UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de un Circuito Integrado con Tecnología de 180nm usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificación de Reglas de Diseño y Corrección de Errores Obtenidos.

Trabajo de graduación presentado por Antonio Altuna Hernández 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 180nm usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificación de Reglas de Diseño y Corrección de Errores Obtenidos.

Trabajo de graduación presentado por Antonio Altuna Hernández 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

A lo largo de los pasados cuatro años de carrera universitaria la metodología de enseñanza esta orientada a investigación y desarrollo de nuevas tecnologías. La carrera de ingeniería en electrónica en la Universidad del Valle de Guatemala tiene en sí muy bien diferenciadas las características de los cursos de electrónica digital y electrónica analógica. Este proyecto es, no solo la colisión de estas dos disciplinas, es la cúspide del aprendizaje adquirido aplicado. La fabricación y diseño de circuitos integrados se considera una de las disciplinas más complejas e integrales en la industria. Un ingeniero electrónico juega el papel fundamental del diseño de estos circuitos integrados a nanoescala. Este proceso en sí es complejo y requiere de múltiples habilidades y dominio de diferentes herramientas y es por esto que cada parte está segmentada. En este trabajo de graduación los diferentes aprendizajes y la metodología modular se ve puesta a prueba. **IMEC** colaborador del proyecto, que es pionero en la región, nos ha proporcionado los recursos tanto de software como las restricciones y los recursos de **TSMC** para poder fabricar un circuito integrado de función personalizada.

Este trabajo de graduación abarca dos de las múltiples etapas del flujo de diseño para circuitos integrados, para ser específicos, estos son: Síntesis Física y *SignOff*. Para ambas etapas se cuenta con documentación y resultados significativos en el entendimiento de las herramientas de diseño de Synopsys. En caso de la síntesis física, el trasladar a la herramienta IC Compiler II los resultados de años anteriores en IC Compiler I mejorando el proceso representó un desafío. Esto se debe a que no solo la forma de trabajar en la nueva herramienta es diferente sino que con la poca experiencia que se cuenta en el área de diseño, el apoyo de la documentación, los trabajos anteriores y el soporte de **IMEC** además del trabajo en equipo es crucial. El proceso de síntesis se escala de forma que se inicia con un circuito de baja complejidad subiéndola hasta hacer una síntesis lo suficientemente robusta capaz de sintetizar una gran cantidad de circuitos. En este caso se inicia con una compuerta NOT, luego una XOR, luego un Sumador Completo, una Unidad Aritmético Lógica y un contador secuencial de 4 bits. Esta es una progresión ambiciosa pero realizable en el tiempo designado a este trabajo de investigación.

Adicional a este proceso de síntesis, la etapa de *SignOff* juega un papel fundamental. **TSMC** proporciona herramientas que en conjunto con los programas EDA de Synopsys permiten verificar la funcionalidad y manufacturabilidad del circuito a fabricar. El proceso de fabricación es tan complejo debido a la escala en la que se realiza por lo que verificar que

cumpla con los requerimientos de diseño establecidos por el fabricante es indispensable. Esta etapa de verificación contiene múltiples procesos entre estos: **LVS**, **LPE**, **ERC** y *Atenna Rule Check*. Cada una de estas verificaciones distribuida en el equipo de trabajo deben de ser concluidas con éxito, para cada circuito propuesto y así en algún momento este diseño se pueda mandar a fabricar.

Debido a la complejidad, el trabajo en equipo con los demás integrantes de esta investigación es indispensable y juega un rol crucial. No solo en asignación de tareas y cumplir con las metas y objetivos individuales, sino que para que en conjunto se llegue al objetivo principal que en conjunto con la UVG es la fabricación del primer circuito a escala nanométrica de la región.

Índice

Lista de figuras

Lista de cuadros

Resumen

El objetivo de esta investigación es mejorar la síntesis física actual, como parte del flujo de diseño para el desarrollo de circuitos integrados digitales a nanoescala. Esta por ser la tercera iteración del proyecto, para realizar el flujo de diseño, se cuenta con la documentación de las promociones pasadas. El primer paso para realizar este proyecto es el análisis de los resultados anteriores. Para hacer esto se replicarán los trabajos realizados anteriormente por los ingenieros: Luis Nájera y Luis Abadilla, que fueron los encargados de la síntesis física en su respectiva promoción. Luego de esto se migrarán estos procesos de la herramienta de Custom Compiler I a Custom Compiler II. Además se verificarán las librerías de **TSMC** que se están utilizando para corroborar que estén actualizadas y sean compatibles con el proceso de manufactura para el que se está diseñando.

Para etapa de verificación y validación de los resultados se realizarán los siguientes tres procesos de verificación: *Design Rule Check*, *Electrical Rule Check* y *Antenna Rule Check*. Para estos procesos de verificación también se cuenta con la documentación de los trabajos de los ingenieros Matthias Sibrian y Marvin Flores. Con las librerías en orden en conjunto con la síntesis física concluida, se estarán haciendo múltiples pruebas con diferentes circuitos eléctricos digitales como: una compuerta NOT, una compuerta XOR, un Full Adder, una Unidad Aritmética Lógica (ALU), un contador de 4 bits, una memoria RAM y un chip de diseño personalizado. Estos cinco circuitos serán pasados por el proceso de síntesis física y las verificaciones mencionadas anteriormente para validar que los resultados obtenidos por la síntesis física se puedan manufacturar por la empresa **TSMC**.

Abstract

The objective of this investigation is to improve the latest iteration of the physical synthesis, as part of the design flow to develop nanoscale integrated circuits. This is the third iteration of the project thus we have repositories and documentation regarding the physical synthesis process. The first step in the process of developing the physical synthesis for an integrated circuit is the analysis of the previous iterations results. This process consists in the replication of previous results. To achieve this goal as a team we will replicate the whole design flow, in the case of this layer of the process specifically the research of the engineers Luis Nájera and Luis Abadilla. After this process is concluded, the next step is to migrate the results form Synopsys EDA tood Custom compiler I to Custom Compiler II. In addition to this process, the validation and updating of Synopsys and **TSMC** software and repositories is intended.

For the SingOff layer in the design flow the following three process of validation and verification will be done: Design Rule Check, Electrical Rule Check and Antenna Rule Check. Each of these processes of verification was previously researched on and the results are at hand in the available repositories inherited from engineers Matthias Sibrian and Marvin Flores. With the updated Synopsys and **TSMC** software's and data altogether with the physical synthesis process concluded multiple runs of this flow will be done. The circuits that will be tested in this process are: a NOT gate, a XOR gate, a Full Adder Circuit, an Arithmetic Logical Unit (**ALU**) circuit, a 4 bit counter, a RAM and a personalized circuit design. These five circuits will be physically synthesized and validated in the SingOff layer of design to ensure the results obtained are eligible to manufacturing by **TSMC** foundary.

CAPÍTULO 1

Introducción

Ahora más que nunca los dispositivos que utilizan electrónica para funcionar forman parte de nuestro día a día. Estos dispositivos facilitan las tareas diarias, optimizan nuestro tiempo y proveen servicios a nuestros hogares y trabajos. A medida que la demanda de productos electrónicos crece la industria también. La innovación es consecuencia de esta alta demanda ya que busca satisfacer más necesidades de las personas, esto por el gran campo de aplicación que la electrónica ofrece. Algunas de las formas en la que la industria de la electrónica puede innovar son: haciendo dispositivos más pequeños, con menos consumo energético y mayor capacidad computacional. Sin embargo, a medida que los dispositivos electrónicos se hacen más pequeños las leyes de la física se hacen más complejas y las perturbaciones externas en conjunto con nuevas problemáticas debido a las dimensiones tan reducidas se convierten en un desafío difícil de superar.

Debido a que este proceso se hace tan complejo e imposible de manejar a mano. Las herramientas Electronic Design Automation (EDA) proporcionan una solución integral al proceso de diseño. Es más, debido a que aún así el proceso es demasiado complejo y requiere del dominio de múltiples disciplinas, existen un gran conjunto herramientas que buscan satisfacer las diferentes necesidades de los diseñadores. La estandarización de un proceso de diseño fue indispensable para que los diseñadores pudiesen especializarse y así perfeccionar el flujo de diseño.

En lationamérica, actualmente, los países que cuentan con las herramientas para el diseño de circuitos integrados a nanoescala son pocos. Menos son los países que cuentan con un equipo dedicado al diseño de circuitos integrados orientado a su fabricación. Actualmente la Universidad del Valle de Guatemala cuenta con un conjunto de herramientas de la *suite* de Synopsys y que en conjunto con **Interuniversity Microelectronics Centre (IMEC)** está desarrollando un flujo de diseño de circuitos a Nanoescala:. El objetivo principal de este flujo es enviar a fabricar a **Taiwan Semiconductor Manufacturing Company (TSMC)** un circuito integrado con tecnología de 180nm que impacte y motive a la región y en especial a Guatemala a formar parte de la industria de semiconductores la cual actualmente es liderada por los países del primer mundo.

El proceso de diseño y fabricación utilizado para este trabajo de graduación es *Cell/celda: based*. En este proceso la empresa que fabrica el circuito integrado proporciona al diseñador un conjunto de librerías que contienen las celdas base. Las celdas base son conjuntos de transistores encapsulados en dimensiones estándar que realizan tareas básicas como lo son las compuertas lógicas o *flip flops*. Para luego integrarlas y crear circuitos más complejos sin necesidad de revelar toda la información del diseño de estas celdas.

Este trabajo de investigación se centra en dos partes del flujo de diseño. La síntesis física y *SignOff*. En la parte de síntesis física principalmente se traducen los resultados lógicos del proceso de síntesis lógica validados y se reproduce una librería resultante con la información del circuito en silicio conformado por las celdas de las librerías de **TSMC**. En el proceso de *SignOff* la verificación de reglas de diseño verifica que los resultados generados por la síntesis física cumplan con los requerimientos de fabricación de **TSMC**.

Para poder generar el flujo de la síntesis física y verificación de las reglas de diseño y validarlo se debe someter a múltiples circuitos con diferentes características y grados de complejidad. Así mismo que estos cumplan con los procesos de verificación con la menor cantidad de errores posibles y también con la menor cantidad de modificaciones al proceso de síntesis debido a que eso le otorga robustez al proceso. Los circuitos que se estarán efectuando son: una compuerta NOT, una compuerta XOR, un circuito Full Adder una Unidad Aritmético Lógica, un contador de 4 bits y una RAM. Circuitos que tienen características diferentes y grados de complejidad significativamente más complicados a medida que se avanza.

CAPÍTULO 2

Antecedentes

La Universidad del Valle de Guatemala (UVG) a lo largo de los años se ha caracterizado por estar a la vanguardia en la investigación de nuevas tecnologías y metodologías en el área de las ciencias aplicadas. El Ing. Carlos Esquit, en el 2009, ingresó como nuevo director del departamento de Ing. Electrónica y Mecatrónica de la universidad y consigo trajo las destrezas y aprendizajes adquiridos en Estados Unidos en las ramas de micro y nanoelectrónica. Al iniciar, realizó una reforma al mapa curricular de Ing. Electrónica para enseñar dichas disciplinas aprendidas en el extranjero.

En el 2013, se inicia a impartir el curso de Introducción al diseño de sistemas **VLSI**. Al inicio las herramientas con las que se impartió este curso fueron gratuitas y permitían realizar diseños básicos a micro y nanoescala de circuitos. Al año siguiente, el ingeniero formó una alianza con la empresa Synopsys. Esta se encuentra a la vanguardia en el diseño en Silicio: de chips y su verificación. Esta proporcionó herramientas que actualmente utilizan los líderes de la industria de semiconductores. Con estas herramientas se realizó el primer diseño en silicio a nanoescala de 28nm, realizado por el Ing. Jonathan de los Santos [1] en 2014.

En la nueva reforma curricular, que entró en vigencia en 2015, se añadieron los cursos de Nanoelectrónica 1 y 2. El fin de este cambio, que actualmente sigue vigente, es impartir teoría de **VLSI** y desarrollar en los alumnos competencias en el área de investigación del diseño de chips a micro y nano escala. Esto utilizando las herramientas que el software Synopsys le proporciona a la UVG. Con estas herramientas en el 2019 y 2020 se realizó el diseño para el primer chip con tecnología CMOS a nanoescala en Guatemala.

En el año 2019 los alumnos Luis Nájera y Steven Rubio en conjunto con el apoyo de Interuniversity Microelectronics Centre (**IMEC**) y bajo la supervisión del Ing. Carlos Esquit desarrollaron lo que fue la primera iteración de este proyecto. Los resultados de este equipo fueron la implementación de las librerías de **TSMC** en el desarrollo de un flujo de diseño para la fabricación del primer circuito integrado en la historia de Guatemala, abriendo paso a una nueva área de investigación en la UVG. Los trabajos que documentan dichos resultados se encuentran en [2] y [3].

Los antecesores directos a este proyecto que trabajaron en el ciclo 2020 a 2021 en lo que se refiere a la síntesis física, Verificación Electrical Rule Check (ERC), Antenna Rule Check y verificación Design Rule Check (DRC) fueron el equipo de ingenieros: Luis Abadilla [4], Marvin Flores [5] y Matthias Sibrian [6]. Los resultados obtenidos por ellos fueron diferentes simulaciones funcionales, con las librerías de **TSMC** en los programas IC Validator y IC Compiler I. Adicionalmente dejaron documentación para utilizar los programas y *scripts* para replicar su trabajo.

CAPÍTULO 3

Justificación

La fabricación de Circuito Integrado (CI) puede ser un proceso largo y riguroso. El elaborar un flujo de diseño para la fabricación de estos **CI** abre nuevas oportunidades para lo que es el diseño y manufactura de estos dispositivos facilitando y automatizando el proceso. En el mercado actualmente hay una gran variedad de **CI** de propósito general. Sin embargo el poder crear **CI** dedicados permite que los dispositivos electrónicos que los utilizan mejoren su desempeño (velocidad, consumo de potencia, etc.). Esto se debe a que están diseñados para cumplir una única funcionalidad. A diferencia de aquellos de propósito general que cuentan con múltiples módulos internos que realizan tareas varias, que si bien lo hacen más versátil, hacen que este no dedique el 100 % de sus recursos a una tarea.

Es por esto que el crear un flujo de diseño funcional, con herramientas de estado del arte, permite impulsar el desarrollo de la industria de semiconductores en Guatemala. Por medio de la creación de plazas laborales que permitan el diseño y manufactura de estos dispositivos en el país así como la comercialización de estos productos.

Los beneficios inmediatos de este proyecto son el facilitar la utilización del flujo de diseño a quienes proceden esta investigación. Para que se pueda invertir tiempo en mantener actualizado el flujo de diseño y permitir que aquellos que se quieran beneficiar de este puedan crear proyectos de alto nivel en los departamentos de Electrónica, Mecatrónica y Biomédica de la UVG. Además, a largo plazo, que otras carreras puedan colaborar de forma interdisciplinaria para la creación de proyectos más ambiciosos y con un mayor alcance para beneficio de la comunidad UVG y del país.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Mejorar el proceso actual de síntesis física como parte del flujo de diseño para el desarrollo de un circuito integrado a nanoescala, trabajando en conjunto con los demás grupos, para mejorar la interconexión con los demás módulos, adicionalmente someter el *layout* a las diferentes verificaciones y trabajar en conjunto con los integrantes de mi grupo para solventar violaciones a las normas de diseño de forma eficiente para así obtener un *layout* funcional sin errores de DRC.

4.2. Objetivos específicos

- Migrar la síntesis física de la herramienta IC Compiler I a IC Compiler II.
- Verificar la compatibilidad de las librerías de **TSMC** utilizadas en el desarrollo de la síntesis física.
- Posicionar de forma estratégica los diferentes componentes del circuito integrado para que el proceso de ruteo sea eficiente y minimice los problemas en las etapas posteriores del flujo de diseño, adicionalmente que se cumpla con las restricciones de área, brindada por **TSMC**, que es de $2mm^2$.
- Validar los resultados de la síntesis física por medio de la verificación DRC, comprendiendo las diferentes violaciones a las normas de diseño y corrigiéndolas hasta que el *layout* este libre de errores.
- Mantener la comunicación con mi grupo de trabajo y los demás para solucionar los errores obtenidos de forma rápida y consolidar el flujo de diseño.
- Obtener y delegar los archivos y *scripts* resultantes del proceso de síntesis física y DRC al grupo encargado de la automatización del flujo de diseño.

CAPÍTULO 5

Alcance

El alcance de este trabajo de graduación se limita a tres grandes ejes. El primero es migrar la síntesis física, heredada por los estudiantes de la cohorte anterior, de la herramienta *IC Compiler I* a la herramienta de *IC Compiler II*. Esta migración se hace con el fin de actualizar el flujo de diseño ya que las herramientas necesarias para las otras partes del flujo de diseño como *Custom Compiler* han empezado a desistir de algunas facilidades con la versión de *IC Compiler I*.

El segundo eje es la mejora del proceso de síntesis física en la herramienta de *IC compiler II* utilizando los archivos, *runsets* y documentación de **TSMC** por el intermediario y patrocinador **IMEC** utilizando los programas de la suite de Synopsys a nuestra disposición. Este trabajo pretende ser una guía para los futuros equipos de investigación con el fin de facilitar la comprensión de los archivos, los comandos y el proceso de síntesis física en general. Esto es de vital importancia ya que agiliza la curva de aprendizaje de la siguiente generación y sirve como guía para replicar resultados previos.

El tercer y último eje es la verificación y corrección de reglas de diseño por medio de la verificación *Design Rule Check* o **DRC** utilizando los *runsets*, archivos y documentación de **TSMC** por medio de **IMEC** utilizando las herramientas de la suite de Synopsys a nuestra disposición. Esta verificación se hace con el fin de minimizar los errores de diseño de los diferentes diseños a nanoescala realizados y documentados en este trabajo.

CAPÍTULO 6

Marco teórico

6.1. VLSI

Very Large Scale Integration (VLSI) es el proceso completo por el cual se fabrica un circuito integrado (**CI**), los cuales están compuestos, actualmente, de millones de transistores a nanoescala. El proceso de diseño y simulación se le conoce como Flujo de Diseño. [3] Dicho proceso sigue una estructura *top-down* la cual implica que el proceso se lleva a cabo desde la jerarquía más alta (macro) hasta el proceso más específico (micro).

Actualmente las computadoras, teléfonos y demás dispositivos electrónicos con capacidad de computo cuentan con un **CI** denominado procesador. El procesador de estos dispositivos es el encargado de realizar el cómputo de la información y distribuirla a lo largo de las diferentes piezas de Hardware: que integran a estos dispositivos. Las empresas líder en el diseño **VLSI** para procesadores de dispositivos electrónicos actualmente son Intel, AMD, Samsung y recientemente se incorporó a este campo, de diseñó a gran escala, Apple.

Apple actualmente cuenta con su primera generación de procesadores diseñados por ellos especializados para mejorar el actual rendimiento de sus computadoras, tablets y laptops. Este procesador se conoce como *M1* y cuenta con 16 mil millones de transistores con una tecnología de 5nm[7]. A continuación una imagen:

Figura 1: Primer procesador dedicado para computadoras Apple M1 con 16 miles de millones de transistores con una tecnología de 5nm. [7]

6.2. Flujo de diseño [6]

Es el proceso por el cual se obtiene el diseño de un **CI** fabricable y funcional. Este proceso puede dividirse en múltiples niveles de abstracción que conforman las diferentes capas del diseño de un **CI**. Las seis capas en las que se divide este proceso son: sistema, algoritmo, arquitectura, lógica, física y *SingOff*. A continuación se describe a mayor detalle cada uno de ellas. Cabe resaltar que la necesidad de dividir todo el proceso de fabricación en capas surge ya que la complejidad de diseño es tal que requiere de especialización en diferentes disciplinas. El segmentar en capas permite que cada especialista trabaje en un nivel de abstracción específico aportado al flujo de diseño flexibilidad, cooperación y calidad.

Ingenieros de diseño de Intel Corp pertenecientes al equipo de NIKE Design Technology mencionan que en sus procesos de trabajo para el diseño **VLSI** es necesario particionar las diferentes tareas a realizar en el flujo de diseño. Cada fase a cargo de ingenieros especializados. Sin embargo es importante recalcar que puede existir una partición innecesaria (sobresegmentación del flujo) de tareas que puede resultar en optimización local de cada uno de los procesos. Esto en mayor esquema puede presentar problemas en la optimización general del **CI** ya que en la marcha surgen cambios que afectan diferentes segmentos del flujo de diseño y puede tomar días arreglar estos problemas. [8] Es por esto que es importante la segmentación apropiada del flujo del diseño y que sean equipos de ingenieros especializados que permiten agilizar este proceso. A continuación se sugiere un particionamiento apropiado para el equipo de trabajo con el que se cuenta.

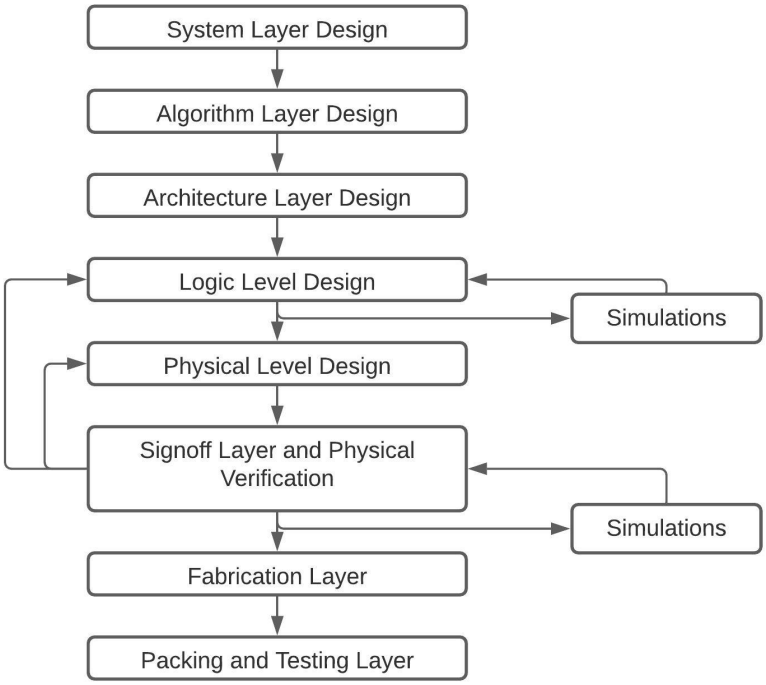

Figura 2: Flujo de diseño simplificado para el diseño de un **CI** digital a nanoescala

6.2.1. Diseño en la capa sistema

Esta es la jerarquía más alta de diseño, el objetivo que se desea conseguir en esta etapa es diseñar un sistema que cumpla con las necesidades básicas de su aplicación. Las principales características en las que se especializa esta capa son: funcionalidad, desempeño, condiciones de trabajo, dimensiones físicas, empaquetado, pinout, tecnología de fabricación, costos, la potencia consumida y la necesidad de comunicación interna y externa. [9]

6.2.2. Diseño en la capa algoritmo

Esta capa busca definir una estructura en Software: que permita una implementación a hardware funcional y sencilla. En esta etapa es importante considerar la capacidad de computo del **CI**, la cantidad de memoria que este requiere, se define la arquitectura del conjunto de instrucciones (**ISA**) por sus siglas en inglés [10]. En este punto debe evaluarse que tan complejo, que tantos recursos computacionales y que tanta exactitud se requiere según los objetivos establecidos en la capa superior como los protocolos de comunicación y las tareas que debe realizar.

6.2.3. Diseño en la capa arquitectura

Así como la capa superior busca distribuir los recursos computacionales de forma eficiente, esta capa busca distribuir de forma eficiente los recursos físicos (hardware) para cada una de las tareas que se van a realizar. Aquí se debe de prestar atención a los objetivos del diseño en la capa de sistema como: desempeño, costo, potencia, dimensiones, etc... En este punto la operación de la máquina (**MO**), por sus siglas en inglés, toma lugar y se refiere a como el hardware implementa el **ISA** definido en la capa superior [10]. La definición de *datapaths*, tamaño de los buses, memorias de acceso aleatorio (**RAM**) y su relación con los demás componentes son las tareas de prioridad.

6.2.4. Diseño en la capa lógica

En esta capa se busca generar una descripción lógica del circuito y hacer una síntesis de la misma para generar lo que se denomina *Netlist:*. Ambas tareas críticas para el diseño del **CI**.

Ya con una arquitectura, que define todo el proceso de interacción del hardware, se requiere definir la conectividad de cada módulo. En el nivel de abstracción más alto de esta capa se traduce en *black boxes* las cuales contienen entradas, salidas y una sincronización temporal definida (reloj). La cual se puede describir utilizando un lenguaje descriptor de hardware o **HDL** por sus siglas en inglés. Los lenguajes más comunes para poder describir el harware son: Verilog y VHDL. Estos módulos deben ser simulados y verificados para asegurar su funcionalidad.

El siguiente paso es la traducción de estos bloques de **HDL** a elementos de bajo nivel de un circuito y como estos están interconectados entre sí. Al resultado de esta traducción se le conoce como *netlist*. Herramientas de síntesis lógica permiten automatizar estos procesos, lo que facilita esta tarea de traducir de **HDL** a un *netlist*.

Los *netlists* utilizan algo llamado *cell librarys* que contienen los componentes más elementales de un circuito. Los transistores son definidos y conforman subcircuitos que representan compuertas lógicas las cuales en conjunto con los elementos básicos como capacitores, resistencias, entre otros; son descritos tanto sus propiedades como su interconexión. Esto se puede simular y validar utilizando una herramienta de simulación *hspice*.

6.2.5. Diseño en la capa física

A partir de este punto cada subcircuito descrito en el *netlist* es colocado en un espacio físico en una oblea de material semiconductor (Silicio) la cual está conformada de las diferentes difusiones, metales, pines y sus interconexiones. Debido a que el fabricante tiene requerimientos de diseño, que se deben cumplir para garantizar la funcionalidad del diseño, es imperativo que se tomen en cuenta y se verifiquen de forma rigurosa. Ya que este proceso toma en cuenta factores como: rendimiento, área, confiabilidad, potencia, entre otros... el proceso se divide en diferentes secciones: *Partitioning, FloorPlanning, Placement, Power and Ground Routing, Signal Routing y Closure* de las cuales se discutirán a mayor detalle. Al

final de todo este proceso un *netlist* asociado es generado el cual contiene todos los cambios y restricciones colocadas en los diferentes pasos de esta capa de síntesis física.

6.2.6. Diseño en la capa *Signoff* y verificación física

Una vez se culmina el diseño este debe de ser verificado de forma rigurosa para asegurar que este cumpla con los estándares de fabricación, que concuerde el diseño en silicio con el del circuito propuesto, etc... Cualquier problema debe ser solucionado en la capa anterior de diseño. Las verificaciones que se realizan se mencionan a continuación:

Design rule checking (DRC)

Verifica que el *Layout:* cumpla con todos los requerimientos de fabricación los cuales están directamente relacionadas a la tecnología que se está trabajando. El realizar un proceso de manufactura a escala nanométrica es complejo debido a que se ve limitado por la tecnología de las herramientas de fabricación con las que cuenta la fábrica. Limitaciones como las distancias entre los metales, difusiones, policilicio, o complicaciones como la pérdida de resolución, las esquinas redondeadas, las conexiones reducidas durante el proceso de manufactura, etc... deben mitigares y esto se hace cumpliendo con los requerimientos de los fabricantes ya que ellos toman en cuenta estos problemas a la hora de fabricarlo. [11]

Layout vs. schematic (LVS)

Verifica la funcionalidad del diseño. El *netlist* generado por la capa superior, síntesis física, y el *netlist* generado por la capa de síntesis lógica son comparados para verificar si ambos concuerdan en términos de funcionalidad.[12]

Antenna rule checking

Esta fase sirve para prevenir los efectos de antena en el circuito. Los conductores al acumular cargas pueden presentar efectos de antena. Estos pueden acumularse debido a las descargas del rayo láser al momento del grabado, generando descargas no deseadas en las compuertas del transistor. Esto representa un problema ya que puede causar daños a los transistores, en especial a la compuerta del gate, o reducir su tiempo de vida.

Electrical rule checking (ERC)

Verifica la calidad de la conexión de tierra y alimentación así como los tiempos de transmisión de señales. Adicionalmente verifica cargas capacitivas y la correcta conexión de los *fanouts*. Una compuerta lógica puede excitar un número finito de entradas a otras compuertas en un mismo nodo (*fanout*). Esta excitación se ve limitada por la corriente que provee el circuito. Por lo tanto debe tomarse en cuenta para la frecuencia de operación y la potencia

entregada máxima de una compuerta. Factores que afectan tanto el rendimiento como el tiempo de vida del *CI*. [5]

Parasitic extraction (LPE)

Este deriva de los componentes geométricos, a nivel nanométrico, sus parámetros eléctricos (Capacitancias y Resistencias) para verificar las características eléctricas del circuito. A esta verificación se ingresa un *netlist* y con la ayuda de la herramienta de Synopsys *starRC* se genera un *hspice* con todas estas capacitancias y resistencias parásitas presentes en el *layout*.

Final simulation

Luego de haber pasado todas las simulaciones en *CLEAN* o *PASS*, utilizando el *hspice* generado en la verificación de **LPE**, se hace una simulación que sirve para validar que los resultados obtenidos por el circuito descrito por el *layout* que se diseñó cumplan con las mismas funcionalidades que el circuito descrito en la capa de síntesis lógica. Si esto es congruente, entonces se puede concluir la etapa de verificaciones.

