Diseño de un circuito integrado con tecnología de 180 nm, utilizando las librerías de diseño de TSMC y las librerías educativas de Synopsys: corrección de errores de densidad y polisilicio, verificación DRC y antena.

Noel Francisco Prado Búcaro

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de un circuito integrado con tecnología de 180 nm, utilizando las librerías de diseño de TSMC y las librerías educativas de Synopsys: corrección de errores de densidad y polisilicio, verificación DRC y antena.

Trabajo de graduación presentado por Noel Francisco Prado Búcaro
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

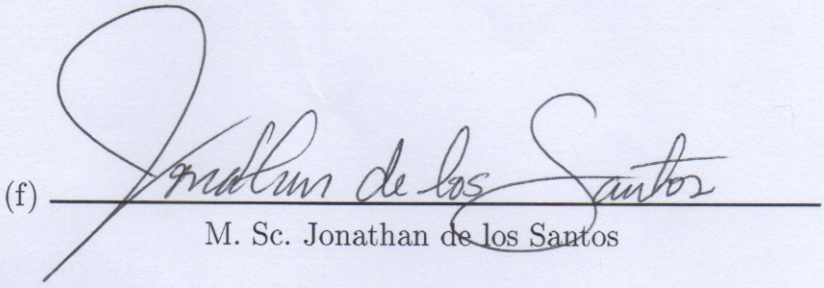
Diseño de un circuito integrado con tecnología de 180 nm, utilizando las librerías de diseño de TSMC y las librerías educativas de Synopsys: corrección de errores de densidad y polisilicio, verificación DRC y antena.

Trabajo de graduación presentado por Noel Francisco Prado Búcaro
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2024

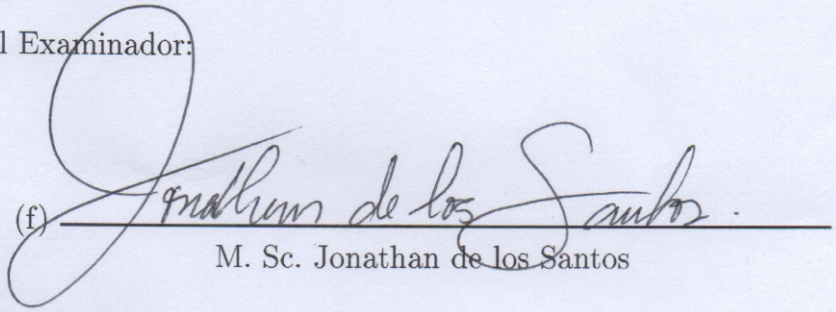
Vo.Bo.:



(f) _____

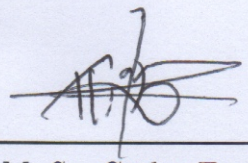
M. Sc. Jonathan de los Santos

Tribunal Examinador:



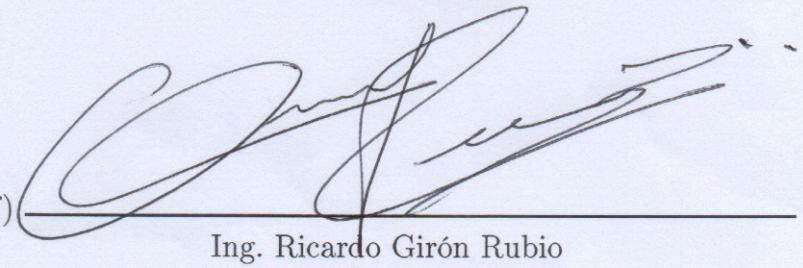
(f) _____

M. Sc. Jonathan de los Santos



(f) _____

M. Sc. Carlos Esquit



(f) _____

Ing. Ricardo Girón Rubio

Fecha de aprobación: Guatemala, 19 de Junio de 2024.

Prefacio

Agradezco profundamente a Dios por darme la fortaleza y la oportunidad de completar este trabajo. También, mi sincero agradecimiento a mi familia y amigos por su constante apoyo durante mi carrera universitaria.

Un especial reconocimiento al MSc. Carlos Esquit, cuya contribución ha sido fundamental en el campo de la nanoelectrónica en la Universidad del Valle de Guatemala. Asimismo, agradezco a mi asesor, el Ing. Jonathan de los Santos, por su orientación, paciencia y valiosos consejos durante el desarrollo de este proyecto.

Índice

Lista de figuras

Lista de cuadros

Resumen

Este trabajo busca realizar mejoras al flujo de diseño de circuitos integrados propuesto en años anteriores, el cual utiliza las librerías de diseño de TSMC. Se busca también replicar el flujo de diseño con mejoras implementadas, haciendo uso de las librerías de diseño educativas de *Synopsys*.

Una de las principales mejoras realizadas al flujo de diseño es la simplificación de la jerarquía de directorios que se utiliza para realizar tanto la síntesis lógica como la síntesis física. Esto permite facilitar la comprensión del flujo de diseño a futuros integrantes de la línea de investigación.

Dentro del flujo de diseño de TSMC, uno de los principales problemas, por el cual no se ha logrado concluir el diseño del circuito, es la densidad de metal en distintas capas del circuito; el problema consiste en que no se llega el requisito mínimo para fabricación. Sin embargo, este año, se ha conseguido un *runset* a través del cual se debería de solucionar este problema. Sin embargo a pesar de que este programa reduce el porcentaje de error, no lo soluciona totalmente.

La implementación del flujo de diseño en las librerías de *Synopsys* fue exitoso, en lo que se refiere a las etapas de síntesis lógica y síntesis física; sin embargo a través de la verificación DRC no se obtuvieron los resultados deseados; se determinó que esto era un problema del *runset* utilizado para realizar la verificación; dado que se realizaron múltiples pruebas con circuitos simples, y en todos los casos en los que existía más de una celda se obtenían múltiples errores.

Abstract

This study aims to improve the integrated circuit design flow proposed in previous years, which utilizes the TSMC design libraries. There is also an attempt to replicate the design flow with implemented enhancements, making use of the educational design libraries of *Synopsys*.

