportar señales y energía eficazmente desde el exterior del *die* hacia el interior del *core*. Esto ya que en esta área es donde se encontrarán las celdas estándar, las cuales ejecutan la función lógica designada del circuito. Por lo tanto, si ya se tiene definido cómo llegarán las señales a estas celdas, el siguiente paso es que estas últimas sean posicionadas dentro del *core*. Para realizar el *placement* se puede utilizar el siguiente comando:

create_placement -floorplan -timing_driven -congestion
-effort *very_low/low/medium/high*
-congestion_effort *low/medium/high*

Este último es uno de los comandos más importantes de flujo de síntesis física ya que define donde irán posicionadas las celdas de acuerdo a ciertos parámetros internos y otros especificados por el usuario. Es un comando muy poderoso y cuenta con varias opciones que resultan útiles para mejorar el desempeño del circuito, así como reducir problemas en el diseño. Es importante mencionar que el comportamiento de este comando puede ser modificado de dos maneras distintas, con las opciones propias del comando y con las opciones de aplicación. Las opciones de aplicación que permiten modificar el comportamiento del *placement* son todas las que pertenecen a la categoría "*place*".

El comando inicia indicando que se debe tomar en cuenta las opciones de aplicación relacionadas al *floorplan* que tienen un impacto sobre el *placement*. Estas son las opciones de aplicación que se encuentran en la categoría "*plan*", subcategoría "*place*". La opción **timing_driven** del comando únicamente tiene efecto si se indicó la opción de **floorplan**. Utilizando las dos en conjunto se indica a la herramienta que durante el *placement* debe buscar reducir el *timing*. De igual manera, la opción de **congestion** debe indicarse en conjunto a la de **floorplan** y con esto se indica a la herramienta que también debe buscar reducir la congestión de celdas. Con la opción de **effort** se indica el nivel de esfuerzo de la herramienta al realizar el *placement*. Se debe tomar en cuenta que entre mayor sea el esfuerzo se tendrá mejores resultados, pero este proceso será más tardado y se usarán más recursos del sistema. Por último, la opción de **congestion_effort** funciona de forma similar a la anterior, pero para el esfuerzo que se hará al reducir congestiones. Para modificar el proceso de reducción de *timing* y congestiones se puede modificar los parámetros de las opciones de aplicación "**plan.place.timing_driven_mode**" y "**plan.place.congestion_driven_mode**", respectivamente.

Continuando con el ejemplo de la síntesis física para una compuerta NOT, en la Figura 36 se muestra el resultado luego de haber utilizado el comando **create_placement**. Se puede ver que las tres celdas estándar importadas durante la lectura del Verilog (Fig. 24b) han sido colocadas dentro del *core*. Sin embargo es importante notar que, a pesar que ya se encuentran dentro del *core*, las celdas no están alineadas con las filas y por lo tanto con los *tracks* internos de alimentación. Para que estas sean alineadas se debe utilizar el comando:

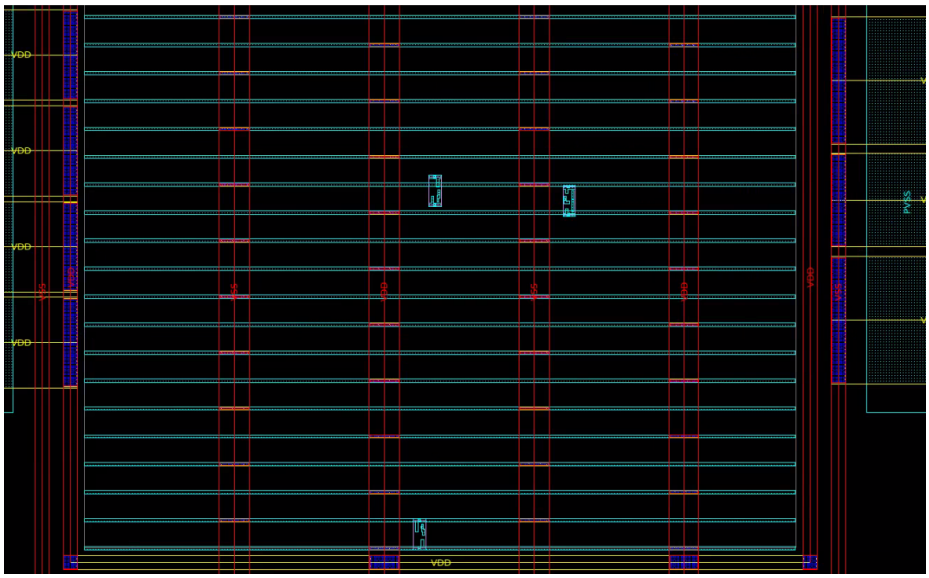


Figura 36: *Placement* de celdas estándar en ICC2

legalize_placement

En la Figura 37 se muestra el resultado de utilizar el comando sobre el diseño de la NOT. Cómo se puede ver, este comando se encarga de alinear las celdas a los *tracks* y de voltearlas (en caso de ser necesario) para que los pines de VDD y VSS de la celda se encuentren sobre el *track* correcto. Esto último se puede visualizar de mejor manera en la Figura 38. En la Figura 38a se puede ver cómo se encuentra una celda luego del comando **create_placement**. Esta se encuentra dentro del *core*, pero no se encuentra alineada dentro de las filas. Lo que es peor, su pin de VDD se encuentra sobre el *track* de VSS y el pin de VSS sobre el *track* de VDD, en efecto creando un cortocircuito que seguramente dejaría la celda inutilizable. Luego

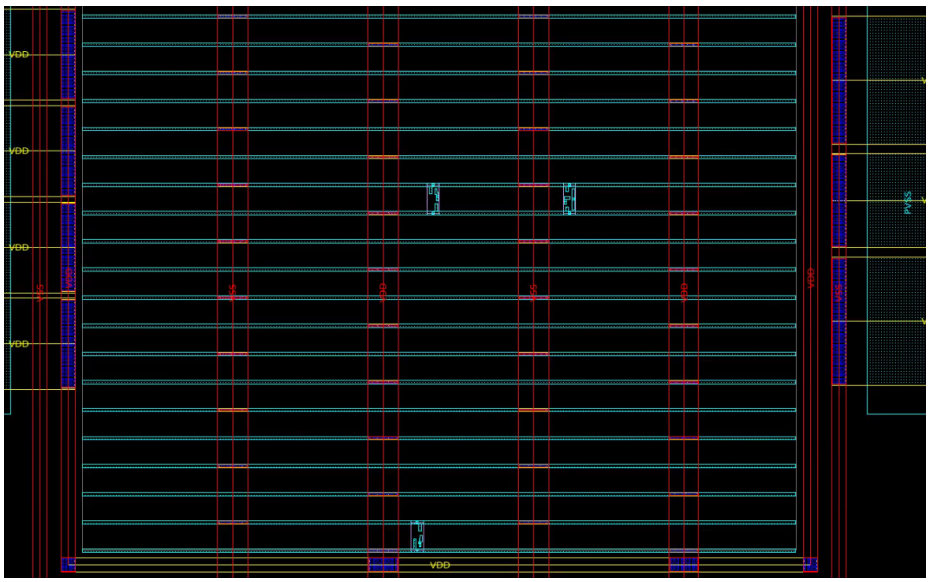
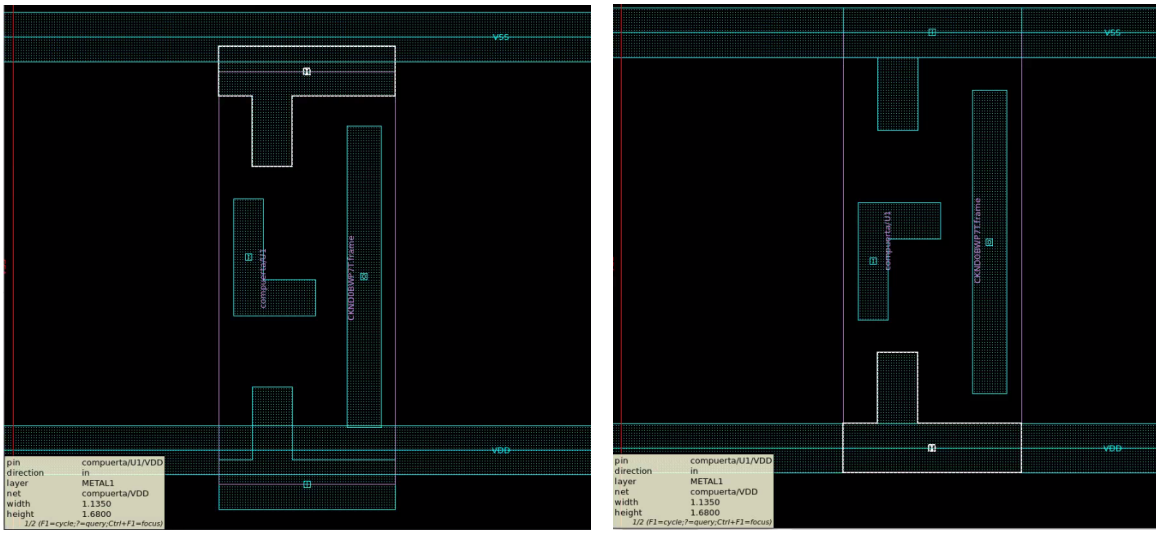


Figura 37: *Placement* y legalización de celdas estándar en ICC2



(a) Celda posicionada en dentro del *core*

(b) Celda legalizada dentro del *core*

Figura 38: *Placement* vs Legalización de una celda estándar en ICC2

de ejecutar el comando **legalize_placement** estos problemas son corregidos. Se puede ver cómo dentro de las filas se posiciona la celda de forma exacta y en este caso es girada de manera que su pines de alimentación se encuentren sobre el *track* correcto.

A pesar que los pines ya se encuentran sobre el *track* correcto estos elementos aún no se encuentran conectados, únicamente están sobrepuestos. Para que ICC2 realice esta conexión se puede utilizar el siguiente comando:

connect_pg_net -automatic

Utilizando este último comando la herramienta resuelve automáticamente la relación entre las *nets* y conecta los pines de alimentación de las celdas al *track* correspondiente. Con estos resultados se puede dar por finalizada la etapa del *placement*, por lo que se puede continuar con el siguiente paso del flujo de síntesis física.

7.4. Enrutamiento

Una vez se tiene las celdas posicionadas dentro del *core* y conectadas a las *nets* de poder, el siguiente paso es realizar las conexiones que se utilizarán para transportar las *nets* de señalización en el *layout*. Estas señales se introducen al circuito por medio de los *pads* IO para luego ser transportadas al interior del *core* y ser conectadas a las celdas estándar. Este es el último paso para tener el circuito totalmente conectado.

Previo a realizar el enrutamiento, ICC2 permite verificar el estado actual del diseño para poder determinar si existen errores que puedan dar problemas durante esta etapa. Esto se puede realizar con el siguiente comando:

check_design -checks pre_route_stage

El comando comprueba varios factores en el diseño, por ejemplo, la existencia de un *miss-match* entre las *nets*. Esto último sucede cuando se comprueba que una *net* se encuentra definida físicamente pero no lógica. Con el comando también se verifica posibles violaciones de *timing*, si el diseño se encuentra preparado para el enrutamiento, la capacidad que se tiene de crear rutas óptimas con el diseño actual, etc. El resultado de ejecutar este comando sobre el diseño del NOT (luego de la etapa de *placement*) se muestra en la Figura 39. En este caso no se detectaron errores en el diseño, aunque se tienen algunos *warnings* que por el momento pueden ser ignorados.

```
check_design -checks pre_route_stage ****************************************
 Report : check_design
 Options: { pre_route_stage }
 Design : Not_IO
 Version: R-2020.09-SP3
 Date   : Sat Sep  4 23:42:59 2021
****************************************

Running mega-check 'pre_route_stage':
    Running atomic-check 'design_mismatch'
    Running atomic-check 'scan_chain'
    Running atomic-check 'mv_design'
    Running atomic-check 'timing'
    Running atomic-check 'routability'
    Running atomic-check 'hier_pre_route'

  *** EMS Message summary ***
  ------------------------------------------------------------------------------------------------------
  Rule         Type   Count      Message
  ------------------------------------------------------------------------------------------------------
  DFT-011      Info   1          The design has no scan chain defined in the scandef.
  MV-208       Warn   1          Pin(s) '%portStr' of library cell '%module' is(are) missing the ...
  TCK-001      Warn   1          The reported endpoint '%endpoint' is unconstrained. Reason: '%re...
  ------------------------------------------------------------------------------------------------------
  Total 3 EMS messages : 0 errors, 2 warnings, 1 info.
  ------------------------------------------------------------------------------------------------------

  *** Non-EMS message summary ***
  ------------------------------------------------------------------------------------------------------
  Rule         Type   Count      Message
  ------------------------------------------------------------------------------------------------------
  ZRT-026             2          Layer %s pitch %.3f may be too small: wire/via-down %.3f, wire/v...
  ZRT-027             1          Ignore %d top cell ports with no pins.
  ------------------------------------------------------------------------------------------------------
  Total 3 non-EMS messages : 0 errors, 3 warnings, 0 info.
  ------------------------------------------------------------------------------------------------------

Warning: EMS database "check_design.ems" already exists, over-writing it. (EMS-040)

Information: EMS database is saved to file 'check_design.ems'.
Information: Non-EMS messages are saved into file 'check_design2021Sep04234259.log'.
1
```

Figura 39: Verificación del diseño previo a la etapa de enrutamiento en ICC2

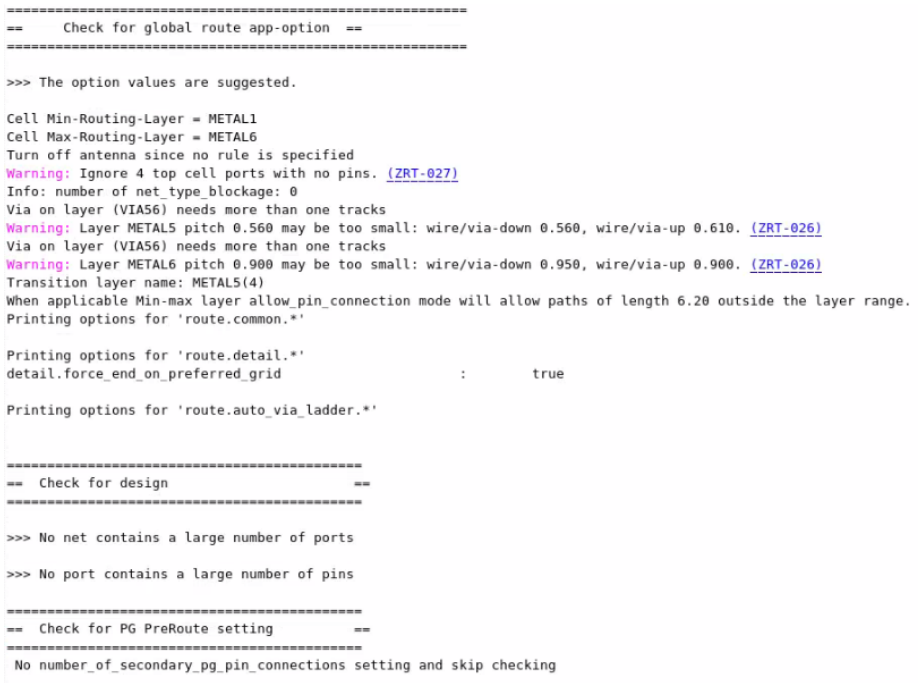


Figura 40: Verificación de *routability* previo a la etapa de enrutamiento en ICC2

Si se desea validar exclusivamente la capacidad que se tiene de producir un buen enrutamiento en el diseño actual, se puede utilizar el comando **check_routability**. Con este último la herramienta genera un reporte donde se evalúan varios factores que puedan afectar negativamente el desempeño, tanto de la herramienta como del diseño final. Un ejemplo de este reporte se muestra en la Figura 40. La figura muestra únicamente una parte del reporte generado ya que este es bastante extenso. Un reporte sin problemas permite verificar que el diseño se encuentra listo para la etapa de enrutamiento.