6.2.7. Diseño en la capa de fabricación

Al finalizar todas las simulaciones, se genera un archivo de extensión **.GDS2** el cual contiene toda la información que el fabricante necesita para empezar el proceso de manufactura. Este archivo es enviado a una fábrica que se dedica a manufacturar chips en silicio. En el caso del proyecto, el fabricante es **TSMC**. El proceso en donde se manda el archivo a la fábrica se le conoce como *tapeout*. Aquí se generan las máscaras y por el proceso de fotolitografía, patrones expuestos a los rayos láser se graban en el silicio, así como los diferentes dopajes. El diseño se hace sobre obleas de silicio redondas las cuales varían en tamaño desde los 200mm hasta los 300mm. Por último se clasifican según su estado: funcional o defectuoso, establecido por las pruebas de potencia y velocidad que se establecieron.[11]

6.2.8. Diseño en la capa de empaquetado y pruebas

Aquí se cortan los **CI** individualmente de la oblea de silicio donde se fabricaron. A cada **CI** se le conoce como *die* y este es empaquetado según la necesidad del cliente. Los empaquetados pueden ser: **DIP, PGS, BGA, etc...** Se conectan los pines dentro del empaquetado con el *die* ya sea con *bonding wires o solder bumps* y se sella. Como punto final del flujo de diseño, este se prueba en un laboratorio por el cliente para ver que este funcionando de la forma deseada y con los requerimientos propuestos. Concluyendo el flujo de diseño del **CI**.

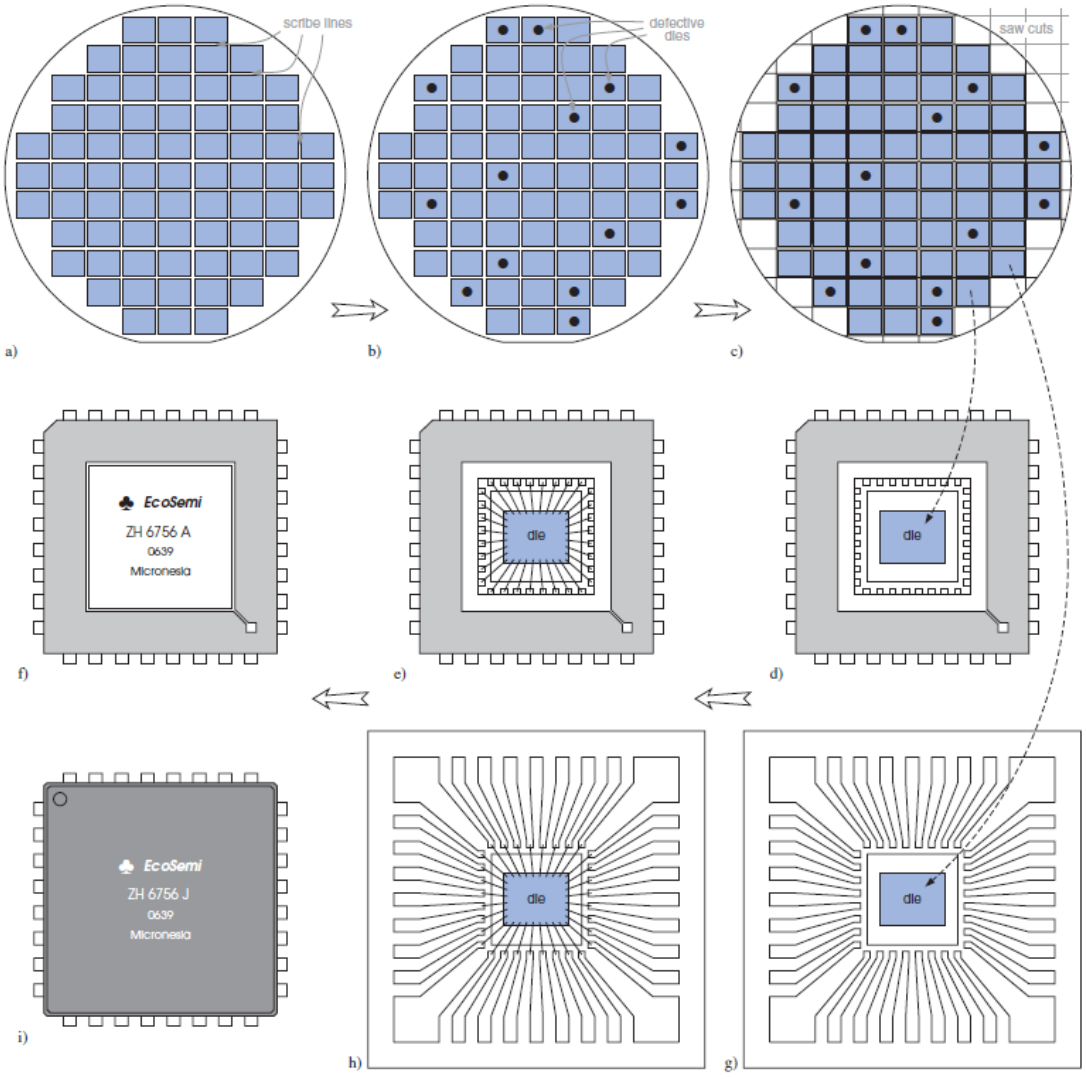

Fig. 11.11 Testing and encapsulation steps (simplified, not drawn to scale). Processed wafer (a), probed wafer with defective circuits inked (b), wafer after sawing (c), a good die attached to package cavity (d), wires bonded to lead frame (e), final IC after sealing, testing, and stamping (f). (g,h,i) correspond to (d,e,f) for a plastic package.

Figura 3: Segmentación de la oblea de silicio resultante en **die**, proceso de empaquetado, colocación de *bonding wires* y **CI** final [13]

6.3. Síntesis física, *Design rule check (DRC), electrical rule check (ERC) y antenna rule check*

Esta es la parte del flujo de diseño que se realizará este trabajo de investigación. Por lo tanto se desarrolla a más detalle la información de los procesos que se realizarán.

6.3.1. Síntesis física

La síntesis física es una de las partes fundamentales para el desarrollo de este proyecto, ya que aquí se transformará la síntesis lógica a un conjunto de componentes físicos posicionados en silicio y unidos a través de *interconects*. Esta etapa de diseño está segmentada en diferentes secciones las cuales se describen a continuación:

Partitioning

En esta etapa se secciona el circuito en subcircuitos con el fin de que estos puedan luego ser posicionados y colocados de forma estratégica.[9]

Floor planning

En esta etapa se hace una relación espacial entre los pines de entras y salidas del **CI** y los subcircuitos. Esto se hace con la finalidad de establecer prioridades a las secciones del circuito original que deben estar más cerca de estos puertos ya que se desea minimizar retardos y material utilizado.[14]

Placement

En esta etapa se plantean secciones del espacio que se va a utilizar para realizar el **CI**. En el caso de este proyecto se cuentan con $2mm^2$ de espacio para la fabricación del circuito deseado. Las partes seccionadas en la primera etapa se les asigna de forma estratégica una posición física en el silicio con el fin de obtener los mejores resultados en cuanto a la minimización de retardos y distancia entre las entradas y salidas de estos subcircuitos con los demás. [14]

Power and ground routing

En esta etapa se hace un Ruteo: de los planos de alimentación y tierra. Esto se hace de primero ya que todo circuito lógico compuesto de **MOSFETs** tendrá la necesidad de estos, lo que hace crítico que estas señales se hagan primero. [15]

Signal routing

Se generan *interconects* entre todos los circuitos lógicos. El software utilizado permite optimizar estas trayectorias a manera de que utilicen menor material. Ya que entre más distancias y más material la cantidad de parásitos incrementará lo que impacta directamente con el rendimiento del **CI**. [14]

Closure

Este es el paso antes de pasar a las verificaciones físicas, en esta fase se realizan las últimas revisiones del *layout*. Se puede realizar un **DRC** básico que proporciona la herramienta de IC Compiler II. [15]

Proceso de síntesis física anterior [16]

A continuación se muestra el proceso de la síntesis física realizada por el estudiante Luis Abadilla en el 2020. Los comandos que utilizó se describen y están colocados en el orden en el que fueron utilizados. Toda la síntesis física se realiza en la herramienta de IC Compiler I, y cuenta con cuatro partes fundamentales.

- **Importación de librerías y creación de *MilkyWay Library*:** En esta etapa se importan las librerías de **TSMC** a utilizar, se crean los archivos donde se documenta el historial de todas las acciones que se realizan en la síntesis física. Además se importan los archivos que provienen de la síntesis lógica. Adicionalmente se crea un directorio de trabajo y se llaman a las librerías link y target las cuales definen el caso típico.
- **Floor Plan**
  - **Conexiones lógicas de alimentación y tierra:** Esta sección se dedica a crear las variables y las celdas que le corresponden a los puertos de alimentación.
  - **Creación de pads y corners:** Aquí se plantean las esquinas y los *pads*, llamados celdas, los cuales definen y acotan el contorno del **CI**
  - **Configuración de las restricciones:** Una de las partes fundamentales del Floor Plan, ya que aquí se restringe todas las posibles orientaciones, posiciones, conexiones o arreglos que puedan causar conflictos en el ruteo. Indispensable para realizar un *layout* de calidad.
  - **Creación del Floor Plan:** se ejecuta y verifica el Floor Plan propuesto por todas las restricciones y configuraciones previas.
- **Placement:** Aquí se colocan los componentes según las reglas de Floor Plan, además se pueden configurar arreglos que optimicen: *delays*, *fanouts*, distancias y *X-Talk*. También es verificable y reconfigurable.
- **Routing:** Por último se realizan todas las interconexiones, empezando por los puertos de alimentación y luego por las demás interconexiones. Aquí se realiza nuevamente una verificación de la efectividad del proceso y se concluye la síntesis física.

Cuadro 1: Proceso de síntesis física realizado en el 2020, comandos y su descripción.

Comando	**Funcionalidad**
Comandos para la importación de librerías y creación de la *MilkyWay* y directorios necesarios para el proyecto	
./ icc_shell -gui - shared_license	Iniciar el programa y su interfaz gráfica en consola
lappend search_path / home / administrador / Escritorio /FA_ TSMC_ DRC / Libs / tcb018gbwp7t_ 290a _FE / tcb018gbwp7t /LM	Comando para definir el directorio en donde se encuentran las librerías que se van a utilizar
set link_ library " tcb018gbwp7ttc .db tcb018gbwp7twc .dbtcb018gbwp7tbc .db tpd018nvtc .db "*	Hacer mención directa de las librerías que se van a utilizar de todo el catálogo de **TSMC** (link)
set target_ library "tcb018gbwp7ttc .db "	Hacer mención directa de la librería (target) con la que se estará trabajando: caso típico
set_ tlu_ plus_ files -max_ tluplus / home / administrador / Escritorio/ FA_ TSMC_ DRC / Libs / tcb018gbwp7t_ 290a_ FE / tluplus / t018lo_ 1p6m_ typical . tluplus - min_ tluplus / home / administrador / Escritorio / FA_ TSMC_ DRC / Libs / tcb018gbwp7t_ 290a_ FE / tluplus / t018lo_ 1p6m_ typical . tluplus - tech2itf_ map / home / administrador / Escritorio / FA_ TSMC_ DRC / Libs / tcb018gbwp7t_ 290a_ FE / tluplus / star . map_ 6M	Comando para agregar la TUPLUS y el archivo .MAP específico dentro de las librerías de **TSMC**
create_ mw_ lib -technology / home / administrador / Escritorio / FA_ TSMC_ DRC / Libs / tcb018gbwp7t_ 290a_ FE /tf/ tsmc018_ 6lm .tf - mw_ reference_ library {/ home / administrador / Escritorio / FA_ TSMC_ DRC / Libs / tcb018gbwp7t_ 290a_ FE / tcb018gbwp7t / home / administrador / Escritorio / FA_ TSMC_ DRC / Libs / iolib / tpd018nv } - bus_ naming_ style {[% d]} -open / home / administrador / Escritorio / LUIS_ ABADIA / Milky_ Aba	Crear un directorio y una librería *MilkyWay* para la documentación de todos los cambios que se harán dentro de la síntesis física.
Hasta este punto se importan los archivos de extensión verilog (.v) y extensión (.sdc) resultantes de la síntesis física.	

Floor Plan Parte 1: Conexiones lógicas de VDD y VSS	
set_app_var mw_logic1_net "VDD"	Definición de una variable para la net de poder del voltaje
set_app_var mw_logic0_net "VSS"	Definición de una variable para la net de poder de tierra
derive_pg_connections -power_net VDD -power_pin VSS -ground_net VSS -ground_pin VSS	Configuración de los pines de poder y las nets de poder según las variables que se definieron en los comandos anteriores.
derive_pg_connections -power_net VDD -ground_net VSS -tie	
report_cell_physical -connections	Verificación de que los comandos anteriores se ejecutaron con éxito. Si el resultado es 1 es exitoso, si es 0 es fallido.
Floor Plan Parte 2: Creación de celdas (pads y corners)	
create_cell {C1 C2 C3 C4} PCorner	Creación de las 4 esquinas del chip.
create_cell { VDD} PVDD1CDG	Creación de la conexión de VDD (Pad)
create_cell { VDD} PVSS1CDG	Creación de la conexión de VSS (Pad)
Floor Plan Parte 3: Configuración de las restricciones	
set_pad_physical_constraints -pad_name ç1side 1	Configuración del primer pad de esquina en el lado 1
set_pad_physical_constraints -pad_name ç2side 2	Configuración del segundo pad de esquina en el lado 2
set_pad_physical_constraints -pad_name ç3side 3	Configuración del tercer pad de esquina en el lado 3
set_pad_physical_constraints -pad_name ç4side 4	Configuración del cuarto pad de esquina en el lado 4
set_pad_physical_constraints -pad_name "VDDside 3 -order 2	Configuración del pad de VDD en el lado 3 en segunda posición
set_pad_physical_constraints -pad_name "VSSside 4 -order 2	Configuración del pad de VSS en el lado 4 en segunda posición
Floor Plan Parte 4: Creación del Floor Plan	
create_floorplan - control_type width_and_height - core_utilization 0.7 - core_width 5 -core_height 50 - no_double_back - top_io2core 10 - bottom_io2core 10 - left_io2core 5 - right_io2core 5	Creación de la segmentación del espacio a utilizar por los componentes (Floor Plan)
check_physical_design	Verificación del proceso de diseño hasta este punto. Si el resultado es 1 es correcto, si es 0 es incorrecto y se debe corregir.
adjust_fp_floorplan	Este comando termina de hacer los ajustes necesarios para asegurar que este esté bien acotado.
Configuración y creación del placemnet	
set_fp_placement_strategy - adjust_shapes on	Ajuste de las geometrías dentro del Floor Plan

create_fp_placement -effort High -max_fanout 800 - optimize_pins - congestion_driven - timing_driven - consider_scan	Comando para realizar el posicionamiento , con mejor esfuerzo, un gran *fanout*, optimizando la congestión, el *delay* y la posición de los pines de entradas y salidas con respecto a los *pads*.
check_physical_design - post_initial_placement	Este comando permite realizar una verificación del placement si el resultado es 0 esta mal y si es 1 esta correcto.
place_opt -effort high	Optimizar el placement al final para tener mejores resultados maximizando el esfuerzo para que se cumplan las condiciones previamente establecidas.
Creación del Routing	
create_rectangular_rings -nets {VDD VSS } -around core	Creación del anillo de alimentación al rededor de todos los componentes para hacer el routing más eficiente.
create_power_straps -direction vertical -start_at 155 - num_placement_strap 4 - increment_x_or_y 50 -nets {VDD} -layer METAL2 -width 2 - do_not_route_over_macros	Este comando crea también un fácil acceso a los planos de alimentación y tierra. Los *power straps* dependen de la cantidad de componentes y saturación del espacio.
set_route_mode_options -zroute false	Para cambiar el comando clásico se utiliza este comando.
check_routeability	Este preceso sirve para hacer un análisis preliminar para saber si la herramienta es capaz de rutear el circuito propuesto. De ser cierto el resultado es 1, si es 0 se debe cambiar o la configuración o el placement.
set_route_opt_strategy	Este comando permite analizar si el circuito tiene problemas de delay, de X-talk o errores generales de diseño.
set_preroute_drc_strategy	En este punto se hace un análisis de DRC muy básico, esto es solamente como una pre revisión, antes del análisis DRC por IC Validator.
preroute_standard_cells -mode net -connect both -nets { VDD}	Comando para conectar networks con el anillo de VDD.
190 preroute_standard_cells -mode net -connect both -nets { VSS}	Comando para conectar networks con el anillo de VSS.
preroute_instances	Coando para conectar todas las celdas con los anillos.
route_opt	Conexión entre celdas y fuentes de alimentación.

verify_route	La última verificación de routing se hace aquí, si el valor es 0 debe de verificarse nuevamente y probar con nuevas estrategias de ruteo, si es 1 todo está en orden y se puede proceder a DRC.
Finaliza la síntesis física.	

6.3.2. DRC

Design Rule Check, esta etapa es crítica y juega un papel fundamental en conjunto con la síntesis física. Aquí se comparan los resultados obtenidos de la síntesis física con un *runset*, que en nuestro caso es proporcionado por **TSMC**, el cual contiene todas las restricciones de fabricación. Aquí se hace una exhaustiva revisión de las normas como el espaciado de metales, las interconexiones, el tamaño de los *pads*, etc... se cumplan y concuerden con todas las normas de **TSMC**. Esta etapa no trata acerca de la funcionalidad ni la coherencia entre el diseño lógico y físico que es tarea del **LVS**. Cualquier problema resultante de esta verificación debe ser gestionado por la capa de diseño superior (síntesis física) y de forma iterativa solucionar cualquier problema que se presente hasta obtener un resultado *CLEAN*. [6]

Para esta etapa se pueden presentar una gran cantidad de errores y no existe un orden específico, ni un formato definido para presentarlos. Es por esto que se debe de contar con la documentación del software que se estará utilizando para la realización de este proceso de validación. En el caso de Synopsys, la herramienta que ofrecen es IC Validator. Debido a que existen demasiadas reglas de diseño a continuación se mencionan algunas reglas de diseño comunes. [11]

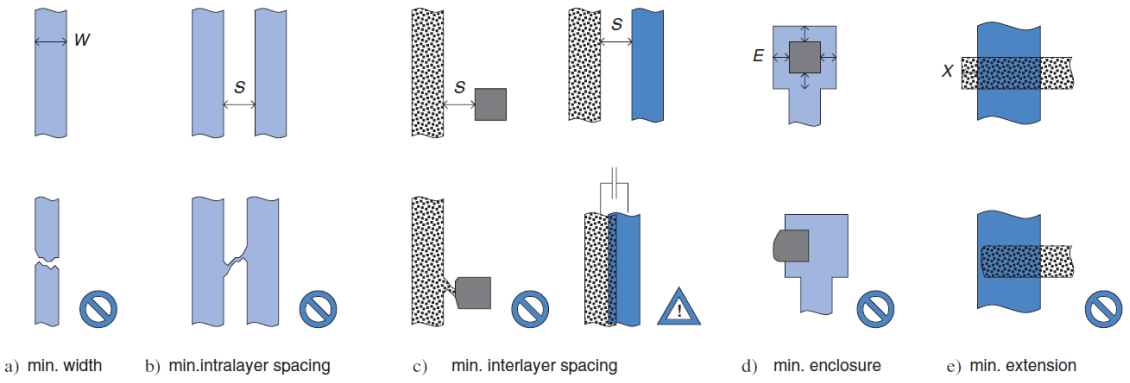

Fig. 11.1 Minimum size rules (top row) and the likely consequences of violating them (bottom row).

Figura 4: Ilustración de las reglas de diseño de condiciones mínimas y como estas se infringen [13]

Minimum width

Se establece un grosor mínimo de las estructuras para evitar que estas colapsen o se abra el circuito por falta de material. Particularmente las lineas largas y delgadas, sin elementos en sus vecindades, son más susceptibles a sobre-estirarse a diferencia que un conjunto de lineas del mismo grosor. Es por esto que se necesita un mayor grosor.[13]

Minimum intralayer spacing

Esta regla limita la distancia entre las diferentes estructuras. Esto se hace para evitar cortos circuitos entre los elementos adyacentes, También puede depender si estas estructuras tienen una conexión eléctrica existente. Por ejemplo, en un proceso de 130nm exige una separación de *Well-to-Well* de 630nm si estos manejan el mismo potencial. De no ser así este es de 1000nm. [13]

Minimum interlayer spacing

Esta regla permite verificar la distancia entre un plano y otro, para evitar acoplamientos e interacciones indeseadas entre sí. Problemas como la inflación de capacitancias por contactos entre polisilicio y difusiones o corrientes de fuga por acoplamientos ente uniones PN y una *gate* de policilicio. [13]

Minimum enclosure

Estas reglas están directamente relacionadas con las estructuras del *layout* en diferentes planos. Estas reglas son uniformes a lo largo de todos los bordes. Por ejemplo para las vias que el contacto sea el adecuado con las superficies que conecta ya que el proceso de manufactura tiene cierta tolerancia en las máscaras y su alineación además de las imperfecciones del proceso de *etching*. Otro ejemplo es que el metal de más alto nivel este debajo de *overglass passivation layer* a lo largo de todas las aperturas para los *pads* así el sellado es hermético. [13]

Minimum extension

Debido a que existen múltiples capas es importante mantener reglas que delimiten como se sobreponen las estructuras. La forma en la que esto se verifica es por medio de la orientación en la que se sobreponen. El *self-aligned perocess* permite que el policilicio actúe como una máscara para el proceso de implantación de iones [11]. Por lo que si estas capas de policilicio no están correctamente alineadas es posible que ya sea el *drain* o el *source* no se dopen de forma correcta. Esto afectaría considerablemente el desempeño y la funcionalidad del **CI**. Estas reglas ya toman en cuenta las imperfecciones del proceso de *etching* y problemas en la alineación de las máscaras. [13]

Maximum width

Las vías y en general los contactos se fabrican por medio del proceso de *etching* en la capa de dieléctrico para luego depositar rellenos de tungsteno para llenar las cavidades, antes de depositar la siguiente capa metálica. Para asegurar una calidad uniforme y una superficie plana es necesario delimitar el grosor máximo de estas cavidades. Lo más común es delimitar un único tamaño de vía o contacto. Si en algún caso se desea agregar un contacto

más grande, estas deben segmentares en subgrupos de contactos. A esta técnica se le conoce como *stipple contact/via.*[13]

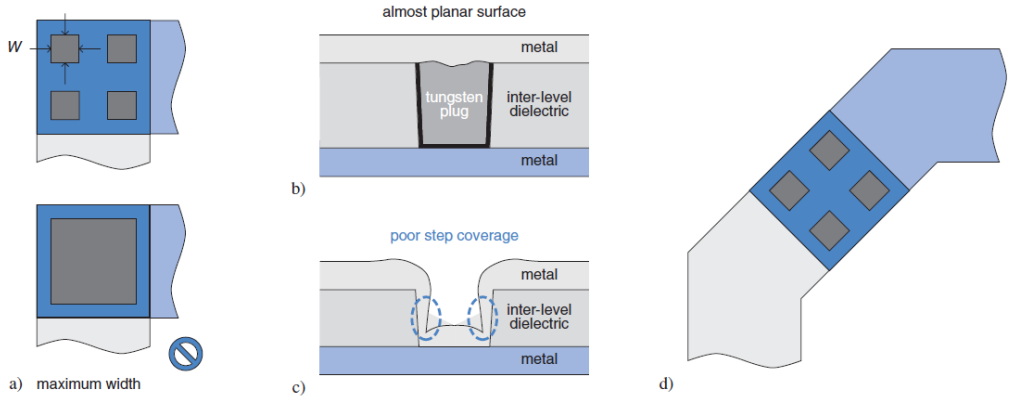

Fig. 11.2 Maximum size rule for contacts and vias. Stipple contact/via versus an oversize single contact/via (a). Cross section of a plugged via (b) and of a historical sink-in via (c). Electromigration-aware stipple contact/via (d), to be explained in section 11.6.1.

Figura 5: Ilustración de la regla de *Maximum Width*, como esta afecta el diseño y la técnica de *stipple contact/via* [13]

Density rules

La densidad planar de una capa de un layout es definida como el área ocupada por todos los elementos en esa capa, dividido el área de toda la capa. Estas normas permiten limitar la ocupación del plano con cota inferior y superior. Por ejemplo se puede definir que la ocupación esté en promedio entre el 20 % y el 80 % por cada milímetro por milímetro de región. Algunas prácticas como llenar de *dummys*, sin conectar, una región en donde la ocupación no se llegue a cumplir resultan útiles en el proceso de diseño.[13]

Extension rules

Si bien estas reglas están mejor definidas en el proceso de *Antenna Rule Checking*, el proceso de **DRC** hace una verificación únicamente de la extensión del material conductor. Estas reglas se detallaran con mayor detenimiento en la verificación de reglas de antena.[13]

Verificación de DRC anterior

En la iteración previa a este proyecto, la cual fue dirigida y ejecutada por Matthias Sibrian y precedida por Luis Nájera el año anterior, cuenta con una metodología específica para realizar el proceso de verificación DRC. El proceso de ejecución de esta verificación esta

segmentado en cuatro pilares fundamentales: Instalar y abrir Custom Compiler, Cargar la celda sintetizada, abrir la vista de *layout* y correr la verificación DRC.

La parte fundamental de este proceso, para el nivel al que se trabajará este proyecto, se centra principiante en la ejecución de la verificación de las reglas de diseño, comprensión, corrección de errores y revalidación. Es un proceso iterativo que requiere consultar la documentación de TSMC y sus restricciones de diseño. Está directamente enlazado con el proceso de síntesis física ya que en este se corrigen los errores que esta verificación presenta. A continuación se muestra un diagrama que muestra el proceso de corrección de errores de DRC.

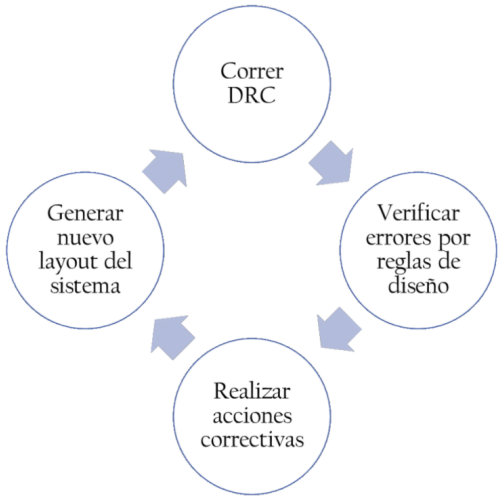

Figura 6: Imagen del proceso de verificación DRC de años anteriores [17]

Debido a problemas con la compatibilidad de IC Validator con IC Compiler I los grupos anteriores se vieron obligados a utilizar Custom Compiler como la herramienta de verificación y su integración con IC Validator. Es por esto que el proceso mostrado debe ser abordado de forma manual, utilizando una interfaz gráfica. Uno de los desafíos que presenta el actual proyecto es utilizar IC Compiler II para poder utilizar IC Validator en consola y abrir paso a una verdadera automatización del flujo de diseño.

6.3.3. ERC

Electrical Rule Check es una etapa de la capa de *singoff* la cual permite verificar que el rendimiento eléctrico del circuito cumpla con las especificaciones del fabricante. Las reglas que verifica esta etapa son las siguientes: [5]

Soft connect check

Esta regla establece si la conectividad (unidireccional) de las difusiones que se encuentran en el *N-Well* y la capa superior es la especificada. [5]

Path check

Esta regla valida lo siguiente: Nodos con un *path* conectado a alimentación pero no tierra, Nodos con un *path* a tierra pero no a alimentación, Nodos sin *path* a tierra o alimentación o Nodos sin un *path* a cualquier otra *net*. [5]

PTAP/NTAP connectivity check

Verifica si el *PTAP* se conecta a alimentación y el *NTAP* a tierra. [5]

MOSFET power and ground check

Verifica que para cualquier **MOSFET**, ya sea tipo N o P, *source* **MOSFET** y *drain* **MOSFET** estén conectados correspondientemente a alimentación y a tierra. [5]

Gate directly connected to power or ground

Esta regla sirve para verificar que exista protección contra descargas electro-estáticas. Si el *gate* de un **MOSFET** tipo P se conecta a alimentación de forma directa podría dañarse el *gate*, así como un **MOSFET** tipo N conectado a tierra directamente. [5]

Floating gate

Verifica si los contactos están interfiriendo con el *poly gate* [5]

Floating well

Verifica si existe alguna conexión entre el *N-Well* o *P-Subrate* y alimentación o tierra. [5]

6.3.4. *Antenna rule checking* [5]

Esta verificación busca evitar que el **CI** tenga efectos de antena que pueden dañar las terminales de los transistores (*gate, drain, source o bulk*) durante el proceso de fabricación. Esto puede ocurrir en el proceso de grabado de los caminos, por medio de rayos plasma o en

la implantación de iones. Estos rayos pueden hacer que se acumulen cargas no deseadas en los cables metálicos, los cuales pueden estar conectadas a nodos de uniones PN. Algunos de los efectos presentes pueden ser: daño de oxido inducido por plasma (efecto antena), también conocido como *Fowler-Nordheim tunneling*, e inducción de corriente por altas temperaturas.[13] Efectos que pueden quemar o reducir el tiempo de vida de un transistor afectando el rendimiento y tiempo de vida del **IC** completo.

Esta verificación al igual que el **DRC** cuenta con un *runset* específico y utiliza el software de IC Compiler II. Las reglas de antena se centran en especificar el área máxima de metal que se puede conectar a un gate sin el source o drain para actuar como elemento de descarga. Las relaciones entre gate y los demás metales son: 100:1 hasta 5000:1 para que la carga sobre el metal no dañe la compuerta. Algunas formas de solucionar estos problemas pueden ser: seccionar el metal en conexiones de metales más pequeños para reducir la potencia del rayo plasma durante la fabricación o la colocación de un diodo entre el metal y tierra para evitar la conducción de corriente y que esta pueda causar una descarga no deseada por las altas temperaturas.