One of the primary improvements made to the design flow is the simplification of the directory hierarchy used for both logical and physical synthesis. This adjustment aids in clarifying the design flow for future members of the research team.

Within the TSMC design flow, one of the main challenges preventing the completion of the circuit design is the metal density in various layers of the circuit; the issue lies in not meeting the minimum manufacturing requirement. However, this year, a *runset* has been acquired which should address this problem. Even though this program reduces the error percentage, it does not entirely solve the issue.

The implementation of the design flow using the *Synopsys* libraries was successful regarding the stages of logical and physical synthesis. Nonetheless, the desired results were not obtained through DRC verification; it was determined that this was an issue with the *runset* used for verification. Multiple tests were conducted with simple circuits, and in all cases where more than one cell was present, numerous errors were identified.

CAPÍTULO 1

Introducción

En recientes décadas, la electrónica ha transformado nuestro mundo, gracias a la habilidad de desarrollar circuitos detallados en áreas muy pequeñas. Esta innovación se conoce como micro y nanoelectrónica y ha emergido como una de las principales industrias en los países más avanzados.

Con el tiempo, los circuitos se han integrado cada vez más en nuestra vida cotidiana. Reconociendo su importancia, la Universidad del Valle de Guatemala inauguró una línea de investigación dedicada a la creación de un circuito integrado utilizando tecnología de 180 nm.

Desde 2019, diversos investigadores han explorado el diseño de este circuito, centrándose en las librerías de diseño proporcionadas por TSMC. Sin embargo, han surgido problemas, como errores de densidad en ciertas capas del circuito. En este trabajo se continúa con la investigación, proponiendo el uso de diferentes librerías, como las educativas de *Synopsys*, y abordando los mencionados errores.

CAPÍTULO 2

Antecedentes

La Universidad del Valle de Guatemala ha estado trabajando en el campo de la nanoelectrónica desde el año 2013, cuando se impartió por primera vez el curso *Introducción al diseño de sistemas VLSI*. En el año 2014, se estableció una alianza con la empresa *Synopsys*, lo que permitió el acceso a herramientas necesarias para avanzar en el campo de la nanoelectrónica, esto dio lugar al primer trabajo de graduación relacionado con nanoelectrónica [1], el cual ha servido como referencia para trabajos realizados posteriormente.

En 2015 se agregaron al mapa curricular de la carrera ingeniería electrónica los cursos de nanoelectrónica 1 y 2. Este mismo año se logró un acuerdo con el *Inter-university Microelectronics Centre* (IMEC) para la manufactura de un chip, el cual ha sido objeto de varios trabajos de graduación, y lo será a su vez de este.

En 2019, se inicio el proyecto de realizar un *nanochip*, nombrado El Gran Juguar, desde ese año varios estudiantes han logrado avances significativos en la simulación y verificación funcional, así como en la síntesis lógica y física del *nanochip*. A pesar de estos avances, no se ha logrado concluir el diseño del *nanochip*, como consecuencia de errores de densidad de metal, problemas en el proceso de LVS y en la extracción de parásitos, los cuales aún no han sido resueltos.

A continuación se presentan los avances en el proyecto a lo largo de los años; los cuales sirven como antecedentes para este trabajo de graduación.

2.1 Primera iteración (año 2019)

Este año tuvo inicio el proyecto de diseño y fabricación del chip. Dado que era el primer año en el que se trabajaba en este proyecto, fue necesario investigar la funcionalidad de las distintas aplicaciones provistas por *Synopsys*. Este año se planteó un primer flujo de diseño para llevar a cabo tanto el diseño lógico como el físico de un circuito integrado (el chip), esto puede confirmarse en el trabajo de graduación de Steven Rubio [2].

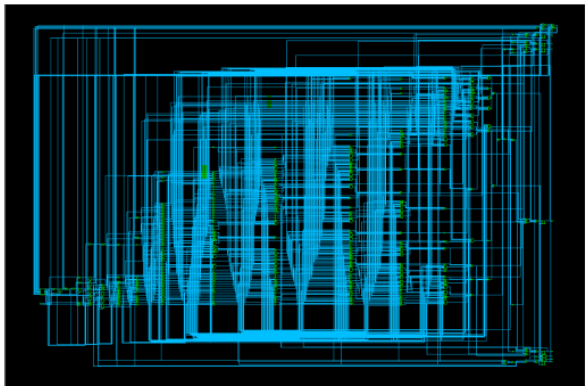


Figura 1: Síntesis lógica obtenida en 2019, a través de la herramienta *Design Vision*

Durante este primer año del proyecto también fue planteada la funcionalidad del chip que se diseñaría y fabricaría. La funcionalidad es mostrar un texto por medio de caracteres ASCII. Esto fue planteado en el trabajo de graduación de Luis Najera [3]; en este también se definen los pines que tendrá el circuito integrado, siendo estos en total 12: un pin para Vdd, otro para Vss, un pin para la señal de reloj; y 8 pines de salida para la generación del carácter ASCII; y uno de los pines queda sin funcionalidad, este existe solamente para mantener una simetría en el chip.

Dentro del trabajo de graduación de Luis Najera, también se generó el archivo HDL (hardware description language), para describir el circuito con la funcionalidad planteada previamente. Se utilizó la herramienta *Design Vision*, para sintetizar el HDL a nivel de celdas, y se comprobó que el comportamiento fuera equivalente al descrito por el HDL través de la herramienta *Formality*.

2.2 Segunda iteración (año 2020)

El principal enfoque del segundo año del proyecto consistió en dar continuidad al flujo de diseño planteado el año anterior. Durante este año se trabajaron partes específicas del flujo de diseño, dentro de las cuales se generaron *scripts* para automatizarlas. Las principales partes del flujo de trabajo que fueron trabajadas durante este año son: *VCS, DRC, LVS* y la extracción de parásitos para posteriormente analizar el efecto que tienen en *HSPICE*. Siguiendo el flujo de diseño planteado el año anterior, se realizó la síntesis física, para realizarla se utilizó por primera vez la herramienta *IC Compiler I*. Esta herramienta solamente fue utilizada durante este año, ya que el año siguiente fue sustituida por *IC Compiler II*.