Se debe tomar en cuenta que estas verificaciones son opcionales ya que realmente no tienen un impacto durante la ejecución del flujo. Sin embargo, permiten validar si el flujo propuesto es efectivo. Para realizar el enrutamiento se puede ejecutar el siguiente comando:

route_auto

Aunque este último comando parece simple, realiza tres funciones de suma importancia. La primera es el *global routing* y es donde se calculan y generan todas las rutas para la conexión de *nets* que se encuentran desconectadas. La siguiente función es la de *track assignment* y es donde a cada *track* creado en la función anterior se le asigna la *net* que debe transportar. Por último se tiene la función de *detail routing* y es la que se encarga de optimizar las rutas por medio de iteraciones, buscando eliminar cualquier error de diseño. Por defecto la cantidad de iteraciones realizadas es 40 y este valor puede cambiarse al momento de ingresar el comando si se especifica la opción de **-max_detail_route** seguido del número de iteraciones que se desea. La cantidad de iteraciones puede ir desde 1 hasta 1000.

En la Figura 41 se muestra el resultado de ejecutar este comando en el diseño de la

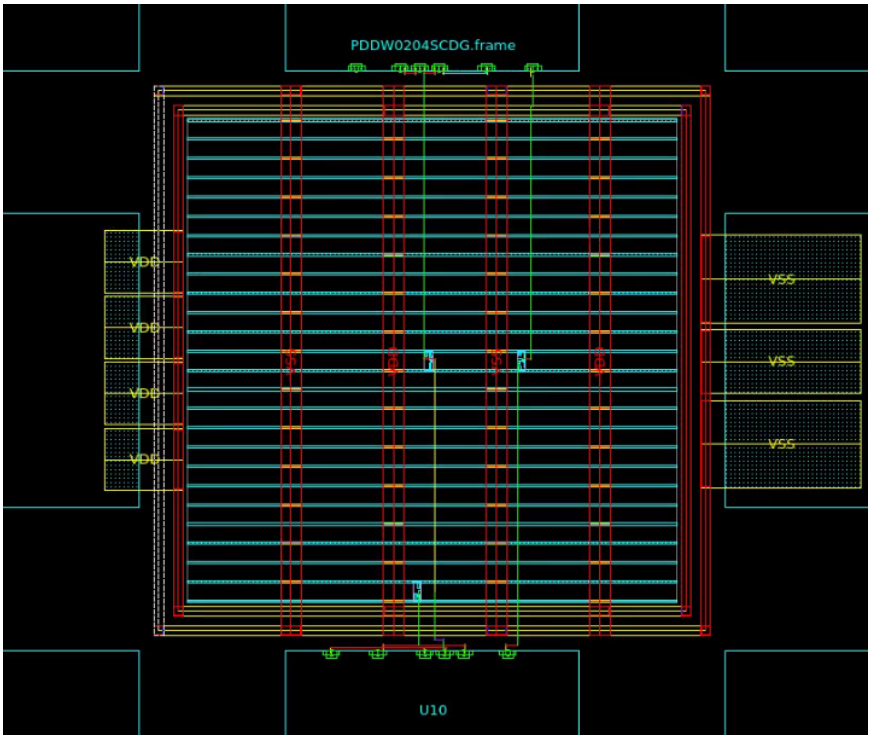


Figura 41: Enrutamiento de celdas en ICC2

NOT. Cómo se puede observar las celdas ya se encuentran conectadas a los *pads* IO. También es importante notar que los pines lógicos de estos *pads* no se encuentran conectados directamente a VDD y VSS, sino que a las celdas "TIEHBWP7T" y "TIELBWP7T" que, tal como se mencionó en la sección 7.2.1, realizan la traducción de valores de voltaje a un valor lógico de uno o cero. Estos resultados permiten validar y dar por terminada la etapa de enrutamiento.

7.4.1. Sintetización de relojes

Hasta el momento todos los procesos comentados permiten la creación de diseños que implementan lógica combinacional. El flujo propuesto funciona de igual manera para circuitos secuenciales, sin embargo, estos últimos requieren llevar a cabo un proceso previo al enrutamiento el cual se conoce como la sintetización de relojes. En la elaboración de circuitos secuenciales la integridad de la señal de reloj es esencial para el funcionamiento correcto del mismo. Por lo tanto, la *net* del reloj no puede ser tratada como una *net* de señalización normal ya que muchos de los elementos del circuito dependerán de ella. Esto significa que se debe planear cuidadosamente cómo será distribuida esta señal en todo el circuito, de forma que todos estos elementos tengan fácil acceso a ella y al mismo tiempo, evitando la congestión de *nets* en un área para reducir el riesgo de generar errores en el diseño durante el enrutamiento. Para realizar la sintetización en ICC2 se puede utilizar el siguiente comando:

synthesize_clock_trees -clocks *Nombre-net-de-reloj*

Este comando genera y optimiza el árbol para el reloj indicado. Se dice que es un árbol porque el reloj ingresa al circuito mediante el pin designado y luego se ramifica hacia los distintos puntos del circuito. El árbol debe ser optimizado para mantener la integridad del reloj y reducir el *skew* en la señal. Esto último resulta importante para mantener la sincronía en todo el circuito. Seguido de este comando se puede ejecutar:

clock_opt

Con este último se optimiza todos los relojes definidos en el diseño y se realiza el enrutamiento para cada uno. Luego de esto optimiza las conexiones creadas para reducir el *timing* y la potencia e intenta corregir cualquier error de DRC que se pudo haber generado durante el ruteo del reloj. Con la señal de reloj sintetizada se puede proseguir con el flujo y llevar a cabo el enrutamiento del resto de *nets* del circuito, tal como se describió anteriormente.

7.5. Inserción de *fillers*

Con todo lo realizado hasta el momento queda una última etapa en el flujo, la cual consiste en realizar las preparaciones finales para que el diseño pueda ser fabricado. Para esto se debe insertar las celdas *filler* tanto en anillo IO como en el *core*. Por otro lado, es necesario insertar *fillers* de metal en cada capa hasta lograr cumplir con el requerimiento de densidad impuesto por el *foundry*. Estos procesos se definen a continuación.

7.5.1. Celdas *filler*

Cómo se mencionó en secciones anteriores, el propósito de estas celdas es el de tener un *layout* donde la densidad de celdas sea uniforme. Esto es necesario ya que parte de las reglas de diseño de los *foundries* incluye tener cierto porcentaje de utilización de área. Si estas no se cumplen puede que durante la verificación de DRC se obtenga errores de baja utilización o errores de espaciado debido a que las celdas se encuentran muy separadas unas de otras.

Una solución a estos problemas podría ser reducir el tamaño del *die* de forma que este sea exacto para la cantidad de celdas que tiene el diseño. Sin embargo, al juntar demasiado las celdas se corre el riesgo de crear congestiones durante el enrutamiento, al no haber suficiente espacio para crear conexiones óptimas. Para evitar esto se puede colocar estos *fillers* que, cómo su nombre lo dice, cumplen el propósito de rellenar los espacios vacíos de forma que el diseño no tenga violaciones a estos requerimientos de utilización. Existen dos tipos de celdas *filler*: IO y estándar.

7.5.1.1. *Fillers* IO

Para importar estas celdas y posicionarlas en el diseño se puede utilizar el siguiente comando:

create_io_filler_cells -io_guides [get_io_guides {*nombre-del-anillo-IO.lados*}**]**
-reference_cells {*lista de celdas filler IO*}

En este último comando se debe indicar primero sobre que lados del anillo IO se desea colocar estos *fillers*. Como se mencionó anteriormente, un anillo IO tiene cuatro lados que son identificados por sus nombres en inglés *left*, *right*, *top* y *bottom*. Por lo tanto, si se desea insertar *fillers* en todo el anillo, en el comando se debe indicar una lista donde se encuentren los cuatro lados. Por ejemplo, si el anillo IO se llamase IO_R la lista que a indicar debe ser: {*IO_R.left IO_R.right IO_R.top IO_R.bottom*}. Luego de esta lista se debe indicar propiamente que celdas son las que deben colocarse y para esto se debe identificar en la CLIB cuál es el nombre de las celdas cuyo propósito es el de funcionar como IO *fillers*. En el caso de la librería de 180 nm de TSMC, las celdas que se puede utilizar son las presentadas en el Cuadro 7.

Cuadro 7: Celdas *filler* IO en la librería de TSMC de 180 nm

Nombre de la celda
PFILLER1
PFILLER5
PFILLER05
PFILLER0005
PFILLER10
PFILLER20

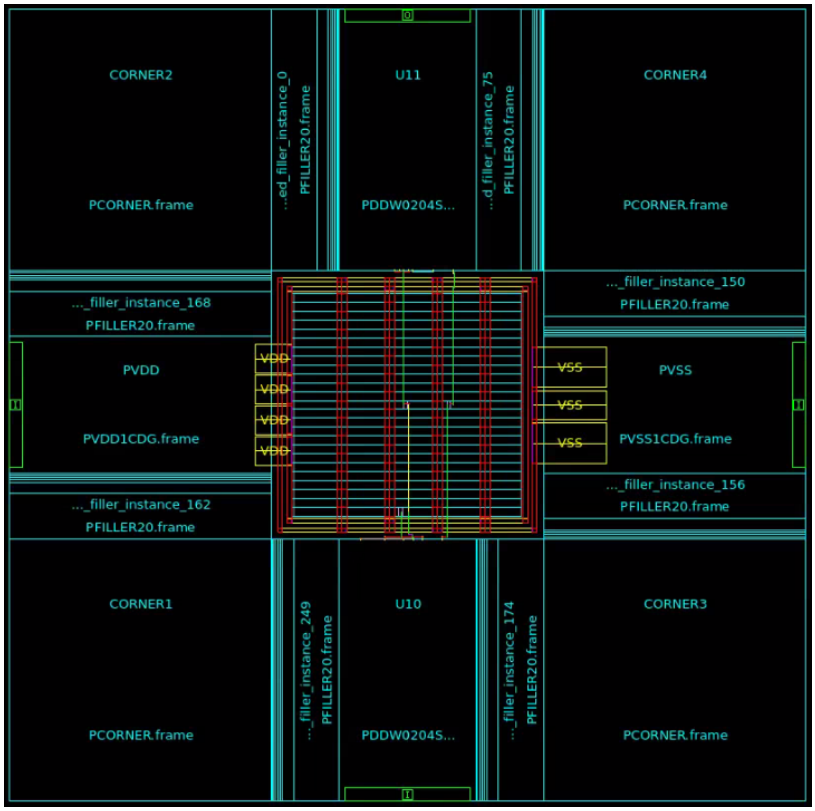


Figura 42: Inserción de celdas *filler* para el anillo IO en ICC2

Se recomienda incluir todas las celdas ya que estas tienen distintos tamaños que permiten a ICC2 rellenar más eficientemente los espacios sin utilizar en el anillo IO. Luego de ejecutar el comando se debería poder observar que estas celdas han sido insertadas en el diseño en las posiciones indicadas. En la Figura 42 se muestra el resultado luego de insertar estas celdas sobre el diseño de la compuerta NOT que se ha utilizado de ejemplo a lo largo del trabajo. Cómo se puede observar, ICC2 ha rellenado automáticamente los espacios libres en el anillo y para esto ajusta utilizando todas las celdas de referencia que fueron indicadas de forma que los espacios libres sean mínimos, tal como se muestra en la Figura 43.

7.5.1.2. *Fillers* estándar

Este tipo de *fillers* es utilizado dentro del *core* y para su inserción se puede utilizar el siguiente comando:

create_stdcell_fillers -lib_cells [get_lib_cells {*lista de celdas filler std*}**]**

Al ser celdas estándar ICC2 automáticamente las colocará dentro del *core*. En este caso solo se debe indicar el nombre de las celdas que se desea utilizar. Nuevamente este debe ser el nombre de la celda dentro de la CLIB y en el caso de la librería de TSMC de 180 nm se puede indicar las celdas de el Cuadro 8.

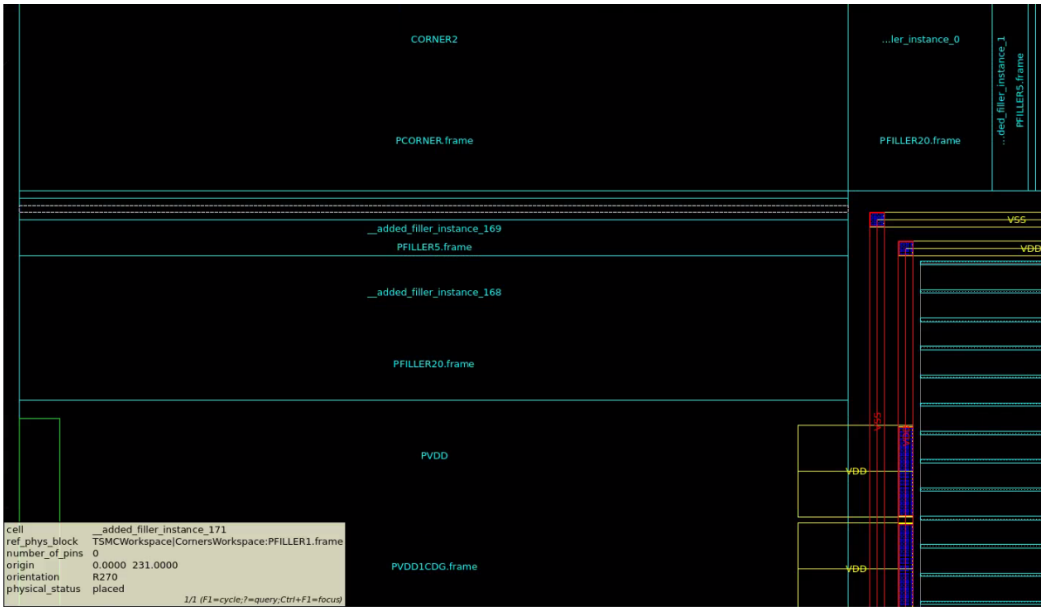


Figura 43: Distintos tamaños de celdas *fillers* colcadas en el anillio IO en ICC2

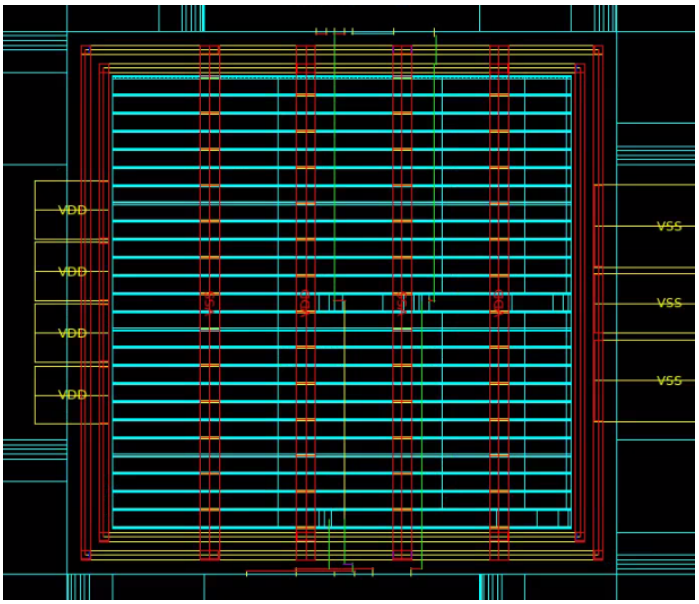
Cuadro 8: Celdas *filler* estándar en la librería de TSMC de 180 nm

Nombre de la celda
FILL64BWP7T
FILL32BWP7T
FILL16BWP7T
FILL8BWP7T
FILL4BWP7T
FILL2BWP7T
FILL1BWP7T

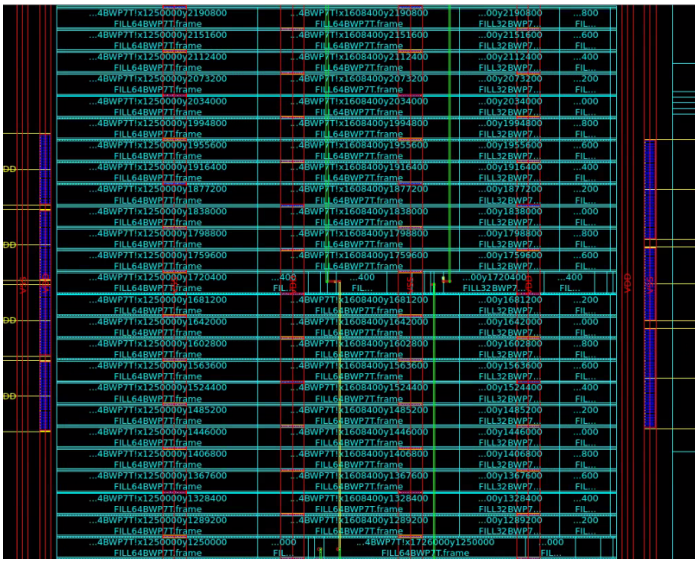
Luego de haber importado los *fillers* estos deben ser conectados a los *tracks* de alimentación. Esto se debe hacer no porque estas celdas requieran ser alimentadas, si no debido a que al igual que una celda normal, estas se ordenarán dentro de las filas del *core*. Esto implica que también serán posicionadas sobre los *tracks* de alimentación y pueden generar errores de *nets* flotantes. Para realizar la conexión se puede utilizar nuevamente el comando:

connect_pg_net -automatic

Como se mencionó, este comando resuelve automáticamente las relaciones entre los distintos elementos que deben ir conectados a las *nets* de alimentación y realiza las conexiones pertinentes. Al ejecutar el comando para la inserción de estas celdas puede que parezca que ICC2 no insertó nada dependiendo del nivel de zoom que se tenga en la herramienta, como se puede ver en la Figura 44a. Sin embargo, si se acerca suficiente al *core* se puede ver que estas celdas sí están presentes tal como se muestra en la Figura 44b. Observando esta última se puede ver que ahora toda el área del *core* está siendo utilizada, al haber rellenado todas



(a)



(b)

Figura 44: Inserción de celdas *filler* estándar en ICC2

las filas con estas celdas. Al igual que en el caso de los *fillers* para el anillo IO, se puede ver que se utilizan todas las celdas de referencia que fueron indicadas, ya que ICC2 aprovecha sus distintos tamaños para ajustar y rellenar de la mejor manera cualquier espacio dentro del *core* que no esté siendo utilizado.