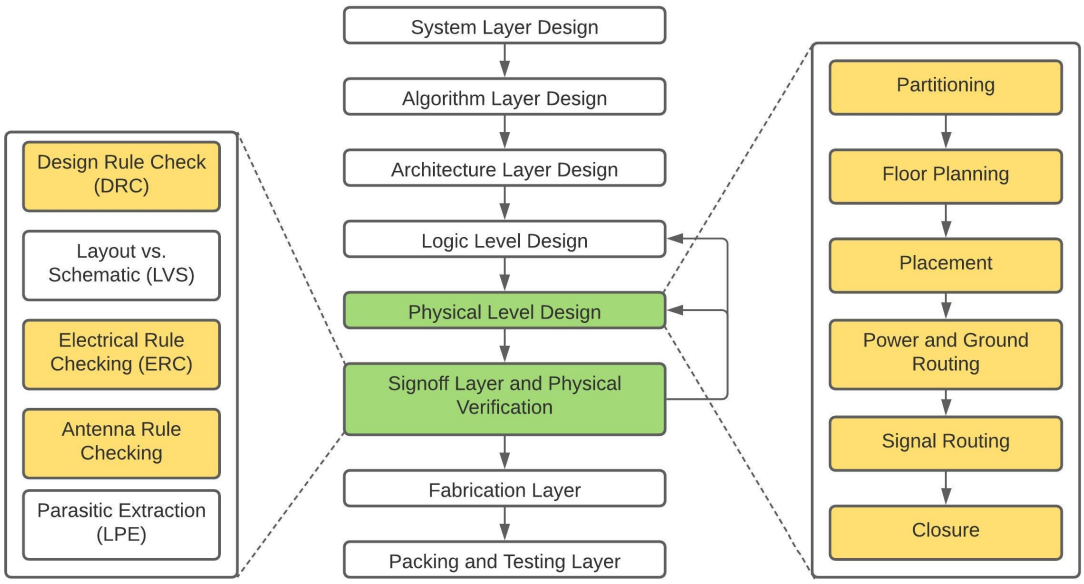

Figura 7: Flujo de diseño que ilustra las partes específicas a realizar de la síntesis física y la etapa de *Signoff* en amarillo

6.4. *Electronic design automation (EDA)*

La industria **EDA** existe con el fin de proporcionar herramientas de software para diseñadores electrónicos, para automatizar los procesos de diseño y fabricación de **CIs**. Esto permite que los precios para la fabricación de estos dispositivos disminuya, además de la especialización en cada una de las herramientas de software y separación del proceso de fabricación con el de diseño. Esto genera equipos de trabajo especializados en el área de diseño

de **CIs** y por otro lado las compañías que fabrican dichos dispositivos pueden proporcionar las restricciones y normas de diseño con las que pueden fabricar los dispositivos. Si bien ambas disciplinas se separan estas convergen con las herramientas **EDA**, siendo el puente en entre los diseñadores y la industria de semiconductores, reduciendo costos, generando equipos especializados y creando una nueva industria en el mundo.[18]

Actualmente las dos compañías más grandes en la industria **EDA** son Cadence y Synopsys. Para fines de esta investigación se estarán utilizando las herramientas **EDA** de Synopsys. Esta cuenta con todas las herramientas para completar el flujo de diseño propuesto en la Figura 3 hasta la etapa de *Signoff and Physical Verification*. Esto se debe a que Synopsys no es una empresa que se dedica a la fabricación de **CIs**. Además como se mencionó anteriormente este es el puente entre el fabricante y el diseñador, por lo que las restricciones de diseño y fabricación han sido proporcionadas por **TSMC** la empresa a la que se mandará a hacer el **CI**, en donde se podrán concluir las últimas dos etapas del flujo de diseño y obtener el **CI** físico.[19]

6.5. Synopsys

Synopsys es una herramienta de diseño y simulación para cada una de las diferentes etapas del flujo de diseño. Estas herramientas son proporcionadas por el departamento de Ing. Electrónica, Mecatrónica y Biomédica de la UVG. Las cuales se aprenden a utilizar en los cursos de Nano Electrónica 1 y 2 son: HSpice, Wave Viewer, IC Compiler, IC Compiler II, Custom Compiler, Verdi3, VCS, Design Vision, Nanotime, IC Validator, StarRC, Formality, Milky Way, entre otras. En los siguientes apartados se desarrolla acerca de las herramientas que son de interés para el desarrollo del proyecto de graduación en lo que serán las etapas de Síntesis Física y en la etapa de *Signoff* y la Verificación Física: **DRC**, **ERC** y *Anttenna Rule Checking* [1]

6.5.1. IC Compiler II

La sucesora a la herramienta IC Compiler I, es una de las herramientas líder para lo que es Place and Route. Esta herramienta es esencial para la síntesis física. En esta herramienta permite procesar el *verilog* proveniente de la síntesis lógica para convertirlo en un *layout* que represente el circuito físico en silicio, el cual será luego verificado.[20]

6.5.2. IC Validator

IC Validator es una de las herramientas de Synopsys para la etapa de *Signoff* y Verificación física. Este software permite realizar **DRC**, *Antenna* y **ERC**, verificaciones necesarias para asegurar que el *layout* generado en la etapa anterior pueda ser fabricado.[21]

CAPÍTULO 7

Proceso de síntesis física

Anteriormente el proceso de síntesis física se realizó en la herramienta de Synopsys IC Compiler I (**ICCI**). Sin embargo, debido a las recomendaciones de los proyectos anteriores y programas actuales que están dejando de utilizarlo de forma progresiva se realizó una migración completa de esta etapa a IC Compiler II (**ICCII**). Además, durante el proceso de migración, se aprendieron nuevas técnicas que permitieron mejorar el flujo de diseño anterior.

7.1. Composición de las librerías estándar *New data model* (NDM)

La primera diferencia significativa entre el proceso de síntesis física anterior y el actual es que la herramienta **ICCII** trabaja con librerías denominadas *New Data Model* o por sus siglas en inglés (**NDM**) y su extensión es *.ndm*. A diferencia de **ICCI** que utiliza la librería *MilkyWay* extensión *.mw*.

Las librerías **NDM** al igual que las *MilkyWay* son una colección de librerías que contienen toda la información de las celdas que se estarán usando dentro de un diseño, esta puede ser tanto una colección para tener todos los archivos de referencia como una librería que contenga toda la síntesis física. Nuestro *foundary*: **TSMC** nos brinda tres tipos de archivos que se utilizarán a lo largo de este proyecto para lo que es la librería de referencia. Estos tres archivos son: *Technology File*, *DB-Technology File* y *Library Exchange Format File*.

7.1.1. *Technology file*

El primer requerimiento para poder empezar a trabajar dentro de la síntesis física es el *Technology File*, también conocido como *techfile* en el entorno de Synopsys, es un archivo de extensión *.tf* que contiene toda la información referente a las reglas específicas que debe seguir el diseño según la tecnología que se está trabajando. Este archivo contiene información como: tamaño de una celda estándar, tamaño de la tecnología que se está trabajando, colores para las diferentes capas de metal y características de los diseños, dimensiones de los metales, extensiones máximas, mínimas, entre otras. En esta investigación se utiliza un *techfile* proporcionado por **TSMC** para una tecnología de 180nm y 6 *layer map* que se refiere a las capas de metal que se están utilizando.

Nombre del archivo:

- ***tsmc018_6lm.tf***

Directorio:

- *TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/-Back_End/milkyway/tcb018gbwp7t_270a/techfiles*

7.1.2. *DB-Technology file*

Tanto las celdas estándar como los *pads* de entradas y salidas cuentan con este tipo de archivo de extensión *.db*. Este archivo es una versión compilada de otro tipo de archivo que también proporciona **TSMC** denominado *Liberty File* extensión *.lib*. Estos dos archivos análogos contienen la información lógica de las celdas y *Pad:* que se van a estar utilizando en la síntesis física. Algunas de estas características lógicas son: timing, área, potencia, voltaje, entre otros.

Nombre de los archivos para celdas estándar:

- ***tcb018gbwp7tbc.db***
- ***tcb018gbwp7tlt.db***
- ***tcb018gbwp7tml.db***
- ***tcb018gbwp7ttc.db***
- ***tcb018gbwp7twc.db***
- ***tcb018gbwp7twcl.db***

Directorio para las celdas estándar:

- *TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/-Front_End/timing_power_noise/NLDM/tcb018gbwp7t_270a*

Nombre del archivo para los *pads* de entradas y salidas:

- ***tpd018nvtc.db***

Directorio para los *pads* de entradas y salidas:

- *TSMC/180/CMOS/G/IO3.3V/iolib/LINEAR/tpd018nv_280a_FE/TSMCHOME/digital/Front_End/timing_power_noise/NLDM/tpd018nv_280a*

7.1.3. *Library exchange format file*

Por el momento se han mencionado únicamente las características lógicas de las celdas y *pads*, sin embargo la representación física de estos es igual de indispensable para la síntesis física. Estos archivos de extensión *.lef* son los que contienen todas las representaciones de las celdas y *pads* en silicio, así como algunas reglas e información de las celdas. Este archivo por ser de **TSMC** únicamente nos permite observar la simplificación de caja negra de las celdas y sus pines en metal 1, así como algún *Routing Blockage* o **RB**. La descripción es legible ya que está en texto plano. A continuación se especifica el archivo y el directorio que se usó para este proyecto:

Nombre del archivo para las celdas estándar:

- ***tcb018gbwp7t_6lm.lef***

Directorio para las celdas estándar:

- *TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/lef/tcb018gbwp7t_270a/lef*

Nombre del archivo para los *pads* de entradas y salidas:

- ***tpd018nv_6lm.lef***

Directorio para los *pads* de entradas y salidas:

- *TSMC/180/CMOS/G/IO3.3V/iolib/LINEAR/tpd018nv_280a_FE/TSMCHOME/digital/Back_End/lef/tpd018nv_280a/mt_2/6lm/lef*

7.2. Creación de librerías con IC Compiler II *library manager tool*

Debido a que los trabajos de años anteriores se basan en librerías *MilkyWay* el proyecto se vio obligado a cambiar desde el inicio. Explorando en la documentación de Synopsys

se encontró una interfaz auxiliar de **ICCII** la cual se llama *Library Manager*. Para poder invocar a esta herramienta se utiliza el comando: ***icc2_ lm_ shell -gui***. Esta herramienta sirve para crear librerías *.ndm* a través de diferentes flujos. El flujo depende de los archivos con los que se cuente y las celdas que uno desee generar en las librerías. Sin embargo todas las librerías siguen la misma metodología para generar una **NDM**. Este proceso se describe a continuación:

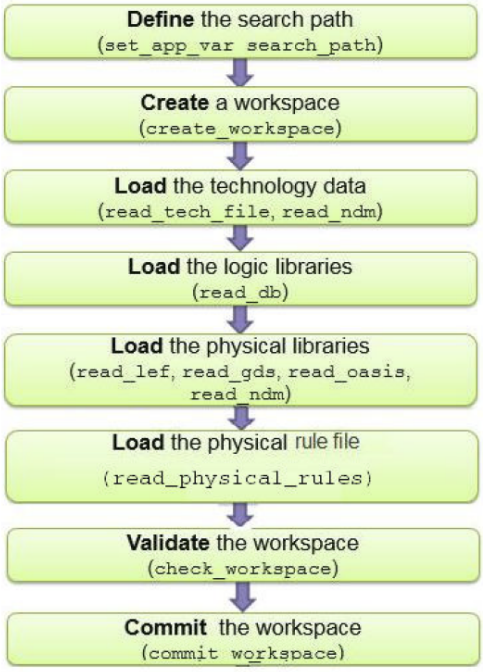

Figura 8: Proceso para crear una librería **NDM** y sus comandos respectivos extraído de: [22]

Para generar la librería de referencia del proyecto se utilizaron 3 flujos distintos: *Normal Flow, Physical Only Flow* y *Aggregate Flow*. Todos estos flujos requieren de un *technology file*, que para fines de este proyecto es el mismo para todas las celdas ya que se está trabajando con la misma tecnología de 180nm. A continuación se describe cada uno de los flujos que se implementaron y los comandos necesarios para ejecutarlos.

7.2.1. *Normal flow*

Este flujo permite crear una librería **NDM** que cuente con: *tech file, db file* y *lef file*. Las celdas generadas en esta **NDM** serán exclusivamente aquellas que su información física tenga asociada una información lógica. En otras palabras que tanto en el *.db* como en el *.lef* exista la misma celda describiendo la información lógica y física respectivamente. Por lo tanto estas celdas contienen tanto información del *layout* como información **PVT**. Si en algún caso no existe una pareja, es decir hay celdas sin alguna de las dos descripciones, estas son expulsadas de la *.ndm*. Estas librerías son útiles para la creación de *pads* de entradas y salidas como las celdas estándar con fines de fabricación y simulación. A continuación la ejecución de las librerías utilizadas en el proyecto en el *shell* del *library manager*.

Ejemplo de un *Normal Flow* para la creación de una **NDM** de celdas estándar:

1. Creamos un *workpace* de flujo normal:

```
> 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
```

2. Importamos los *.db files* donde están las celdas estándar:

```
> 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 }
```

3. Importamos el *.lef file* de las celdas estándar:

```
> 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
```

4. Hacemos el *check* del *workspace* y desplegamos la ventana de errores para verificar que todo esté en orden:

```
> current_workspace; check_workspace
```

```
> gui_create_window -type MessageBrowserWindow
```

```
> open_ems_database check_workspace.ems
```

5. Hacemos *commit* y *save* a la librería *.ndm* con el nombre **StandardWorkspace.ndm**

```
> current_workspace NormalWorkspace; commit_workspace  -output
  StandardWorkspace.ndm
```

Ejemplo de un *Normal Flow* para la creación de una **NDM** de *pads* de entradas y salidas:

1. Creamos un *workpace* de flujo normal:

```
> 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
```

2. Importamos los *.db files* donde están los *pads* de entradas y salidas:

```
> read_db { TSMC/180/CMOS/G/IO3.3V/iolib/LINEAR/tpd018nv_280a_FE/
  TSMCHOME/digital/Front_End/timing_power_noise/NLDM/tpd018nv_280a/
  tpd018nvtc.db }
```

3. Importamos el *.lef file* donde están los *pads* de entradas y salidas:

```
> read_lef TSMC/180/CMOS/G/IO3.3V/iolib/LINEAR/tpd018nv_280a_FE/
  TSMCHOME/digital/Back_End/lef/tpd018nv_280a/mt_2/6lm/lef/
  tpd018nv_6lm.lef
```

4. Hacemos el *check* del *workspace* y desplegamos la ventana de errores para verificar que todo esté en orden:

```
> current_workspace; check_workspace
```

```
> gui_create_window -type MessageBrowserWindow
```

```
> open_ems_database check_workspace.ems
```

5. Hacemos *commit* y *save* a la librería *.ndm* con el nombre **PadsWorkspace.ndm**

```
> current_workspace NormalWorkspace; commit_workspace  -output
  PadsWorkspace.ndm
```

7.2.2. *Physical only flow*

Este flujo, a diferencia del *Normal Flow*, escoge únicamente a aquellas celdas cuya información se encuentre exclusivamente en el archivo *.lef*, es decir solamente toma en cuenta la información física de la celda, como su nombre lo indica. Estas librerías se crearon ya que hay ciertas celdas que son indispensables para la creación del chip como las esquinas y los *non-metal fillers* y que no cuentan con información **PVT** por parte del *foundary*. Estas librerías contienen celdas cuya funcionalidad se ve en el proceso de fabricación, pero no en el proceso de simulación. A continuación se describen las librerías creadas para el proyecto.

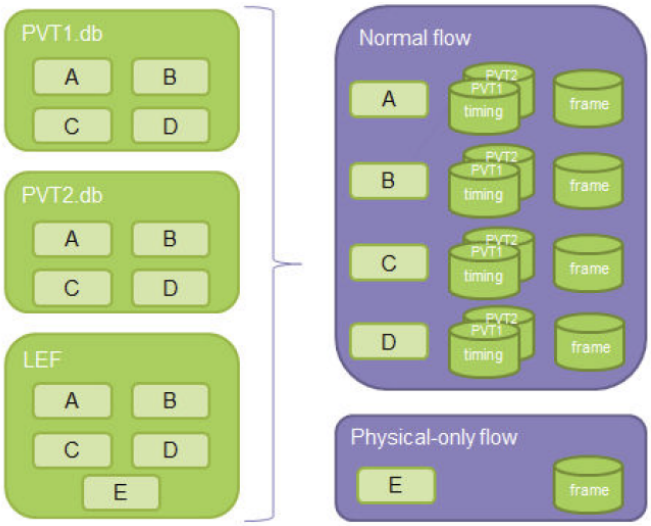

Figura 9: Diagrama que muestra como se crean las diferentes **NDM** según la información y el flujo utilizado, extraído de: [22]

Ejemplo de un *Physical Only Flow* para la creación de una **NDM** de celdas estándar que contengan los *non-metal fillers*:

1. Creamos un *workpace* de flujo únicamente físico:

```
> create_workspace -flow physical_only -technology usr/synopsys/
TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital
/Back_End/milkyway/tcb018gbwp7t_270a/techfiles/tsmc018_6lm.tf
PhysicalOnlyWorkspace
```

2. Importamos los *.db files* donde están las celdas estándar:

```
> 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 }
```

3. Importamos el *.lef file* de las celdas estándar:

```
>       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
```

4. Hacemos el *check* del *workspace* y desplegamos la ventana de errores para verificar que todo esté en orden:

```
>       current_workspace; check_workspace
```

```
>       gui_create_window -type MessageBrowserWindow
```

```
>       open_ems_database check_workspace.ems
```

5. Hacemos *commit* y *save* a la librería *.ndm* con el nombre **FillersWorkspace.ndm**

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

Ejemplo de un *Physical Only Flow* para la creación de una **NDM** *pads* de entradas y salidas que contengan los *I/O non-metal fillers* y las esquinas:

1. Creamos un *workpace* de flujo únicamente físico:

```
>       create_workspace -flow physical_only -technology usr/synopsys/
TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital
/Back_End/milkyway/tcb018gbwp7t_270a/techfiles/tsmc018_6lm.tf
PhysicalOnlyWorkspace
```

2. Importamos los *.db files* donde están los *pads* de entradas y salidas:

```
> read_db { TSMC/180/CMOS/G/IO3.3V/iolib/LINEAR/tpd018nv_280a_FE/
TSMCHOME/digital/Front_End/timing_power_noise/NLDM/tpd018nv_280a/
tpd018nvtc.db }
```

3. Importamos el *.lef file* donde están los *pads* de entradas y salidas:

```
> read_lef TSMC/180/CMOS/G/IO3.3V/iolib/LINEAR/tpd018nv_280a_FE/
TSMCHOME/digital/Back_End/lef/tpd018nv_280a/mt_2/6lm/lef/
tpd018nv_6lm.lef
```

4. Hacemos el *check* del *workspace* y desplegamos la ventana de errores para verificar que todo esté en orden:

```
> current_workspace; check_workspace
```

```
> gui_create_window -type MessageBrowserWindow
```

```
> open_ems_database check_workspace.ems
```

5. Hacemos *commit* y *save* a la librería *.ndm* con el nombre **CornersWorkspace.ndm**

```
> current_workspace PhysicalOnlyWorkspace; commit_workspace  -
output CornsersWorkspace.ndm
```

7.2.3. *Aggregate flow*

Una vez creadas estas cuatro librerías *.ndm* se debe crear una única librería a la que se le refiere en el proceso de la síntesis física como *reference library*. Esta librería cuenta con toda la información necesaria para poder hacer un mapeo de las celdas de la síntesis lógica, entre otras, necesarias para realizar la síntesis física. Un *Aggregate Flow* es una compilación de múltiples librerías *.ndm* referidas en una única librería. A continuación se crea la librería usada en este proyecto:

Ejemplo de un *Aggregate Flow* para la creación de una **NDM** que contenga a las cuatro librerías anteriormente creadas.

1. Creamos un *workspace* de flujo agregado:

```
> create_workspace -flow aggregate AggregateWorkspace
```

2. Importamos las *.ndm* previamente creadas:

```
> 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
```

4. Hacemos el *check* del *workspace* y desplegamos la ventana de errores para verificar que todo esté en orden:

```
> current_workspace; check_workspace
```

```
> gui_create_window -type MessageBrowserWindow}
```

```
> open_ems_database check_workspace.ems}
```

5. Hacemos *commit* y *save* a la librería *.ndm* con el nombre **TSMCWorkspace.ndm**

```
> current_workspace AggregateWorkspace; commit_workspace -output
TSMCWorkspace.ndm
```

7.3. Síntesis física en IC Compiler II *tool*

Una vez finalizada la librería de referencia, el proceso de síntesis física puede iniciar. Para que esto se pueda efectuar de forma correcta se debe seguir una línea de trabajo ordenada. La herramienta de **ICCII** cuenta con un *Task Assistant* el cual segmenta, organiza y guía al usuario por el proceso de síntesis con una interfaz gráfica. Sin embargo, no todas las opciones para el proceso de este proyecto se encuentran dentro de esta guía, por lo tanto,

el apoyarse con los manuales de guía de usuario[15] y guía de diseño [14] es un excelente forma de complementar a esta interfaz. Adicionalmente el manual de comandos de **ICC2** [23] y las opciones dentro de la aplicación [24] proporcionan mayor detalle y algunos ejemplos complementarios a las guías de usuario. El comando para invocar a **ICCII** desde consola es: `icc2_shell -gui`

7.3.1. Creando un bloque de trabajo

Para poder iniciar a trabajar en el proceso de síntesis física primero se debe crear una librería **NDM** en donde se almacenará toda la información del bloque que se estará creando. Luego se le debe cargar el archivo de síntesis lógica de extensión verilog *.v* que contiene la información de la interconexión de las celdas estándar y los *pads* de entradas y salidas. Es por esto que es de suma importancia la librería de referencia contenga toda la información de estas celdas, tanto física como lógica. Si alguna de estas celdas no existiese se presentarían errores desde un comienzo. A continuación se describe como realizar este proceso en **ICCII**.

Ejemplo de la creación de un bloque y la importación de librerías necesarias para la síntesis física asociadas a este, de una compuerta NOT.

1. Creamos la librería **NDM** que almacenará todas las cosas que se realicen en la síntesis física y le asociamos su librería de referencia. (La librería de referencia es la que fue creada en la sección anterior). El nombre de dicha librería es **NOT_SYN.ndm**

```
> 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
```

2. Se lee el archivo verilog de la compuerta NOT que ya pasó por el proceso de síntesis lógica. El nombre de este archivo es **NOT_IO_syn.v**. A partir de aquí se crea un bloque con el nombre del módulo de mayor jerarquía del archivo de verilog. En este caso el nombre de este módulo es **NOT_IO**. Esta información será relevante más adelante.

```
> read_verilog /mnt/nfs/compartida/NOT/SintesisLogica/
  salidas_not_io/NOT_IO_syn.v
```

3. Se lee el archivo de extensión *.sdc* para importar los requerimientos de diseño impuestos en la síntesis lógica. Este archivo se llama **NOT_IO.sdc**. Se le quitó el argumento de `-syntax_only` ya que ese no incluía los requerimientos, únicamente revisaba la validez del archivo *.sdc*

```
> read_sdc -echo /mnt/nfs/compartida/NOT/SintesisLogica/
  salidas_not_io/NOT_IO.sdc
```

4. Por último, se importan los archivos TLU+, estos contienen toda la información de los coeficientes de resistencia y capacitancia, esenciales para la extracción de parásitos. Adicionalmente se importa en conjunto el *layermap* que es una descripción de las características de la cantidad de capas de metal que el diseño tiene.

```
> read_parasitic_tech -tlup /usr/synopsys/TSMC/180/CMOS/G/stclib
/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/
tcb018gbwp7t_290a/tluplus/t018lo_1p6m_typical.tluplus -layermap /usr
/synopsys/TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/
TSMCHOME/digital/Back_End/milkyway/tcb018gbwp7t_290a/tluplus/star.
map_6M
```

Figura 10: Instancias del archivo verilog en el bloque de trabajo

7.3.2. Instanciar *pads* de alimentación y esquinas

Antes de iniciar con el proceso de *floorplaning* es indispensable que todas las celdas que se van a utilizar en las entradas y salidas estén definidas. Por lo tanto en esta etapa se definen los *pads* de VDD y VSS que no se definieron en la síntesis lógica. Adicionalmente, las esquinas del chip se deben definir, esto ya que son las únicas celdas de entradas y salidas que son cuadradas y le otorgan orden, simetría al chip. Además carecen de conexiones físicas en el circuito por la posición en donde se encuentran.

A continuación se instancian las celdas de VDD y VSS, así como las esquinas de una NOT.

1. De primero instanciamos los *pads* de entradas y salidas. Estos provienen de la librería de referencia, por lo que es indispensable que estén definidos en ella. En nuestro caso

se encuentran dentro del flujo normal *PadsWorkspace.ndm* dentro de *TSMCWorkspace.ndm* y los nombres de estas celdas son: **PVDD1CDG** y **PVSS1CDG**. Los nombres de cada instancia son: **PVDD** y **PVSS**, respectivamente, para distinguirlos de las nets de alimentación.

```
> create_cell {PVDD} PVDD1CDG
```

```
> create_cell {PVSS} PVSS1CDG
```

2. Definimos las esquinas del chip, estas las definimos como **CORNER1, CORNER2, CORNER3** y **CORNER4** y provienen de la librería de referencia *TSMCWorkspace.ndm* en la librería *CornersWorkspace.ndm*. El nombre de la celda que se está utilizando es: **PCORNER**.

```
> create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER
```

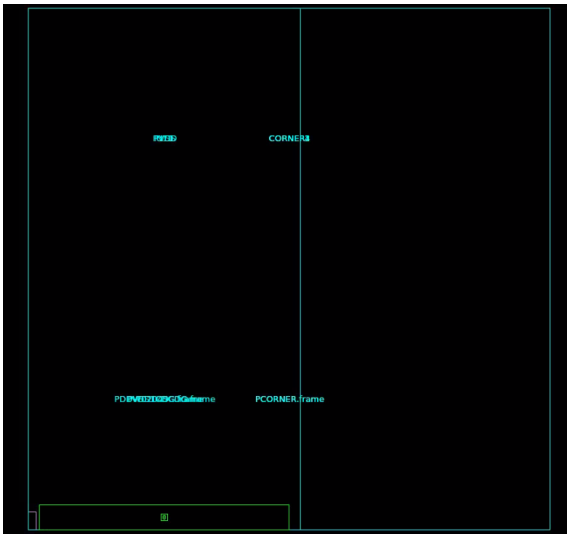

Figura 11: Instancias de las esquinas y *pads* de VDD y VSS

7.3.3. Creación de PG *nets*

Debido a que en la síntesis lógica los *pads* de alimentación no existen, es crítico que se defina una conexión lógica de todos los pines de las celdas con esa nomenclatura entre sí. Asimismo es importante que estas conexiones estén dirigidas hacia los pines de acople con los *pads* de entradas y salidas. De omitir este paso las conexiones de **PG** causan problemas de **DRC** ya que no son celdas de una misma *Net:*.

Flujo para definir las conexiones lógicas de las celdas, *pads* y pines de **PG**. Extraído del manual de usuario *Design Planning User Manual* [14].

1. Debido a que no estamos haciendo un diseño que tenga un *Unified Power Format* **UPF**, sino que el flujo esta basado en un archivo de verilog, entonces el siguiente comando deriva dos conexiones por defecto con los nombres de **VDD** y **VSS** los cuales y los establece como fuentes de alimentación principales.

```
> resolve_pg_nets
```

2. En seguida especificamos los nombres de las *nets* de **PG** que vamos a crear.

```
> create_net -power VDD
```

```
> create_net -ground VSS
```

3. Luego de crear estas **nets**, se deben asignar a todos los pines, para esto el siguiente comando agarra todos aquellos elementos físicos cuya interconexión se llame **VDD** o **VSS** y los asocia a las **nets** que se crearon.

```
> connect_pg_net -net VDD [get_pins -physical_context *VDD]
```

```
> connect_pg_net -net VSS [get_pins -physical_context *VSS]
```

4. Luego se debe asegurar que todas aquellas celdas que por algún motivo el programa obvió, se asocien a estas *nets*.

```
> connect_pg_net -automatic
```

5. Por último se solicita un reporte de estas interconexiones para validar que estas existan y esten asociadas a una *net*.

```
> report_cells -power
```

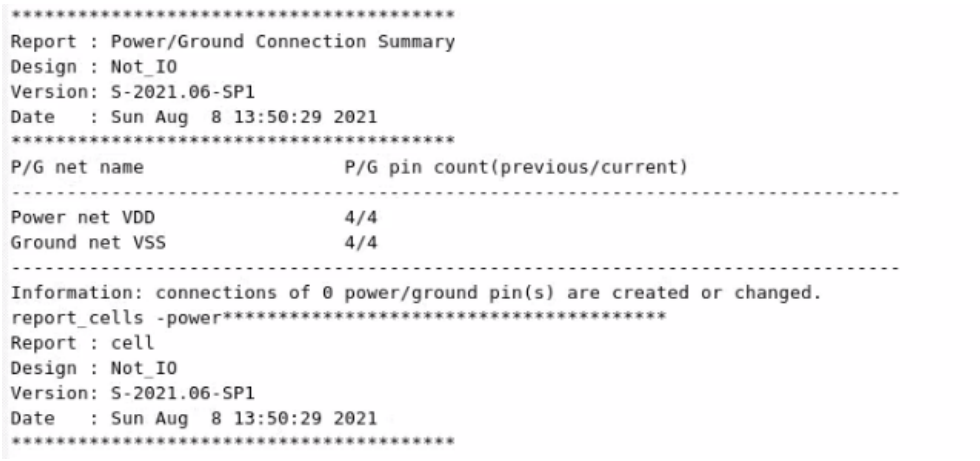

Figura 12: Resumen de consola del reporte de conexiones con 4 de 4 conexiones exitosas de **PG**

7.3.4. Creación del *FloorPlan* inicial con anillo de entradas y salidas

Una vez instanciadas todas aquellas celdas que se están utilizando para el diseño, así como todas aquellas conexiones lógicas entre ellas, se puede empezar a crear un *FloorPlan* y el anillo de Entradas y salidas. El *FloorPlan* es la parte del proceso de diseño en donde se define el espacio físico en donde se estarán posicionando los elementos, tanto del núcleo del **CI** como el espacio designado para los *pads* de entradas y salidas. El anillo de entradas y salidas define el espacio en donde estarán encerrados los pads de entradas y salidas. Son guías virtuales que luego pueden ser utilizadas como directrices de posicionamiento para los pads.