La síntesis física obtenida durante este año a través de *IC Compiler I*, presentaba problemas al correr DRC, siendo el principal que no existía conexión hacia los pines Vdd y Vss.[4]

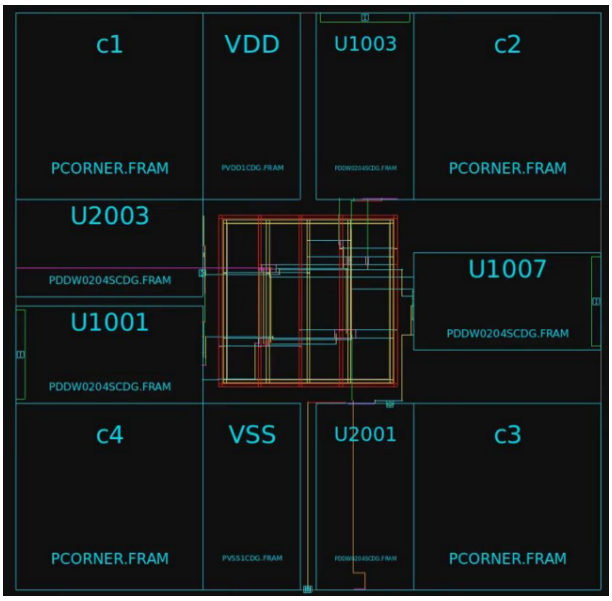


Figura 2: Síntesis física obtenida a través de *IC Compiler I*

2.3 Tercera iteración (año 2021)

Durante el tercer año en el que se trabajó el proyecto, se tomó la decisión de utilizar el programa *IC Complier II* para realizar la síntesis física, en contraste con el utilizado el año anterior *IC Compiler I*; esta decisión fue tomada debido a que la herramienta fue conocida por participantes en el proyecto el año anterior, y al ser una opción más moderna para realizar la síntesis física, se optó por explorarla.

Gran parte del avance realizado durante este año fue, la adquisición de conocimiento acerca de la herramienta *IC Compiler II*, y la adaptación de los flujos realizados en años anteriores a esta.

Este año se implementaron flujos en *IC Compiler II* para efectuar la síntesis física, las verificaciones de antena [5], la verificación de reglas de diseño[6] y la verificación de reglas eléctricas [7].

El *layout* obtenido para el circuito planteado en 2019, pasa las pruebas de verificación de antena, y las reglas eléctricas; sin embargo, en el caso de las reglas de diseño, se obtuvieron 6 errores de densidad, los cuales no pudieron ser resueltos. Estos errores consistían en que no se llega al requerimiento mínimo de densidad de metal en todas las capas en las que se está trabajando el *chip*. Se encontró una posible solución, la cual era generar metales *dummy* para llegar a este requisito. Sin embargo, para poder ejecutar esta acción se requería un *runset*, que no estaba disponible; este fue solicitado a IMEC, pero no se obtuvo respuesta durante este año. [7]

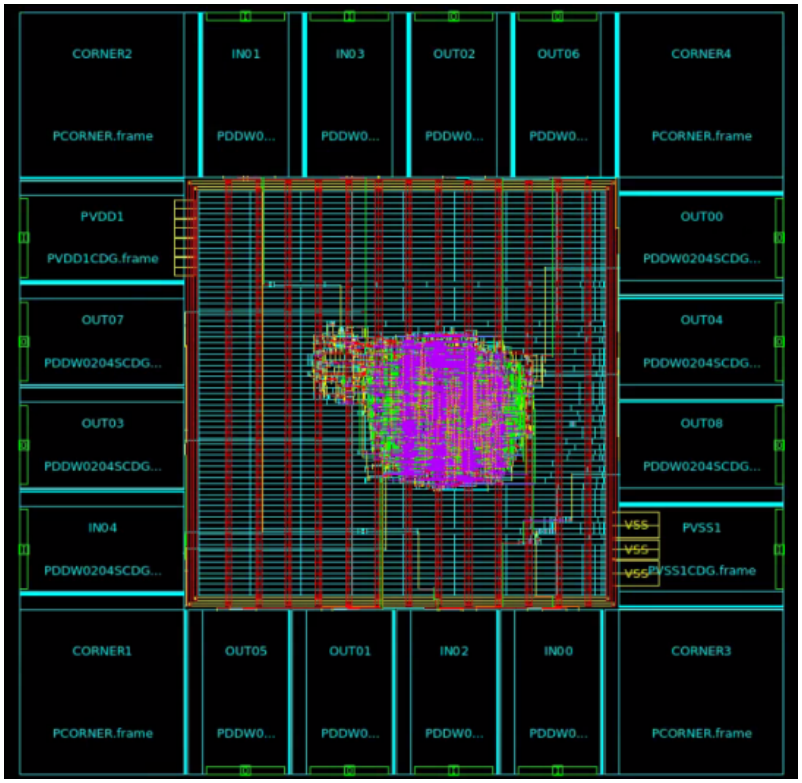


Figura 3: Síntesis física obtenida a través de *IC Compiler II*

2.4 Cuarta iteración (año 2022)

El cuarto año que se trabajó el proyecto, se automatizó aún más el flujo de diseño trabajado en años anteriores. Unos de los principales avances obtenidos como consecuencia de esto, es una interfaz gráfica, a través de la cual es posible accionar el flujo de diseño, con el cual se generan los archivos necesarios para realizar verificaciones físicas como: DRC, ERC, LVS, Antena y extracción de parásitos. Los errores de densidad obtenidos en 2021, fueron resueltos parcialmente, reduciéndolos solamente a 2. Para esto se válido un *runset* para relleno de metal. El *runset* que fue dado no era compatible, por lo que fue necesario realizar una traducción de *Calibre* a uno compatible. La traducción del *runset* tuvo varias fases, ya que se intentó realizar la traducción de forma manual, lo cual no dio resultados exitosos; luego se utilizó una herramienta proveída por *Synopsys*, y de la traducción obtenida a través de está herramienta, fue necesario realizar un proceso manual para solucionar los problemas obtenidos al ejecutarlo. [8].