7.5.2. Metal *fillers*

Los *foundries* establecen reglas de densidad que indican cuanto porcentaje de utilización debe haber en cada capa de metal para lograr una metalización uniforme y evitar problemas

durante el proceso de fabricación. Para esto se pueden crear polígonos de metal que cubran los espacios vacíos en cada capa hasta lograr la densidad requerida. En el caso de la librería de TSMC utilizada estas reglas se encuentran definidas en el *technology file*, como se muestra en la Figura 45. En este caso la densidad mínima requerida por el *foundry* es de 30 % en todas las capas de metal.

```
DensityRule     {
                layer                   = "METAL1"
                minDensity              = 30
}

DensityRule     {
                layer                   = "METAL2"
                minDensity              = 30
}

DensityRule     {
                layer                   = "METAL3"
                minDensity              = 30
}

DensityRule     {
                layer                   = "METAL4"
                minDensity              = 30
}

DensityRule     {
                layer                   = "METAL5"
                minDensity              = 30
}

DensityRule     {
                layer                   = "METAL6"
                minDensity              = 30
}
```

Figura 45: Reglas de densidad establecidas por TSMC

A diferencia de la inserción de celdas *filler*, los *fillers* de metal son añadidos al diseño mediante un *runset* que se encarga de realizar este proceso. Uno de los problemas que surgieron al realizar esta etapa fue que no se contaba con este *runset* entre los archivos de TSMC. Al encontrar este inconveniente se contactó a la organización IMEC, la cual es patrocinadora de este proyecto y ha apoyado brindando todos los archivos necesarios para su ejecución. Sin embargo, no se obtuvo la respuesta esperada por parte de ellos y no se logró obtener este *runset*, por lo que este proceso del flujo de diseño no pudo ser llevado a cabo.

Si se tuviese el *runset* el proceso para la inserción de los *fillers* puede ser ejecutado en dos pasos, primero se debe referenciar el archivo en ICC2 y posterior a esto realizar la inserción como tal. Esto se puede realizar con los siguientes comandos:

set_app_options -name signoff.create_metal_fill.runset -value *Runset.rs*

signoff_create_metal_fill

Luego de ejecutar los comandos anteriores se puede dar por finalizado el proceso de inserción de *fillers* y por lo tanto, la síntesis física. El último paso es guardar el bloque, lo que se puede ejecutar con el comando:

save_block *Nombre_librería.ndm:Nombre_bloque*

7.6. Validación de la metodología

Una vez se tuvo definida toda la metodología para el flujo era necesario validarla para comprobar que esta fuera lo suficientemente robusta y permitiera llevar a cabo el proceso de síntesis física para cualquier circuito que se solicitará. Esto significa que se debían seguir exactamente todos los pasos anteriormente descritos para el *floorplan*, *placement*, enrutamiento e inserción de *fillers* y poder obtener un diseño sin errores durante la ejecución del flujo. Es importante mencionar que los procesos para la preparación de librerías y creación de la CLIB del *foundry* solo deben ser ejecutados una vez. Esto ya que con la librería creada simplemente debe ser referenciada para poder utilizar las celdas del *foundry* en los diseños que se desee.

Para facilitar el proceso de validación de la metodología fue necesario generar un *script* donde se encuentran los comandos de ICC2 correspondientes a cada etapa del flujo. Esto permite que para el proceso de validación solo fuera necesario cambiar el archivo *Verilog* donde se encuentra descrito el circuito que se desea sintetizar y cambiar el tamaño del *die* para que tuviera un tamaño adecuado según el número de *pads* en el diseño. Para esto último se debía alterar los parámetros del comando utilizado para crear el *floorplan*:

initialize_floorplan -control_type *die/core* **-site_def** *nombre-del-site*
-use_site_row -keep_all -side_length {*X Y*} **-core_offset** {*número*}

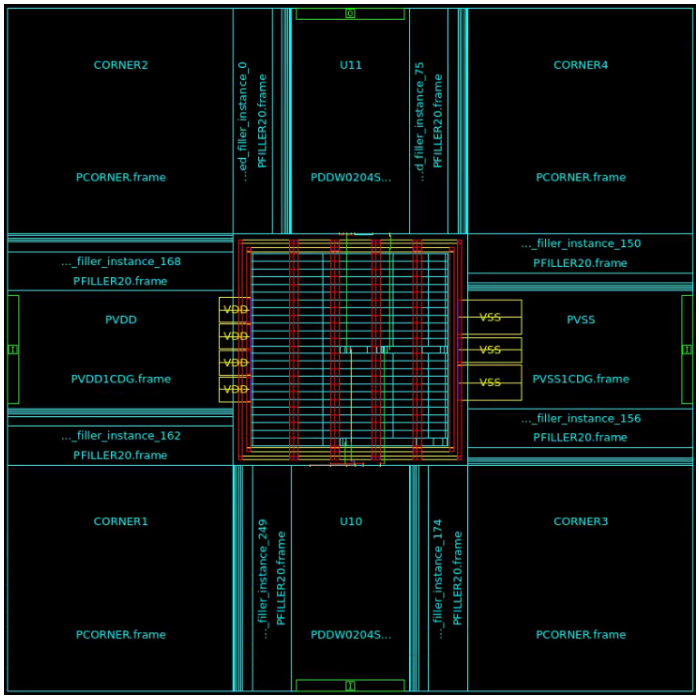
De acuerdo a lo comentado en secciones anteriores, en todos los diseños se indicó la opción de "*core*" para el argumento **control_type** y "*unit*" para el argumento **site_def**. Los parámetros que sí se alteraron entre diseños fueron los del **side_length**. Estos fueron determinados de acuerdo al número de *pads* en el diseño, de forma que el *die* fuera lo suficientemente grande para contenerlos. Los parámetros utilizados por cada diseño sintetizado durante la validación de la metodología se muestran en el Cuadro 9. En todos los casos se utilzó un **core_offset** de 125, ya que el largo de los *pads* es 115 μm y de esta forman queda un espaciado de 10 μm entre el *core* y los *pads* donde se posicionará el anillo de poder. El *script* se puede encontrar en los anexos de este trabajo. En las siguientes secciones se muestran los resultados de aplicar el flujo propuesto para la síntesis de distintos circuitos de prueba y finalmente del Chip UVG que se planea fabricar con TSMC. En todos los casos se logró obtener resultados exitosos, validando el funcionamiento correcto del flujo.

Cuadro 9: Parámetros utilizados durante la validación de la metodología

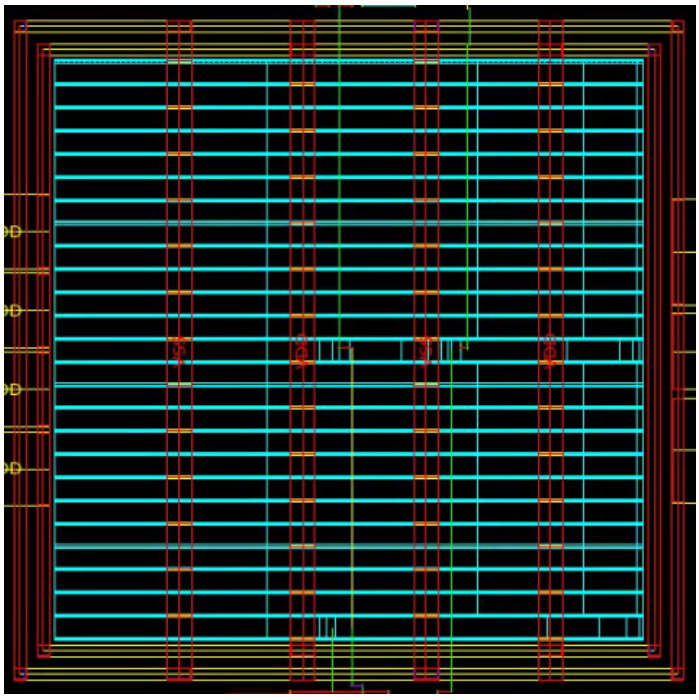
Circuito	***Número de pads***	**side_length (μm)**	**core_offset (μm)**	**Área total core (mm^2)**	**Área total die (mm^2)**
NOT	4	{100 100}	125	0.01	0.1225
XOR	8	{135 135}	125	0.018225	0.140625
Full Adder	16	{285 285}	125	0.081225	0.286225
ALU (4b)	22	{350 350}	125	0.1225	0.286225
Contador (4b)	8	{150 150}	125	0.0225	0.16
RAM (5b)	17	{350 350}	125	0.1225	0.286225
Chip UVG	16	{285 285}	125	0.081225	0.286225