1. El siguiente comando tiene múltiples argumentos. En conjunto estos crean el *FloorPlan*.

```
> initialize_floorplan -site_def unit -use_site_row -keep_all -
  side_length {100 100} -core_offset {125}
```

   a. El primer argumento `-site def unit` define el tamaño de una celda básica. En el *techfile* está definido este argumento que es de $3.92\mu m$ bajo el nombre de **unit**. Esto se hace ya que el núcleo debe ser un múltiplo de este argumento para generar filas exactas. Por ejemplo, en este caso se define un núcleo de $100 * 100\mu m^2$ sin embargo el programa aproxima a $98 * 98\mu m^2$ debido a que son 25 filas exactas de $3.92\mu m$.
   b. El segundo argumento `-use_site_row` especifica que se desea utilizar la dimensión **unit** pra definir las filas del core.
   c. El tercer argumento `-keepall` mantiene todas aquellas celdas que se instanciaron, esto para no deshacer procesos anteriores que se encuentren sobre la misma área de trabajo.
   d. El cuarto argumento `-side_length {100 100}` define un núcleo con las dimensiones de $100 * 100\mu m^2$.

e. El último argumento `-core_offset {125}` es la distancia a la que se encuentra el borde del núcleo al borde del margen externo.

2. Se define el anillo de entradas y salidas en el margen externo del **CI** además de especificarle que la altura de las esquinas es de $115\mu m$. El nombre de la instancia del anillo es: **anillo_IO**.

```
> create_io_ring -name anillo_IO -corner_height 115
```

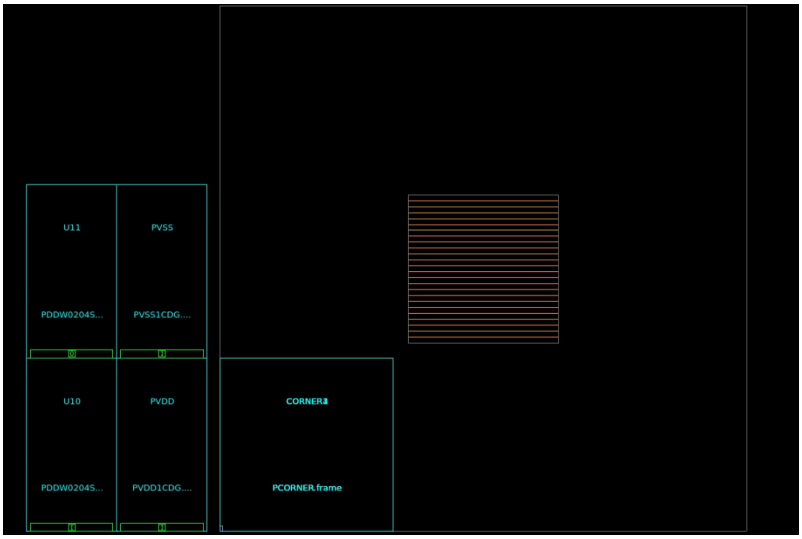

Figura 13: *FloorPlan* con las *site rows* desplegadas y el anillo creado al rededor del margen externo

7.3.5. IO *Placement* y creación del anillo de PG

Para poder crear conexiones funcionales entre los *pads* de alimentación y el circuito es necesario crear un **PG** *ring*; este es un anillo que sirve como puente entre las celdas y los *pads* de alimentación. Para esto primero hay que posicionar los pads en el anillo de IO y luego crear el anillo.

A continuación se muestra un ejemplo de un *placement* de IO así como la creación de un anillo de entradas y salidas para una compuerta NOT.

1. Este comando es el indicado cuando se desea hacer el *placement* de los *pads* de entradas y salidas. Cabe resaltar que como esta es una compuerta NOT no se requirió de ninguna especificación adicional. A diferencia de otros casos que se presentarán más adelante.

```
> place_io
```

2. Los siguientes comandos son necesarios para crear y definir el anillo de **PG**. De primero se crea un patrón de anillo. Este patrón se define de forma que los *tracks* horizontales estén en metal 2 y los *tracks* verticales en metal 3. Esta decisión se tomó ya que los pads de **VDD** y **VSS** cuentan con sus pines en metal 2, lo que en un futuro evitará cortos en las redes de alimentación. El espaciado y las distancias entre *straps* fueron extraídas de los ejemplos del *Task Assistant*

```
> create_pg_ring_pattern ring_pattern -horizontal_layer METAL2
 -horizontal_width {2} -horizontal_spacing {2}    -vertical_layer
METAL3 -vertical_width {2} -vertical_spacing {2}
```

3. Luego se crea una estrategia. Las estrategias utilizan patrones para luego ser compiladas. Esta estrategia se crea en el anillo del núcleo, utiliza la estrategia definida anteriormente y la replica 2 veces, una para la net **VDD** y la otra es para la net **VSS** de forma concéntrica. Adicionalmente estas están separadas del núcleo por $1\mu m$

```
> set_pg_strategy core_ring    -pattern {{name: ring_pattern}
{nets: {VDD VSS}} {offset: {1 1}}} -core
```

3. Por último se compila la estrategia definida para que esta sea colocada en el diseño.

```
> compile_pg -strategies core_ring
```

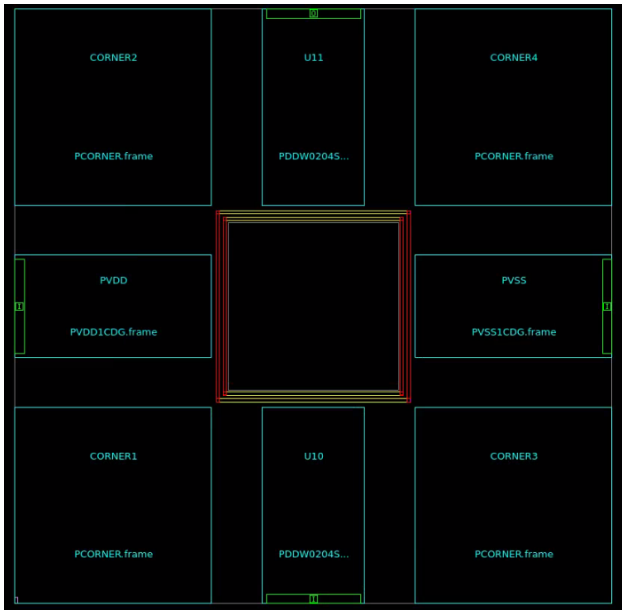

Figura 14: Colocación del anillo de **PG** y *placement* de los *pads* en el anillo de IO

7.3.6. *Cell placement* y legalización

Para poder crear un *placement* de celdas exitoso es importante que siga el flujo descrito a continuación.

1. Esta opción se coloca en **FALSE** para que cuando se haga el *placement* este se ejecute y no coloque las celdas en posiciones fijas por si requieren de algún movimiento. Si en algún caso se desean fijas, esto se coloca en **TRUE**. Además se colocan los bloqueos como automáticos. Esto se hace en dado caso sean necesarios, el diseñador no tenga que especificarlos manualmente.

```
> set_app_options -name place.coarse.fix_hard_macros -value false
```

```
> set_app_options -name plan.place.auto_create_blockages -value
  auto
```

2. Se crea el *FloorPlan* con las opciones de: optimizar para *timing*, congestiones y que el esfuerzo sea el máximo posible. Para diseños pequeños y sin relojes, no hay mucha diferencia que estos argumentos estén sin embargo, son muy útiles y es preferible dejarlos.

```
> create_placement -floorplan -timing_driven -congestion -effort
  high -congestion_effort high
```

3. Esta es la última parte del flujo del *placement*. Aquí se efectúan todas las descripciones anteriores y se colocan las celdas. Todo esto se registra en la **NDM** del proyecto.

```
> legalize_placement
```

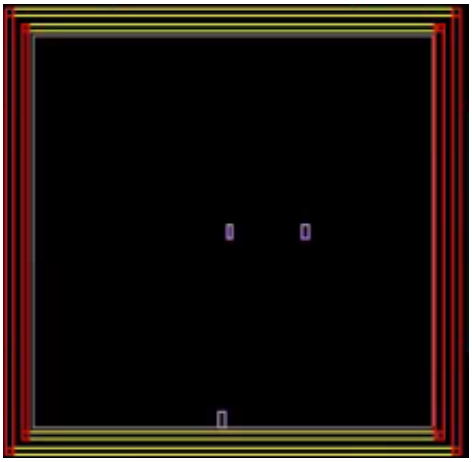

Figura 15: *Cell Placement* del **CI** vista únicamente del core

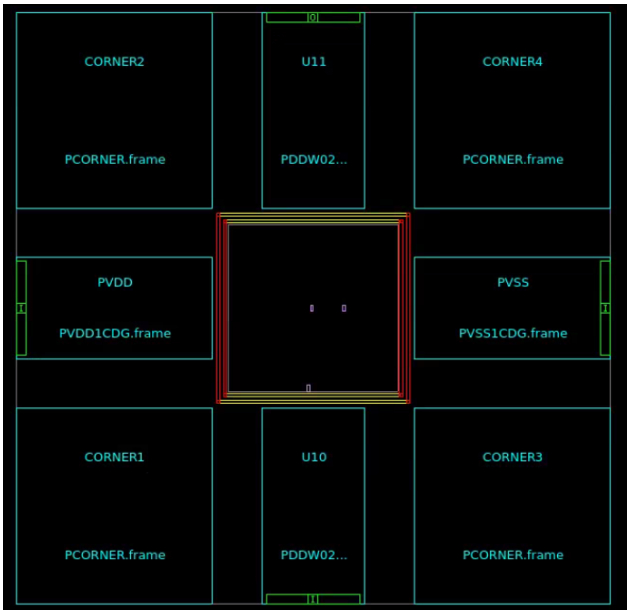

Figura 16: *Cell Placement* con vista de todo el **CI**

7.3.7. Conexión del anillo de PG con los *pads* de alimentación

Hasta este punto únicamente se ha creado un anillo de alimentación sin embargo es necesario que los pads de alimentación sean conectados a dicho anillo. Este paso es un agregado proceso de síntesis física anterior. El proceso de desglosa a continuación.

1. Se crea un **PG** *Pattern*, este sirve para dar instrucciones de en que metal y en que orientación se colocan los metales para el proceso que prosigue. Para el caso de estos *pads* es necesario colocarlos en METAL 2 ya que los demás metales tienen bloqueos en la vista de diseño del *pad*. Asimismo se especifica que la conexión que se hará es con un pin y las nets involucradas son VDD y VSS.

```
> create_pg_macro_conn_pattern hm_pattern -pin_conn_type
scattered_pin -layers {METAL2 METAL3} -nets {VDD VSS} -pin_layers {
METAL2}
```

2. Se configura la herramienta para que trate a los *pads* de **PG** como macros y así poderlos conectar por medio del flujo actual.

```
> set_app_options -name plan.pgroute.treat_pad_as_macro -value
true
```

3. Se crea la estrategia asociada a las conexiones macro previamente definidas. Se le especifica que celdas son las macro, que en este caso son los pads de alimentación.

Adicionalmente se especifica el patrón que fue creado con anterioridad y las nets a las que esta estrategia se conectará. El * seguido del los *pads* indica si existen más instancias con este mismo nombre también las toma en cuenta para la conexión.

```
> set_pg_strategy macro_conn -macros [get_cells {PVDD* PVSS*}] -
pattern {{name: hm_pattern} {nets: {VDD VSS}}}
```

4. Antes de compilar la estrategia se debe definir un conjunto de reglas para las vías que se colocarán. Para este proyecto se extrajo dicha instrucción de un repositorio de ejemplos del *Task Assistant* de la herramienta de **ICCII**.[15].

```
> 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}}}
```

5. Por último se compila la estrategia deseada en el diseño, para que se coloque todo según fue indicado. Tanto la estrategia que depende del patron, como de las reglas de vías.

```
> compile_pg -strategies macro_conn -via_rule macro_conn_via_rule
 -tag test
```

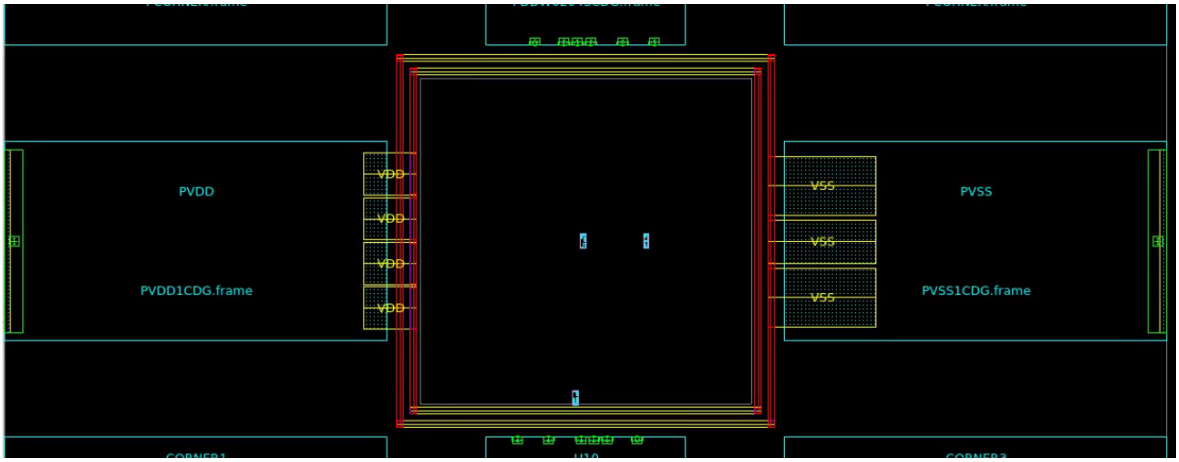

Figura 17: Conexión de los *pads* de **PG** con el anillo de alimentación vista aumentada

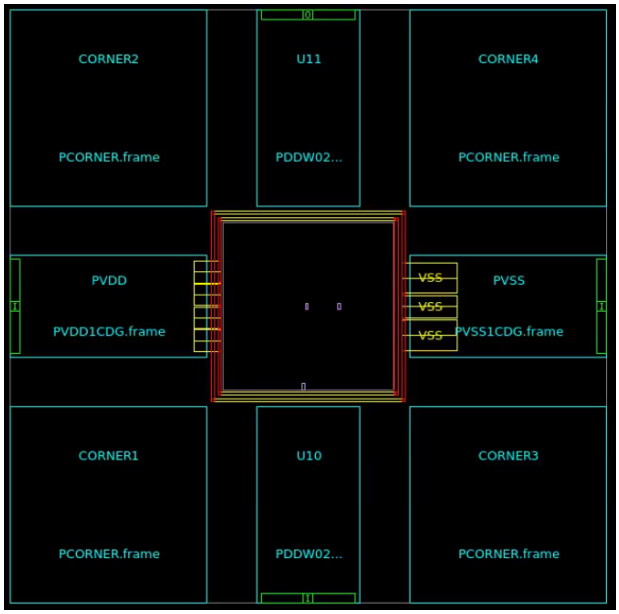

Figura 18: Conexión de los *pads* de **PG** con el anillo de alimentación vista de todo el **CI**

7.3.8. Creación del *mesh* y los *cell rows*

Una vez creada la estrategia de alimentación fuera del núcleo se crea la estrategia interna del núcleo. Para esto se hace un *Mesh:* que crea *cell rows* de ancho de una *cell unit* la cual fue definida en el *FloorPlan*. En la capa de METAL1 se crean lineas horizontales dentro de la región del núcleo especificada. Adicionalmente se colocan *straps* en dirección vertical en METAL 3 para interconectar tanto el anillo como las *cell rows*. Es por esto que en el anillo se escogen esos metales, para que no causen cortos entre las *nets* de VDD y VSS.

1. Se define el patrón de los *straps* y el metal en el que trabajan, así como su ancho.

```
> create_pg_mesh_pattern mesh_pattern -layers { {{vertical_layer:
  METAL3} {width: 4.2} {pitch: 42} {spacing: interleaving}} }
```

2. Se define el área en donde se instanciará el *mesh* y los *straps* así como las celdas involucradas.

```
> set_pg_strategy mesh_strategy -polygon {{125.000 118.000}
  {224.670 230.000}} -pattern {{pattern: mesh_pattern}{nets: {VDD VSS
  }}} -blockage {macros: all}
```

3. Se define un patrón de conexión con el patrón definido anteriormente.

```
> create_pg_std_cell_conn_pattern std_cell_pattern
```

4. Se crea un patrón de *cell rows* para conectar las celdas estándar en el núcleo. Se coloca unicamente en el núcleo para que este no estorbe las conexiones de los *pads* de **PG**.

```
> set_pg_strategy std_cell_strategy -core -pattern {{pattern:
std_cell_pattern}{nets: {VDD VSS}}}
```

5. Se compila todo lo definido anteriormente para obtener como resultado el *mesh*.

```
> compile_pg
```

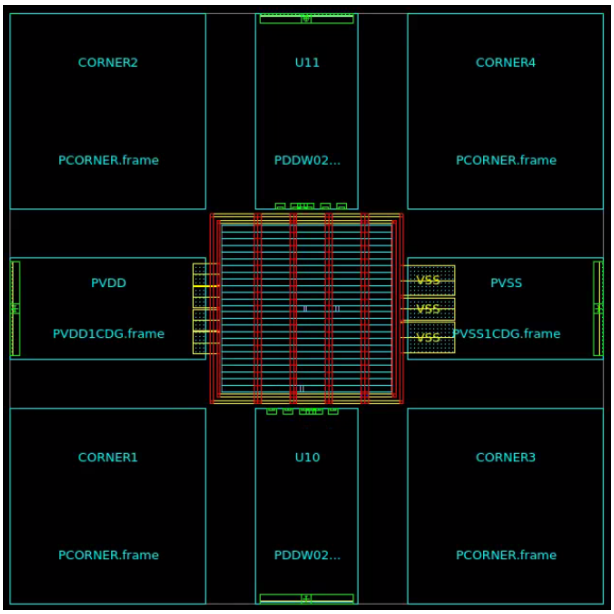

Figura 19: Conexión de las celdas estándar al anillo por medio de un *mesh* utilizando *cell rows* y *straps*

7.3.9. Conexión de PG *nets* y unión del PG *planning*

Luego de realizar el *mesh* y los *cell rows* para alimentar a las celdas estándar y conectarse con el anillo de **PG** se hace una conexión de todas las celdas, del *mesh*, los *straps* y las celdas.

1. De primero se conectan todas las celdas de alimentación que pudiesen estar desconectadas con su respectiva *net*.

```
> connect_pg_net -automatic
```

2. En seguida se hace una fusión del anillo con el *mesh* para que estos estén en el mismo contexto de alimentación. De no ser así, esto causaría problemas de **LVS** de corto circuito.

```
> merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {
  METAL2 METAL3}
```

7.3.10. Ruteo

Para esta parte se utiliza la aplicación de Z-Route que trae la herramienta de **ICCII**. A continuación se describe el flujo para hacer un ruteo exitoso.

1. Primero se revisa si el programa puede hacer todas las conexiones con el *placement* actual. El argumento que tiene colocado es para que pueda verificar también si algún puerto tiene problemas de acceso por algún bloqueo generado o proveniente de alguna celda.

```
> check_routability -check_pg_blocked_ports true
```

2. Luego se hace una verificación hasta la etapa actual, la cual corre diferentes procesos internos, pero para fines de este proyecto habilita el poder hacer el ruteo. Si alguna verificación falla esta puede ser observada desde el *Error Browser* de lo contrario devuelve un 1.

```
> check_design -checks pre_route_stage
```

3. Por último se ejecuta el *routing* para que todos los pines se conecten con sus respectivas nets.

```
> route_auto
```

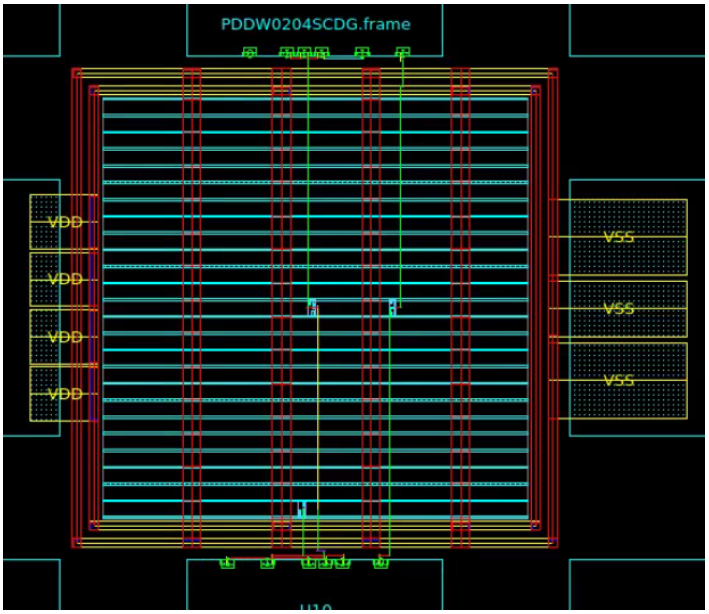

Figura 20: Ruteo de celdas estándar y pads de entradas y salidas vista del núcleo.

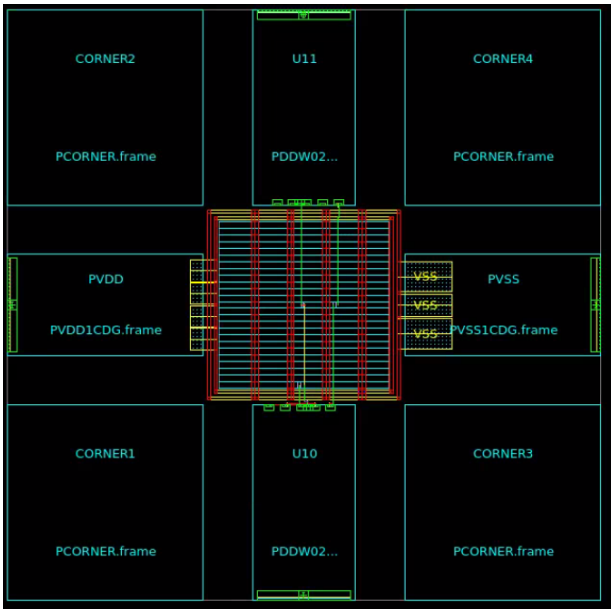

Figura 21: Ruteo de celdas estándar y pads de entradas y salidas vista completa.

7.3.11. Creación de *filler cells* para los *IO*

Estas celdas sirven para llenar cualquier espaciado que se encuentre entre los pads de entradas y salidas. Su función principal es evitar Corto circuito: y cumplir con los requerimientos de fabricación. Estas celdas no tienen metal.

1. Creamos los *fillers* correspondientes a los puertos de entradas y salidas. Estos se encuentran en las *frame only* de la librería de pads de entradas y salidas dentro de la librería de *TSMCWorkspace.ndm*.

```
> 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}
```

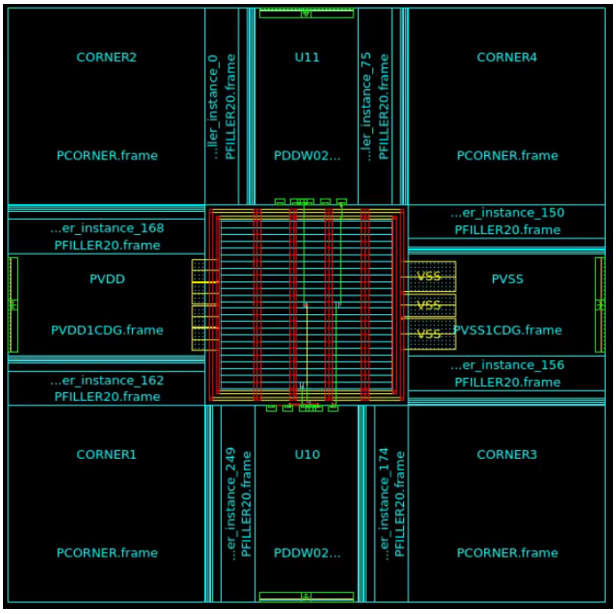

Figura 22: *Fillers* correspondientes a los pads e entradas y salidas colocados.

7.3.12. Creación de *filler cells* para el *core*

Estas celdas son parte del flujo de diseño de un **CI**. Sirven para llenar el espacio que otras celdas no ocupan dentro del núcleo y así evitar problemas en el proceso de fabricación. Estas celdas no tienen metal.

1. Se colocan los *cell fillers* dentro del núcleo para que se cumplan los requerimientos de disñeo. Estos están dentro de la librería *frame only* de las celdas estándar dentro de la librería *TSMCWorkspace.ndm*.

```
> 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}]
```

2. Se vuelven a realizar todas las conexiones de alimentación. Debido a que se insertaron los *fillers* se espera que existan nuevas conexiones para estas celdas. Si esto no se hace se presentan errores de corto circuito en la verificación de **LVS**

```
> connect_pg_net -automatic
```

3. Se indica si algún *filler cell* tiene violaciones, es decir: corto circuito, sobre-posición de celdas, etc. que sean removidos del diseño. En consola se puede observar si se hizo alguna modificación al diseño. Es importante hacerlo en este punto ya que si se hace antes muchas celdas tendran violación ya que no estarán asignados sus *pads* de **PG**.

```
> remove_stdcell_fillers_with_violation
```

4. Se revisa que todo este en orden hasta este punto. Si algo está mal lo desplegará en este resumen. Este se puede ver con mas detalle en el *Error Browser* de **ICCII**.

```
> check_legality
```

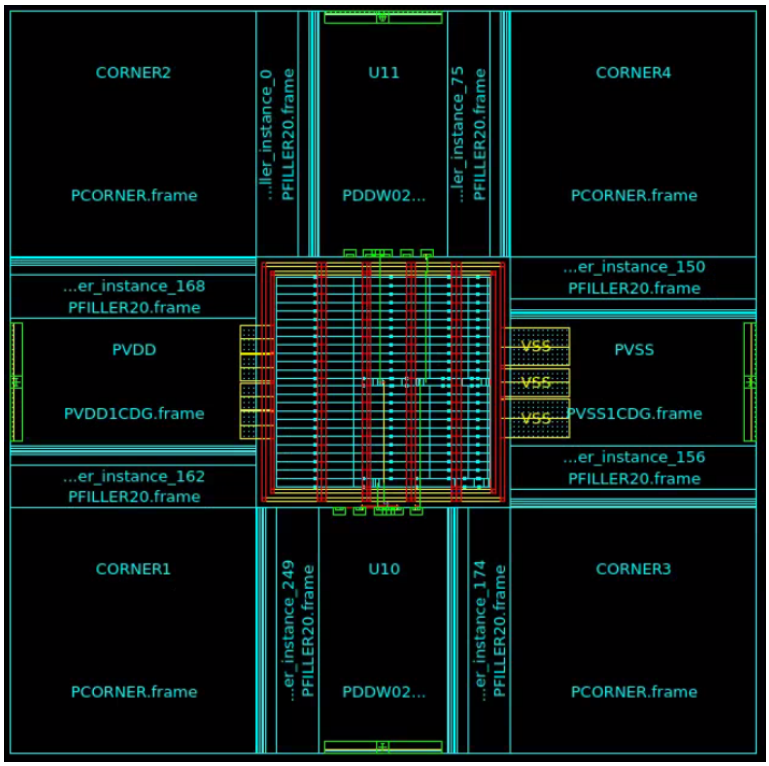

Figura 23: *Fillers* correspondientes a las celdas estándar dentro del núcleo con vista completa del **CI**.

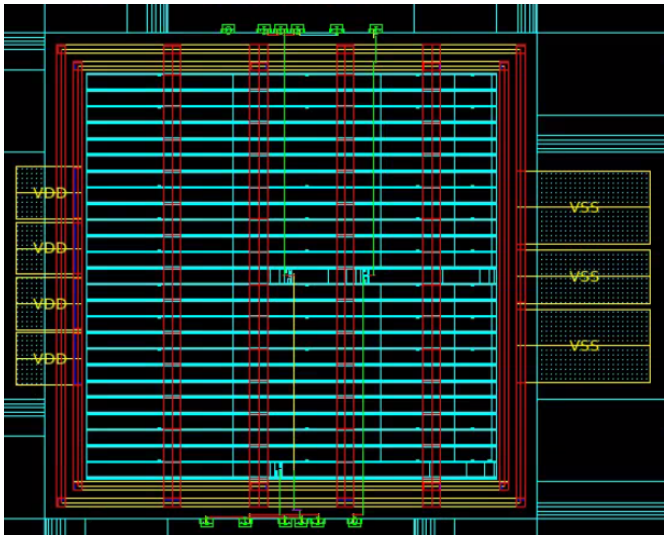

Figura 24: *Fillers* correspondientes a las celdas estándar dentro del núcleo.

```
****************************************
Report : Power/Ground Connection Summary
Design : Not_IO
Version: S-2021.06-SP1
Date   : Wed Aug 11 00:18:37 2021
****************************************
P/G net name                  P/G pin count(previous/current)
--------------------------------------------------------------------------------
Power net VDD                 4/141
Ground net VSS                4/141
--------------------------------------------------------------------------------
Information: connections of 274 power/ground pin(s) are created or changed.
```

Figura 25: Reporte de todas las conexiones nuevas realizadas por haber instanciado a todos los *fillers*.

7.3.13. Exportación de archivos de interés

El primer archivo de interés es la librería **NDM** que contiene toda la información del *layout*, una copia de las librerías de referencia y otra información importante como el archivo de tecnología, entre otros.

1. Para poder guardar la librería **NDM** la sintaxis es la siguiente: [NOMBRE.ndm]:[Nombre del módulo de más alta jerarquía en el archivo verilog]. El nombre debe coincidir con el nombre de la librería creada al inicio del proceso de síntesis física de lo contrario el comado falla.

```
> save_block NOT_SYN.ndm:Not_IO
```

La librería creada será una carpeta dentro del directorio de trabajo en donde se instanció la terminal. A continuación algunas imágenes de referencia:

Figura 26: **NDM** resultante con la síntesis física de una **ALU**.