A través del *runset* fue posible obtener múltiples síntesis físicas, en las cuales destacan dos. Una en la que se disminuyeron los errores de densidad a solamente dos, y otra en la que se eliminaron por completo los errores de densidad, pero en la cual existían errores de reglas, los cuales no se pudieron solucionar.

Durante este año también se realizó la automatización de la etapa de síntesis lógica para poder obtener archivos *Verilog* del circuito deseado, y la eliminación de cualquier archivo duplicado en el proceso. En este caso fue posible obtener la síntesis lógica de manera

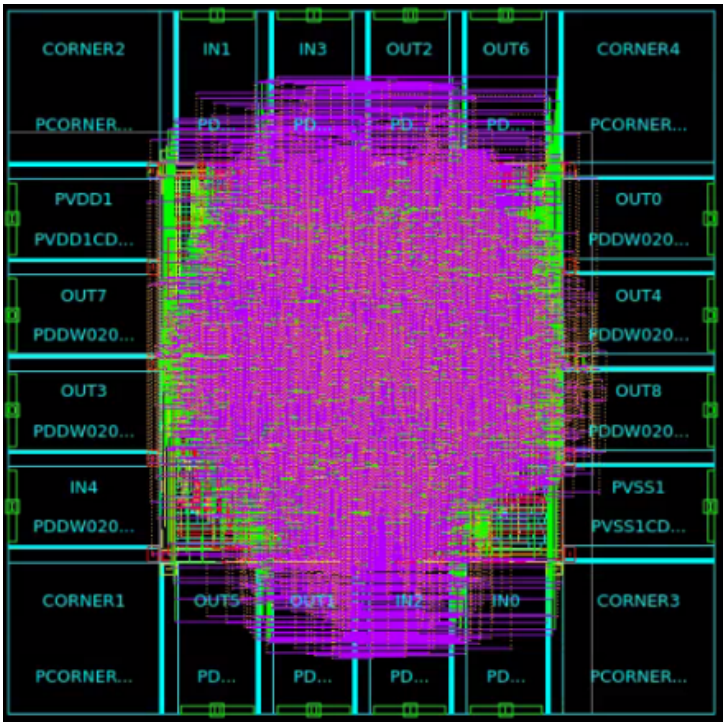


Figura 4: Síntesis física con 2 errores de densidad

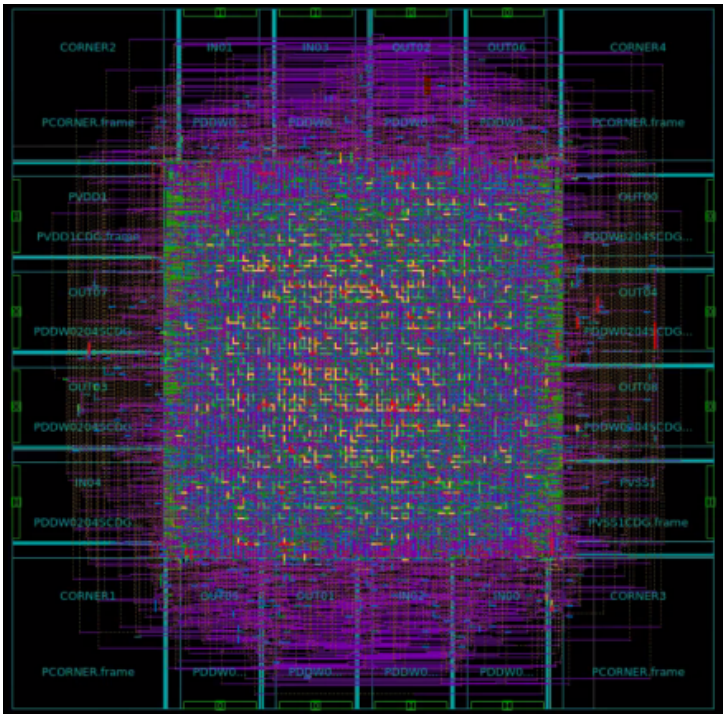


Figura 5: Síntesis física sin errores de densidad

automática del circuito planteado en 2019; esto permitió cargar la síntesis lógica a un *FPGA* [9]. También se realizó una aplicación para comprobar que la salida del circuito sea la

esperada, esta aplicación consistió que en que la salida de un *FPGA*, con el circuito cargado, fuera recibida a través de un bus de datos de 8 bits, y transmitido a través de comunicación serial (*UART*) a una computadora; en la cual se recibe la cadena de texto y se convierte en audio mediante una aplicación en *Python 3*. Esta aplicación permitió comprobar el correcto funcionamiento de la síntesis lógica, y fue un avance en lo que corresponde al método en el que se expondrá el chip, cuando sea fabricado [10]. En la figura 6 se puede observar el *hardware* utilizado para implementar esta aplicación.

De manera que en este punto se tiene automatizado el proceso de síntesis lógica, síntesis física y las verificaciones, por lo que se puede determinar que el flujo de diseño está totalmente automatizado.

Figura 6: Circuito implementado en FPGA para comprobar que las salidas sean las esperadas

CAPÍTULO 3

Justificación

Los circuitos integrados, también conocidos como *chips*, son una pieza fundamental de la tecnología moderna. Estos dispositivos electrónicos contienen miles o incluso millones de transistores en un solo paquete, lo que permite un mayor rendimiento y una menor cantidad de espacio requerido en comparación con los circuitos electrónicos antiguos. La historia de los circuitos integrados se remonta a la década de 1950, cuando los primeros transistores fueron desarrollados. Los científicos y técnicos de todo el mundo trabajaron en el desarrollo de tecnologías que permitieran a los transistores y otros componentes electrónicos ser fabricados en un solo chip. Los primeros circuitos integrados fueron creados alrededor de 1958 por Jack Kilby de *Texas Instruments* y Robert Noyce de *Fairchild Semiconductor*.