7.6.1. Circuitos combinacionales

7.6.1.1. Compuerta NOT



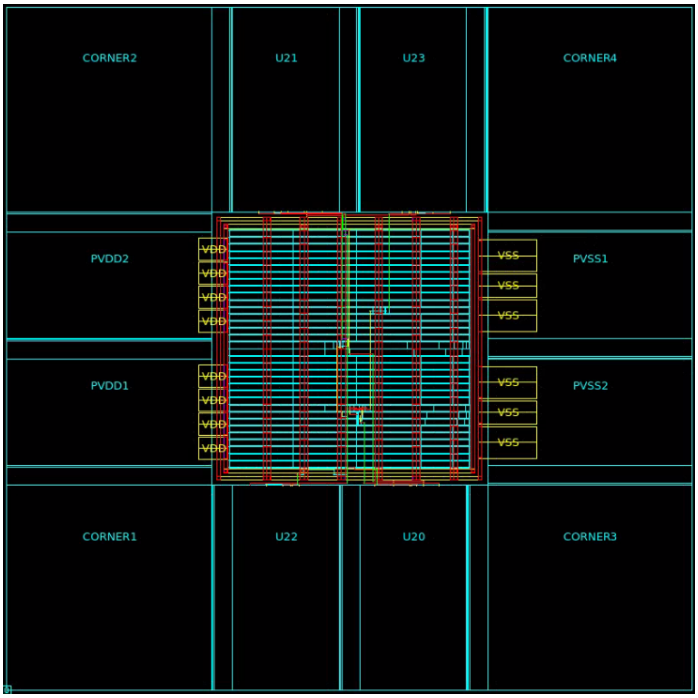
(a) *Die*



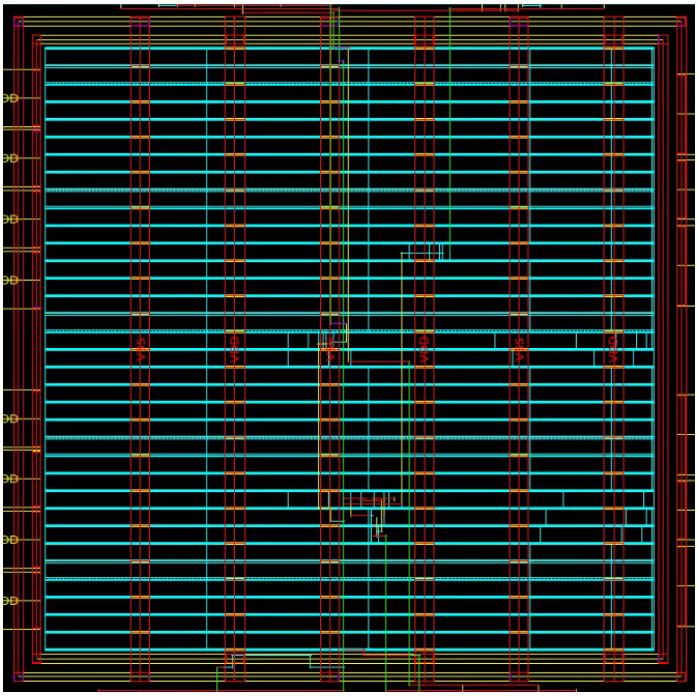
(b) *Core*

Figura 46: Síntesis física de una compuerta NOT en ICC2

7.6.1.2. Compuerta XOR



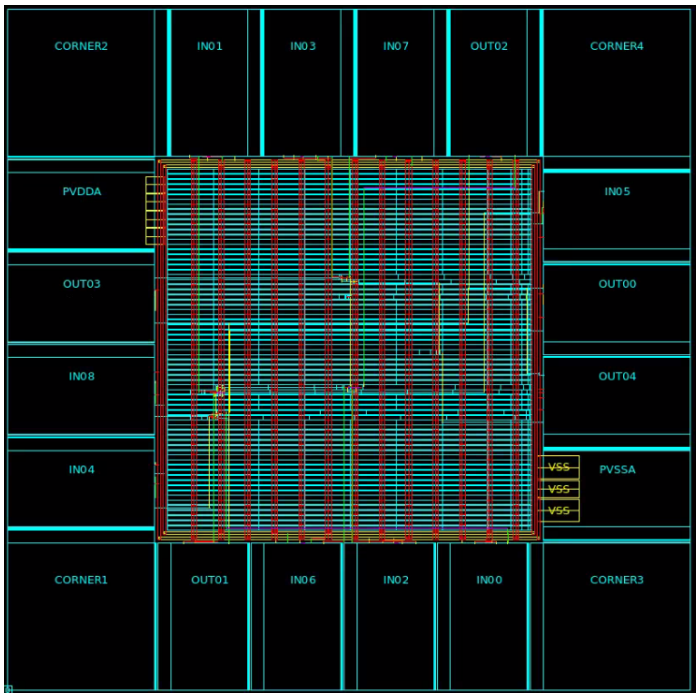
(a) *Die*



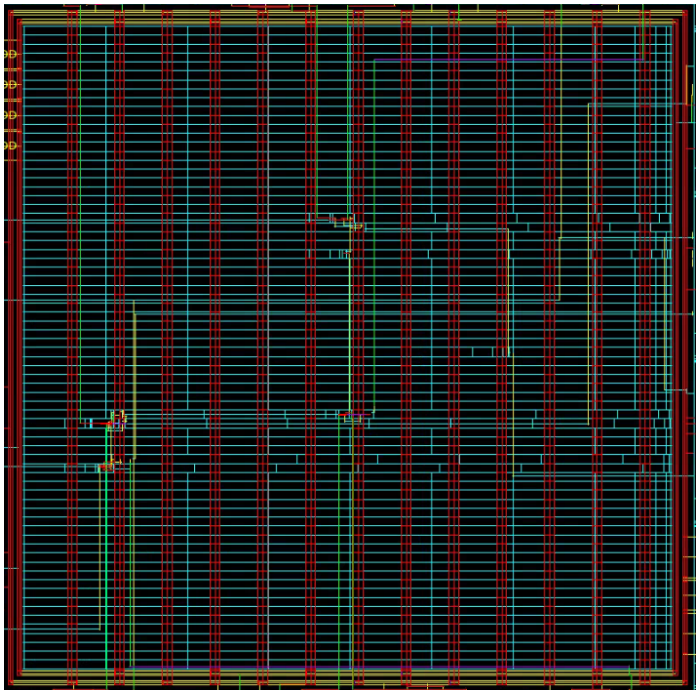
(b) *Core*

Figura 47: Síntesis física de una compuerta XOR en ICC2

7.6.1.3. Circuito *Full Adder*



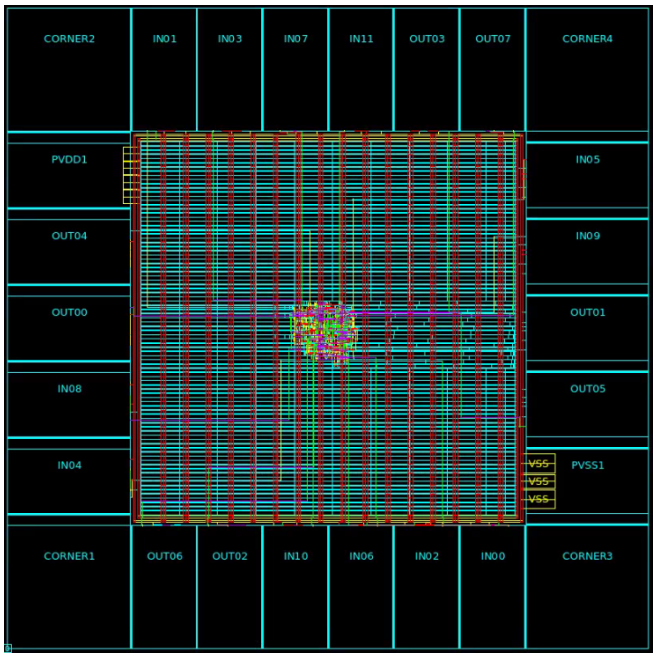
(a) *Die*



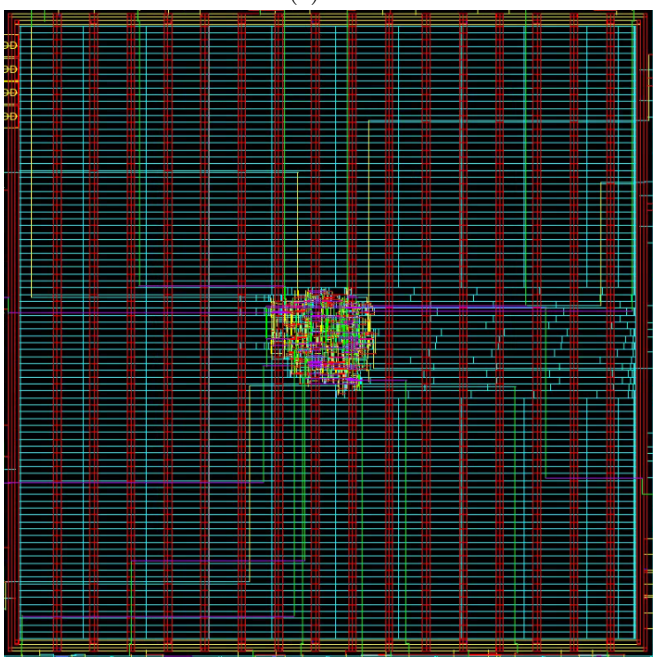
(b) *Core*

Figura 48: Síntesis física de un *Full Adder* en ICC2

7.6.1.4. ALU de 4 bits



(a) *Die*



(b) *Core*

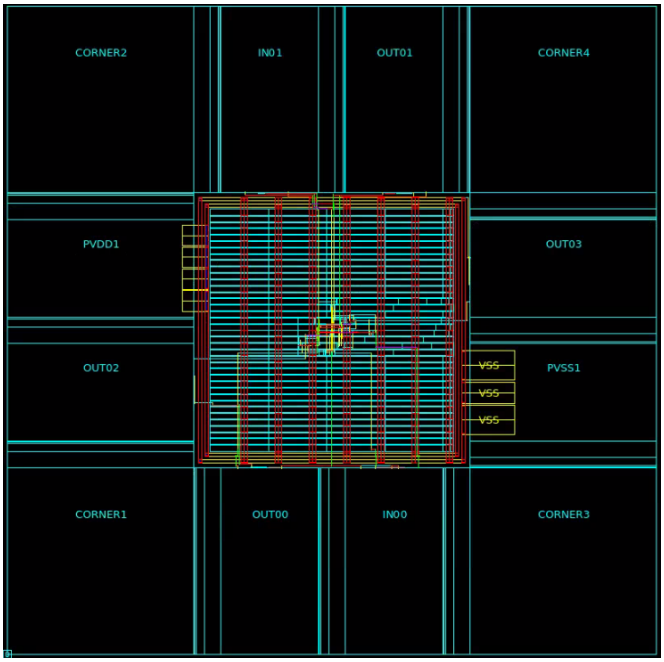
Figura 49: Síntesis física de una ALU en ICC2

7.6.2. Circuitos secuenciales

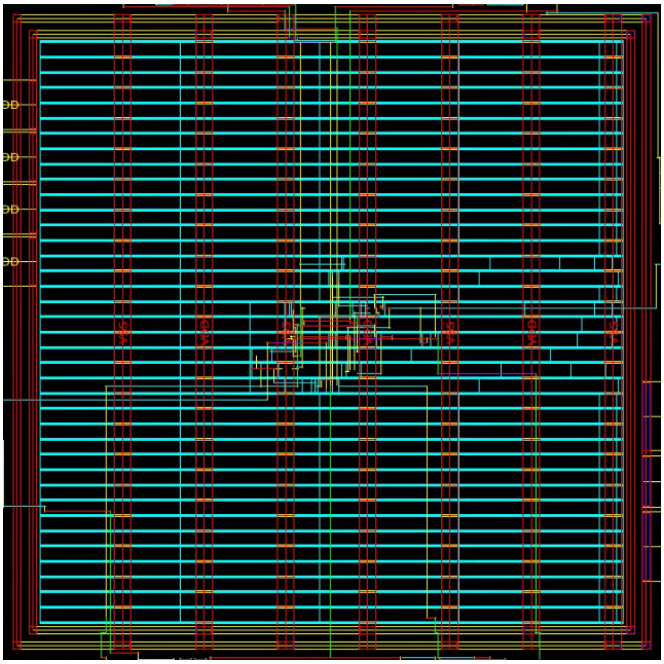
Para el caso de los circuitos secuenciales se realizó un cambio adicional y es que se agregó al *script* los comandos mencionados en la Sección 7.4.1 para la sintetización de relojes previo

al enrutamiento. El resto del flujo se ejecutó de la misma manera.

7.6.2.1. Contador de 4 bits



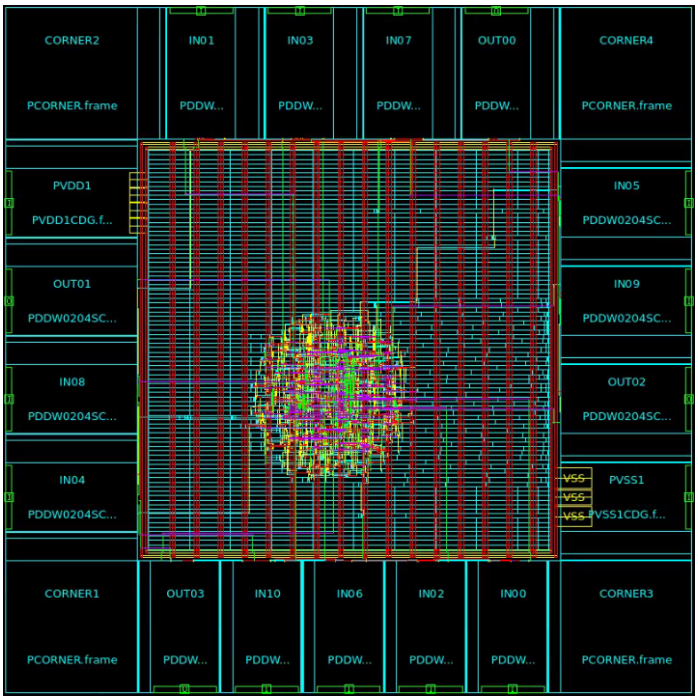
(a) *Die*



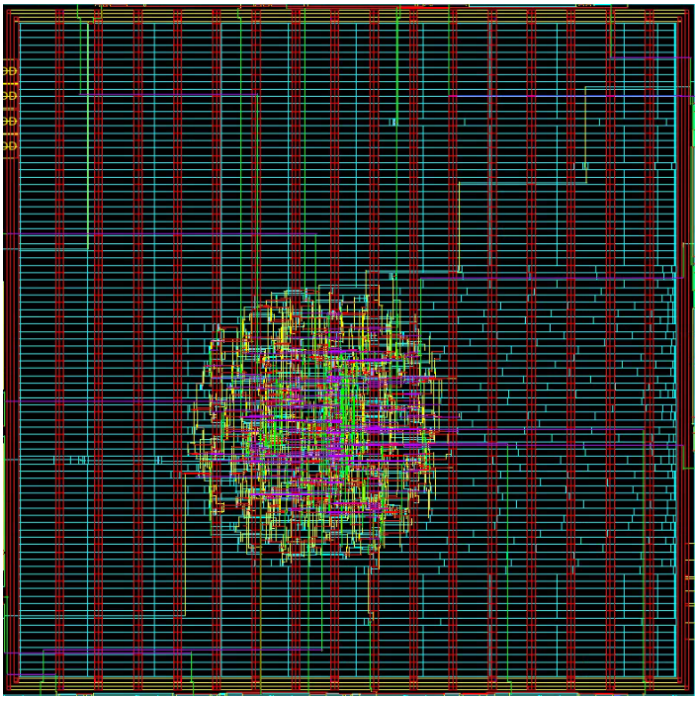
(b) *Core*

Figura 50: Síntesis física de un Contador en ICC2

7.6.2.2. RAM de 5 bits



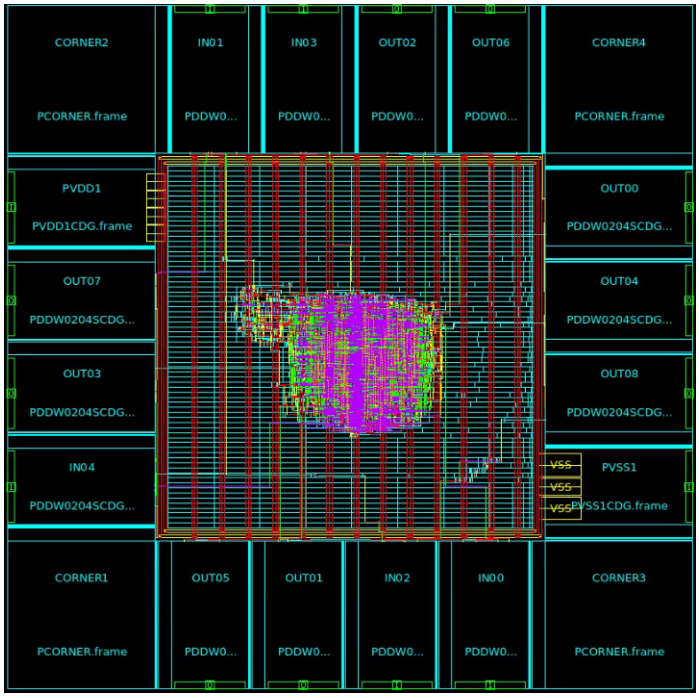
(a) *Die*



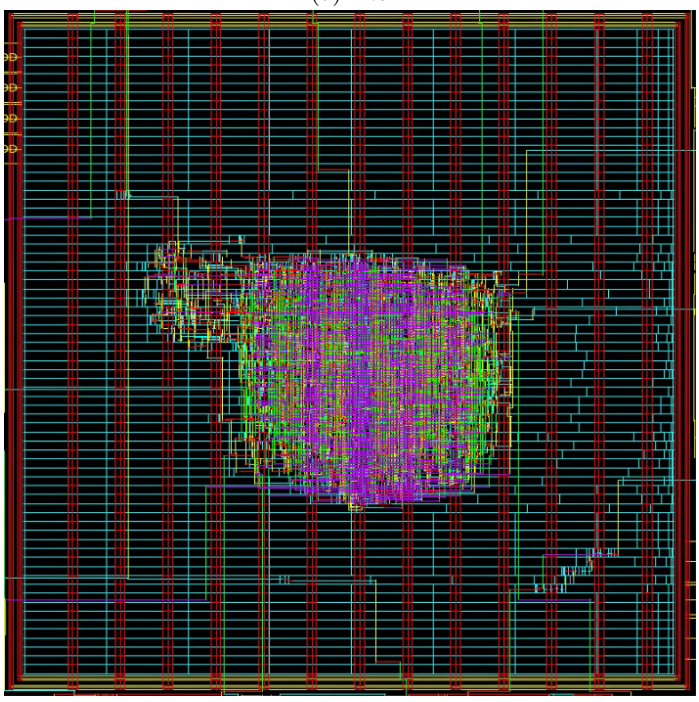
(b) *Core*

Figura 51: Síntesis física de una RAM en ICC2

7.6.2.3. Chip UVG



(a) *Die*



(b) *Core*

Figura 52: Síntesis física del Chip UVG en ICC2

7.7. Generación de archivo GDSII

Con el diseño sintetizado y aprobado el siguiente paso es su exportación a un archivo en formato *Graphic Design System II* (GDSII). Este archivo es sobre el cual se llevan a cabo las verificaciones de LVS, ERC y Antena debido a que es el que debe ser enviado al *foundry* para proseguir con la fabricación del *chip*. El archivo puede ser generado desde ICC2 con el siguiente comando:

write_gds -library *Nombre_librería.ndm* **-design** *Nombre_del_bloque* **-view design -hierarchy all -lib_cell_view frame** *Diseño.gds*

Para ejecutar este comando primero se debe indicar el nombre de la librería de diseño y seguido de esto el nombre del bloque dentro de esta librería que se desea exportar. Luego se debe definir la vista de este bloque desde la cual se desea generar el archivo. Utilizando el flujo propuesto la única vista que se tendrá disponible es la de "*design*" ya que esta es la que crea por defecto ICC2 al realizar la lectura del archivo Verilog. Luego se debe indicar que partes de la jerarquía de la librería de diseño se desea exportar. Esta jerarquía se puede visualizar la Figura 53, donde se muestra la librería que contiene el diseño de la compuerta NOT sintetizada. En la figura se puede ver que la librería contiene tanto los diseños generados por el usuario como las librerías de referencia generadas durante la etapa de preparación de librerías.

Según lo que se especifique con el argumento **hierarchy** se puede generar un archivo GDSII solamente con el bloque principal (*NOT_IO* en la figura), con todos los bloques creados por el usuario en la librería de diseño (si es que hubiera más de uno) o con todo el contenido de la librería de diseño, incluyendo las librerías de referencia. Esta última opción es la que debe ser indicada para exportar correctamente los diseños generados con el flujo propuesto, principalmente porque de lo contrario no se exportaría ninguna de las celdas pertenecientes a librerías de referencia que hayan sido utilizadas en el diseño.