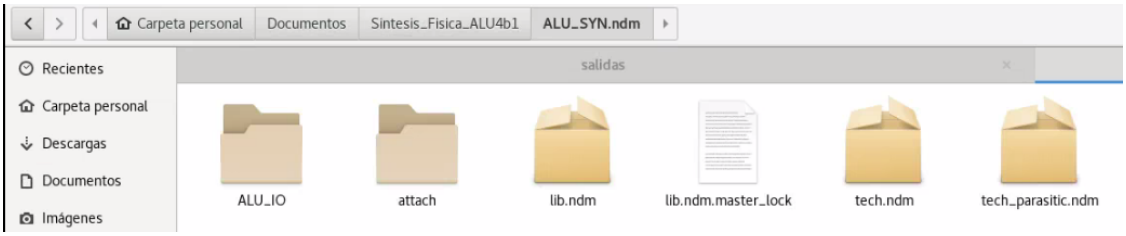

Figura 27: Interior de la librería **NDM** con los archivos correspondientes a la librería.

El segundo archivo de interés es el archivo verilog correspondiente al final de la síntesis. Este archivo es importante ya que durante el proceso de síntesis física instancias de *pads, filler cells* e *IO fillers* se colocan en el diseño.

1. Para poder generar el archivo verilog se corre el siguiente comando con el argumento `-include all` para que este archivo incluya todas las instancias generadas, *tracks*, etc...

```
> write_verilog -include all NOT_SYN.v
```

Figura 28: Archivo verilog generado dentro del directorio de trabajo.

Por último el proceso que concluye toda la síntesis física es la generación del archivo GDSII. Este archivo es indispensable ya que es el que se utiliza para validar el diseño en

la etapa de *SignOff*. El archivo *.gds* es un resumen de todas las conexiones físicas y las instancias de todos los componentes utilizados en la síntesis física. El archivo *.gds* también es indispensable ya que es el que se le entrega a **IMEC** y **TSMC** para que pueda fabricarse el diseño en silicio. Para poder generar este archivo se debe colocar el siguiente comando en **ICCII**:

1. Para poder generar el archivo *.gds* se coloca el comando *write_ gds*. El argumento *library* y *design* establecen el nombre de la .ndm y el nombre del bloque. La vista especifica la jerarquía mas alta que es el diseño y el argumento más importante es especificar que las jerarquías que se van a utilizar todas. Esto se debe a que si no se especifica no toma en cuenta las librerías de referencia. Por último es la vista que es *frame* que es la única vista que se utiliza para diseñar y el nombre del archivo generado.

```
> write_gds -library NOT_SYN.ndm -design Not_IO -view design -
hierarchy all -lib_cell_view frame NOT.gds
```

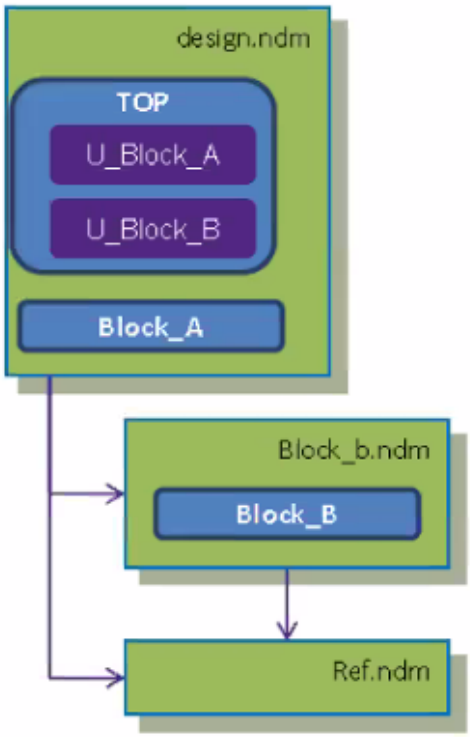

Figura 29: Jerarquías de un diseño para exportar el archivo *.gds* en **ICCII**

CAPÍTULO 8

Configuración y ejecución de **DRC** *in design*

La siguiente sección se enfoca en lo que es la verificación y corrección de errores de diseño. Para esto es importante contar con la herramienta de validación y los archivos necesarios para ejecutar dicha verificación. La herramienta de Synopsys que se encarga de hacer las verificaciones de reglas de diseño **DRC** es IC Validator **ICV**. Sin embargo el proceso de síntesis física y **DRC** se hacen de forma consecutiva en el flujo de diseño. Además el proceso de verificación y corrección de errores consiste en un intercambio de información directo y continuo de resultados. Es por esto que la herramienta de **ICCII** cuenta con la capacidad de configurar e invocar a la herramienta de **ICV** dentro de su propia interfaz. Así mismo los resultados,que consisten en el reporte de errores, pueden ser visualizados y corregidos desde la misma herramienta. Esto agiliza el flujo de diseño y da lugar a que la síntesis física generada sea más robusta.

8.1. TSMC DRC *runset file*

Debido a que nuestro objetivo es mandar a fabricar el **CI**, **TSMC** nos ha otorgado ciertos archivos para realizar las verificaciones correspondientes para el diseño y así asegurar que el diseño es manufacturable. Si dicho diseño se envía con errores, de primero no pasaría a la etapa de fabricación y se devolvería el diseño para que los diseñadores corrijan los errores correspondientes. Además, si en dado caso no se hacen las verificaciones correspondientes el diseño podría llegar a fabricarse, pero no aseguraría al diseñador que el funcionamiento de este es el que se planteo desde un inicio.

Dentro del repositorio de TSMC es posible encontrar distintos runsets. Sin embargo estos no pueden ser utilizados en las herramientas disponibles. Debido a esto IMEC proporciono un runset compatible el cual debe ser descargado mediante un servidor, utilizando un usuario y contraseña. Si en dado caso se quisiera acceder a este servidor para extraer información de

interés se debe solicitar la ubicación del archivo a IMEC y luego extraerlo con las respectivas credenciales que se otorgan. Si en dado caso se quisiera acceder a este servidor para extraer información de interés se debe solicitar la ubicación del archivo a **IMEC** y luego extraerlo con las respectivas credenciales que se otorgan.

Para este proyecto se utiliza un *runset* de **ICV** para poder verificar que todas las reglas de diseño se cumplan a cabaldada y garantizar que el diseño cumple con los estándares de fabricación de **TSMC**. El *runset* utilizado es: `ICVLM18_LM16_LM152_6M.215a_pre041518`. Este archivo es específico para diseños que trabajan en tecnologías de: $0.18\mu m$, $0.16\mu m$ y $0.152\mu m$. Este archivo es un *script* que al ser ejecutado se verifica el *layout* generado por la síntesis física.

8.2. IC Compiler II *tool in design singOff check*

Ya que la herramienta de **ICCII** cuenta con los recursos de **ICV**, para la verificación **DRC**, se efectúa la verificación de **DRC** dentro del mismo proceso de síntesis física. Antes de correr la verificación de **DRC** se debe configurar el entorno y la herramienta de **ICCII**. La primera recomendación es crear una carpeta que se llame "DRC"dentro de la carpeta donde se está trabajando. Esto se hace con el fin de que todos los resultados y archivos generados por esta verificación se guarden dentro de ella y no se confunda con los archivos de importancia generados por el proceso de síntesis física. Además se debe colocar el *runset file* dentro de la carpeta de trabajo a continuación una imagen como ejemplo.

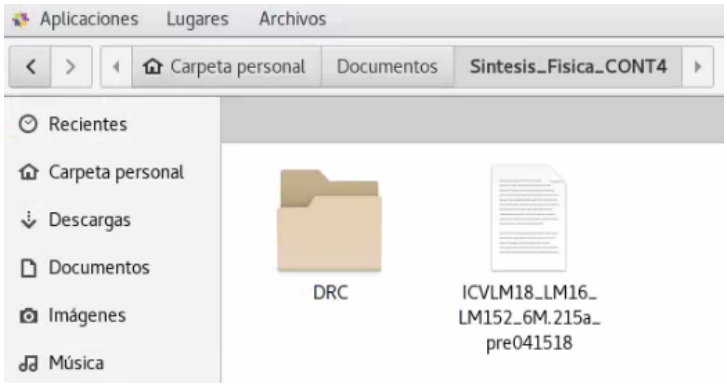

Figura 30: Entorno dentro de la carpeta de trabajo listo para efectuar verificación **DRC**

Como segundo paso, dentro de la herramienta de **ICCII** se debe configurar el *path* donde se estarán almacenando los resultados de las verificaciones y el nombre del archivo del *runset*. a continuación se muestra la secuencia de comandos para efectuar esta preparación:

1. Este comando indica el lugar en donde los resultados del comando `check_design` se guardarán.

```
> set_app_options -list {signoff.check_design.run_dir {/home/
  nanoelectronica2021/Documentos/SintesisFisica_CONT4/DRC/}}
```

2. Este comando indica el lugar en donde los resultados del comando `check_drc` se guardarán.

```
> set_app_options -list {signoff.check_drc.run_dir {/home/
  nanoelectronica2021/Documentos/SintesisFisica_CONT4/DRC/}}
```

3. Se especifica el *runset* que el comando `check_design` ejecutará para hacer la revisión del *layout*.

```
> set_app_options -list {signoff.check_design.runset {
  ICVLM18_LM16_LM152_6M.215a_pre041518}}
```

4. Se especifica el *runset* que el comando `check_drc` ejecutará para hacer la revisión del *layout*.

```
> set_app_options -list {signoff.check_drc.runset {
  ICVLM18_LM16_LM152_6M.215a_pre041518}}
```

Una vez configurado el entorno dentro de la herramienta de **ICCII** y el entorno de la carpeta de trabajo se procede a correr la verificación de **DRC** a continuación se muestra el flujo para realizarla:

1. De primero se manda a la herramienta a hacer correcciones minoritarias de **DRC** estas correcciones no son de todas las clases de errores. Para fines de este proyecto el uso de este comando fue casi como hacerle caso omiso, ya que no corregía nada. Sin embargo para flujos de diseño más complicados puede resultar útil. Se dejó ya que se consideró como una precaución.

```
> signoff_fix_drc
```

2. El siguiente comando corre el runset de **DRC** especificado en las opciones de la aplicación.

```
> signoff_check_drc
```

3. Este comando es adicional al flujo de **DRC** es una revisión previa del **LVS** oficial. No utiliza **ICV** pero si compara el *layout* generado con las definiciones de las celdas en el *verilog* que se crea a raíz del final de la síntesis física.

```
> signoff_lvs
```

4. Se guarda el proyecto nuevamente.

```
>          save_block NOT_SYN.ndm:Not_IO
```

La primera vez que se realizó **DRC** a una compuerta NOT los resultados fueron desalentadores. El circuito tenía más de 400 errores. La mayoría de estos errores tenían la siguiente estructura:

```
    Minimum extension of MX beyond the overlap area that VIAX and VIAY
  are fully or partially touching < n
```

La imagen mostrada a continuación muestra el ejemplo del error dentro del *runset*.

```
// VIA2 checks
//=============
VIA2_CORE = VIA2 not SR_VIA2;
rVIA2_W_1 @= { @ "VIA2.W.1 : VIA2 must be 0.26 x 0.26";
    A = not_rectangles( VIA2_CORE, orientation = ORTHOGONAL, sides = { == 0.26, == 0.26 } );
    A outside RNGX;                    /* exclude from metal fuse protection ring area */
} /* end of rule : VIA2.W.1 */
rVIA2_S_1 @= { @ "VIA2.S.1 : Min. VIA2 space < 0.26";
    external( VIA2, < 0.26, extension = RADIAL, intersecting = {  }, intersection_angle = < 90, look_thru = NOT_ADJACENT, relational = { POINT_TOUCH } );
} /* end of rule : VIA2.S.1 */
rVIA2_E_1 @= { @ "VIA2.E.1 : Min. extension of a M2 region beyond a VIA2 region < 0.01";
    enclose( VIA2, M2, < 0.01, extension = RADIAL, extension_look_past = POINT_TO_POINT, intersecting = {  }, intersection_angle = < 90, line_touch_shape = INSIDE, look_thru = NOT_CONTAINED, relational =
{ POINT_TOUCH }, width = 0.004 );
    VIA2 not M2;
} /* end of rule : VIA2.E.1 */
rVIA2_E_2 @= { @ "VIA2.E.2 : Min. extension of M2 end-of-line region beyond VIA2 region < 0.06";
    X = enclose_edge( VIA2, M2, < 0.06, extension = NONE, extension_look_past = POINT_TO_POINT, intersecting = {  }, intersection_angle = < 90, look_thru = NOT_CONTAINED, output_layer = LAYER1, projection_filter
= MUTUAL_NON_ORTHOGONAL, projection_mode = ASYMMETRIC );    /* a narrow side */
    internal1_error( X, < 0.26, extension = RADIAL, intersecting = {  }, intersection_angle = == 90, orientation = { } );    /* adjacent narrow sides */
} /* end of rule : VIA2.E.2 */
#ifdef RECOMMEND
   rVIA2_E_3 @= { @ "VIA2.E.3 : Minimum extension of M2 beyond the overlap area that VIA2 and VIA1 are fully or partially touching < " + VIA2_E_3;
       X = VIA2 and VIA1;
       enclose( X, M2, < VIA2_E_3, extension = RADIAL, extension_look_past = POINT_TO_POINT, intersecting = {  }, intersection_angle = < 90, line_touch_shape = INSIDE, look_thru = NOT_CONTAINED, relational =
{ POINT_TOUCH }, width = 0.004 );
   } /* end of rule : VIA2.E.3 */
#endif /* end of #ifdef RECOMMEND  */
```

Figura 31: Linea del *runset* que presenta errores dentro de la categoría de *VIA2 checks*

Esto quiere decir que las vías instanciadas dentro del *layout* estaban incumpliendo requerimientos de diseño. Sin embargo al momento de indagar más acerca de los errores que presentaba el diseño dentro del *runset* pudimos identificar que estos estaban dentro de una condicional bajo la variable *#RECOMEND*. Las siguientes acciones se basaron en las siguientes dos conjeturas: primero el *runset* describe las opciones recomendadas como no indispensables y que solamente se colocan para garantizar que problemas como el *X-Talk* o ruido no afecten en la extracción de parásitos; como segundo punto se concluyó que el hecho de que las vías y sus arreglos son extraídos del *technology file* que es un documento proporcionado por el fabricante al que no se le deben realizar modificaciones las opciones recomendadas tenían una menor prioridad que el *technology file*

Por lo tanto se decidió comentar esta configuración dentro del *runset* de **DRC**. La configuración que se muestra a continuación es la que se utilizó para correr **DRC** en todos los diseños realizados en este trabajo de graduación.

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

//#define CHECK_SRAM_EXCL                    /* turn on only when you want
    check M2 and upward layers in SRAM region covered by EXCL. otherwise
    please turn off it */
```

```
#define CHECK_ALRDL                        /* turn on only when you want
    check MD&VIAD rules. otherwise please turn off it */
//#define CHECK_DUMMY_ODPO                 /* turn on only when you want
    check Dummy OD&PO. otherwise please turn off it */
#define GUIDELINE_LUP                      /* turn on only when you want
    check latchup guideline rules. otherwise please turn off it */
//#define DISCONNECT_ALL_RESISTOR          /* turn on only when you want to
    disconnect all resistors to check latchup rules, otherwise please turn
    off it */
//#define CONNECT_ALL_RESISTOR             /* turn on only when you want to
    connect all resistors to check latchup rules, otherwise please turn off
    it */
#define GUIDELINE_ESD                      /* turn on only when you want
    check ESD guideline rules. otherwise please turn off it */
//#define RECOMMEND                        /* turn on only when you want to
    check Recommended rules. otherwise please turn off it */
//#define MIX_MODE                         /* turn on only when you use Mixed
    -Singal/RF process. otherwise please turn off it */
//#define THICK_40K                        /* turn on only when 40KA Thick
    Top Metal is used. otherwise please turn off it */
//#define THICK_20K                        /* turn on only when 20KA Thick
    Top Metal is used. otherwise please turn off it */
//#define LP                               /* turn on only when you use Low
    Power process. otherwise please turn off it  */
//#define C016                             /* turn on only when you use 0.16
    um process. otherwise please turn off it */
//#define C0152                            /* turn on only when you use 0.152
    um process. otherwise please turn off it */
#define _3_3V                               /* turn on only when HIGH_VOLTAGE
     is 3.3V. otherwise please turn off it */
//#define _5V                               /* turn on only when HIGH_VOLTAGE
     is 5V. otherwise please turn off it */
//#define _2_5V                             /* turn on only when HIGH_VOLTAGE
     is 2.5V. otherwise please turn off it */
#define _1_8V                               /* turn on only when CORE_VOLTAGE
     is 1.8V. otherwise please turn off it */
//#define _1_5V                             /* turn on only when CORE_VOLTAGE
     is 1.5V. otherwise please turn off it */
//#define CHECK_LATCHUP_BY_TEXT            /* Turn on to recognize IO PAD by
    following text   */
```

Listing 8.1: Configuración del *runset* de **DRC** especificando el voltaje de los **IO**, el voltaje del núcleo y las opciones recomendadas desactivadas

Con esto de forma satisfactoria se logró reducir de más de 400 errores de diseño dentro de una compuerta NOT a 6 errores. A continuación se muestran las imágenes que permiten desplegar el monitor de errores de la herramienta **ICCII** en conjunto con los resultados

obtenidos hasta el momento.

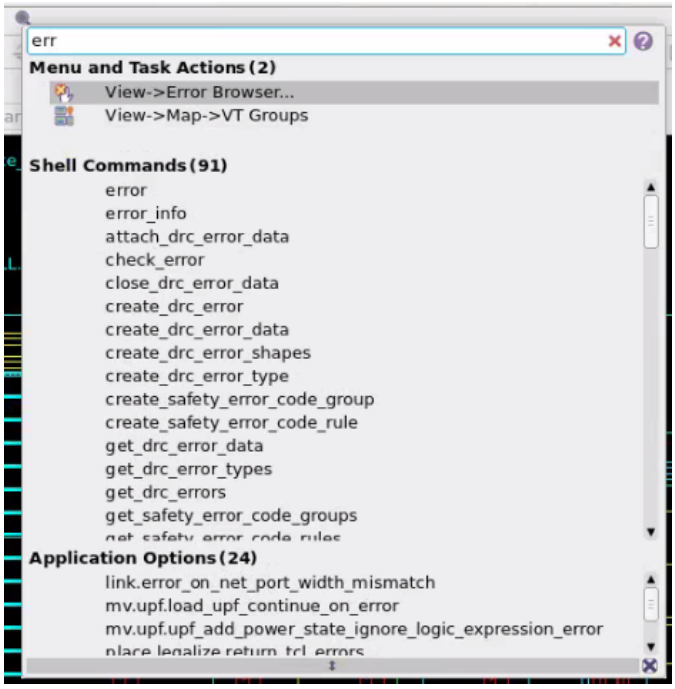

Figura 32: Se utiliza el buscador de comandos dentro de la herramienta de **ICCII** para invocar al *Error Browser*

Esto también puede hacerse desde consola con el comando `gui_show_error_data`

Los reportes generados hasta el momento son los siguientes: legalización del proceso de *placement*, **LVS** interno, legalidad de todo el proceso hasta la finalización de la síntesis, chequeo de ruteabilidad, **DRC** y reporte de la herramienta de ruteo.

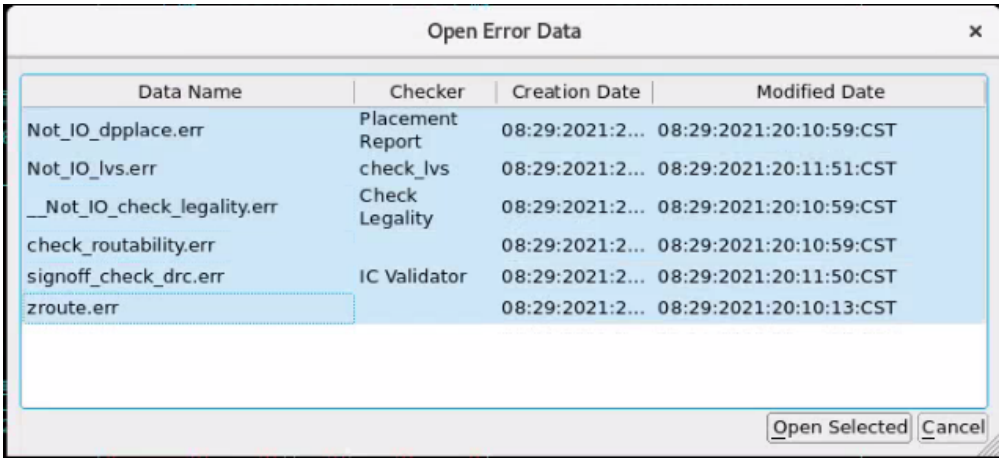

Figura 33: Se seleccionan todos los reportes de errores generados al momento

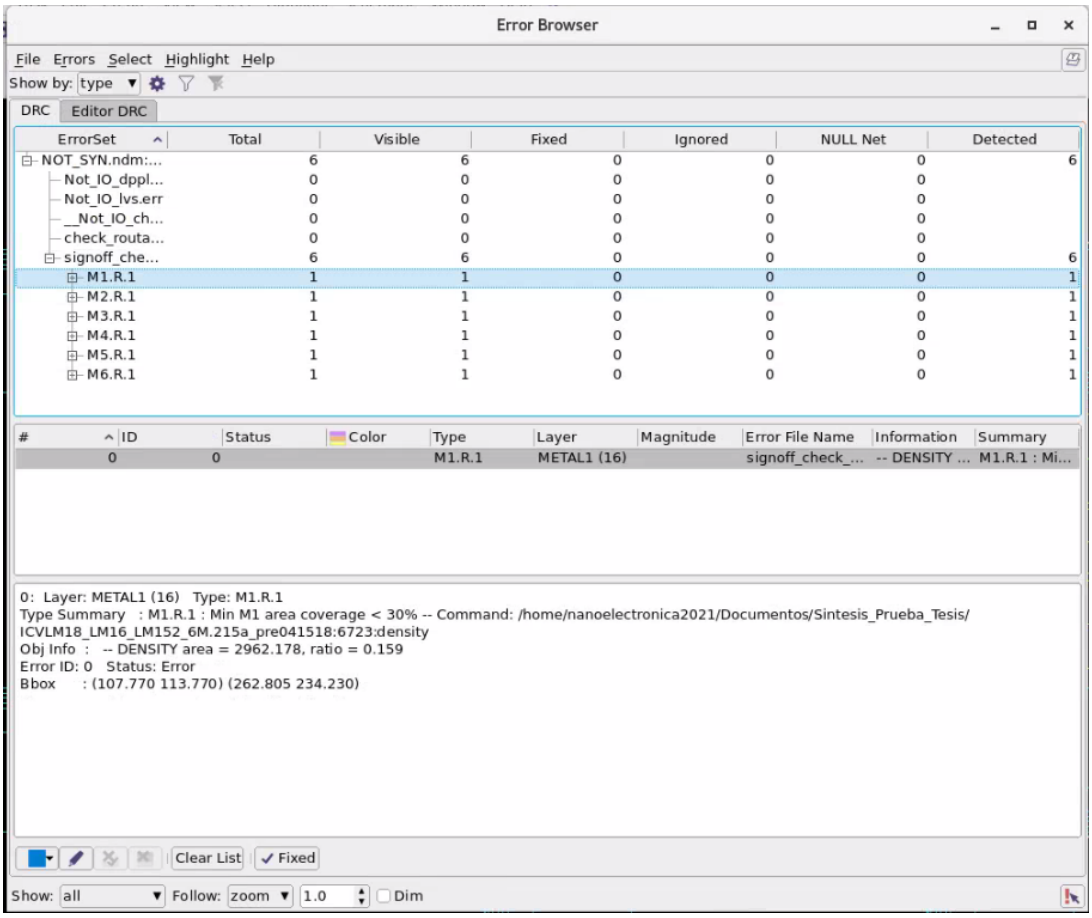

Figura 34: Se despliegan los resultados dentro de la interfaz gráfica mostrando únicamente los 6 errores de **DRC** presentes hasta el momento

Los 6 errores restantes se presentaron en todos los diseños realizados. Este error se da cuando dentro del área del núcleo no se cumple con el requisito de utilización mínima definida en el *techfile*. A este error se le conoce como error de densidad y es un error corregible. Al momento de realizar la colocación de las *filler cells* se omitieron las celdas de relleno con metal. Debido a que para colocar estos *metal fillers* se requiere de un *runset* especializado, este se solicitó a **TSMC** por medio de **IMEC** y así poder corregirlo.

A continuación el resultado oficial que ofrece ICV *in design* de realizar una compuerta NOT con los recursos actuales:

```
RESULTS: NOT CLEAN

#   #   ###   #####      ####  #      #####   ###   #   #
##  #  #   #    #       #      #      #      #   #  ##  #
# # #  #   #    #       #      #      ####   #####  # # #
#  ##  #   #    #       #      #      #      #   #  #  ##
#   #   ###     #        ####  #####  #####  #   #  #   #
```

```
================================================================================


--------------------------------------------------------------------------------
ICV Execution
--------------------------------------------------------------------------------


                                IC Validator

          Version R-2020.09-SP2-2 for linux64 - Jan 28, 2021 cl#6206082

                    Copyright (c) 1996 - 2021 Synopsys, Inc.
   This software and the associated documentation are proprietary to
    Synopsys,
 Inc. This software may only be used in accordance with the terms and
    conditions
 of a written license agreement with Synopsys, Inc. All other use,
    reproduction,
             or distribution of this software is strictly prohibited.

Called as: icv -icc2 -f NDM -i NOT_SYN.ndm -p /home/nanoelectronica2021/
    Documentos/SintesisFisica_NOT3 -c Not_IO -clf /home/nanoelectronica2021/
    Documentos/SintesisFisica_NOT3/DRC/slnFile.txt -icc_density_blockage -
    icc2_error_categories -I /home/nanoelectronica2021/Documentos/
    SintesisFisica_NOT3/DRC -icc2_error_browser INST -icc2_error_cell
    signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
    SintesisFisica_NOT3/DRC/signoff_check_drc.rc -I /home/nanoelectronica2021
    /Documentos/SintesisFisica_NOT3/DRC /home/nanoelectronica2021/Documentos/
    SintesisFisica_NOT3/ICVLM18_LM16_LM152_6M.215a_pre041518

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

User name:        nanoelectronica2021
Layout format:    NDM
Input file name:  NOT_SYN.ndm
Top cell name:    Not_IO
Time started:     2021/07/31 10:03:46PM
Time ended:       2021/07/31 10:03:57PM

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

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

Rule and DRC Error Summary
```

```
192 total rules were run.
226 rules NOT EXECUTED.
6 rules have violations.
There are 6 total violations.
Refer to Not_IO.LAYOUT_ERRORS
```

Listing 8.2: Resultados de la verificación DRC para una compuerta NOT

8.3. Corrección de errores de diseño

8.3.1. Corrección de *density rules*

Para corregir estos errores es necesario un *runset* que genere los rellenos metálicos que se mencionaron anteriormente. Sin embargo el problema es que por el memento no se cuenta con el *runset* de **TSMC** para la generación de los *metal fills*. Sin embargo el flujo de colocación de estas celdas es descrito por la documentación de Synopsys que se muestran a continuación:

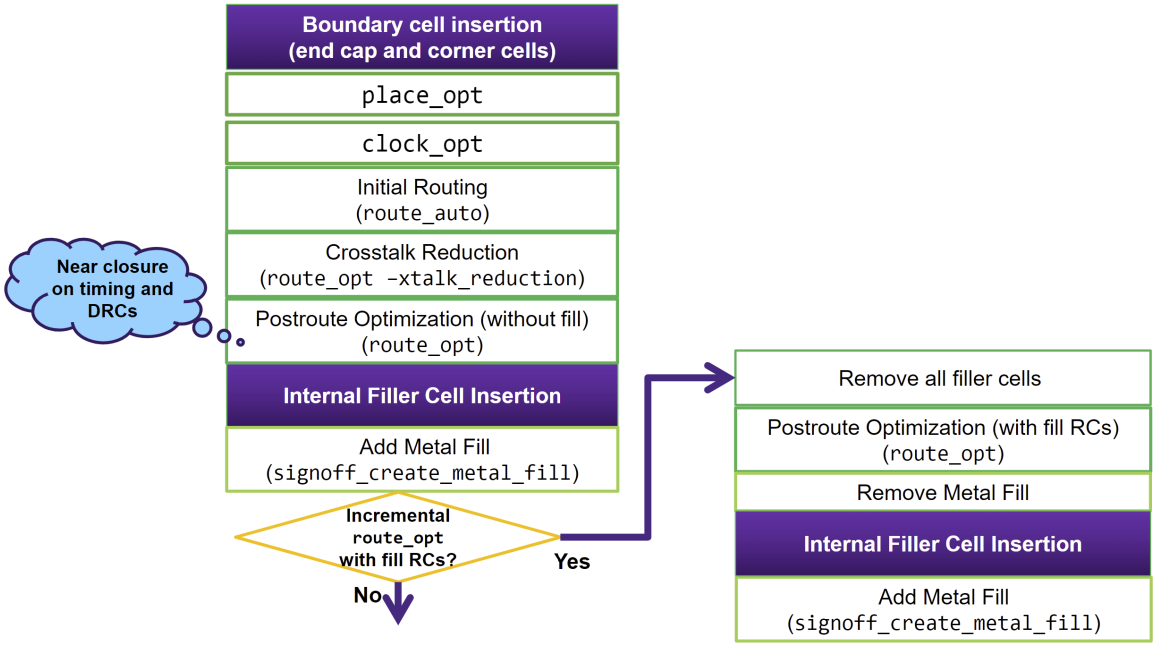

Figura 35: Colocación de *metal fill cells* dentro del flujo de la sintesis física. [25]

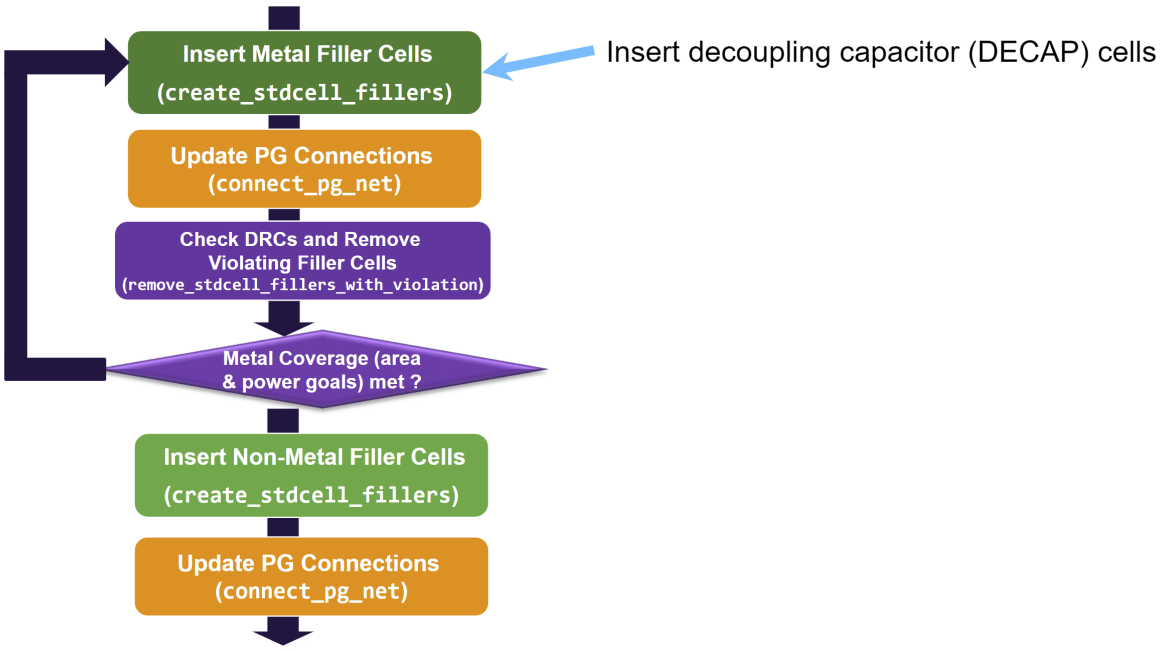

Figura 36: Procedimiento para colocar los *metal fills cells* con un *runset*. [25]

Los resultados de no realizar este proceso son los únicos 6 errores de DRC obtenidos en todo el proceso de la síntesis física. A continuación se muestran los resultados de ICV *in design* que corre la herramienta de ICCII para la verificación DRC de una compuerta NOT.