Desde entonces, los circuitos integrados se han vuelto cada vez más pequeños y más potentes, y se han utilizado en una amplia variedad de dispositivos electrónicos, desde teléfonos móviles y computadoras hasta automóviles y sistemas de control de vuelo en aviones. Los avances en la fabricación de chips también han permitido el desarrollo de tecnologías como la inteligencia artificial, el aprendizaje automático y la realidad virtual.

Actualmente, en la Universidad del Valle de Guatemala se cuenta con un flujo de diseño automatizado trabajado principalmente con las librerías de diseño de TSMC, con el cual se han obtenido avances significativos en el diseño del *nanochip* El Gran Jaguar. Un problema que existe como consecuencia de utilizar las librerías de TSMC, es que no se puede acceder a toda la información que se desearía para tener una mayor comprensión sobre el modo en el que el circuito se está diseñando, no es posible acceder a esta información debido a que esta es propiedad intelectual de TSMC. Como consecuencia de esto se busca replicar los avances obtenidos en librerías educativas proveídas por *Synopsys*, dentro de las cuales se tendría un acceso completo a toda la información acerca del circuito que se está diseñando.

Se busca realizar esta réplica también, para poder comprender y solucionar errores de densidad y polisilicio que existen en el diseño del circuito actualmente. La réplica puede implicar también utilizar otro flujo de diseño, uno sugerido por *Synopsys*, el cual fue explorado el año anterior [11] [12], dentro de esta misma línea de investigación, y el cual también podría formar parte de la solución de los errores de densidad y polisilicio.

Además de realizar la réplica de los avances obtenidos en el proyecto El Gran Jaguar hacia las librerías de diseño de *Synopsys*, se busca también mejorar etapas del flujo de diseño del mismo proyecto, en las librería de diseño de TSMC. Esto con el fin de solucionar todos los errores en el diseño, y cumplir los requisitos para la fabricación.

En el contexto de Guatemala, la realización del diseño y fabricación de un *nanochip* en la Universidad del Valle de Guatemala puede ser un paso importante hacia el desarrollo de una industria tecnológica más fuerte y competitiva en el país. Además, la investigación y el desarrollo de *nanochips* también pueden tener aplicaciones importantes en áreas como la medicina, la energía y la comunicación, lo que podría tener un impacto significativo en la sociedad guatemalteca y el mundo en general.

CAPÍTULO 4

Objetivos

4.1 Objetivo general

Replicar el avance del proyecto del *nanochip* El Gran Jaguar, logrado con las librerías de diseño de 180 nm de TSMC, hacia las librerías educativas proveídas por *Synopsys*, con el fin de encontrar una solución definitiva a los errores de densidad y polisilicio, detectados luego de la realización de las pruebas DRC y antena.

4.2 Objetivos específicos

- Realizar la síntesis lógica del *nanochip* El Gran Jaguar en librerías de *Synopsys*, y buscar mejoras en la síntesis lógica del proyecto El Gran Jaguar, que utiliza las librerías de diseño de TSMC.
- Realizar la síntesis física del *nanochip* El Gran Jaguar en librerías de *Synopsys*, y buscar mejoras en la síntesis física del proyecto El Gran Jaguar, que utiliza las librerías de diseño de TSMC.
- Realizar la verificación física DRC del *nanochip* El Gran Jaguar en librerías de *Synopsys*, y buscar mejoras de DRC del proyecto El Gran Jaguar, que utiliza las librerías de diseño de TSMC.
- Realizar la verificación de antena del *nanochip* El Gran Jaguar en librerías de *Synopsys*, y buscar mejoras en prueba de antena del proyecto El Gran Jaguar, que utiliza las librerías de diseño de TSMC.
- Realizar la corrección de errores, que surjan luego de cada iteración, en conjunto con el equipo de trabajo.
- Dejar todos los avances debidamente documentados, para que puedan servir de referencia en trabajos futuros.

CAPÍTULO 5

Alcance

El alcance de este trabajo consiste en implementar mejoras al flujo de diseño con el que se cuenta actualmente, y adaptarlo a las librerías educativas de *Synopsys*. Las mejoras implementadas al flujo de diseño buscan facilitar la comprensión de este, así como obtener mejores resultados en el diseño de distintos circuitos.

La adaptación del flujo a las librerías de diseño de *Synopsys* permite observar que aspectos se pueden generalizar dentro del flujo, para hacerlo más adaptable a distintas tecnologías. La adaptación permite también tener un conocimiento más profundo acerca de lo que sucede en el circuito, dado que es posible observar que hay dentro de cada celda.

Se busca implementar mejoras en el flujo de diseño actual, el cual utiliza las librerías de TSMC, a través de las cuales se puedan resolver los errores de densidad, motivo por el cual la fabricación del circuito no se ha podido efectuar.

CAPÍTULO 6

Marco teórico

6.1 Nanoelectrónica

El diseño de circuitos a escala nanométrica introduce desafíos únicos que requieren un enfoque especializado y sofisticado para el flujo de diseño. En esencia, la nanoeléctronica no solo se ocupa de la miniaturización de los dispositivos, sino también de cómo se organizan y se conectan entre sí para formar circuitos y sistemas completos.

6.2 Flujo de diseño

Existen múltiples flujos de diseño dentro de la nanoeléctronica, sin embargo, la mayoría siguen la siguiente estructura: se comienza con la definición del concepto del sistema y los requisitos del circuito. Esto abarca la funcionalidad deseada, las restricciones de rendimiento, eficiencia energética, y otros factores como el costo y la fiabilidad.