La Figura 54 muestra las advertencias indicadas por ICC2 sí se exporta el diseño sin incluir las librerías de referencia. En estas advertencias la herramienta indica que el bloque que se está exportando contiene celdas que no se encuentran en la librería de diseño o en el nivel de jerarquía especificado. Inspeccionando el nombre de las celdas que generan la advertencia se puede ver claramente que estas son las celdas de las librerías de TSMC, por

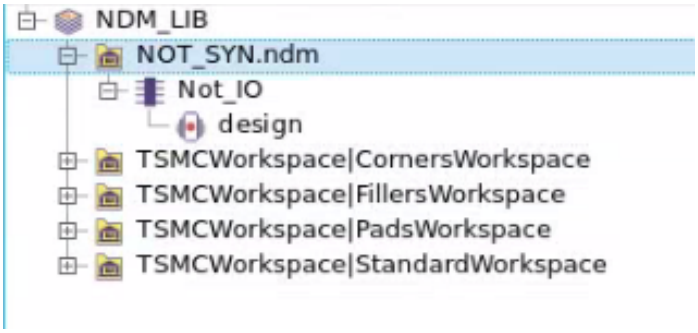


Figura 53: Jerarquía de la librería de diseño

```
Warning: The user specified view ' design frame layout' for lib-cell 'CKND0BWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PDDW0204SCDG' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'TIELBWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'TIEHBWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PCORNER' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PVDD1CDG' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PVSS1CDG' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PFILLER20' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PFILLER5' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PFILLER1' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PFILLER05' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'PFILLER0005' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'FILL64BWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'FILL16BWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'FILL2BWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'FILL8BWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'FILL4BWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'FILL1BWP7T' is not available. (GDS-034)
Warning: The user specified view ' design frame layout' for lib-cell 'FILL32BWP7T' is not available. (GDS-034)
1
```

Figura 54: Problemas de generación del archivo GDSII

lo que se trata de un problema en el nivel de jerarquía indicado. Para solucionarlo se debe indicar la opción de "*all*" en el argumento de la jerarquía. Aunque estas advertencias no impiden que el archivo GDSII sea generado, el exportar el diseño sin las celdas significa que el diseño en el GDSII no es una representación válida del diseño generado durante la síntesis física. Por lo tanto no aprobará ninguna de las verificaciones que se deben realizar previo a enviar los archivos al *foundry*.

Para finalizar la exportación del diseño el comando también requiere que se especifique la vista a exportar de las celdas utilizadas en el diseño. Para el flujo propuesto esta vista siempre debe ser la de *frame*. Esto ya que según lo indicado en la Sección 7.1.1, esta es la vista que se crea automáticamente al momento de crear la CLIB con las librerías del *foundry*. Por último se debe indicar el nombre que tendrá el archivo GDSII, el cual puede ser cualquiera que el usuario desee. Con este archivo GDSII generado se puede dar por finalizado el flujo de trabajo propuesto para la ejecución de una de síntesis física completa en la herramienta *IC Compiler 2*.

CAPÍTULO 8

Verificación de antena

Que un circuito haya podido ser traducido exitosamente a un *layout* de silicio no significa que este diseño generado es fabricable. Para verificar esto se debe comprobar que el diseño cumple con las reglas impuestas por el fabricante. Esto se logra cuando se aprueba las verificaciones de DRC, LVS, ERC y Antena, siendo esta última la de interés en este trabajo. Durante la verificación se hace una serie de pruebas al circuito por medio de un *script*, comúnmente conocido como *runset*, y el cual debe ser brindado por el *foundry*. Este *script* evalúa distintos aspectos del diseño y comprueba que en este último no haya violaciones a las reglas de antena.

La verificación debe llevarse a cabo sobre el archivo GDS II generado al finalizar la síntesis física, dado que este es el que debe ser entregado al *foundry* y contiene el diseño final que se pretende fabricar. Es importante tomar esto en cuenta ya que el *foundry* realizará sobre el diseño las mismas pruebas por su cuenta y si determinan que este contiene violaciones será rechazado, solicitando que se realicen las correcciones necesarias. Por lo tanto, se debe buscar que el diseño se encuentre libre de errores previo a ser enviado al *foundry*, de forma que se pueda agilizar el proceso de fabricación.

Para llevar a cabo la verificación de antena en el ambiente de trabajo de *Synopsys* deben realizarse dos pasos. Primero es necesario definir las propiedades y reglas de antena, lo que debe hacerse previo a la etapa de enrutamiento. Cuando en esta última etapa se tiene los parámetros definidos ICC2 realizará el enrutamiento tomando en cuenta dichas reglas y automáticamente evitará crear violaciones a las mismas. Esto reduce considerablemente las violaciones a las reglas en el diseño final. El segundo paso es la verificación como tal y para esto se puede utilizar la herramienta *IC Validator*, de aquí en adelante referenciada como ICV. Esta requiere únicamente dos parámetros, el archivo GDS II y el *runset* del *foundry*. Este proceso puede ser ejecutado con un solo comando y al finalizar se generan los archivos necesarios para confirmar que el diseño haya aprobado la verificación de antena.

8.1. Definición de propiedades de antena de las celdas

Para que ICC2 evite crear violaciones a las reglas de antena durante el enrutamiento primero debe conocerlas. Estas reglas deben ser definidas en dos pasos, primero se definen las propiedades de antena de las celdas y luego las reglas como tal. Las propiedades de antena de las celdas pueden definirse con dos métodos distintos: definición en librería física (**.lef**) y definición en CLIB (**.ndm**). En las siguientes secciones se detalla cómo pueden ejecutarse ambos métodos utilizando los archivos de TSMC para un proceso de 180 nm y 6 capas de metal.

8.1.1. Propiedades de antena en la librería física

De acuerdo a lo comentado anteriormente, la librería física (**.lef**) contiene la descripción física tanto de las celdas como de los metales y vías utilizados para interconectarlas. Por lo tanto, es razonable pensar que en este archivo se pueda definir las propiedades de antena al ser una verificación únicamente sobre la estructura física del diseño, no lógica. Cómo se observa dada su extensión, estos archivos utilizan el lenguaje *Library Exchange Format* (LEF), el cual es un formato abierto creado por la empresa *Cadence Design Systems*. El estudio de este lenguaje no se encuentra dentro del alcance de este trabajo, sin embargo, para entender cómo son definidas las propiedades de antena en estos archivos se considera importante mencionar dos conceptos básicos del mismo:

- ***Statement***: Permiten indicar que tipo de objeto físico se empezará a definir. Estos objetos pueden ser capas de metal, vías, macros (celdas), etc.
- ***Keywords***: Permiten definir las propiedades físicas que tendrá un objeto creado mediante un *statement*.

En el Ejemplo 8.1 se muestra cómo se encuentra definida la capa de metal 1 en la librería física de TSMC. El *statment* "LAYER" indica que se estará definiendo una capa de metal cuyo nombre será "METAL1". Dentro de este *statment* encontramos sus respectivas *keywords*. Se puede pensar que estas funcionan como variables que permiten indicar los parámetros físicos del objeto. En el ejemplo se puede ver la definición de parámetros como "PITCH", "SPACING", "THICKNESS", etc.

Ejemplo 8.1: Definición de METAL1 en la librería física de TSMC

```
LAYER METAL1
    TYPE ROUTING ;
    DIRECTION HORIZONTAL ;
    OFFSET       0.280 ;
    PITCH        0.560 ;
    WIDTH        0.230 ;
    SPACING      0.230 ;
    AREA         0.202 ;
    THICKNESS    0.530 ;
    HEIGHT       1.100 ;
    MINIMUMDENSITY       30 ;
    SPACING 0.600 RANGE 10.001 200.0 ;

    ANTENNASIDEAREARATIO        400.000000 ;
    ANTENNADIFFSIDEAREARATIO PWL ( ( 0 400 ) ( 0.2029 400 ) ( 0.203 2281.2 ) ( 0.5
        2400 ) ( 1 2600 ) ( 1.5 2800 ) ) ;
```

```
    ACCURRENTDENSITY    AVERAGE
    FREQUENCY   500 ;
        WIDTH           0.230   1.000 ;
        TABLEENTRIES    1.000   1.000 ;
    ACCURRENTDENSITY    RMS
    FREQUENCY   500 ;
        WIDTH           0.230   1.000 ;
        TABLEENTRIES    8.000   8.000 ;
    ACCURRENTDENSITY    PEAK
    FREQUENCY   500 ;
        WIDTH           0.230   1.000 ;
        TABLEENTRIES    28.284   28.284 ;
    DCCURRENTDENSITY    AVERAGE
        WIDTH           0.230   1.000 ;
        TABLEENTRIES    1.000   1.000 ;

    RESISTANCE RPERSQ 0.0780000000 ;
    CAPACITANCE CPERSQDIST 0.0000473913 ;
    EDGECAPACITANCE 0.0000640000 ;
END METAL1
```

Cómo se ve en el Cuadro 10, existen varias *keywords* que permiten establecer las propiedades de antena. Durante la verificación se revisa que la relación de antena se cumpla para todas las conexiones de metal hacia *gate*. Esto significa que sí para todas las conexiones se cumple la ecuación 1, el diseño aprobará la verificación de antena.

$$\text{Relación de antena máxima} > \text{Relación de antena del } \textit{input} \text{ pin} \tag{1}$$

$$\text{Relación de antena del } \textit{input} \text{ pin} = \frac{\text{Área metal}}{\text{Área } \textit{gate}} \tag{2}$$

Observando nuevamente el cuadro se logra ver que no solo se definen estas propiedades de

Cuadro 10: *Keyowrds* para definir las reglas de antena en un archivo LEF

Tipo de regla	***Keyword***
Área del *gate* o difusiones	ANTENNAGATEAREA ANTENNADIFFAREA
Factor de multiplicación para el área	ANTENNAAREAFACTOR ANTENNASIDEAREAFACTOR
Relación de antena	ANTENNAAREARATIO ANTENNASIDEAREARATIO ANTENNADIFFAREARATIO ANTENNADIFFSIDEAREARATIO ANTENNACUMAREARATIO ANTENNACUMSIDEAREARATIO ANTENNACUMDIFFAREARATIO ANTENNACUMDIFFSIDEAREARATIO

antena para el *gate*, sino que también para las difusiones, en específico la terminal de *drain*. Esto se debe a que para transmitir señales usualmente la terminal del *drain* se conecta al *gate* de otro grupo de transistores. Por ser una difusión, esta terminal actúa como un diodo el cual puede brindar cierta protección al *gate* en caso el metal que los conecta se llegara a cargar a un voltaje suficiente para causar la ruptura en el diodo y permitir el flujo de corriente. De esta forma descargando el metal a través del diodo y protegiendo al resto del circuito al evitar una descarga a través del *gate* que deje inoperable al transistor. En este caso se utiliza el área de la difusión para determinar que tanta protección puede brindar este diodo. El valor de la relación máxima permitida varía según este nivel de protección. En secciones posteriores se explicará cómo se utiliza este valor al momento de realizar las validaciones a las reglas de antena.

El valor utilizado para el área del *gate* es definido en el *statement* de cada celda. En el Ejemplo 8.2 se muestra como está definida la celda "CKND0BWP7T" en lenguaje LEF. Esta es la celda para implementar una compuerta NOT que fue utilizada durante el ejemplo del flujo de síntesis física en el capítulo anterior. La figura muestra la definición de dos parámetros de interés, en el pin de salida "ZN" se define el área de la difusión y en el pin de entrada "I" el área del *gate*.

Ejemplo 8.2: Definición de una compuerta NOT en la librería física de TSMC

```
MACRO CKND0BWP7T
    CLASS CORE ;
    FOREIGN CKND0BWP7T 0.000 0.000  ;
    ORIGIN 0.000 0.000 ;
    SIZE 1.680 BY 3.920 ;
    SYMMETRY x y ;
    SITE core7T ;
    PIN ZN
        ANTENNADIFFAREA 0.8215 ;
        DIRECTION OUTPUT ;
        PORT
        LAYER METAL1 ;
        RECT  1.215 0.545 1.540 3.405 ;
        END
    END ZN
    PIN I
        ANTENNAGATEAREA 0.2790 ;
        DIRECTION INPUT ;
        PORT
        LAYER METAL1 ;
        RECT  0.420 1.605 0.915 1.945 ;
        RECT  0.140 1.605 0.420 2.710 ;
        END
    END I
    PIN VSS
        DIRECTION INOUT ;
        USE ground ;
        SHAPE ABUTMENT ;
        PORT
        LAYER METAL1 ;
        RECT  0.700 -0.235 1.680 0.235 ;
        RECT  0.320 -0.235 0.700 0.925 ;
        RECT  0.000 -0.235 0.320 0.235 ;
        END
    END VSS
    PIN VDD
        DIRECTION INOUT ;
        USE power ;
        SHAPE ABUTMENT ;
        PORT
        LAYER METAL1 ;
        RECT  0.700 3.685 1.680 4.155 ;
        RECT  0.320 3.020 0.700 4.155 ;
        RECT  0.000 3.685 0.320 4.155 ;
        END
    END VDD
END CKND0BWP7T
```

(a) Área superior del *track*

(b) Área lateral del *track*

Figura 55: Áreas de metal a utilizar durante la verificación de antena [15]

Con esto se define únicamente una parte de las propiedades, el valor de área de *gate* a utilizar en los cálculos. El valor del área de metal no puede definirse como un valor fijo ya que este no será determinado hasta el momento de realizar el enrutamiento. El área de metal indica que tanta capacidad para recolectar cargas tiene un *track* y entre mayor sea el área, mayor será la carga acumulada. Debido a que un *track* es un objeto tridimensional el área del metal se puede calcular de dos formas distintas. La primera es utilizando el área de la cara superior del *track*, como se muestra en la Figura 55a. La segunda es utilizando el área total de las caras laterales, tal como se muestra en la Figura 55b.

El área de metal a utilizar en los cálculos es definida en la declaración de la regla para la relación de antena (Cuadro 10), si se utiliza "ANTENNAAREARATIO" se especifica que el área a utilizar en los cálculos es la superior. Por otro lado, utilizando "ANTENNASIDEAREARATIO" se indica que se debe utilizar el área lateral. Para ilustrar mejor esto se puede observar nuevamente la descripción de la capa de metal 1 mostrada en el Ejemplo 8.1 y se puede notar que se define que el área a utilizar es la lateral y el valor máximo de la relación es 400. Por lo tanto, si se tuviera una conexión de metal 1 hacia la compuerta NOT definida en el Ejemplo 8.2, el área máxima permitida para el *track* de metal es la siguiente:

$$400 = \frac{\text{Área lateral máxima de metal}}{0.2790}$$
$$\text{Área lateral máxima de metal} = 111.6\mu\text{m}^2 \tag{3}$$

Durante el proceso de enrutamiento ICC2 evalúa constantemente que la relación se cumpla de forma que no se generen violaciones a las reglas de antena. Ya en su implementación el cálculo es distinto al realizado anteriormente, primero se crea la conexión y se obtiene la relación área de metal/área del *gate* y luego se verifica que está no sea mayor a la relación máxima establecida por las reglas de antena. Si lo es, intenta corregir el error realizando un salto intermedio hacia otro metal o colocando un diodo de descarga.

Es importante mencionar que al encontrarse definidas en la librería física no es necesario indicar nuevamente las propiedades de antena durante el proceso. Por lo que si se trabaja con la librería física de 180 nm de TSMC realmente no es necesario que el usuario defina estas propiedades manualmente ya que son importadas junto con la librería física. Si este no

fuera el caso se puede utilizar el segundo método para la definición de dichas propiedades.

8.1.2. Propiedades de antena en CLIB

Este método para definir las propiedades de antena de las celdas se puede utilizar cuando ya se tiene creada la CLIB, pero las propiedades aún no se encuentran definidas. Para el flujo propuesto en la Figura 19 las propiedades de antena deben ser definidas únicamente en la CLIB que contiene las celdas estándar y esto debe hacerse previo a la creación de la librería unificada. Si en el *Library Manager* no se tiene abierta la librería para su edición se puede utilizar el siguiente comando:

create_workspace -flow edit *Nombre_CLIB_std_cells.ndm*

Con la librería abierta el siguiente paso es importar las propiedades de antena por medio de la lectura de un archivo tipo *Cell Library Format* o **.clf**. Esto puede hacerse con el siguiente comando:

read_clf_antenna_properties *Nombre-del-archivo.clf*

En las librerías de TSMC este archivo está identificado con el siguiente nombre: "*antenna_tcb018gbwp7t.clf*". En el Ejemplo 8.3 se muestra cómo se encuentran definidas las propiedades para dos celdas, una compuerta AND y la compuerta NOT "CKND0BWP7T" que se ha estado revisando durante los ejemplos.