```
                LAYOUT ERRORS RESULTS: ERRORS

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


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


         Library name:    NOT_SYN.ndm
         Structure name:  Not_IO
         Generated by:    IC Validator RHEL64 R-2020.09-SP2-2.6206082
2021/01/28
         Runset name:     /home/nanoelectronica2021/Documentos/
SintesisFisica_NOT3/ICVLM18_LM16_LM152_6M.215a_pre041518
         User name:       nanoelectronica2021
         Time started:    2021/07/31 10:03:50PM
         Time ended:      2021/07/31 10:03:57PM
```

```
        Called as: icv -icc2 -f NDM -i NOT_SYN.ndm -p /home/
nanoelectronica2021/Documentos/SintesisFisica_NOT3 -c Not_IO -clf /home/
nanoelectronica2021/Documentos/SintesisFisica_NOT3/DRC/slnFile.txt -
icc_density_blockage -icc2_error_categories -I /home/nanoelectronica2021/
Documentos/SintesisFisica_NOT3/DRC -icc2_error_browser INST -
icc2_error_cell signoff_check_drc.err -rc /home/nanoelectronica2021/
Documentos/SintesisFisica_NOT3/DRC/signoff_check_drc.rc -I /home/
nanoelectronica2021/Documentos/SintesisFisica_NOT3/DRC /home/
nanoelectronica2021/Documentos/SintesisFisica_NOT3/ICVLM18_LM16_LM152_6M
.215a_pre041518
        CLF: -sln "16 18 28 31 33 38 17 27 29 32 39 "

           ERROR SUMMARY

         M1.R.1 : Min M1 area coverage < 30%
           density ..................................... 1
violation found.

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

         M3.R.1 : Min M3 area coverage < 30%
           density ..................................... 1
violation found.

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

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

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

Listing 8.3: Especificación de errores de DRC para una compuerta NOT

CAPÍTULO 9

Circuitos adicionales y cambios en el proceso de síntesis física

Adicional a la compuerta NOT, como parte de la metodología propuesta, se sintetizaron los siguientes circuitos: XOR, *Full Adder*, **ALU** y un Contador de 4 bits. Esto se hizo así ya que la complejidad de estos circuitos va escalando. El que la complejidad escale de forma progresiva permite que se encuentren fallas o posibles mejoras en el proceso de síntesis.

9.1. Compuerta XOR

El primer nivel de dificultad se eleva al momento de agregar un *pad* adicional de entradas y salidas. Se pierde cierta simetría en el circuito. Con el fin de agregar simetría al diseño se coordinó con el proceso de síntesis lógica para realizar el diseño con 4 en vez de 3 *pads* de entradas y salidas. Complementando con *pads* dobles de **PG** en los laterales. Adicionalmente, esta compuerta no fue instanciada como una única celda, para subir la dificultad se creo la compuerta a partir de otras compuertas lógicas como lo son las NOT, OR y ADD.

Para este proceso de síntesis se añadieron, movieron y modificaron algunos comandos dentro del proceso de síntesis. Por ejemplo, debido a que había que insertar más pads de VDD y VSS se agregaron al siguiente conjunto de comandos:

```
> create_cell {PVDD1 PVDD2} PVDD1CDG

> create_cell {PVSS1 PVSS2} PVSS1CDG
```

Debido a que a medida que el circuito crece y sus entradas y salidas también surgió la necesidad de especificar la posición en donde los *pads* de entradas y salidas se colocan. Las aristas en donde se colocan debido a la forma en la que se hizo el diseño es que los *pads* de **VDD** se colocan en la arista izquierda del *PG Ring* y los pads de **VSS** se colocan en la arista derecha del *PG Ring*. Antes de hacer el *IO Placement* se ejecutan los siguientes comandos:

1. Estos dos comandos sirven para alinear con la respectiva arista del anillo de **PG** las instancias de los pads de **PG**. El asterisco, como se mencionó anteriormente, sirve como una mascara en donde todas las intancias con el nombre **PVSS** y **PVDD** con cualquier terminación sean incluidas en esta acción.

```
> add_to_io_guide [get_io_guides anillo_IO_FA.left] PVDD*

> add_to_io_guide [get_io_guides anillo_IO_FA.right] PVSS*
```

También se movilizó la parte de *Cell Placement and Legalization* especificada en: 7.3.6 hasta despues de *Create Mesh and Cell Rows* especificada en: 7.3.8 Para separar completamente el *FloorPlanning* del *Placement*. Esto no afecta el resultado final pero mantiene una mejor secuencia del flujo de diseño.

A continuación se muestra el resultado de este proceso de síntesis modificado para la compuerta XOR:

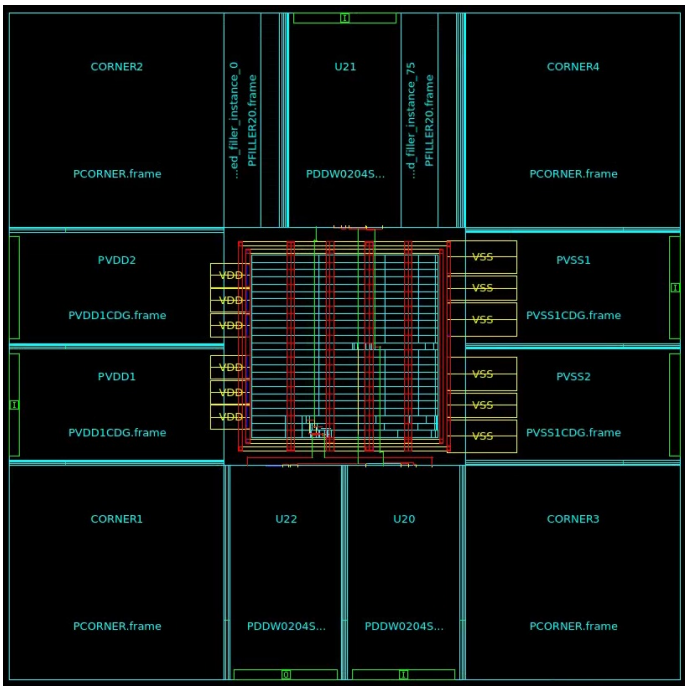

Figura 37: Compuerta XOR sintetizada con **ICCII**.

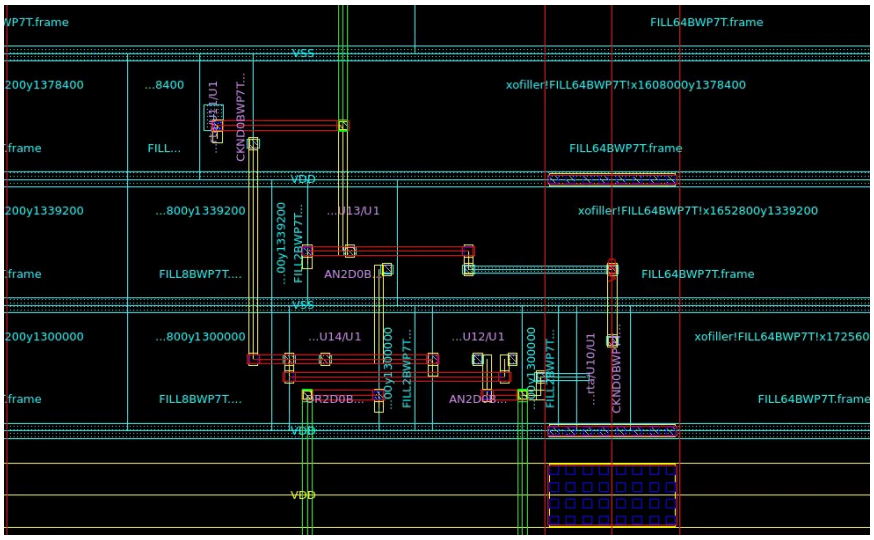

Figura 38: Compuerta XOR sintetizada enfocando las celdas del núcleo.

A continuación se muestra el resultado de la verificación DRC y la especificación de los errores obtenidos para la compuerta XOR:

```
                              RESULTS: NOT CLEAN

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

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


--------------------------------------------------------------------------------
ICV Execution
--------------------------------------------------------------------------------


                                  IC Validator

              Version R-2020.09-SP2-2 for linux64 - Jan 28, 2021 cl#6206082

                        Copyright (c) 1996 - 2021 Synopsys, Inc.
     This software and the associated documentation are proprietary to
      Synopsys,
    Inc. This software may only be used in accordance with the terms and
      conditions
    of a written license agreement with Synopsys, Inc. All other use,
```

```
    reproduction,
            or distribution of this software is strictly prohibited.

Called as: icv -icc2 -f NDM -i XOR_SYN2.ndm -p /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_XORV2 -c Xor_IO -clf /home/nanoelectronica2021
    /Documentos/Sintesis_Fisica_XORV2/DRC/slnFile.txt -icc_density_blockage -
    icc2_error_categories -I /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_XORV2/DRC -icc2_error_browser INST -icc2_error_cell
    signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_XORV2/DRC/signoff_check_drc.rc -I /home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_XORV2/DRC /home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_XORV2/
    ICVLM18_LM16_LM152_6M.215a_pre041518

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

User name:        nanoelectronica2021
Layout format:    NDM
Input file name:  XOR_SYN2.ndm
Top cell name:    Xor_IO
Time started:     2021/09/01 04:43:47PM
Time ended:       2021/09/01 04:44:07PM

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

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

Rule and DRC Error Summary

192 total rules were run.
226 rules NOT EXECUTED.
6 rules have violations.
There are 6 total violations.
Refer to Xor_IO.LAYOUT_ERRORS
```

Listing 9.1: Resultados de la verificación DRC para una compuerta XOR

```
       LAYOUT ERRORS RESULTS: ERRORS

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

==============================================================================
```

```
Library name:    XOR_SYN2.ndm
Structure name:  Xor_IO
Generated by:    IC Validator RHEL64 R-2020.09-SP2-2.6206082 2021/01/28
Runset name:     /home/nanoelectronica2021/Documentos/Sintesis_Fisica_XORV2/
   ICVLM18_LM16_LM152_6M.215a_pre041518
User name:       nanoelectronica2021
Time started:    2021/09/01 04:44:01PM
Time ended:      2021/09/01 04:44:07PM

Called as: icv -icc2 -f NDM -i XOR_SYN2.ndm -p /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_XORV2 -c Xor_IO -clf /home/nanoelectronica2021
   /Documentos/Sintesis_Fisica_XORV2/DRC/slnFile.txt -icc_density_blockage -
   icc2_error_categories -I /home/nanoelectronica2021/Documentos/
   Sintesis_Fisica_XORV2/DRC -icc2_error_browser INST -icc2_error_cell
   signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
   Sintesis_Fisica_XORV2/DRC/signoff_check_drc.rc -I /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_XORV2/DRC /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_XORV2/
   ICVLM18_LM16_LM152_6M.215a_pre041518
CLF: -sln "16 18 28 31 33 38 17 27 29 32 39 "

   ERROR SUMMARY

 M1.R.1 : Min M1 area coverage < 30%
   density ..................................... 1 violation found.

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

 M3.R.1 : Min M3 area coverage < 30%
   density ..................................... 1 violation found.

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

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

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

Listing 9.2: Especificación de errores de DRC para una compuerta XOR

9.2. Circuito *full adder*

Para el circuito del *Full Adder* la única modificación restante del proceso de síntesis física es la dimensión del circuito. Esto ya que a medida que crecen la cantidad de entradas y salidas y las celdas que el circuito utiliza se requiere más área. Los comandos que se vieron afectas por este cambio fueron:

1. Cuando se define el *FloorPlan* se hace más grande para que quepan los *pads* de entradas y salidas.

```
> initialize_floorplan -site_def unit -use_site_row -keep_all -
  side_length {285 285} -core_offset {125}
```

2. Se cambia el área en donde se colocan los **PG** *straps* ya que como creció el área ahora deben ser más largos y deben generarse más para mejorar la densidad ocupación de los metales.

```
> set_pg_strategy mesh_strategy -polygon {{125.000 118.000}
  {409.480 414.240}} -pattern {{pattern: mesh_pattern}{nets: {VDD VSS
  }}} -blockage {macros: all}
```

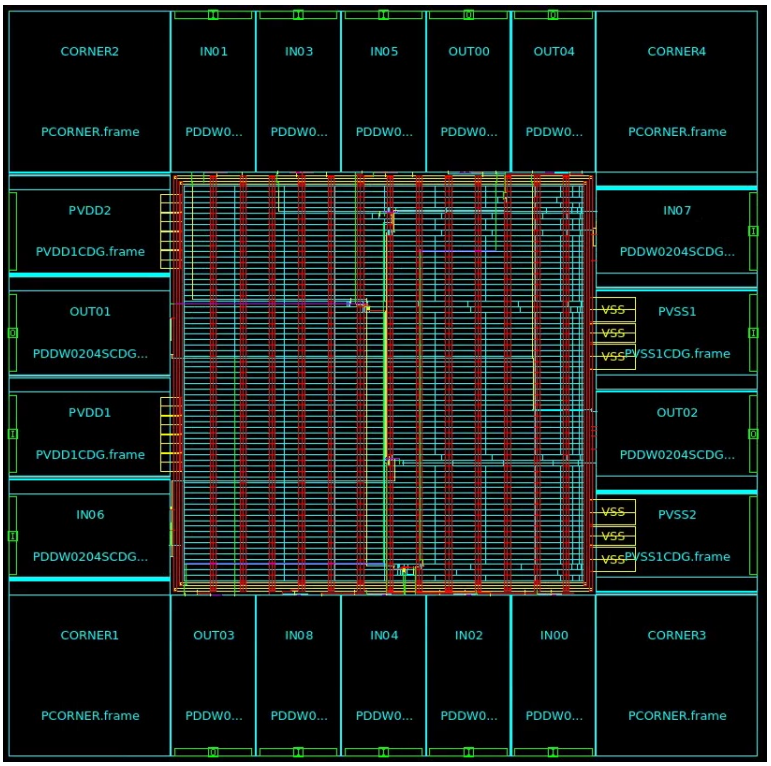

Figura 39: Circuito *Full Adder* sintetizada con **ICCII**.

A continuación se muestra el resultado de la verificación DRC y la especificación de los errores obtenidos para el circuito Full Adder:

```
                              RESULTS: NOT CLEAN

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

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


--------------------------------------------------------------------------------
ICV Execution
--------------------------------------------------------------------------------


                                  IC Validator

            Version R-2020.09-SP2-2 for linux64 - Jan 28, 2021 cl#6206082

                      Copyright (c) 1996 - 2021 Synopsys, Inc.
      This software and the associated documentation are proprietary to
       Synopsys,
    Inc. This software may only be used in accordance with the terms and
       conditions
    of a written license agreement with Synopsys, Inc. All other use,
       reproduction,
                or distribution of this software is strictly prohibited.

Called as: icv -icc2 -f NDM -i FA_SYN.ndm -p /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_FA -c fulladd_io -clf /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_FA/DRC/slnFile.txt -
   icc_density_blockage -icc2_error_categories -I /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_FA/DRC -icc2_error_browser INST -
   icc2_error_cell signoff_check_drc.err -rc /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_FA/DRC/signoff_check_drc.rc -I /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_FA/DRC /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_FA/ICVLM18_LM16_LM152_6M
   .215a_pre041518

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

User name:        nanoelectronica2021
Layout format:    NDM
Input file name:  FA_SYN.ndm
```

```
Top cell name:    fulladd_io
Time started:     2021/09/01 06:21:49PM
Time ended:       2021/09/01 06:22:10PM

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

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

Rule and DRC Error Summary

192 total rules were run.
226 rules NOT EXECUTED.
6 rules have violations.
There are 6 total violations.
Refer to fulladd_io.LAYOUT_ERRORS
```

Listing 9.3: Resultados de la verificación DRC para circuito Full Adder

```
       LAYOUT ERRORS RESULTS: ERRORS

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

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

Library name:    FA_SYN.ndm
Structure name:  fulladd_io
Generated by:    IC Validator RHEL64 R-2020.09-SP2-2.6206082 2021/01/28
Runset name:     /home/nanoelectronica2021/Documentos/Sintesis_Fisica_FA1/
   ICVLM18_LM16_LM152_6M.215a_pre041518
User name:       nanoelectronica2021
Time started:    2021/08/19 10:58:47PM
Time ended:      2021/08/19 10:58:55PM

Called as: icv -icc2 -f NDM -i FA_SYN.ndm -p /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_FA1 -c fulladd_io -clf /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_FA1/DRC/slnFile.txt -
   icc_density_blockage -icc2_error_categories -I /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_FA1/DRC -icc2_error_browser INST -
   icc2_error_cell signoff_check_drc.err -rc /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_FA1/DRC/signoff_check_drc.rc -I /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_FA1/DRC /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_FA1/ICVLM18_LM16_LM152_6M
```

```
    .215a_pre041518
CLF: -sln "16 18 28 31 33 38 17 27 29 32 39 "

   ERROR SUMMARY

 M1.R.1 : Min M1 area coverage < 30%
   density ..................................... 1 violation found.

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

 M3.R.1 : Min M3 area coverage < 30%
   density ..................................... 1 violation found.

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

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

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

Listing 9.4: Especificación de errores de DRC para el circuito Full Adder

9.3. Circuito de una unidad aritmética lógica ALU

La **ALU** no necesitó tampoco ningún cambio en el proceso de síntesis y únicamente modificando las dimensiones de algunos comandos como en el caso del *Full Adder*. A continuación se muestran las nuevas dimensiones:

1. Cuando se define el *FloorPlan* se hace más grande para que quepan los *pads* de entradas y salidas.

```
> initialize_floorplan -site_def unit -use_site_row -keep_all -
  side_length {350 350} -core_offset {125}
```

2. Se cambia el área en donde se colocan los **PG** *straps* ya que como creció el área ahora deben ser más largos y deben generarse más para mejorar la densidad ocupación de los metales.

```
> set_pg_strategy mesh_strategy -polygon {{125.000 118.000}
  {475.000 480.880}} -pattern {{pattern: mesh_pattern}{nets: {VDD VSS
  }}} -blockage {macros: all}
```

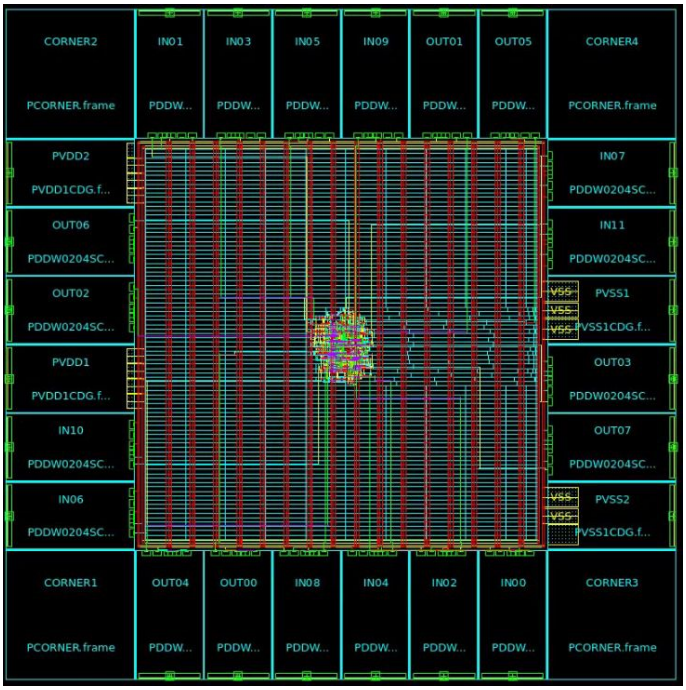

Figura 40: **ALU** sintetizada con **ICCII**.

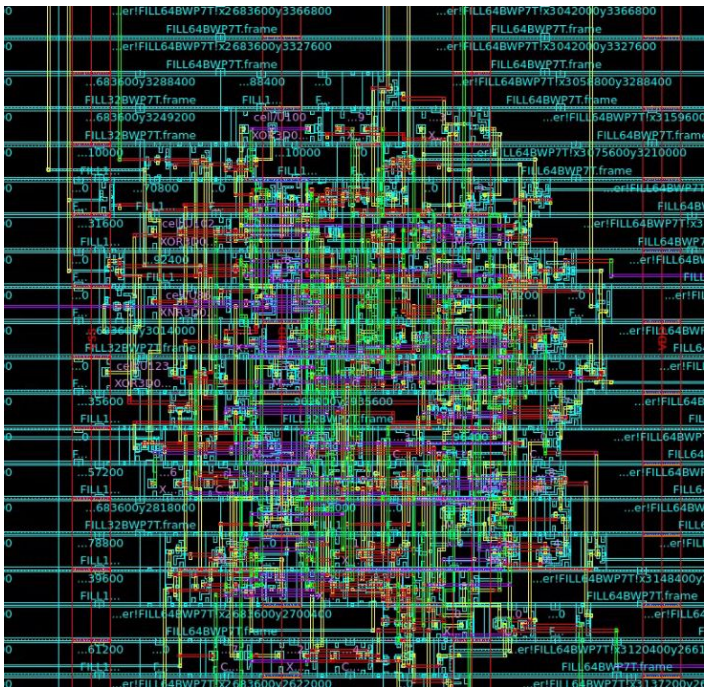

Figura 41: Síntesis de la **ALU** enfocando las celdas dentro del núcleo.

A continuación se muestra el resultado de la verificación DRC y la especificación de los errores obtenidos para la **ALU**:

```
                                RESULTS: NOT CLEAN

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

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


--------------------------------------------------------------------------------
ICV Execution
--------------------------------------------------------------------------------


                                    IC Validator

                Version R-2020.09-SP2-2 for linux64 - Jan 28, 2021 cl#6206082

                        Copyright (c) 1996 - 2021 Synopsys, Inc.
           This software and the associated documentation are proprietary to
            Synopsys,
         Inc. This software may only be used in accordance with the terms and
            conditions
         of a written license agreement with Synopsys, Inc. All other use,
            reproduction,
                  or distribution of this software is strictly prohibited.

Called as: icv -icc2 -f NDM -i ALU_SYN.ndm -p /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_ALU4B -c ALU_IO -clf /home/nanoelectronica2021
    /Documentos/Sintesis_Fisica_ALU4B/DRC/slnFile.txt -icc_density_blockage -
    icc2_error_categories -I /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_ALU4B/DRC -icc2_error_browser INST -icc2_error_cell
    signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_ALU4B/DRC/signoff_check_drc.rc -I /home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4B/DRC /home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4B/
    ICVLM18_LM16_LM152_6M.215a_pre041518

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

User name:         nanoelectronica2021
Layout format:     NDM
Input file name:   ALU_SYN.ndm
Top cell name:     ALU_IO
Time started:      2021/09/01 06:35:56PM
Time ended:        2021/09/01 06:36:18PM
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

Rule and DRC Error Summary

192 total rules were run.
226 rules NOT EXECUTED.
6 rules have violations.
There are 6 total violations.
Refer to ALU_IO.LAYOUT_ERRORS
```

Listing 9.5: Resultados de la verificación DRC para la **ALU**

```
       LAYOUT ERRORS RESULTS: ERRORS

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

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

Library name:    ALU_SYN.ndm
Structure name:  ALU_IO
Generated by:    IC Validator RHEL64 R-2020.09-SP2-2.6206082 2021/01/28
Runset name:     /home/nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4B/
   ICVLM18_LM16_LM152_6M.215a_pre041518
User name:       nanoelectronica2021
Time started:    2021/09/01 06:36:10PM
Time ended:      2021/09/01 06:36:18PM

Called as: icv -icc2 -f NDM -i ALU_SYN.ndm -p /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_ALU4B -c ALU_IO -clf /home/nanoelectronica2021
   /Documentos/Sintesis_Fisica_ALU4B/DRC/slnFile.txt -icc_density_blockage -
   icc2_error_categories -I /home/nanoelectronica2021/Documentos/
   Sintesis_Fisica_ALU4B/DRC -icc2_error_browser INST -icc2_error_cell
   signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
   Sintesis_Fisica_ALU4B/DRC/signoff_check_drc.rc -I /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4B/DRC /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4B/
   ICVLM18_LM16_LM152_6M.215a_pre041518
CLF: -sln "16 18 28 31 33 38 17 27 29 32 39 "
```

```
  ERROR SUMMARY

M1.R.1 : Min M1 area coverage < 30%
  density ..................................... 1 violation found.

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

M3.R.1 : Min M3 area coverage < 30%
  density ..................................... 1 violation found.

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

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

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

Listing 9.6: Especificación de errores de DRC para la **ALU**

9.4. Contador de 4 bits

Luego de verificar el proceso de síntesis física para lógica combinacional y concluir con los objetivos del proyecto de graduación se decidió escalar el proyecto y subir un grado más la dificultad de la síntesis física. El proceso hasta este punto no contaba con componentes secuenciales por lo que el uso de un reloj y definición de una frecuencia de operación no eran procesos necesarios. En el proceso de síntesis lógica se adicionan estas dos etapas. En lo que concierne a la síntesis física, el proceso está orientado a la estructura de ruteo del árbol de la *net* de reloj y su optimización.

Un árbol de reloj o *Clock Tree* es la forma en la que se refiere a la net de reloj en un circuito secuencial. Esta nomenclatura se utiliza debido a que la forma en la que se distribuye una red de un reloj dentro de un circuito secuencial es como un árbol con tronco y ramas que se bifurcan. Debido a que los circuitos secuenciales escalan a medida que su funcionalidad se hace más compleja problemas en esta *net* del reloj empiezan a surgir. Diferentes técnicas para reforzar, sincronizar y mantener la integridad en general de la señal como *Fish Bone* y creación de *Clock Mesh* o *Clock Straps* permiten que el circuito no tenga problemas al momento de ser fabricado debido a la escala. Algunas otras posibles soluciones a problemas con retrasos, *signal skew* y *high fanout* son: colocación de *buffers* y segmentación del reloj son tambien utilizadas.

Durante el proceso de síntesis física la estrategia, colocación y optimización del *clock tree* se hace luego del *Placement*. Se hace así ya que el reloj tiene una mayor prioridad que las señales. Esta preferencia es debido a que el reloj es una de las *nets* más importantes con las

de **PG**. La herramienta de **ICCII** tiene todas las optimizaciones y soluciones mencionadas anteriormente. Permite crear troncos, *straps*, *buffers*, compuertas y *flip flops* con el propósito de regenerar las señales. Debido a que este circuito es un contador de 4 bits su complejidoad es reducida, la cantidad de compuertas que usan reloj son pocas y la extensión del circuito no es significativa, la creación de un *mesh* o tronco no es necesaria.

El proceso para sintetizar el reloj de este contador de 4 bits es el siguiente:

1. Se hacen las verificaciones necesarias para validar que el circuito está listo para sintetizar la *net* del reloj. En la síntesis lógica el nombre de **clk** se le dio a la *net* que se encarga de este proceso.

```
> check_clock_trees -clocks clk


check_design -checks pre_clock_tree_stage
```

2. Como siguiente paso se sintetiza la *net*, la opción A hará el ruteo únicamente de la *net* del reloj. Sin embargo la opción B tiene también incluido el ruteo de todas las demás nets. Ambas son correctas, pero debido a la prioridad de la net de reloj, se prefiere la opción A.
   Opción A:

```
> synthesize_clock_trees -clocks clk -postroute -
routed_clock_stage detail
```

   Opción B:

```
> synthesize_clock_trees -clocks clk -postroute -
routed_clock_stage detail_with_signal_routes
```

3. En este paso se optimiza la forma del ruteo y si hay errores o rutas críticas demasiado significativas estas se corrigen.