A continuación, se realiza la etapa de diseño lógico. Aquí, el comportamiento del sistema se describe en términos de operaciones lógicas y se realizan las primeras simulaciones a nivel de sistema. Se utilizan lenguajes de descripción de hardware, como VHDL o *Verilog*, para este propósito [13]. Durante esta etapa se debe de generar también un archivo HDL con mayor detalle sobre las conexiones, a este proceso se le conoce como síntesis lógica.

El siguiente paso es el diseño físico. Se determina cómo se organizarán los dispositivos en el chip y se establecen las conexiones entre ellos. En este paso, se consideran factores como la densidad del dispositivo, la disipación de calor, y la resistencia y capacitancias parásitas. Este paso puede ser particularmente desafiante en la nanoeléctronica debido a los fenómenos cuánticos y de interferencia que se vuelven significativos a esta escala. Este paso se conoce también como síntesis física.

Posteriormente, el diseño se verifica mediante una serie de pruebas y simulaciones para asegurarse de que cumple con los requisitos originales. También se realizan análisis de

tolerancia para comprender cómo las variaciones en los parámetros del dispositivo pueden afectar al rendimiento del circuito.

Una vez que el diseño ha sido verificado, se procede a la fabricación, que a nivel nanométrico puede implicar técnicas avanzadas como la litografía ultravioleta extrema (EUV) o el auto ensamblaje dirigido.

Finalmente, después de la fabricación, se realizan pruebas y validaciones en el dispositivo real para asegurarse de que funciona como se esperaba.

Cabe destacar que el flujo de diseño en nanoeléctronica debe estar integrado con la investigación y desarrollo continuos de nuevos materiales y procesos, dado que los límites de la tecnología actual se están alcanzando rápidamente y la innovación es crucial para seguir avanzando. Por lo tanto, el diseño de circuitos a escala nanométrica es un proceso interdisciplinario que combina la física, la química, la ingeniería eléctrica, la ciencia de materiales y la informática [13].

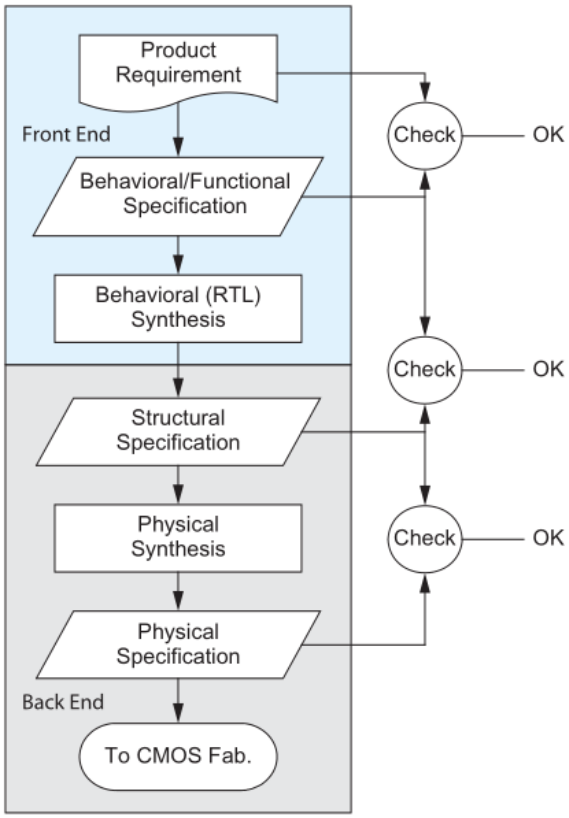


Figura 7: Flujo de diseño tradicional (obtenido de [13])

A continuación se habla con mayor detalle sobre las etapas de mayor interés, para este trabajo de graduación, dentro del flujo de diseño.

6.3 Síntesis lógica

La síntesis lógica es el proceso de convertir una descripción de alto nivel del circuito en un diseño implementable utilizando compuertas lógicas básicas. Esta descripción de alto nivel se suele realizar en lenguajes de descripción de hardware (HDL) como *Verilog*. Durante la síntesis lógica, se optimiza el circuito, se lleva a cabo el mapeo a nivel de compuertas y se realiza la asignación de registros y componentes de sincronización. Además, se realiza la verificación funcional y de restricciones de temporización para asegurar que el circuito cumple con los requisitos de funcionamiento y tiempo especificados.

6.4 Síntesis física

La síntesis física es una etapa fundamental y extensa dentro del flujo de diseño en nanoelectrónica. Esta parte del proceso se encarga de convertir cada subcircuito del diseño lógico en su correspondiente representación física, donde se incluyen elementos como difusiones, pines y el uso de silicio. A partir de un *netlist*, se genera un *layout* en formato GDS que representa el diseño físico completo del circuito. Este diseño está listo para ser manufacturado, pero es esencial tener en cuenta que se cumplan las características y requisitos específicos del fabricante. Es necesario asegurarse de que el diseño físico se ajuste a las limitaciones tecnológicas y las capacidades del proceso de fabricación específico que se utilizará. Para realizar la síntesis física se utiliza la herramienta *IC Compiler II*.

6.5 *Design Rule Check* (DRC)

Design Rule Check (DRC) es una etapa del flujo de diseño, en la que se verifica si el diseño físico del circuito cumple con las reglas de diseño específicas y las limitaciones impuestas por el proceso de fabricación. Cabe destacar que estas reglas y especificaciones varían según el *foundry* (fábrica de semiconductores, por ejemplo TSMC) en el que se llevará a cabo la fabricación. Cada *foundry* tiene sus propias capacidades tecnológicas, restricciones de diseño y tolerancias. Por lo tanto, el DRC depende del *runset* definido por el *foundry* específico en el que se fabricará el circuito. Las reglas de diseño establecidas por el *foundry* correspondiente se utilizan para verificar y asegurar que el diseño físico cumpla con los requisitos y capacidades del proceso de fabricación. Al realizar el DRC, se verifican y corrigen las posibles violaciones de las reglas de diseño establecidas, lo que garantiza que el diseño sea compatible y adecuado para la fabricación en ese entorno particular. Esta dependencia resalta la importancia de colaborar estrechamente con el *foundry* y seguir sus directrices para lograr un diseño físico exitoso y cumplir con los estándares de calidad de fabricación.