El comando **defineDiodeProtecion** es utilizado para definir el área de la difusión, funcionando de manera similar al *keyword* ANTENNADIFFAREA en el lenguaje LEF. De igual manera se puede hacer la relación entre el comando **defineGateSize** y el *keyword* ANTENNAGATEAREA. Esto puede ser validado comparando los valores de las propiedades de la celda "CKND0BWP7T" mostrados en los ejemplos 8.2 y 8.3 para ambos métodos y donde se puede notar que los valores definidos son iguales.

Es importante notar que a diferencia de las definiciones realizadas en la librería física, en este archivo únicamente se definen las propiedades de antena para las celdas, no se hacen definiciones para los metales o vías. Esto no presenta un inconveniente ya que al momento de definir las reglas de antena estas propiedades son establecidas nuevamente, tal como se explicará en la siguiente sección.

Ejemplo 8.3: Definición de propiedades de antena en el archivo *Cell Library Format* de TSMC

```
#######################################################
TSMC Library/IP Product
Filename: antenna_tcb018gbwp7t.clf
Technology: CL018G
Product Type: Standard Cell
Product Name: tcb018gbwp7t;Version: 270a
#######################################################

defineDiodeProtection "AN2D0BWP7T" "Z"  (0.5688)
defineGateSize        "AN2D0BWP7T" "A2" (0.2133)
```

```
defineGateSize         "AN2D0BWP7T" "A1" (0.2133)

defineDiodeProtection "CKND0BWP7T" "ZN" (0.8215)
defineGateSize         "CKND0BWP7T" "I"  (0.2790)
```

8.2. Definición de reglas de antena

Con los parámetros de antena definidos se puede proceder formalmente a establecer las reglas de antena. Este proceso puede realizarse en cualquier punto del flujo de diseño siempre y cuando sea antes de realizar el enrutamiento. Esto ya que mientras ICC2 realiza este último proceso verifica que las conexiones creadas no presenten violaciones a las reglas de antena establecidas. Para definir las reglas de antena en ICC2 se puede utilizar el siguiente comando:

source Reglas_Antena.tcl

Estas reglas se encuentran definidas en un *script* **.tcl** brindando por TSMC, el cual tiene por nombre: "*antennaRule_018_6lm.tcl*". Las definiciones principales para las reglas aplicadas a los metales se muestran en el Ejemplo 8.4.

Ejemplo 8.4: Definición de reglas de antena para metales en el *script* de TSMC

```
# Single-layer drawn ratio of a metal sidewall area to the active
# poly gate area
# M1~M6 : ratio=400
# M1~M5 : ratio=diode_area*400+2200      if diode_area >=  0.203
# M6    : ratio=diode_area*8000+30000    if diode_area >=  0.203


set_parameter -name doAntennaConx -value 4
set lib [current_lib];
remove_antenna_rules -library $lib

set topMetalLayer 6;

##### Single metal layer sidewall area rule #####
define_antenna_rule $lib \
  -mode 4 \
  -diode_mode 4 \
  -metal_ratio 400 \
  -cut_ratio 0

define_antenna_layer_rule $lib \
            -mode 4 \
            -layer "METAL$topMetalLayer" \
            -ratio 400 \
            -diode_ratio {0.203 0 8000 30000}

define_antenna_layer_rule $lib \
            -mode 4 \
            -layer "METAL5" \
            -ratio 400 \
            -diode_ratio {0.203 0 400 2200}
```

Es importante notar que la migración de este proceso de la herramienta ICC hacia ICC2 fue bastante sencilla. Las propiedades de antena ya se encontraban definidas en el LEF por lo que automáticamente son importadas y por otro lado, los comandos para definir las reglas

de antena son los mismos en ambas herramientas. Sin embargo, sí se tuvo que hacer una sola modificación en el *script* de TSMC ya que a pesar de ser los mismos comandos, este *script* está diseñado para ser importado en ICC no ICC2. En el ejemplo anterior se puede ver que al inicio se elimina cualquier regla de antena que estuviera definida previamente, en el *script* original está línea se encuentra de la siguiente manera:

```
remove_antenna_rules $lib
```

La modificación que se debe realizar es agregar el argumento "*-library*" al comando. Con esto se puede importar y definir las reglas en ICC2 sin problemas.

En ICC2 se puede definir reglas de antena generales o específicas por capa de metal, a continuación se detalla cómo se encuentran definidas estas reglas en el *script* de TSMC y cómo afectan durante el proceso de enrutamiento.

8.2.1. Definición de reglas de antena generales

Las reglas generales se pueden definir con el comando **define_antenna_rule**. En el *script* de TSMC se define una sola regla, la cual se encuentra configurada como se muestra en el ejemplo 8.4. El primer argumento a definir es el modo (*mode*) utilizado para calcular el área de metal. Se tiene un total de 6 modos y en este caso TSMC define el número 4. Con este se indica que el área de metal a utilizar es el área lateral (Figura 55b) y que para los cálculos se debe utilizar únicamente el metal de la capa actual, no los metales en capas superiores o inferiores. Esto significa que se desea que la relación de antena para los pines de entrada sea calculada de acuerdo a la ecuación 2. Por otro lado, el área lateral del metal es calculada de la siguiente manera:

$$\text{Área lateral} = 2 * (\text{Ancho} + \text{Largo}) * (\text{Grosor}) \tag{4}$$

El siguiente argumento que se define es el modo a utilizar para el cálculo de la protección brindada por los diodos de las difusiones (*diode_mode*). Se tiene un total de 18 modos y en este caso nuevmante TSMC define el número 4. Este modo indica que, para calcular la relación máxima de antena cuando se tiene uno o más diodos conectados, primero se debe realizar la sumatoria de los valores de protección de cada diodo (*dp*) y luego utilizar la siguiente fórmula:

- Si $dp_{total} > v_0$ y $v_4 \neq 0$

$$Relación\ de\ antena\ máxima = min(((dp_{total} + v_1) * v_2 + v_3), v_4) \tag{5}$$

- Si $dp_{total} > v_0$ y $v_4 = 0$

$$Relación\ de\ antena\ máxima = (dp_{total} + v_1) * v_2 + v_3 \tag{6}$$

- Si $dp_{total} \leq v_0$

$$Relación\ de\ antena\ máxima = Relación\ de\ antena\ de\ la\ capa\ de\ metal \tag{7}$$

Donde:

$$dp_{total} = \sum_{i=1}^{n} dp_i \tag{8}$$

$$\{v_0 \quad v_1 \quad v_2 \quad v_3 \quad [v_4]\} = \vec{v} \tag{9}$$

En este caso el valor para *dp* (*diode protection*) se encuentra especificado para cada celda, ya sea en el LEF (ANTENNADIFFAREA) o el **.clf** (**defineDiodeProtection**) como se explicó en las secciones anteriores. Por otro lado, el vector $\vec{v}$ se conoce como el vector de relación de diodo y se especifica por cada capa de metal, como se demostrará en la siguiente sección. El elemento v_4 se encuentra entre corchetes ya que su definición es opcional y si no se define por defecto su valor es 0.

Finalmente, en la definición de la regla general el argumento "*metal_ratio*" define la relación máxima a utilizar si no se tiene conectado un diodo que brinde protección en esa conexión o si se cumple la tercera condicional durante el cálculo de la relación máxima con protección (Ecuación 7). El argumento "*cut_ratio*" especifica la relación máxima en caso se tenga conexión hacia una capa conocida como la capa de corte, sin embargo, cómo este valor es 0 el argumento es ignorado.

8.2.2. Definición de reglas de antena específicas

Las reglas específicas se definen por cada capa de metal con el comando **define_antenna_layer_rule**. En el ejemplo 8.4 se muestra cómo TSMC definió estas reglas para las capas de metal 5 y 6. Junto al comando se definen varios argumentos, el primero es nuevamente el modo utilizado en el cálculo para el área de metal. Este argumento funciona de la misma manera en las reglas generales y específicas. Se puede ver que para ambos metales se escoge el modo 4, indicando que el área de metal a calcular será la lateral (Ecuación 4) y se calculará de forma individual, capa por capa.

Con el argumento "*layer*" se define la capa de metal a la que se desea aplicar la regla. Nuevamente con el argumento "*ratio*" se define la relación máxima a utilizar en caso no se cuente con protección de diodo en esa conexión o la protección brindada no sea la suficiente de acuerdo a los criterios de evaluación para el cálculo de la relación máxima. Como se mencionó, el vector de relación de diodo debe ser definido por cada capa de metal y esto se puede hacer con el argumento "*diode_ratio*". En el ejemplo también se puede observar que TSMC únicamente define 4 elementos del vector, efectivamente indicando que en cualquier cálculo para la relación máxima v_4 será 0 y por lo tanto únicamente se utilizarán las ecuaciones 6 y 7.

Es importante mencionar que en el Ejemplo 8.4 únicamente se decidió mostrar las definiciones para los metales 5 y 6. Esto ya que la regla aplicada para el metal 5 es la misma que para los metales 1 al 4, cambiando únicamente el valor del argumento "*layer*". Lo comentado anteriormente se puede comprobar en las líneas iniciales del ejemplo donde se muestran los comentarios hechos por TSMC en el *script*. En estos indican que para todos los metales la relación máxima es 400 si no hay protección. Por otro lado, para los metales 1 al 5 la expresión para la relación máxima es la misma que la obtenida si se sustituye el vector de relación de diodo definido para la capa 5 en la Ecuación 6:

$$\vec{v} = \begin{Bmatrix} 0.203 & 0 & 400 & 2200 \end{Bmatrix}$$

$$\text{Relación de antena máxima capas 1-5} = (dp_{total}) * 400 + 2200$$

Se puede comprobar de igual manera que este caso es el mismo para la relación máxima de la capa 6. Adicional a lo mencionado anteriormente, es importante comentar que en el *script* de TSMC también se definen reglas de antena para conexiones a vías, estas se muestran en el Ejemplo 8.5 y como se puede observar, su estructura es la misma que para el caso de los metales por lo que aplica todo lo mencionado anteriormente.

Ejemplo 8.5: Definición de reglas de antena para vías en el *script* de TSMC

```
# Single-layer drawn ratio of a via area to the active poly gate area
# VIA    : ratio=20
# VIA    : ratio=diode_area*83.33+75          if diode_area >=  0.203


##### Single via layer area rule #####
define_antenna_rule $lib \
  -mode 1 \
  -diode_mode 4 \
  -metal_ratio 0 \
  -cut_ratio 20

define_antenna_layer_rule $lib \
            -mode 1 \
            -layer "VIA12" \
            -ratio 20 \
            -diode_ratio {0.203 0 83.33 75}

define_antenna_layer_rule $lib \
            -mode 1 \
            -layer "VIA23" \
            -ratio 20 \
            -diode_ratio {0.203 0 83.33 75}
```

Como observación final, si se analiza cuidadosamente la librería física en el archivo LEF se puede comprobar que en esta se definen tanto los parámetros de antena para las celdas como las reglas para metales y vías. Observando nuevamente el Ejemplo 8.1 se puede ver que se define la *keyword* ANTENNASIDEAREARATIO con un valor de 400. De acuerdo a lo comentado, esto es equivalente a definir una regla para la capa de metal 1 especificando que el área de metal a utilizar en los cálculos es la lateral (modo 4) y que la relación máxima sin protección será 400.

Por otro lado, con la *keyword* ANTENNADIFFSIDEAREARATIO se da una lista de relaciones donde de acuerdo a valor del *dp* se asigna un valor para la relación máxima. Estos valores concuerdan con lo que se puede calcular utilizando las ecuaciones 6 y 7, así como los valores definidos en el *script* de TSMC (Ejemplo 8.4). A pesar que ya se pueden encontrar definidas las reglas al importar la librería física, se recomienda de igual manera definirlas previo al ruteo en ICC2 utilizando el *script* de TSMC. Esto para asegurarse que las reglas se encuentren definidas explícitamente en la herramienta mientras se ejecuta el proceso de enrutamiento.

8.3. Verificación sobre diseños generados en la síntesis lógica

Luego de haber validado que el flujo de trabajo propuesto es capaz de generar diseños sin errores durante la ejecución de la metodología y que las reglas de antena pueden ser importadas exitosamente en ICC2, es necesario verificar los diseños no presenten violaciones a las mismas. Para esto se puede utilizar la herramienta ICV, la cual requiere de dos archivos para llevar a cabo la verificación. El primero siendo el GDSII donde se encuentra el diseño y el segundo el *deck* o *runset* del *foundry* donde se indican todas las reglas que deben ser validadas. Debido a que todo lo realizado hasta el momento fue utilizando las librerías de TSMC de 180 nm y 6 capas de metal, se debe utilizar el *runset* provisto por TSMC. En este caso, el nombre del archivo que se debe utilizar es "ICVLM18_LM16_LM152_6M.ANT.215A_-pre041518". Previo a realizar la verificación, se debe realizar ciertas modificaciones al *runset*. En total son 3 modificaciones las cuales se muestran resaltadas en el Ejemplo 8.6.

Ejemplo 8.6: Cambios a ejecutar en el *runset* de antena de TSMC

```
// OPTION SETUP
//===============

//#define MIMCAP_2F
#define THICK_40K
//#define THICK_20K
M1_THICKNESS : double = 0.53;          /* The Thickness of First Metal is 5.3K A */
M2_THICKNESS : double = 0.53;          /* The Thickness of Inter Metal is 5.3K A */
M3_THICKNESS : double = 0.53;          /* The Thickness of Inter Metal is 5.3K A */
M4_THICKNESS : double = 0.53;          /* The Thickness of Inter Metal is 5.3K A */
M5_THICKNESS : double = 0.53;          /* The Thickness of Inter Metal is 5.3K A */
#ifdef THICK_40K
   M6_THICKNESS : double = 4.6;        /* The Thickness of Top Metal is 40K A */
#else
   #ifdef THICK_20K
      M6_THICKNESS : double = 2.34;     /* The Thickness of Top Metal is 20K A */
   #else
      M6_THICKNESS : double = 0.99;     /* The Thickness of Top Metal is 9.9K A */
   #endif
#endif
MD_THICKNESS : double = 0.99;
PO_THICKNESS : double = 0.2;
MX_S_1 : double = 0.28;
BALANCE_RATIO : double = 6000;
UNBALANCE_RATIO : double = 100;
DIO_AREA : double = 0.203;
vDNW_R_7 : double = 500000;
MT_THICKNESS : double = 0.99;

library(
    cell         = "Nombre-top-cell",
    format       = GDSII,
    library_name = "Path-archivo.gds",
);
```

La primera modificación es descomentar la línea: #define THICK_40K. La siguiente modificación es colocar el nombre de la *top-cell* en el diseño. Este debe ser el nombre que se colocó al módulo principal en el archivo Verilog. Como se sabe, este módulo principal es aquel que no es referenciado dentro de ningún otro módulo. Por último, se debe indicar el *path* completo donde se encuentra el archivo GDS II que contiene el diseño. Luego de aplicar los cambios se puede llevar a cabo la verificación como tal. Para esto se debe abrir una nueva terminal e ingresar el siguiente comando:

icv -i *path-completo-archivo.gds* **-c** *nombre-top-cell* **-sf ICV -vue** *path-completo-runset*

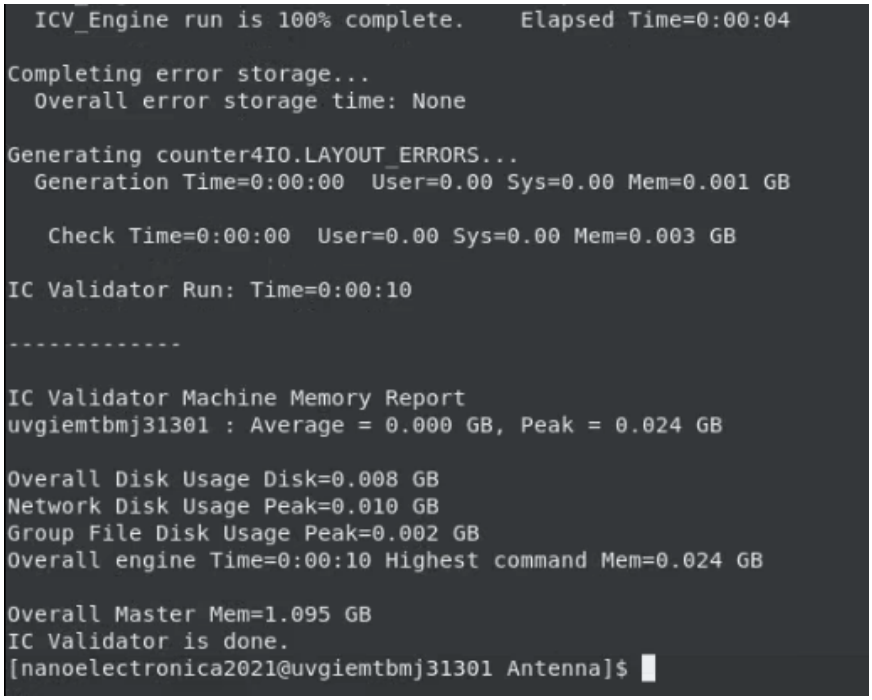


Figura 56: Fin de una verificación de antena en ICV

Este último invoca a ICV y le indica que debe realizar la verificación descrita en el *runset* sobre la *top-cell* del diseño contenido dentro del archivo GDS II. Si todo fue ejecutado correctamente se puede esperar que en la terminal se muestre algo similar a lo mostrado en la Figura 56, donde se indica que ICV ejecutó la verificación exitosamente.