```
> clock_opt -list_only
```

4. Por útimo se pasa la respectiva revisión de la síntesis. El proceso a partir de aquí continua con la etapa de ruteo.

```
> check_design -checks cts_qor
```

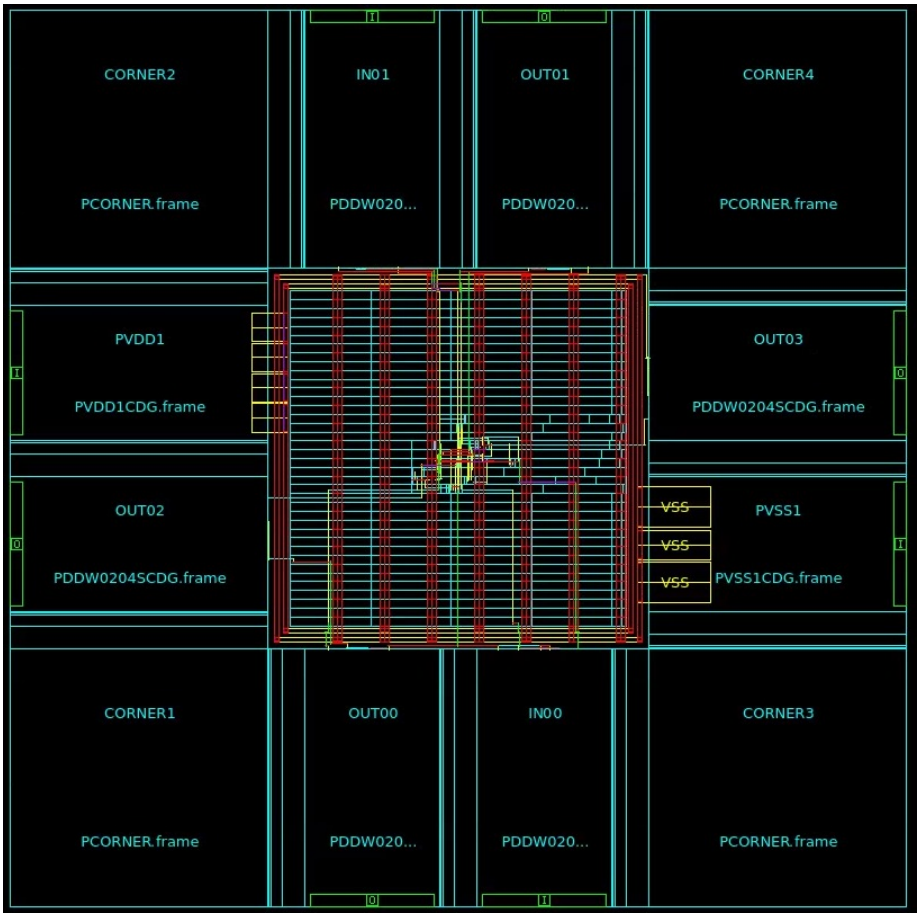

Figura 42: Circuito contador de 4 bits sintetizado con **ICCII**.

La herramienta de **ICCII** cuenta con un visualizador para el árbol del reloj. Para esto se invoca a la herramienta desde el *Task Assistant* y en la sección de *Clock Tree* > *Analysis* > *Color by clock tree*. Esta herramienta despliega cada nivel del árbol desde el puerto hasta el pin, esta herramienta es útil si se desea hacer n análisis exhaustivo y profundo de la síntesis del árbol generado

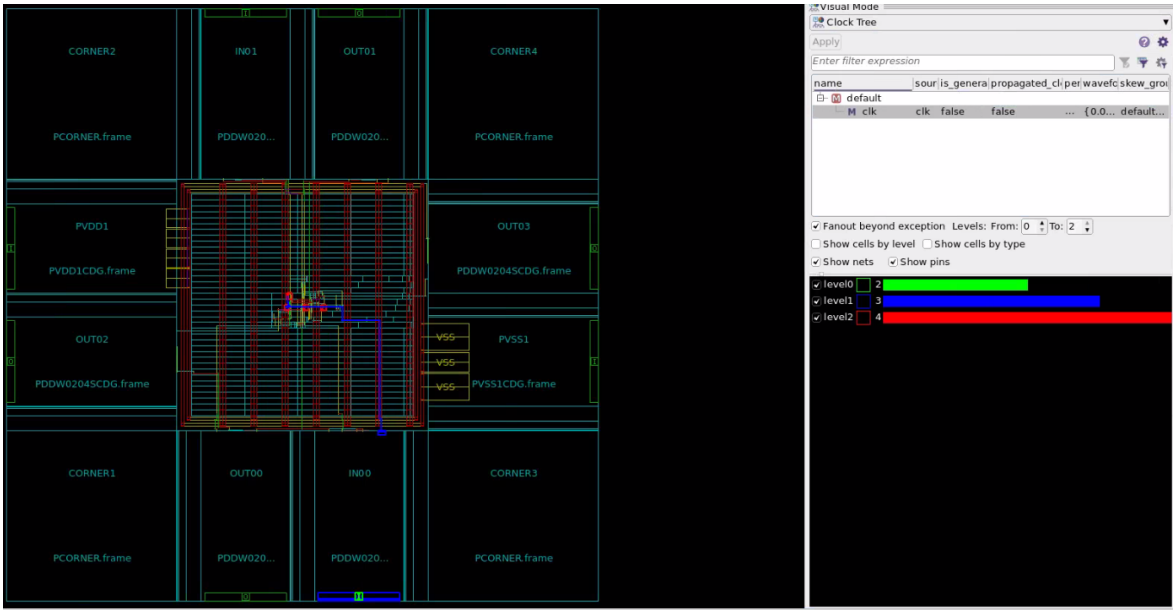

Figura 43: *Color by clock tree* en el contador de 4 bits para la *net* **clk**.

A continuación se muestra el resultado de la verificación DRC y la especificación de los errores obtenidos para un contador de 4 bits:

```
                              RESULTS: NOT CLEAN

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

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


--------------------------------------------------------------------------------
ICV Execution
--------------------------------------------------------------------------------


                                  IC Validator

                Version R-2020.09-SP2-2 for linux64 - Jan 28, 2021 cl#6206082

                        Copyright (c) 1996 - 2021 Synopsys, Inc.
   This software and the associated documentation are proprietary to
    Synopsys,
 Inc. This software may only be used in accordance with the terms and
```

```
   conditions
 of a written license agreement with Synopsys, Inc. All other use,
   reproduction,
            or distribution of this software is strictly prohibited.

Called as: icv -icc2 -f NDM -i CONT_SYN.ndm -p /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_CONT4 -c counter4IO -clf /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4b1/DRC/slnFile.txt -
   icc_density_blockage -icc2_error_categories -I /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_ALU4b1/DRC -icc2_error_browser INST -
   icc2_error_cell signoff_check_drc.err -rc /home/nanoelectronica2021/
   Documentos/Sintesis_Fisica_ALU4b1/DRC/signoff_check_drc.rc -I /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4b1/DRC /home/
   nanoelectronica2021/Documentos/Sintesis_Fisica_CONT4/
   ICVLM18_LM16_LM152_6M.215a_pre041518

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

User name:        nanoelectronica2021
Layout format:    NDM
Input file name:  CONT_SYN.ndm
Top cell name:    counter4IO
Time started:     2021/09/06 09:35:29PM
Time ended:       2021/09/06 09:35:41PM

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

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

Rule and DRC Error Summary

192 total rules were run.
226 rules NOT EXECUTED.
6 rules have violations.
There are 6 total violations.
Refer to counter4IO.LAYOUT_ERRORS
```

Listing 9.7: Resultados de la verificación DRC para un contador de 4 bits

```
       LAYOUT ERRORS RESULTS: ERRORS

    ##### ####  ####   ###  ####   ###
    #     #   # #   # #   # #   # #
    ####  ####  ####  #   # ####   ###
    #     #  #  #  #  #   # #  #      #
    ##### #   # #   #  ###  #   # ####
```

```
================================================================================

Library name:    CONT_SYN.ndm
Structure name:  counter4IO
Generated by:    IC Validator RHEL64 R-2020.09-SP2-2.6206082 2021/01/28
Runset name:     /home/nanoelectronica2021/Documentos/Sintesis_Fisica_CONT4/
    ICVLM18_LM16_LM152_6M.215a_pre041518
User name:       nanoelectronica2021
Time started:    2021/09/06 09:35:34PM
Time ended:      2021/09/06 09:35:41PM

Called as: icv -icc2 -f NDM -i CONT_SYN.ndm -p /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_CONT4 -c counter4IO -clf /home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4b1/DRC/slnFile.txt -
    icc_density_blockage -icc2_error_categories -I /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_ALU4b1/DRC -icc2_error_browser INST -
    icc2_error_cell signoff_check_drc.err -rc /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_ALU4b1/DRC/signoff_check_drc.rc -I /home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4b1/DRC /home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_CONT4/
    ICVLM18_LM16_LM152_6M.215a_pre041518
CLF: -sln "16 18 28 31 33 38 17 27 29 32 39 "

   ERROR SUMMARY

 M1.R.1 : Min M1 area coverage < 30%
   density ..................................... 1 violation found.

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

 M3.R.1 : Min M3 area coverage < 30%
   density ..................................... 1 violation found.

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

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

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

Listing 9.8: Especificación de errores de DRC para un contador de 4 bits

9.5. *Random access memory* RAM

Una vez validado el proceso de síntesis física para diseños de diferente magnitud y complejidad se iniciaron las pruebas con circuitos ya de mayor escala en cuanto a densidad de celdas y conexiones. Una memoria de acceso aleatorio por sus siglas en inglés **Random Access Memory (RAM)** es un circuito integrado altamente utilizado en la industria de los electrónicos de consumo. Para este diseño se implemento una memoria con 5 bits de direccionamiento, lo que da lugar a 32 localidades y cada una de estas es de 4 bits. Por lo tanto tiene una capacidad de 128 bits. Se utilizaron las mismas dimensiones que el circuito de la **ALU** ya que el número de puertos de **IO** es de 17. A continuación se muestra el resultado de la síntesis:

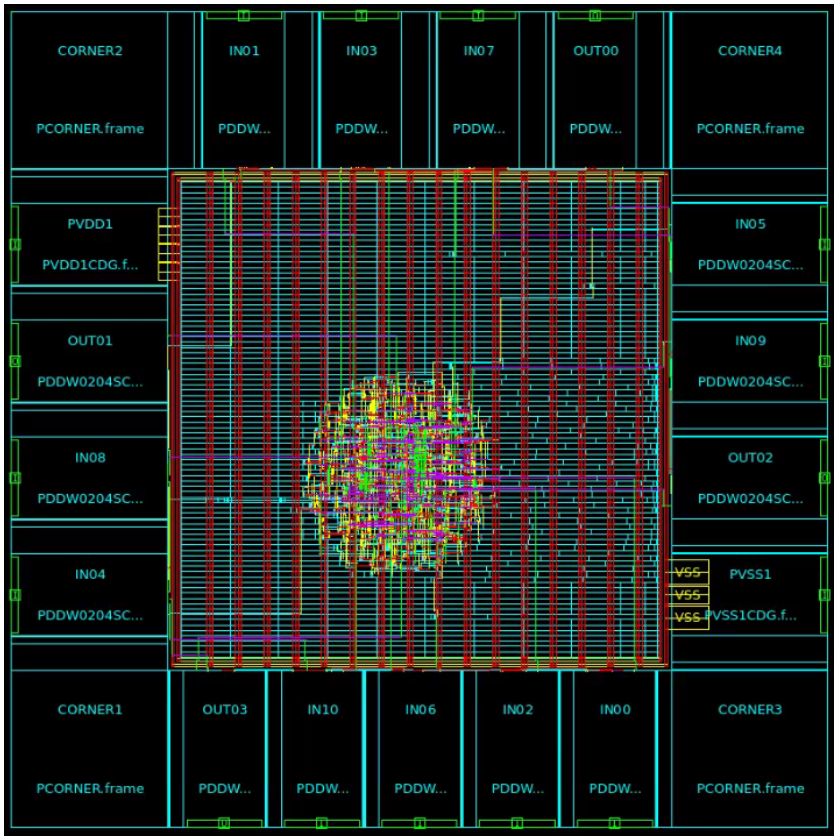

Figura 44: RAM de 4x32 bits sintetizado con **ICCII**.

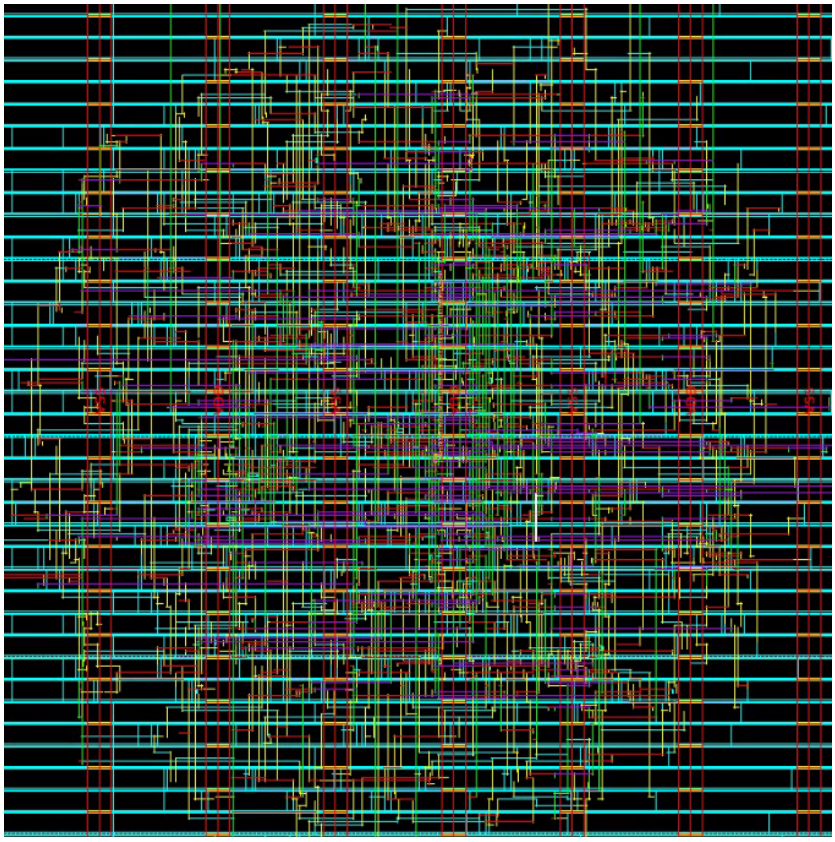

Figura 45: Núcleo de RAM de 4x32 bits sintetizado con **ICCII**.

A continuación se muestra el resultado de la verificación DRC y la especificación de los errores obtenidos para la memoria **RAM**:

```
                              RESULTS: NOT CLEAN

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

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


--------------------------------------------------------------------------------
ICV Execution
--------------------------------------------------------------------------------


                                  IC Validator
```

```
        Version R-2020.09-SP2-2 for linux64 - Jan 28, 2021 cl#6206082

                    Copyright (c) 1996 - 2021 Synopsys, Inc.
   This software and the associated documentation are proprietary to
    Synopsys,
 Inc. This software may only be used in accordance with the terms and
    conditions
 of a written license agreement with Synopsys, Inc. All other use,
    reproduction,
             or distribution of this software is strictly prohibited.

Called as: icv -icc2 -f NDM -i RAM_SYN.ndm -p /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_RAM -c ram_IO -clf /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_RAM/DRC/slnFile.txt -icc_density_blockage -
    icc2_error_categories -I /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_RAM/DRC -icc2_error_browser INST -icc2_error_cell
    signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_RAM/DRC/signoff_check_drc.rc -I /home/nanoelectronica2021
    /Documentos/Sintesis_Fisica_RAM/DRC /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_RAM/ICVLM18_LM16_LM152_6M.215a_pre041518

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

User name:        nanoelectronica2021
Layout format:    NDM
Input file name:  RAM_SYN.ndm
Top cell name:    ram_IO
Time started:     2021/09/09 08:21:54PM
Time ended:       2021/09/09 08:22:22PM

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

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

Rule and DRC Error Summary

192 total rules were run.
226 rules NOT EXECUTED.
6 rules have violations.
There are 6 total violations.
Refer to ram_IO.LAYOUT_ERRORS
```

Listing 9.9: Resultados de la verificación DRC para la memoria **RAM**

```
       LAYOUT ERRORS RESULTS: ERRORS
```

```
    ##### ####  ####   ###  ####   ###
    #     #   # #   # #   # #   # #
    ####  ####  ####  #   # ####   ###
    #     #  #  #  #  #   # #  #      #
    ##### #   # #   #  ###  #   # ####

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

Library name:    RAM_SYN.ndm
Structure name:  ram_IO
Generated by:    IC Validator RHEL64 R-2020.09-SP2-2.6206082 2021/01/28
Runset name:     /home/nanoelectronica2021/Documentos/Sintesis_Fisica_RAM/
    ICVLM18_LM16_LM152_6M.215a_pre041518
User name:       nanoelectronica2021
Time started:    2021/09/09 08:22:08PM
Time ended:      2021/09/09 08:22:22PM

Called as: icv -icc2 -f NDM -i RAM_SYN.ndm -p /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_RAM -c ram_IO -clf /home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_RAM/DRC/slnFile.txt -icc_density_blockage -
    icc2_error_categories -I /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_RAM/DRC -icc2_error_browser INST -icc2_error_cell
    signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_RAM/DRC/signoff_check_drc.rc -I /home/nanoelectronica2021
    /Documentos/Sintesis_Fisica_RAM/DRC /home/nanoelectronica2021/Documentos/
    Sintesis_Fisica_RAM/ICVLM18_LM16_LM152_6M.215a_pre041518
CLF: -sln "16 18 28 31 33 38 17 27 29 32 39 "

   ERROR SUMMARY

 M1.R.1 : Min M1 area coverage < 30%
   density ..................................... 1 violation found.

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

 M3.R.1 : Min M3 area coverage < 30%
   density ..................................... 1 violation found.

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

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

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

Listing 9.10: Especificación de errores de DRC para la memoria **RAM**

9.6. CI personalizado: *El Gran Jaguar*

El propósito de este proyecto, que lleva ya 2 años en vigencia, es el poder enviar a fabricar, por medio de **IMEC**, a **TSMC** el primer Chip a escala nanométrica de la región. Este es una máquina de estados finitos que por medio de un reloj genera una secuencia de caracteres codificados en **ASCII** que luego serán ensamblados para luego reproducir un audio final. Este circuito cuenta con 2 puertos de entradas y salidas de **PG**, 1 reloj, 2 pines de selección para el audio y 8 pines de salida que estará mostrando la codificación de los diferentes caracteres alternando en cada pulso de reloj. Adicionalmente este circuito cuenta con un *ring oscillator* el cual sirve como un reloj compuesto de compuertas NOT enlazadas.

9.6.1. Texto 1

-> Hola, soy El Gran Jaguar, el primer nanochip elaborado en Centroamérica por una Universidad. Desarrollado completamente en la Universidad del Valle de Guatemala por alumnos graduandos de Ingeniería en Electrónica entre 2019 y 2021.

9.6.2. Texto 2

-> El equipo encargado de mi creacion se conformo por: Geovanni Flores, Matthias Sibrian, Steven Rubio, Julio Shin, Karol Cardona, Luis Najera, Luis Abadia, Joel Gonzalez, Ricardo Giron, Carlos Esquit, Charlie Ayenci, Elmer Torres, Gerardo Cardoza, Jonathan de los Santos, Antonio Altuna, Kurt Keller y Luis Rivera.

El nombre *El Gran Jaguar* es un homenaje a la cultura guatemalteca. Siguiendo la dinámica que los desarrolladores de circuitos integrados tienen de enumerar las versiones de sus diseños, *El gran Jaguar* es el Templo 1 de la ciudad de Tikal. La grandeza de esta edificación representa el impacto que este proyecto tiene en la sociedad moderna. A continuación imágenes de los resultados obtenidos de la síntesis física de este diseño.

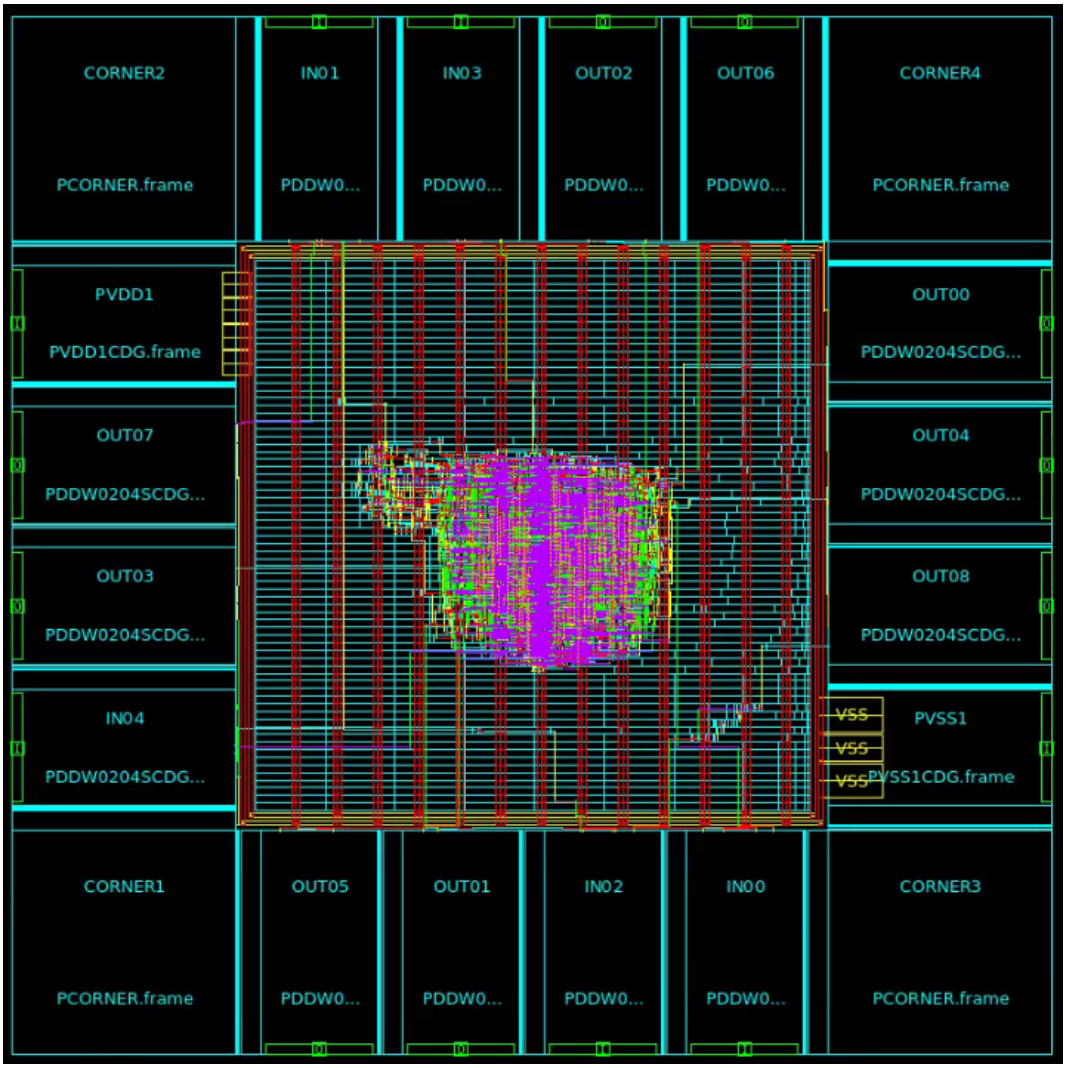

Figura 46: Circuito *El Gran Jaguar* sintetizado con **ICCII**.

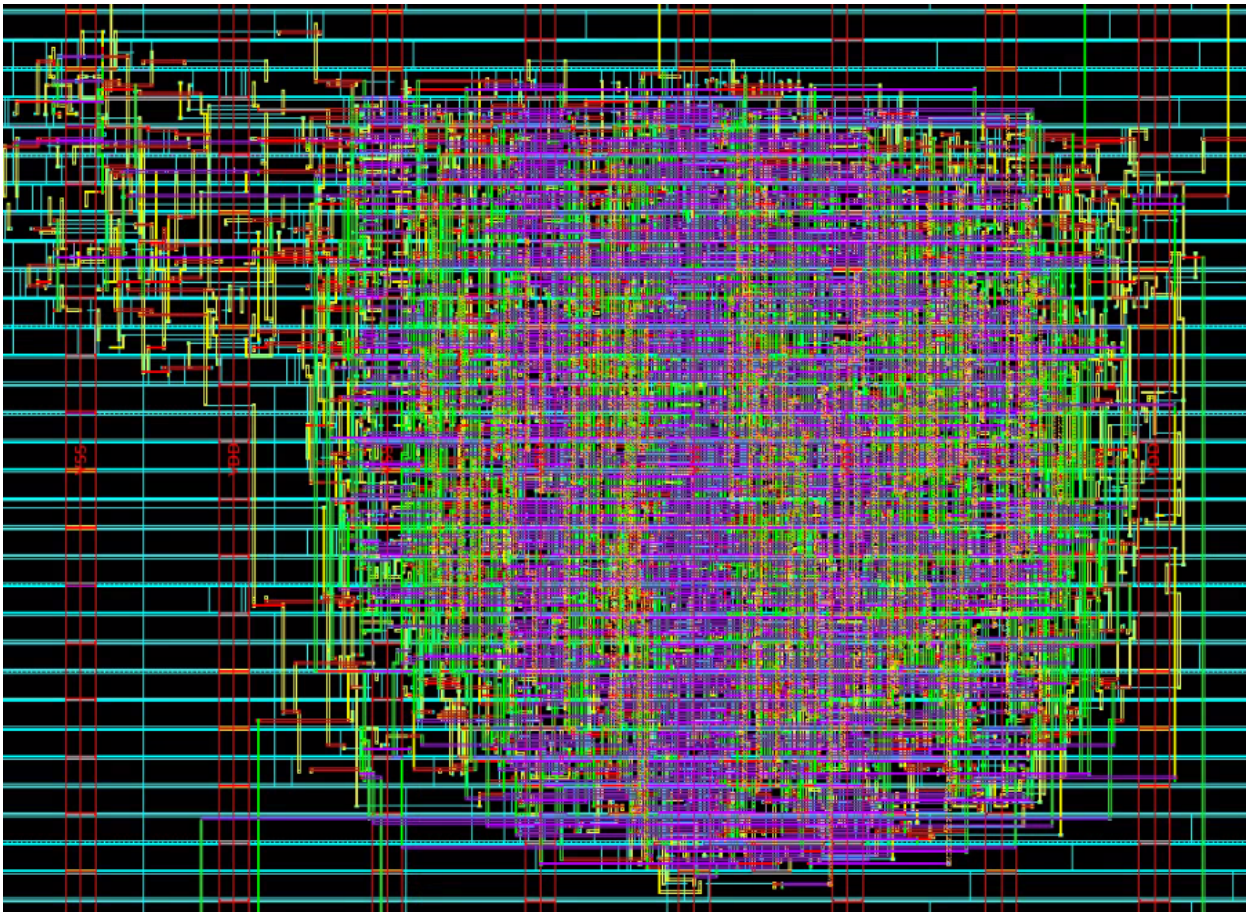

Figura 47: Núcleo del Circuito *El Gran Jaguar* sintetizado con **ICCII**.

A continuación se muestra el resultado de la verificación DRC y la especificación de los errores obtenidos para el circuito *El Gran Jaguar*:

```
                                RESULTS: NOT CLEAN

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

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


--------------------------------------------------------------------------------
ICV Execution
--------------------------------------------------------------------------------
```

```
                                IC Validator

          Version R-2020.09-SP2-2 for linux64 - Jan 28, 2021 cl#6206082

                     Copyright (c) 1996 - 2021 Synopsys, Inc.
    This software and the associated documentation are proprietary to
     Synopsys,
  Inc. This software may only be used in accordance with the terms and
     conditions
  of a written license agreement with Synopsys, Inc. All other use,
     reproduction,
              or distribution of this software is strictly prohibited.

Called as: icv -icc2 -f NDM -i EL_GRAN_JAGUAR.ndm -p /home/
    nanoelectronica2021/Documentos/El_Gran_Jaguar2 -c chip_IO -clf /home/
    nanoelectronica2021/Documentos/El_Gran_Jaguar2/DRC/slnFile.txt -
    icc_density_blockage -icc2_error_categories -I /home/nanoelectronica2021/
    Documentos/El_Gran_Jaguar2/DRC -icc2_error_browser INST -icc2_error_cell
    signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
    El_Gran_Jaguar2/DRC/signoff_check_drc.rc -I /home/nanoelectronica2021/
    Documentos/El_Gran_Jaguar2/DRC /home/nanoelectronica2021/Documentos/
    El_Gran_Jaguar2/ICVLM18_LM16_LM152_6M.215a_pre041518

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

User name:         nanoelectronica2021
Layout format:     NDM
Input file name:   EL_GRAN_JAGUAR.ndm
Top cell name:     chip_IO
Time started:      2021/11/09 11:05:43PM
Time ended:        2021/11/09 11:05:56PM

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

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

Rule and DRC Error Summary

192 total rules were run.
226 rules NOT EXECUTED.
6 rules have violations.
There are 6 total violations.
Refer to chip_IO.LAYOUT_ERRORS
```

Listing 9.11: Resultados de la verificación DRC para el circuito *El Gran Jaguar*

```
       LAYOUT ERRORS RESULTS: ERRORS

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

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

Library name:    EL_GRAN_JAGUAR.ndm
Structure name:  chip_IO
Generated by:    IC Validator RHEL64 R-2020.09-SP2-2.6206082 2021/01/28
Runset name:     /home/nanoelectronica2021/Documentos/El_Gran_Jaguar2/
   ICVLM18_LM16_LM152_6M.215a_pre041518
User name:       nanoelectronica2021
Time started:    2021/11/09 11:05:47PM
Time ended:      2021/11/09 11:05:56PM

Called as: icv -icc2 -f NDM -i EL_GRAN_JAGUAR.ndm -p /home/
   nanoelectronica2021/Documentos/El_Gran_Jaguar2 -c chip_IO -clf /home/
   nanoelectronica2021/Documentos/El_Gran_Jaguar2/DRC/slnFile.txt -
   icc_density_blockage -icc2_error_categories -I /home/nanoelectronica2021/
   Documentos/El_Gran_Jaguar2/DRC -icc2_error_browser INST -icc2_error_cell
   signoff_check_drc.err -rc /home/nanoelectronica2021/Documentos/
   El_Gran_Jaguar2/DRC/signoff_check_drc.rc -I /home/nanoelectronica2021/
   Documentos/El_Gran_Jaguar2/DRC /home/nanoelectronica2021/Documentos/
   El_Gran_Jaguar2/ICVLM18_LM16_LM152_6M.215a_pre041518
CLF: -sln "16 18 28 31 33 38 17 27 29 32 39 "

   ERROR SUMMARY

 M1.R.1 : Min M1 area coverage < 30%
   density ..................................... 1 violation found.