6.6 Verificación de antena

La verificación de antena en el diseño de circuitos es crucial para evitar problemas de radiación electromagnética no deseada. Estos problemas pueden surgir debido a la presencia de estructuras metálicas y las interconexiones en el diseño físico. El efecto de antena ocurre cuando un alambre de metal conectado a una compuerta de transistor se graba mediante

plasma y se carga a un voltaje suficiente para dañar los óxidos de compuerta. Esto puede resultar en fugas de corriente, cambios en el umbral de voltaje y reducción de la vida útil del transistor. Se establecen reglas de antena para limitar el área de metal conectada a una compuerta sin una fuente o drenaje, y así evitar daños. Las correcciones pueden incluir interrupciones en las líneas de metal o la adición de diodos de antena para proporcionar un camino de descarga durante el grabado. La verificación de antena es esencial para garantizar el funcionamiento adecuado y libre de interferencias del circuito [13].

6.7 Synopsys

Las herramientas utilizadas para llevar a cabo el flujo de diseño son proveídas por la empresa *Synopsys, Inc.* en esta sección se da una breve explicación acerca de las herramientas proveídas por esta empresa.

Synopsys, Inc., reconocida mundialmente en la industria de semiconductores y software, juega un papel crucial al proporcionar tecnología de diseño y verificación de semiconductores y soluciones de seguridad de software [14]. Se destacan sus avanzadas herramientas de diseño y verificación de chips de semiconductores, entre las que se incluyen *Fusion Compiler*, *Design Compiler* y *IC Compiler II*, cada una con características únicas que ayudan a los ingenieros a desarrollar diseños más eficientes y precisos.

Además, *Synopsys* ofrece un catálogo de bloques de diseño pre-diseñados o IPs, que abarcan desde interfaces de memoria y interfaces de bus hasta procesadores y bloques analógicos y de señal mixta. Esta gama de IPs facilita enormemente el proceso de diseño de semiconductores, permitiendo a los ingenieros incorporar estos componentes pre-diseñados y probados en sus propios diseños [15].

Más allá del diseño de hardware, *Synopsys* se adentra en la seguridad y calidad del software a través de un conjunto de herramientas de vanguardia. Estas incluyen *Coverity* para pruebas de calidad de software estático, *Black Duck* para la gestión de seguridad de software de código abierto y *Seeker* para la seguridad de aplicaciones interactivas. Estas herramientas ayudan a las organizaciones a mantener la integridad de su software y proteger sus sistemas de posibles vulnerabilidades [16].

Finalmente, la empresa también provee herramientas de verificación de hardware, como la plataforma *ZeBu Server*. Estas herramientas son fundamentales en la etapa final de diseño, permitiendo a los ingenieros validar sus diseños de hardware antes de pasar a la producción en masa, garantizando así la funcionalidad y la fiabilidad de los productos finales [17]. *Synopsys* se posiciona como un proveedor integral de soluciones en el panorama de la electrónica y el software, abordando los desafíos de diseño, verificación, seguridad y calidad.

A continuación se explican brevemente las principales herramientas utilizadas dentro de este trabajo de graduación:

6.8 VCS

VCS es una herramienta de simulación y de verificación. Es recomendada como el primer paso para ejecutar simulaciones del diseño del circuito en VHDL y verificar su funcionamiento, corrigiendo errores en una etapa temprana del flujo de diseño. Se utiliza como un simulador de alto rendimiento y alta capacidad de Verilog, proporcionando una solución eficiente y precisa [18]. Las simulaciones de diseño VHDL requieren un archivo de testbench en VHDL que incluye rutinas de entrada y verificación de salidas para ejecutar pruebas específicas. La simulación lógica se enfoca únicamente en comprobar el funcionamiento lógico del circuito, dejando de lado los aspectos relacionados con el hardware.

A continuación se presentan algunos comandos de esta herramienta utilizados previamente dentro de esta línea de investigación; se incluyen aquellos cuyo uso es probable que vuelva a darse en el presente trabajo:

Origen	**Comando**	**Descripción**
Consola Linux	vcs circuito_testbench.v circuito.v -gui -debug_acc+all -full64	Compila una simulación del circuito descrito en "circuito.v", utilizando como entradas y salidas las descritas en "circuito_testbench.v".

Cuadro 1: Comandos de VCS y su descripción

6.9 *Design Compiler*

Design Compiler es una herramienta esencial dentro del diseño de circuitos integrados. Su principal función es la síntesis lógica, que es el proceso de transformar una descripción de alto nivel de un diseño de circuito (generalmente escrita en lenguajes de descripción de hardware como *Verilog* o VHDL) en una representación de nivel de compuerta. En otras palabras, *Design Compiler* interpreta el diseño abstracto y lo convierte en una red de compuertas lógicas que puede implementarse físicamente en un chip. El resultado de este proceso de síntesis es un *netlist*, que es una descripción detallada de las conexiones de la red de compuertas[19]. *Design Compiler* es vital en el flujo de diseño de circuitos integrados, ya que permite el paso de la representación de alto nivel del diseño a una representación de bajo nivel que se puede fabricar realmente.

En el Cuadro 2 se presetan algunos comandos de *Design Compiler* utilizados previamente dentro de esta línea de investigación; se incluyen aquellos cuyo uso es probable que vuelva a darse en el presente trabajo:

6.10 *IC Compiler II*

IC Compiler II de *Synopsys* es una herramienta de diseño y compilación de circuitos integrados. Esta herramienta permite realizar una serie de tareas esenciales, incluyendo la síntesis física, la optimización del rendimiento, la reducción de la potencia consumida y el cumplimiento de las restricciones de temporización [20].