Con este resultado se puede dirigir a la carpeta donde se abrió la terminal y se debe poder ver que se han generado una serie de archivos. Para verificar que el diseño haya aprobado la verificación se de sebe prestar especial atención a los archivos "Nombre-top-cell.RESULTS" y "Nombre-top-cell.LAYOUT_ERRORS". El primero detalla información de los procesos que fueron ejecutados, las reglas evaluadas e información adicional sobre el desempeño de la herramienta. Se debe buscar la sección de "Results Summary" ya que ahí es donde se indica el total de reglas evaluadas, así como el total de reglas aprobadas y no aprobadas. Esto se muestra en la Figura 57 donde se realizó la prueba sobre el diseño de la NOT y se puede ver que se evaluó un total de 44 reglas y ninguna está siendo violada. También se indica que para más detalles se puede referir al archivo .LAYOUT_ERRORS. Si el diseño ha aprobado todas las reglas este último archivo debe indicar algo similar a lo mostrado en la Figura 58. En este caso la palabra "CLEAN " al inicio indica que no se ha encontrado errores en el diseño y este ha aprobado la verificación de antena.

```
User name:        nanoelectronica2021
Layout format:    GDSII
Input file name:  /home/nanoelectronica2021/Documentos/Trabajo2021/NOT/Antenna/Not.gds
Top cell name:    Not_IO
Time started:     2021/09/21 06:48:29PM
Time ended:       2021/09/21 06:48:42PM

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

--------------------------------------------------------------------------------
Results Summary
--------------------------------------------------------------------------------

Rule and DRC Error Summary

44 total rules were run.
0 rules NOT EXECUTED.
0 rules have violations.
There are 0 total violations.
Refer to Not_IO.LAYOUT_ERRORS
```

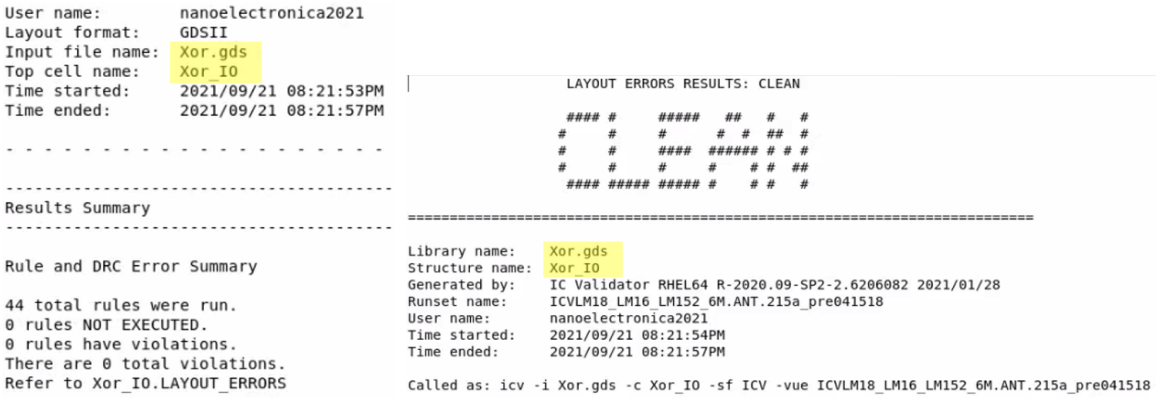
Figura 57: Archivo RESULTS generado durante la verificación de antena exitosa

```
                  LAYOUT ERRORS RESULTS: CLEAN

                   #### #     #####   ##   #   #
                  #    #     #      #  #  ##  #
                  #    #     ####  ###### # # #
                  #    #     #     #    # #  ##
                   #### ##### ##### #    # #   #

================================================================================

Library name:     Not.gds
Structure name:   Not_IO
Generated by:     IC Validator RHEL64 R-2020.09-SP2-2.6206082 2021/01/28
Runset name:      ICVLM18_LM16_LM152_6M.ANT.215a_pre041518
User name:        nanoelectronica2021
Time started:     2021/09/21 06:58:44PM
Time ended:       2021/09/21 06:58:47PM

Called as: icv -i Not.gds -c Not_IO -sf ICV -vue ICVLM18_LM16_LM152_6M.ANT.215a_pre041518
```

Figura 58: Arcvhivo LAYOUT ERRORS generado durante la verificación de antena exitosa

Luego de haber validado el diseño de la compuerta NOT se puede proceder con la validación del resto de diseños generados, estos son los mostrados en la sección 7.6. A continuación se muestran los resultados de estas validaciones los cuales fueron exitosos en todos los casos. En vista de esto, se pudo validar que utilizando el flujo de diseño propuesto se puede llevar a cabo el proceso de síntesis física en ICC2 sin errores de ejecución y que los diseños generados durante el mismo cumplen con las reglas de antena del *foundry* TSMC.

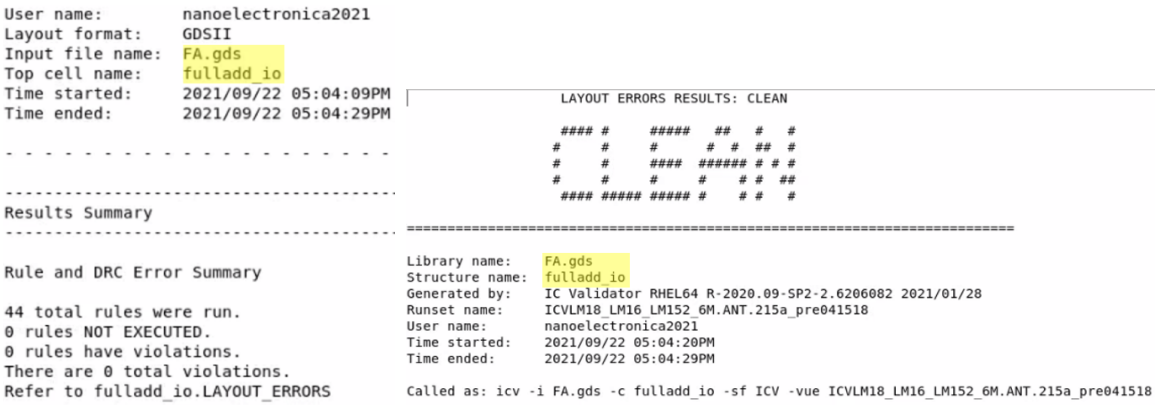
8.3.1. Verificación compuerta XOR



(a) Archivo RESULTS (b) Archivo LAYOUT ERRORS

Figura 59: Verificación de antena sobre el diseño de una compuerta XOR

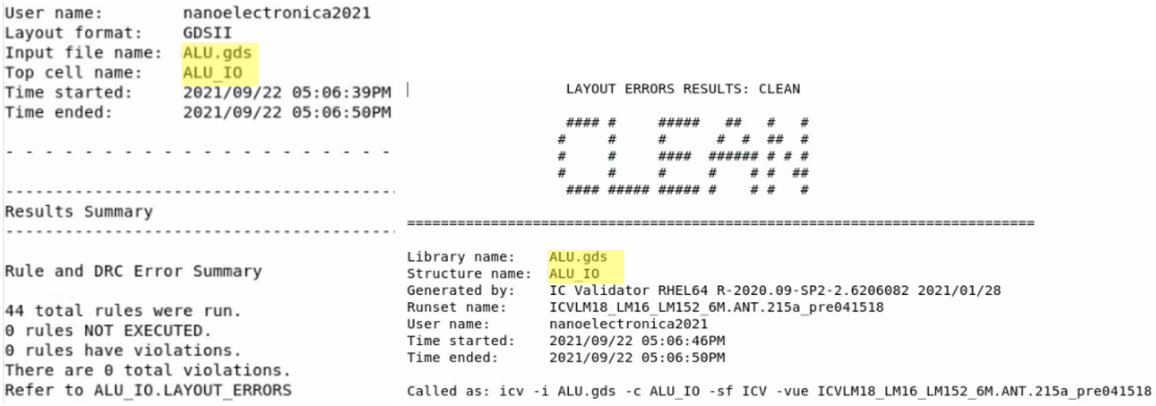
8.3.2. Verificación circuito *Full Adder* de 4 bits



(a) Archivo RESULTS (b) Archivo LAYOUT ERRORS

Figura 60: Verificación de antena sobre el diseño de un *Full Adder*

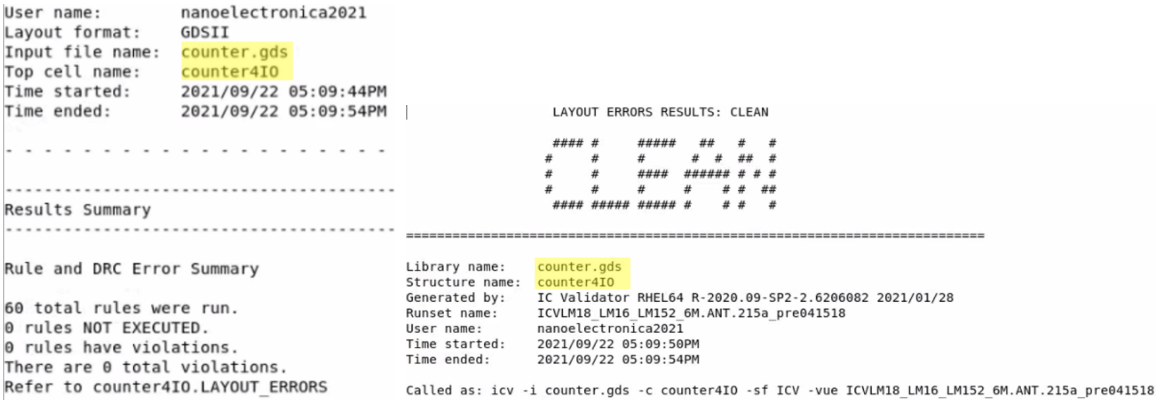
8.3.3. Verificación ALU de 4 bits



(a) Archivo RESULTS (b) Archivo LAYOUT ERRORS

Figura 61: Verificación de antena sobre el diseño de una ALU

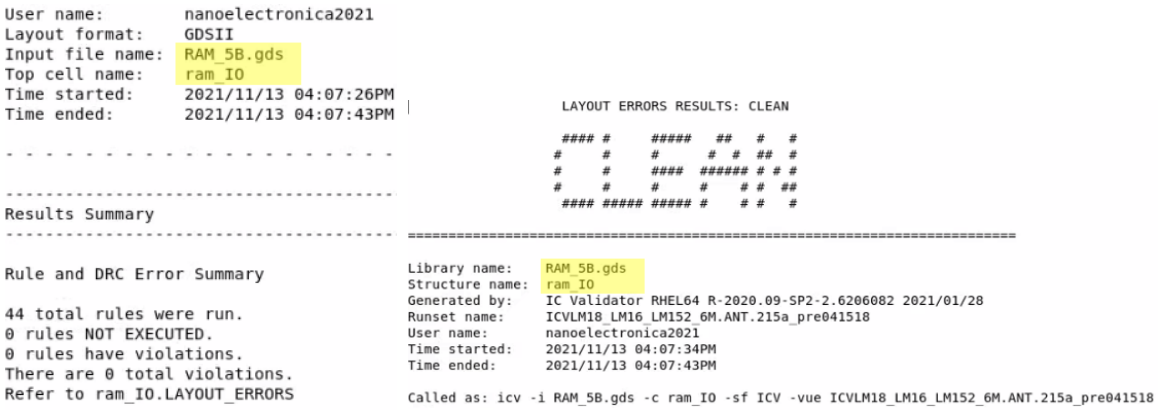
8.3.4. Verificación Contador de 4 bits



(a) Archivo RESULTS (b) Archivo LAYOUT ERRORS

Figura 62: Verificación de antena sobre el diseño de un contador

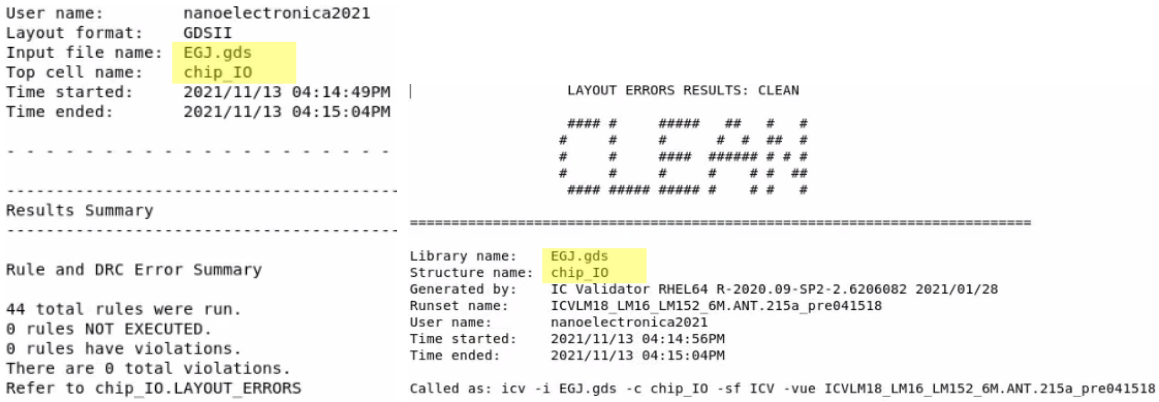
8.3.5. Verificación RAM de 5 bits



(a) Archivo RESULTS (b) Archivo LAYOUT ERRORS

Figura 63: Verificación de antena sobre el diseño de una RAM

8.3.6. Verificación Chip UVG



(a) Archivo RESULTS (b) Archivo LAYOUT ERRORS

Figura 64: Verificación de antena sobre el diseño del Chip UVG

CAPÍTULO 9

Conclusiones

1. Se logró replicar los avances alcanzados en trabajos previos mediante la actualización de los procesos a la herramienta *IC Compiler 2*.

2. Se pudo generar un flujo de trabajo optimizado que permitiera llevar a cabo la etapa de síntesis física en *IC Compiler 2* para una gran variedad de circuitos de distintas complejidades, todo sin errores de ejecución de la herramienta.

3. Se validó que los diseños generados utilizando el nuevo flujo de diseño no presentaran violaciones a las reglas de antena del *foundry*, de esta forma validando la efectividad del flujo.

4. Se elaboraron *scripts* que permitieran ejecutar de forma automática los distintos procesos que conforman el flujo propuesto, de manera que se pueda proseguir con la automatización de todo el flujo de diseño VLSI.