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

 M3.R.1 : Min M3 area coverage < 30%
   density ..................................... 1 violation found.

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

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

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

```
density ..................................... 1 violation found.
```

Listing 9.12: Especificación de errores de DRC para el circuito *El Gran Jaguar*

CAPÍTULO 10

Conclusiones

- Se mejoró el proceso de síntesis física previo actualizando la herramienta de trabajo a IC Compiler II, generando librerías compatibles utilizando el Library Manager y solucionando las diferentes problemáticas presentadas por la verificación **DRC** utilizando documentación de **TSMC** y SolvNet.
- A través del proceso de síntesis física de las diferentes compuertas y circuitos realizados y validados se puede concluir que la migración del proceso de síntesis física de la herramienta de IC Compiler I a IC Compiler II se realizó de forma exitosa.
- Utilizando la herramienta Library Manager de IC Compiler II se pudo crear las librerías de TSMC que se utilizaron y verificar su compatibilidad ya que se obtuvieron librerías generadas libres de errores.
- Se lograron posicionar y rutear las celdas de cada circuito, durante el proceso de síntesis física, de forma eficiente y exitosa utilizando los diferentes comandos de optimización del proceso de *placement* y *pre-routing* y así tener en todas las iteraciones realizadas 0 errores de esa índole.
- Se logró minimizar de forma significativa la cantidad de errores en todos los circuitos diseñados por medio de la correcta configuración del *runset* de **DRC**.
- La comunicación entre los integrantes de este proyecto permitió que las diferentes partes del flujo se realizaran con éxito y validaran de forma correcta debido a que se establecieron metas realistas y hubo apoyo y disponibilidad de cada uno de los integrantes.
- Se generaron *scripts*, reportes de resultados y librerías **NDM** funcionales las cuales fueron delegadas a la verificación de **LVS** y el proceso de automatización del flujo.
- Se documentó todo el proceso de síntesis física y verificación **DRC** con el fin de crear una base sólida en la cual los futuros alumnos a cargo del proyecto puedan minimizar la curva de aprendizaje y mejorar lo que actualmente se presenta.

CAPÍTULO 11

Recomendaciones

- Crear una síntesis física utilizando un circuito secuencial complejo con el fin de crear más robustez en la sintetización del árbol de relojes.
- Establecer comunicación con el equipo de **IMEC** desde principios de año para que cualquier documentación necesaria durante el proceso de realización del trabajo de graduación sea más rápida de conseguir.
- Leer la documentación de la herramienta que se utilice antes de iniciar a trabajar, esto permite familiarizarse con el entorno y sus funcionalidades previo al uso de la herramienta.
- Verificar la vigencia de los archivos, *runsets* y programas utilizados para mantener actualizado el proceso de síntesis física.
- Investigar a profundidad la síntesis física utilizando un flujo con UTF/UTF Gold para lo que es **PG** Planning y verificar si es implementable en el flujo actual.
- Investigar sobre la obtención de parámetros internos del diseño en la herramineta IC Compiler II para crear una síntesis capaz de auto-gestionarse según las características como tamaño, puertos de entradas y salidas, uso de reloj, entre otras.
- Utilizar SolveNet para obtener la documentación necesaria y complementaria para el trabajo de graduación.
- Replicar el proceso actual y familiarizarse con la forma en la que la interfaz se desglosa para minimizar la curva de aprendizaje de estas herramientas y los conceptos nuevos que se presenten.
- Mantener una comunicación con el proceso de síntesis lógica es indispensable ya que hay cosas que se pueden realizar en cualquiera de las dos etapas y estandarizar y delegar las responsabilidades como: colocación de *pads* de alimentación y relojes permite que el proceso esté en sintonía y no se dupliquen o hagan falta estos elementos en las verificaciones posteriores.

CAPÍTULO 12

Bibliografía

[1] J. de los Santos, "Diseño de un sumador/restador completo de 32 bits con tecnología CMOS en un proceso de 28 nanómetros usando aplicaciones de diseño de la empresa Synopsys.," en *Trabajo de Graducacion, Modalidad Tésis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2014.

[2] S. Rubio, "Definición del Flujo de Diseño para Fabricación de un Chip con Tecnologia VLSI CMOS," en *Trabajo de Graducacion, Modalidad Tésis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2019.

[3] L. 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 Graducacion, Modalidad Tésis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2019.

[4] L. 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 Graducacion, Modalidad Tésis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2020.

[5] M. 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 Graducacion, Modalidad Tésis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2020.

[6] M. Sibrian, "Verificación de reglas de diseño (DRC) para el desarrollo de un flujo funcional de un circuito integrado con tecnología nanométrica," en *Trabajo de Graducacion, Modalidad Tésis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2020.

[7] "Apple unleashes M1," *Apple Newsroom*, nov. de 2020. dirección: `https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/`.

[8] K. G. Krishna. B, "Circuit Design Enviroment and Layout Planning," *Intel Technology Journal*, págs. 13-20, 1999.

[9] A. S. Sedra y K. C. Smith, "Devices and Basic Circuits," en *Microelectronic Circuits*, 7th. Oxford University Press, 2015, págs. 220, 315.

[10] C. Esquit, *Computer Architecture Lecture 1: Introduction and Five Components of a Computer*, jul. de 2020.

[11] ——, *Introduction to VLSI System Design Lecture: Fabrication and Layout*, mayo de 2021.

[12] J. Girón, "Etapa de verificación física de Diseño en Silicio vs. Esquemático (LVS) en el flujo de diseño para un chip a nanoescala," en *Trabajo de Graducacion, Modalidad Tésis*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2019.

[13] K. H, "Digital Integrated Circuit Design From VLSI Architectures to CMOS Fabrication," *Intel Technology Journal*, págs. 13-20, 1999.

[14] Synopsys, *Design Planning User Guide: Version S-2021.06, June 2021*, Synopsys Inc., USA, 2021.

[15] ——, *IC Compiler ™II Implementation User Guide: Version S-2021.06, June 2021*, Synopsys Inc., USA, 2021.

[16] L. A. López, "Manual Síntesis Física," en *Manual Síntesis Física para IC Compiler I*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2020.

[17] M. Sibrian, "Manual de DRC," en *MANUAL DE USUARIO PARA LA VERIFICACIÓN DE REGLAS DE DISEÑO EN UN CIRCUITO INTEGRADO CON TECNOLOGÍA NANOMÉTRICA*, Facultad de Ingeniería Universidad del Valle de Guatemala, 2020.

[18] *EDA101 - Introduction to Electronic Design Automation*. Cadence Design Systems, abr. de 2020. dirección: `https://www.youtube.com/watch?v=HkT363NUOBA`.

[19] Synopsys. dirección: `https://www.synopsys.com/silicon-design.html`.

[20] ——, dirección: `https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html`.

[21] ——, dirección: `https://www.synopsys.com/silicon/mask-synthesis/icv-workbench.html`.

[22] ——, *Library Manager User Guide*, Synopsys Inc., USA, 2021.

[23] ——, *IC Compiler ™II Tool Commands: Version S-2021.06, June 2021*, Synopsys Inc., USA, 2021.

[24] ——, *IC Compiler ™II Application Options and Attributes: Version S-2021.06, June 2021*, Synopsys Inc., USA, 2021.

[25] ——, *Filler Cell Insertion in IC Compiler II*, Synopsys Inc., USA, 2017.

CAPÍTULO 13

Anexos

13.1. *Script* para generar las librerías de referencia

```
#Importar el tech file
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

#Importar los db (no incluimos los IO)
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 }

#Importar el LEF
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

#Correr y desplegar el check
current_workspace; check_workspace
gui_create_window -type MessageBrowserWindow
```

```
open_ems_database check_workspace.ems

#Commit y save a la libreria
current_workspace NormalWorkspace; commit_workspace  -output
    StandardWorkspace.ndm

#Creamos el ndm de los pads
create_workspace -flow normal -technology /home/nanoelectronica2021/
    Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf PadsWorkspace
read_db { /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/
    tcb018gbwp7t/LM/tpd018nvtc.db }
read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
    tpd018nv_6lm.lef
current_workspace; check_workspace
current_workspace PadsWorkspace; commit_workspace  -output PadsWorkspace.ndm

#Creamos el ndm para los corners
create_workspace -flow physical_only -technology /home/nanoelectronica2021/
    Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf CornersWorkspace
read_db { /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/
    tcb018gbwp7t/LM/tpd018nvtc.db }
read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
    tpd018nv_6lm.lef
current_workspace; check_workspace
current_workspace CornersWorkspace; commit_workspace  -output
    CornersWorkspace.ndm

#Creamos el ndm para los fillers std cells
create_workspace -flow physical_only -technology /home/nanoelectronica2021/
    Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf PhysicalOnlyWorkspace
read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
    tcb018gbwp7t_6lm.lef
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
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
```

13.2. *Script* para sintetizar una compuerta NOT

```
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
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

#Abrimos el verilog file sintetizado
read_verilog /home/nanoelectronica2021/Escritorio/ELMER_NOT/salidas_not_io/
    FA_syn.v

read_sdc -echo /home/nanoelectronica2021/Escritorio/ELMER_NOT/salidas_not_io
    /FA_sdc_p.sdc

#Importamos las TLU+ y el map
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

#Limpiamos las cosas de PG
remove_pg_strategies -all

#Agregamos 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"

#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

#Creacion de pads para  VDD y VSS
create_cell {PVDD} PVDD1CDG
create_cell {PVSS} PVSS1CDG

#Creacion de 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

#Floorplan inicial
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {100 100} -core_offset {125}

#Creacion del anillo IO
create_io_ring -name anillo_IO -corner_height 115

#Coloca los pines de entradas y salidas (Pads) en un lugar arbitrario de no
    ser especificado en el floorplan
place_io

#Creacion del 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 IO ejemplo 2 -EL MEJOR EJEMPLO-
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}} }
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 pg ring
```

```
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2 METAL3}

#Creamos el placement
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

# Configuramos el DRC runset file
set_app_options -list {signoff.check_design.run_dir {/home/
    nanoelectronica2021/Documentos/T-018-CM-SP-018-W1_1_0A/
    Sintesis_Fisica_NOT/DRC/}}
set_app_options -list {signoff.check_drc.run_dir {/home/nanoelectronica2021/
    Documentos/T-018-CM-SP-018-W1_1_0A/Sintesis_Fisica_NOT/DRC/}}
set_app_options -list {signoff.check_design.runset {ICVLM18_LM16_LM152_6M
    .215a_pre041518}}
set_app_options -list {signoff.check_drc.runset {ICVLM18_LM16_LM152_6M.215
    a_pre041518}}


#Guardamos el bloque y corremos DRC y su Fix
save_block NOT_SYN.ndm:Not_IO
signoff_fix_drc

save_block NOT_SYN.ndm:Not_IO
signoff_check_drc

check_lvs
```

```
save_block NOT_SYN.ndm:Not_IO

#Creamos el verilog equivalente de la sintesis fisica
write_verilog -include all NOT_SYN.v

gui_show_error_data
```

13.3. *Script* para sintetizar una compuerta XOR

```
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
create_lib XOR_SYN2.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

#Abrimos el verilog file sintetizado
read_verilog /mnt/nfs/compartida/XOR2/sintesis_logica/sintesis_cell_io/
    salidas/out_xor_io.v

read_sdc -echo -syntax_only /mnt/nfs/compartida/XOR2/sintesis_logica/
    sintesis_cell_io/salidas/out_xor_io.sdc

#Importamos las TLU+ y el map
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

#Limpiamos las cosas de PG
remove_pg_strategies -all

#Agregamos 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"

#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

#Creacion de pads para  VDD y VSS
create_cell {PVDD1 PVDD2} PVDD1CDG
create_cell {PVSS1 PVSS2} PVSS1CDG

#Creacion de 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

#Floorplan inicial
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {135 135} -core_offset {125}

#Creacion del anillo IO
create_io_ring -name anillo_IO_XOR -corner_height 115

#Coloca los pines de entradas y salidas (Pads) en un lugar arbitrario de no
    ser especificado en el floorplan
add_to_io_guide [get_io_guides anillo_IO_XOR.left] PVDD*
add_to_io_guide [get_io_guides anillo_IO_XOR.right] PVSS*
place_io

#Creacion del 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 IO ejemplo 2 -EL MEJOR EJEMPLO-
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}} }
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {259.960 265.280}}
     -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 pg ring
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2 METAL3}

#Creamos el placement
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_XOR.top
    anillo_IO_XOR.right anillo_IO_XOR.left anillo_IO_XOR.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

#Configuramos el DRC runset file
set_app_options -list {signoff.check_design.run_dir {/home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_XORV2/DRC/}}
set_app_options -list {signoff.check_drc.run_dir {/home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_XORV2/DRC/}}
set_app_options -list {signoff.check_design.runset {ICVLM18_LM16_LM152_6M
    .215a_pre041518}}
set_app_options -list {signoff.check_drc.runset {ICVLM18_LM16_LM152_6M.215
    a_pre041518}}

#Guardamos el bloque y corremos DRC y su Fix
save_block XOR_SYN2.ndm:Xor_IO
signoff_fix_drc

save_block XOR_SYN2.ndm:Xor_IO
signoff_check_drc
```

```
check_lvs
save_block XOR_SYN2.ndm:Xor_IO

#Creamos el verilog equivalente de la sintesis fisica
write_verilog -include all XOR_SYN2.v

gui_show_error_data
```

13.4. *Script* para sintetizar un circuito *full adder*

```
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
create_lib FA_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

#Abrimos el verilog file sintetizado
read_verilog /mnt/nfs/compartida/FullAdder4/sintesis_logica/Sintesis_cell_io
    /salidas/out_Fulladd_io.v

read_sdc -echo -syntax_only /mnt/nfs/compartida/FullAdder4/sintesis_logica/
    Sintesis_cell_io/salidas/out_Fulladd_io.sdc

#Importamos las TLU+ y el map
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

#Limpiamos las cosas de PG
remove_pg_strategies -all

#Agregamos 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"

#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

#Creacion de pads para  VDD y VSS
create_cell PVDD1 PVDD1CDG
create_cell PVSS1 PVSS1CDG
```

```
#Creacion de 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

#Floorplan inicial
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {285 285} -core_offset {125}

#Creacion del anillo IO
create_io_ring -name anillo_IO_FA -corner_height 115

#Coloca los pines de entradas y salidas (Pads) en un lugar arbitrario de no
    ser especificado en el floorplan
add_to_io_guide [get_io_guides anillo_IO_FA.left] PVDD*
add_to_io_guide [get_io_guides anillo_IO_FA.right] PVSS*
place_io

#Creacion del 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 IO ejemplo 2 -EL MEJOR EJEMPLO-
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}} }
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {409.480 414.240}}
     -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 pg ring
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2 METAL3}

#Creamos el placement
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_FA.top
    anillo_IO_FA.right anillo_IO_FA.left anillo_IO_FA.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


# Configuramos el DRC runset file
set_app_options -list {signoff.check_design.run_dir {/home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_FA/DRC/}}
set_app_options -list {signoff.check_drc.run_dir {/home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_FA/DRC/}}
set_app_options -list {signoff.check_design.runset {ICVLM18_LM16_LM152_6M
    .215a_pre041518}}
set_app_options -list {signoff.check_drc.runset {ICVLM18_LM16_LM152_6M.215
    a_pre041518}}


#Guardamos el bloque y corremos DRC y su Fix
save_block FA_SYN.ndm:fulladd_io
```

```
signoff_fix_drc

save_block FA_SYN.ndm:fulladd_io
signoff_check_drc

check_lvs
save_block FA_SYN.ndm:fulladd_io

#Creamos el verilog equivalente de la sintesis fisica
write_verilog -include all FA_SYN.v

gui_show_error_data
```

13.5. *Script* para sintetizar un circuito de una unidad aritmético lógica ALU

```
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
create_lib ALU_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

#Abrimos el verilog file sintetizado
read_verilog /mnt/nfs/compartida/ALU4b/sintesis_logica/Sintesis_cell_io/
    salidas/ALU4_io_out.v

read_sdc -echo /mnt/nfs/compartida/ALU4b/sintesis_logica/Sintesis_cell_io/
    salidas/ALU4_io_out.sdc

#Importamos las TLU+ y el map
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

#Limpiamos las cosas de PG
remove_pg_strategies -all

#Agregamos 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"

#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER
```

```
#Creacion de pads para  VDD y VSS
#create_cell {PVDD2} PVDD1CDG
#create_cell {PVSS2} PVSS1CDG

#Creacion de 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

#Floorplan inicial
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {350 350} -core_offset {125}

#Creacion del anillo IO
create_io_ring -name anillo_IO_ALU -corner_height 115

#Coloca los pines de entradas y salidas (Pads) en un lugar arbitrario de no
    ser especificado en el floorplan
add_to_io_guide [get_io_guides anillo_IO_ALU.left] PVDD*
add_to_io_guide [get_io_guides anillo_IO_ALU.right] PVSS*
place_io

#Creacion del 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 IO ejemplo 2 -EL MEJOR EJEMPLO-
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}} }
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {475.000 480.880}}
     -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 pg ring
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2 METAL3}

#Creamos el placement
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_ALU.top
    anillo_IO_ALU.right anillo_IO_ALU.left anillo_IO_ALU.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

# Configuramos el DRC runset file
set_app_options -list {signoff.check_design.run_dir {/home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4B/DRC/}}
set_app_options -list {signoff.check_drc.run_dir {/home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_ALU4B/DRC/}}
set_app_options -list {signoff.check_design.runset {ICVLM18_LM16_LM152_6M
    .215a_pre041518}}
set_app_options -list {signoff.check_drc.runset {ICVLM18_LM16_LM152_6M.215
    a_pre041518}}
```

```
#Guardamos el bloque y corremos DRC y su Fix
save_block ALU_SYN.ndm:ALU_IO
signoff_fix_drc

save_block ALU_SYN.ndm:ALU_IO
signoff_check_drc

check_lvs
save_block ALU_SYN.ndm:ALU_IO

#Creamos el verilog equivalente de la sintesis fisica
write_verilog -include all ALU4B_SYN.v

gui_show_error_data
```

13.6. *Script* para sintetizar un circuito contador de 4 bits

```
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
create_lib CONT_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/synopsys/Compartida_NFS/ASA/LibreriasNDM/TSMCWorkspace.
    ndm

#Abrimos el verilog file sintetizado
read_verilog /mnt/nfs/synopsys/Compartida_NFS/counter4B/sintesis_logica/
    Sintesis_cell_io/salidas/counter4_io_out.v

read_sdc -echo /mnt/nfs/synopsys/Compartida_NFS/counter4B/sintesis_logica/
    Sintesis_cell_io/salidas/counter4_io_out.sdc

#Importamos las TLU+ y el map
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

#Limpiamos las cosas de PG
#save_block ALU_SYN.ndm:ALU_IO
remove_pg_strategies -all

#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER
```

```
#Creacion de pads para  VDD y VSS
create_cell {PVDD1} PVDD1CDG
create_cell {PVSS1} PVSS1CDG

#Creacion de 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

#Floorplan inicial
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {150 150} -core_offset {125}


# Creacion del anillo IO
create_io_ring -name anillo_IO_CONT -corner_height 115

#Coloca los pines de entradas y salidas (Pads) en un lugar arbitrario de no
    ser especificado en el floorplan
add_to_io_guide [get_io_guides anillo_IO_CONT.left] PVDD*
add_to_io_guide [get_io_guides anillo_IO_CONT.right] PVSS*
place_io

#Creacion del 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 IO ejemplo 2 -EL MEJOR EJEMPLO-
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}} }
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {274.520 280.960}}
     -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 pg ring
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2 METAL3}

#Creamos el placement
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

#Sintetizacion de relojes
check_clock_trees -clocks clk
check_design -checks pre_clock_tree_stage
synthesize_clock_trees -clocks clk -postroute -routed_clock_stage
    detail_with_signal_routes
clock_opt -list_only
check_design -checks cts_qor

#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_CONT.top
    anillo_IO_CONT.right anillo_IO_CONT.left anillo_IO_CONT.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
```

```
# Configuramos el DRC runset file
set_app_options -list {signoff.check_design.run_dir {/home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4b1/DRC/}}
set_app_options -list {signoff.check_drc.run_dir {/home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_ALU4b1/DRC/}}
set_app_options -list {signoff.check_design.runset {ICVLM18_LM16_LM152_6M
    .215a_pre041518}}
set_app_options -list {signoff.check_drc.runset {ICVLM18_LM16_LM152_6M.215
    a_pre041518}}


#Guardamos el bloque y corremos DRC y su Fix
save_block CONT_SYN.ndm:counter4IO
signoff_fix_drc

save_block CONT_SYN.ndm:counter4IO
signoff_check_drc

check_lvs
save_block CONT_SYN.ndm:counter4IO

#Creamos el verilog equivalente de la sintesis fisica
write_verilog -include all CONT_SYN.v

gui_show_error_data
```

13.7. *Script* para sintetizar una memoria RAM de 4x32 bits

```
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
create_lib RAM_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/synopsys/Compartida_NFS/ASA/LibreriasNDM/TSMCWorkspace.
    ndm

#Abrimos el verilog file sintetizado
read_verilog /mnt/nfs/synopsys/Compartida_NFS/ram5b/sintesis_logica/
    Sintesis_cell_io/salidas/ram_io_out.v

read_sdc -echo /mnt/nfs/synopsys/Compartida_NFS/ram5b/sintesis_logica/
    Sintesis_cell_io/salidas/ram_io_out.sdc

#Importamos las TLU+ y el map
```

```
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

#Limpiamos las cosas de PG
#save_block RAM_SYN.ndm:ram_IO
remove_pg_strategies -all

#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

#Creacion de pads para  VDD y VSS
create_cell {PVDD1} PVDD1CDG
create_cell {PVSS1} PVSS1CDG

#Creacion de 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

#Floorplan inicial
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {350 350} -core_offset {125}

#Creacion del anillo IO
create_io_ring -name anillo_IO_CONT -corner_height 115

#Coloca los pines de entradas y salidas (Pads) en un lugar arbitrario de no
    ser especificado en el floorplan
add_to_io_guide [get_io_guides anillo_IO_CONT.left] PVDD*
add_to_io_guide [get_io_guides anillo_IO_CONT.right] PVSS*
place_io

#Creacion del 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 IO ejemplo 2 -EL MEJOR EJEMPLO-
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}} }
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {475.000 480.880}}
     -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 pg ring
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2 METAL3}
check_pg_connectivity

#Creamos el placement
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
report_placement

#Sintetizacion de relojes
check_clock_trees -clocks clk
check_design -checks pre_clock_tree_stage
synthesize_clock_trees -clocks clk -postroute -routed_clock_stage
    detail_with_signal_routes
clock_opt -list_only
check_design -checks cts_qor

#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_CONT.top
```

```
    anillo_IO_CONT.right anillo_IO_CONT.left anillo_IO_CONT.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


# Configuramos el DRC runset file
set_app_options -list {signoff.check_design.run_dir {/home/
    nanoelectronica2021/Documentos/Sintesis_Fisica_ALU4b1/DRC/}}
set_app_options -list {signoff.check_drc.run_dir {/home/nanoelectronica2021/
    Documentos/Sintesis_Fisica_ALU4b1/DRC/}}
set_app_options -list {signoff.check_design.runset {ICVLM18_LM16_LM152_6M
    .215a_pre041518}}
set_app_options -list {signoff.check_drc.runset {ICVLM18_LM16_LM152_6M.215
    a_pre041518}}


#Guardamos el bloque y corremos DRC y su Fix
save_block RAM_SYN.ndm:ram_IO
signoff_fix_drc

save_block RAM_SYN.ndm:ram_IO
signoff_check_drc

check_lvs
```

13.8. *Script* para sintetizar el circuito *El Gran Jaguar*

```
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
create_lib EL_GRAN_JAGUAR.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/synopsys/Compartida_NFS/ASA/LibreriasNDM/TSMCWorkspace.
    ndm

#Abrimos el verilog file sintetizado
```

```
read_verilog /mnt/nfs/synopsys/Compartida_NFS/GRANJAGUAR/El_Gran_Jaguar_v2/
    Sintesis_cell_io/salidas/out_final_io.v

read_sdc -echo /mnt/nfs/synopsys/Compartida_NFS/GRANJAGUAR/El_Gran_Jaguar_v2
    /Sintesis_cell_io/salidas/out_final_io.sdc

#Importamos las TLU+ y el map
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

#Limpiamos las cosas de PG
remove_pg_strategies -all

#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

#Creacion de pads para  VDD y VSS
create_cell {PVDD1} PVDD1CDG
create_cell {PVSS1} PVSS1CDG

#Creacion de 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

#Floorplan inicial
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {285 285} -core_offset {125}

#Creacion del anillo IO
create_io_ring -name anillo_IO_JAGUAR -corner_height 115

#Coloca los pines de entradas y salidas (Pads) en un lugar arbitrario de no
    ser especificado en el floorplan
add_to_io_guide [get_io_guides anillo_IO_JAGUAR.left] PVDD*
add_to_io_guide [get_io_guides anillo_IO_JAGUAR.right] PVSS*
place_io

#Creacion del 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 IO ejemplo 2 -EL MEJOR EJEMPLO-
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}} }
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {409.480 414.240}}
     -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 pg ring
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2 METAL3}

#Creamos el placement
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

#Sintetizacion de relojes
check_clock_trees -clocks clk
check_design -checks pre_clock_tree_stage
synthesize_clock_trees -clocks clk -postroute -routed_clock_stage
    detail_with_signal_routes
clock_opt -list_only
check_design -checks cts_qor

#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_JAGUAR.top
    anillo_IO_JAGUAR.right anillo_IO_JAGUAR.left anillo_IO_JAGUAR.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


# Configuramos el DRC runset file
set_app_options -list {signoff.check_design.run_dir {/home/
    nanoelectronica2021/Documentos/El_Gran_Jaguar/DRC/}}
set_app_options -list {signoff.check_drc.run_dir {/home/nanoelectronica2021/
    Documentos/El_Gran_Jaguar/DRC/}}
set_app_options -list {signoff.check_design.runset {ICVLM18_LM16_LM152_6M
    .215a_pre041518}}
set_app_options -list {signoff.check_drc.runset {ICVLM18_LM16_LM152_6M.215
    a_pre041518}}


#Guardamos el bloque y corremos DRC y su Fix
save_block EL_GRAN_JAGUAR.ndm:chip_IO
signoff_fix_drc

save_block EL_GRAN_JAGUAR.ndm:chip_IO
signoff_check_drc

check_lvs
save_block EL_GRAN_JAGUAR.ndm:chip_IO

#Creamos el verilog equivalente de la sintesis fisica
write_verilog -include all EL_GRAN_JAGUAR.v
```

CAPÍTULO 14

Siglas

CI Circuito Integrado. 5

CMOS Complementary Metal Oxide Semiconductor. 3

DRC Design Rule Check. 4

EDA Electronic Design Automation. 1

ERC Electrical Rule Check. 4

HDL Hardware Descriptive Lenguage. 14

IMEC Interuniversity Microelectronics Centre. 1

NDM New Data Model. 31

RAM Random Access Memory. 91

TSMC Taiwan Semiconductor Manufacturing Company. 1

VLSI Very Large Scale Integration. 11

CAPÍTULO 15

Glosario

Cell/celda: Conjunto de transistores interconectados que proveen una función lógica ya sea secuencial o combinacional.. 2

Corto circuito: Es una conexión entre dos terminales de un elemento de un circuito eléctrico, lo que provoca una anulación parcial o total de la resistencia en el circuito, lo que conlleva un aumento en la intensidad de corriente que lo atraviesa.. 54

Hardware: Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático.. 11

Layout: Diseños bidimensionales o planos de circuitos integrados tridimensionales utilizados en dispositivos con electrónica.. 15

Mesh: Material o diseño compuesto de cable o hilo entrelazado.. 51

Nanoescala: Escala en el orden de 10^{-9}.. 1

Net: nodo de conexión de uno o más puertos dentro de un netlist.. 43

Netlist: Es la descripción de un circuito electrónico en su forma más sencilla, esta compuesto de un listado de los componentes que contienen y los nodos y sus interconexiones.. 14

Pad: Celda en la cual se conectan los bonding wires de un circuito integrado, sirven como interconexión entre el empaquetado y el núcleo.. 32

Ruteo: Proceso por el cual se interconectan las celdas de un circuito según las nets a las que pertence.. 18

Silicio: Elemento químico de número atómico 14, masa atómica 28,086 y símbolo Si ; es un no metal sólido, de color amarillento, que se extrae del cuarzo y otros minerales y es el segundo elemento más abundante en la Tierra después del oxígeno.. 3

Software: Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.. 13