5. Se mantuvo una buena comunicación con los distintos miembros del proyecto de forma que no solo fuera posible validar el correcto funcionamiento de las distintas etapas por separado sino que de todo el flujo de diseño como tal.

CAPÍTULO 10

Recomendaciones

1. Estudiar el uso de las variables de trabajo dentro de *IC Compiler 2*, en especial las que almacenan la información sobre el número de *pads* en el diseño y área del *core*. Esto de forma que se pueda automatizar el proceso de determinar las dimensiones adecuadas para el área del *core*, así como ajustar automáticamente el área del *mesh* de acuerdo a estas dimensiones.

2. Analizar la efectividad de implementar procesos relacionados al *Clock Planning* durante la etapa del *flooplaning*, buscando aumentar la robustez de flujo al incluir procesos que preparen al diseño para la síntetización de relojes.

3. Investigar el uso de archivos con formato *Unified Power Format* (UPF) en el diseño VLSI para determinar su utilidad en el flujo de diseño propuesto y las posibles ventajas y desventajas de su implementación.

4. Automatizar el proceso de verificación de antena para que el nombre de la librería y celda solo deba ser indicado una vez durante el proceso, ya sea editando el *runset* o el comando que invoca a ICV. Esto para eliminar la doble definición de parámetros que se debe hacer en el proceso actual.

CAPÍTULO 11

Bibliografía

[1] J. A. d. l. Santos, "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," en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2014.

[2] L. A. Nájera, "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," en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2019.

[3] S. H. Rubio, "Definición del Flujo de Diseño para Fabricación de un Chip con Tecnologia VLSI CMOS," en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2019.

[4] A. S. Sedra y K. C. Smith, "Devices and Basic Circuits," en *Microelectronic circuits*, 7th. Oxford University Press, 2015.

[5] N. H. Weste y D. Harris, *CMOS VLSI Design: a circuits and systems perspective*. Pearson Education, 2011.

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

[7] C. A. Cruz, "Ejecución y utilización de un flujo de diseño para desarrollo de un chip con tecnología nanométrica: Extracción de componentes parásitos y simulaciones en HSPICE," en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2020.

[8] *An Open-Source Digital Synthesis Flow*, feb. de 2019. dirección: `http://opencircuitdesign.com/qflow/welcome.html`.

[9] *About Us*. dirección: `https://www.synopsys.com/company.html`.

[10] L. E. Abadía, "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," en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2020.

[11] M. G. Flores, "Corrección de anillo de entradas/salidas y pruebas de antenna y ERC para la definición del flujo de diseño del primer chip con tecnología nanométrica desarrollado en Guatemala," en *Trabajo de graduación en modalidad de Tesis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2020.

[12] *Library Manager User Guide*. dirección: https://spdocs.synopsys.com/dow_retrieve/R-2020.09/dg/icc2olh/Default.htm#lmug/pdf/lmug.pdf.

[13] *Design Planning User Guide*. dirección: https://spdocs.synopsys.com/dow_retrieve/R-2020.09/dg/icc2olh/Default.htm#icc2dp/pdf/icc2dp.pdf.

[14] *Implementation User Guide*. dirección: https://spdocs.synopsys.com/dow_retrieve/R-2020.09/dg/icc2olh/Default.htm#icc2ug/pdf/icc2ug.pdf.

[15] *LEF/DEF Language Reference*. dirección: https://www.ispd.cc/contests/18/lefdefref.pdf.

CAPÍTULO 12

Anexos

12.1. Script del *Library Manager*

```
1  #Importamos el tech file
2  create_workspace -flow normal -technology usr/synopsys/TSMC/180/CMOS/G/
       stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/
       tcb018gbwp7t_270a/techfiles/tsmc018_6lm.tf NormalWorkspace
3
4  #Importamos librerias logicas para las celdas estandar (db)
5  read_db { usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/
       TSMCHOME/digital/Front_End/timing_power_noise/NLDM/tcb018gbwp7t_270a/
       tcb018gbwp7tbc.db usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/
       tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/timing_power_noise/NLDM/
       tcb018gbwp7t_270a/tcb018gbwp7tlt.db usr/synopsys/TSMC/180/CMOS/G/stclib
       /7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/
       timing_power_noise/NLDM/tcb018gbwp7tml.db usr/synopsys/TSMC/180/CMOS/G/
       stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/
       timing_power_noise/NLDM/tcb018gbwp7ttc.db usr/synopsys/TSMC/180/CMOS/G/
       stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/
       timing_power_noise/NLDM/tcb018gbwp7twc.db usr/synopsys/TSMC/180/CMOS/G/
       stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/
       timing_power_noise/NLDM/tcb018gbwp7twcl.db }
6
7  #Importamos libreria fisica para las celdas estandar (LEF)
8  read_lef /usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/
       TSMCHOME/digital/Back_End/lef/tcb018gbwp7t_270a/lef/tcb018gbwp7t_6lm.lef
9
10 #Importar propiedades de antena de las celdas estandar (CLF)
11 #Descomentar solo si las propiedades no se encuentran definidas en el LEF
```

```
12 #read_clf_antenna_properties /usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/
       tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/tcb018gbwp7t_270a
       /clf/antenna_tcb018gbwp7t.clf
13
14 #Correr y desplegar el check
15 current_workspace; check_workspace
16 gui_create_window -type MessageBrowserWindow
17 open_ems_database check_workspace.ems
18
19 #Commit y save a la libreria de celdas estandar
20 current_workspace NormalWorkspace; commit_workspace -output
       StandardWorkspace.ndm
21
22 #Creamos la libreria para los pads
23 create_workspace -flow normal -technology /home/nanoelectronica2021/
       Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf PadsWorkspace
24 read_db { /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/
       tcb018gbwp7t/LM/tpd018nvtc.db }
25 read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
       tpd018nv_6lm.lef
26 current_workspace; check_workspace
27 current_workspace PadsWorkspace; commit_workspace -output PadsWorkspace.ndm
28
29 #Creamos la libreria para los corners y pad fillers
30 create_workspace -flow physical_only -technology /home/nanoelectronica2021/
       Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf CornersWorkspace
31 read_db { /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/
       tcb018gbwp7t/LM/tpd018nvtc.db }
32 read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
       tpd018nv_6lm.lef
33 current_workspace; check_workspace
34 current_workspace CornersWorkspace; commit_workspace -output
       CornersWorkspace.ndm
35
36 #Creamos la libreria para los standard cell fillers
37 create_workspace -flow physical_only -technology /home/nanoelectronica2021/
       Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf PhysicalOnlyWorkspace
38 read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
       tcb018gbwp7t_6lm.lef
39 read_db { /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/
       tcb018gbwp7t/LM/tcb018gbwp7tbc.db /home/nanoelectronica2021/Documentos/
       tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/tcb018gbwp7tlt.db /home/
       nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/
       tcb018gbwp7tml.db /home/nanoelectronica2021/Documentos/
       tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/tcb018gbwp7ttc.db /home/
       nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/
       tcb018gbwp7twc.db /home/nanoelectronica2021/Documentos/
       tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/tcb018gbwp7twcl.db }
```

```
current_workspace; check_workspace
current_workspace PhysicalOnlyWorkspace; commit_workspace -output
    FillersWorkspace.ndm

#Integramos los 4 NDM y creamos la CLIB unificada
create_workspace -flow aggregate TSMCWorkspace
read_ndm /mnt/nfs/compartida/ASA/LibreriasNDM/CornersWorkspace.ndm; read_ndm
     /mnt/nfs/compartida/ASA/LibreriasNDM/FillersWorkspace.ndm; read_ndm /mnt
    /nfs/compartida/ASA/LibreriasNDM/PadsWorkspace.ndm; read_ndm /mnt/nfs/
    compartida/ASA/LibreriasNDM/StandardWorkspace.ndm;
current_workspace; check_workspace
current_workspace TSMCWorkspace; commit_workspace -output TSMCWorkspace.ndm
```

12.2. Script de ICC2

```
#Creamos la design library, en este caso es para la compuerta NOT. Modificar
     el nombre acorde al circuito o a los circuitos que va a contener.
create_lib NOT_SYN.ndm -technology /usr/synopsys/TSMC/180/CMOS/G/stclib/7-
    track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/
    tcb018gbwp7t_270a/techfiles/tsmc018_6lm.tf \
 -ref_libs /mnt/nfs/compartida/ASA/LibreriasNDM/TSMCWorkspace.ndm

#Importamos el Verilog y el archivo de restricciones generado durante la
    sintesis logica. Se debe cambiar el nombre del archivo segun el circuito
    que se desea sintetizar
read_verilog /home/nanoelectronica2021/Desktop/salidas_not_io/out_not_io.v
read_sdc /home/nanoelectronica2021/Desktop/salidas_not_io/out_not_io.sdc

#Importamos las TLU+ y el layermap
read_parasitic_tech -tlup /home/nanoelectronica2021/Documentos/
    tcb018gbwp7t_290a_FE/tluplus/t018lo_1p6m_typical.tluplus \
-layermap /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/tluplus/
    star.map_6M

#Importamos y definimos las reglas de antenna
source -echo -verbose "/usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/
    tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/tcb018gbwp7t_270a
    /clf/antennaRule_018_6lm.tcl"

#Creamos los corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

#Creamos los pads para VDD y VSS
create_cell {PVDD} PVDD1CDG
create_cell {PVSS} PVSS1CDG

```

```
#Creamos las nets de VDD y VSS
resolve_pg_nets
create_net -power VDD
create_net -ground VSS
connect_pg_net -net VDD [get_pins -physical_context *VDD]
connect_pg_net -net VSS [get_pins -physical_context *VSS]
connect_pg_net -automatic
report_cells -power

#Creamos el floorplan. Se debe cambiar los parametros del side_length acorde a la complejidad del circuito
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length {100 100} -core_offset {125}

#Creamos el anillo IO
create_io_ring -name anillo_IO -corner_height 115

#Especificamos el posicionamiento deseado para los pads de VDD y VSS luego del placement
add_to_io_guide [get_io_guides anillo_IO.left] PVDD*
add_to_io_guide [get_io_guides anillo_IO.right] PVSS*

#Posicionamos los pads dentro del anillo IO
place_io

#Creamos el anillo de PG
create_pg_ring_pattern ring_pattern -horizontal_layer METAL2 -horizontal_width {2} -horizontal_spacing {2} -vertical_layer METAL3 -vertical_width {2} -vertical_spacing {2}
set_pg_strategy core_ring -pattern {{name: ring_pattern} {nets: {VDD VSS}} {offset: {1 1}}} -core
compile_pg -strategies core_ring

#Creamos conexion del anillo PG hacia los pads de VDD y VSS
create_pg_macro_conn_pattern hm_pattern -pin_conn_type scattered_pin -layers {METAL2 METAL3} -nets {VDD VSS} -pin_layers {METAL2}
set_app_options -name plan.pgroute.treat_pad_as_macro -value true
set_pg_strategy macro_conn -macros [get_cells {PVDD PVSS}] -pattern {{name: hm_pattern} {nets: {VDD VSS}}}
set_pg_strategy_via_rule macro_conn_via_rule -via_rule {{{{strategies: macro_conn}}{{existing: all} {layers: METAL3}} {via_master: default}} {{intersection: undefined}{via_master: NIL}}}
compile_pg -strategies macro_conn -via_rule macro_conn_via_rule -tag test

#Creamos el mesh del circuito para VDD y VSS
connect_pg_net -automatic
create_pg_mesh_pattern mesh_pattern -layers { {{vertical_layer: METAL3} {width: 4.2} {pitch: 42} {spacing: interleaving}} }
```

```
#Se debe modificar las dimensiones del poligono de acuerdo a las dimensiones del core
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {224.670 230.000}} -pattern {{pattern: mesh_pattern}{nets: {VDD VSS}}} -blockage {macros: all}
create_pg_std_cell_conn_pattern std_cell_pattern
set_pg_strategy std_cell_strategy -core -pattern {{pattern: std_cell_pattern}{nets: {VDD VSS}}}
compile_pg

#Merge del mesh con el anillo PG
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2 METAL3}

#Creamos y ejecutamos el placement de celdas estandar dentro del core
set_app_options -name place.coarse.fix_hard_macros -value false
set_app_options -name plan.place.auto_create_blockages -value auto
create_placement -floorplan -timing_driven -congestion -effort high -congestion_effort high
legalize_placement

#Ruteamos
check_routability -check_pg_blocked_ports true
check_design -checks pre_route_stage
route_auto

#Creamos los filler del core y el IO ring
create_io_filler_cells -io_guides [get_io_guides {anillo_IO.top anillo_IO.right anillo_IO.left anillo_IO.bottom}]
-reference_cells {PFILLER1 PFILLER5 PFILLER05 PFILLER0005 PFILLER10 PFILLER20}

create_stdcell_fillers -lib_cells [get_lib_cells {TSMCWorkspace|FillersWorkspace/FILL64BWP7T TSMCWorkspace|FillersWorkspace/FILL32BWP7T TSMCWorkspace|FillersWorkspace/FILL16BWP7T TSMCWorkspace|FillersWorkspace/FILL8BWP7T TSMCWorkspace|FillersWorkspace/FILL4BWP7T TSMCWorkspace|FillersWorkspace/FILL2BWP7T TSMCWorkspace|FillersWorkspace/FILL1BWP7T}]
connect_pg_net -automatic
remove_stdcell_fillers_with_violation
check_legality

#Guardamos el bloque y generamos el archivo GDSII. Cambiar el nombre del bloque y libreria de acuerdo al circuito sintetizado.
save_block NOT_SYN.ndm:Not_IO
write_gds -library NOT_SYN.ndm -design Not_IO -view design -hierarchy all -lib_cell_view frame NOT.gds
```

CAPÍTULO 13

Glosario

Celdas: Bloques lógicos descritos en las librerías del *foundry* los cuales están conformados por transistores y diseñados para cumplir un propósito específico. 14, 15, 16, 19, 24, 25, 30, 32, 34, 38, 39, 40, 41, 44, 45, 46, 48, 49, 50, 61, 68, 80

Celdas estándar: Celdas que ejecutan funciones lógicas y se utilizan en conjunto para cumplir el propósito del circuito. 16, 32, 34, 39, 42, 49, 50, 56, 59

Celdas filler: Celdas que permiten asegurar que el diseño cumplirá con las reglas de densidad. 34, 39, 62, 64, 65, 68

CMOS: Tecnología que utiliza parejas conformadas por un transistor *nMOS* y un *pMOS* para transmitir señales y crear circuitos lógicos de alta complejidad. 12, 13

Core: Área en el centro del chip designada para contener a las celdas estándar. 24, 25, 41, 42, 43, 45, 46, 49, 50, 51, 52, 53, 56, 57, 59, 62, 64, 65

Die: Nombre comúnmente utilizado para referirse al área total de silicio que es ocupada por un chip. 41, 42, 43, 48, 52, 56, 62, 68

Enrutamiento: Etapa del flujo de diseño VLSI que consiste en la creación de las conexiones entre las celdas de acuerdo a la descripción del circuito. 14, 16, 20, 25, 41, 45, 59, 60, 61, 62, 68, 79, 80, 83

Floorplan: Etapa del flujo de diseño VLSI que consiste en la preparación del *layout* para las etapas posteriores. 14, 20, 37, 38, 42, 43, 46, 54, 56, 68

Foundry: Nombre utilizado para referirse a empresas dedicadas a la fabricación chips. 17, 28, 29, 34, 39, 42, 55, 68, 79, 89, 91

Pads IO: Celdas que transportan señales desde el exterior del chip hacia el interior del mismo, este tipo de celdas son conocidas simplemente como *pads*. 32, 34, 39, 40, 43, 44, 48, 54, 59, 61

Placement: Etapa del flujo de diseño VLSI que consiste en el posicionamiento de las celdas dentro del *core*. 14, 15, 20, 45, 56, 58, 59, 68

RTL: Descripción de un circuito a un alto nivel de abstracción y utilizada por los lenguajes HDL. La información contenida en esta se utiliza para generar representaciones del circuito a un menor nivel de abstracción. 13, 14, 30

VLSI: Conjunto de procesos para el diseño y fabricación de circuitos integrados conformados por millones de transistores. 13, 15, 20