Origen	Comando	Descripción
Consola Linux	dc_shell	Inicia terminal de *Design Compiler*.
dc_shell	start_gui	Se abre interfaz gráfica.
dc_shell	set target_library {../ref/db_nldm/saed14rvt_tt0p8v25c.db}	Primer paso para indicar las librerías a utilizar dentro del archivo ".synopsys_dc.setup".
dc_shell	set link_library {* ../ref/db_nldm/saed14rvt_tt0p8v25c.db}	Segundo paso para indicar las librerías a utilizar dentro del archivo ".synopsys_dc.setup".
dc_shell	read_verilog ../source/archivo.v	Lee el diseño del circuito descrito en un archivo "archivo.v".
dc_shell	link	Se enlaza la lectura del archivo con la herramienta.
dc_shell	check_design	Se realiza la revisión del circuito.
dc_shell	read_sdc ../source/archivo.sdc	Se realiza la lectura del archivo que contiene las restricciones de diseño, siendo este "archivo.sdc".
dc_shell	compile -exact_map	Ejecuta la compilación del archivo *verilog* a una representación a nivel de compuerta.

Cuadro 2: Comandos de *Design Compiler* y su descripción

IC Compiler II proporciona un conjunto completo de capacidades para realizar el diseño físico de un circuito integrado. La herramienta utiliza algoritmos y técnicas avanzadas para lograr una implementación eficiente y confiable del diseño físico.

Además, *IC Compiler II* ofrece una amplia gama de funciones para abordar desafíos específicos en el diseño de circuitos integrados. Esto incluye la gestión de la variabilidad del proceso, la integración de bloques de diseño previos, el soporte para tecnologías de vanguardia, como tecnologías de fabricación *FinFET*, y la capacidad de realizar análisis y optimización para el cumplimiento de estándares y regulaciones específicas[20].

En el Cuadro 3 se presentan algunos de los comandos más frecuentemente usados durante la etapa de síntesis física, dado que existe muchísimas opciones de configuración para ejecutar la síntesis física, solamente se incluyen los utilizados con mayor frecuencia:

6.11 *IC Validator*

IC Validator es una herramienta indispensable en el proceso de diseño de circuitos integrados, cuyo principal rol es la verificación de diseño físico. Esta herramienta verifica que el diseño del circuito cumpla con las reglas de diseño específicas para el proceso de fabricación que se utilizará. El conjunto de reglas de diseño (DRC) incluye dimensiones mínimas para las características del circuito, distancias mínimas entre las características, y otras reglas que son críticas para el éxito de la fabricación del circuito. Además de la verificación DRC, *IC Validator* también realiza verificación de *Layout vs Schematic* (LVS), que compara el diseño físico (*layout*) del circuito con el diseño lógico (*schematic*) para asegurar que son equivalentes. También puede realizar la verificación de la Antena, que previene el daño del

Comando	Descripción
create_lib “nombre_de_libreria.ndm” -technology “tech_file.tf”-ref_libs “archivo.ndm”	Creación de la librería a utilizar.
read_verilog ../source/archivo_sintetizado.v	Se abre el archivo *verilog* sintetizado “archivo_sintetizado.v”.
read_sdc ../source/archivo_sintetizado.sdc	Se abre el archivo *verilog* que contiene las restricciones de diseño sintetizado, “archivo_sintetizado.sdc”.
read_parasitic_tech -tlup ../source/t018lo_1p6m_typical.tluplus 15 -layermap ../source/star.map_6M	Se importa TLU+ y el map.
create_cell -lib_cell “libcellname” “instancename”	Se crea una celda, especificando el nombre de la librería que se instanciará “libcellname”, y el nombre que se le asignará a la celda “instancename”.
create_net “netname”	Se crea una nueva *net* (conexión), se debe especificar el nombre que se le asignará, “netname”
initialize_floorplan	Se inicializa el *floorplan*, se pueden definir varios parámetros a través de este comando, tales como área del *chip*, entre otros.
create_io_ring -name “nombre” -corner_-height “tamaño”	Se utiliza para crear el anillo de entradas y salidas.

Cuadro 3: Comandos de *IC Compiler II* y su descripción

dispositivo durante el proceso de fabricación. Al proporcionar estas capacidades de verificación, *IC Validator* ayuda a garantizar que el diseño del circuito integrado esté listo para la fabricación y que tenga una muy alta probabilidad de funcionar correctamente una vez fabricado [21].

En el Cuadro 4 se presenta el comando *icv*, se explican algunos de los parámetros del comando y se da una descripción de la funcionalidad de cada uno. Este comando es utilizado realizar pruebas a través de un *runset*.

En el Cuadro 5 se presentan otros comandos de la herramienta *IC Validator*, sin embargo estos son ejecutados desde *IC Compiler II*.

6.12 *PrimeTime*

PrimeTime es una herramienta crítica en el proceso de diseño de circuitos integrados, que se utiliza principalmente para la verificación de temporización estática (STA). Este proceso es esencial para garantizar que un diseño de circuito cumplirá con sus especificaciones de temporización. Específicamente, *PrimeTime* analiza todas las rutas posibles que una señal puede tomar a través de un diseño de circuito para determinar si las señales llegarán a sus destinos dentro de los límites de tiempo especificados [22]. De esta manera, se asegura

Comando	Parámetro	Descripción
`icv`	`-i chip.gds`	Especifica el archivo de entrada para la verificación. `chip.gds` es el nombre del archivo de diseño en formato gds que se verificaría.
`icv`	`-c Chip_IO`	Define el nombre del archivo de configuración de la celda que se utilizará durante la verificación. `Chip_IO` es el nombre de este archivo.
`icv`	`Metal_Calibre_0.18umV2.rs`	Es el nombre del *runset* de reglas de verificación que se utilizará. Este archivo contiene las reglas de diseño específicas para el proceso de fabricación que se están utilizando.

Cuadro 4: Comando *icv* de *IC Validator*, parámetros y descripción.

Comando	Descripción
`signoff_check_drc`	Este comando se utiliza para realizar una verificación DRC. Comprueba que el diseño cumple con las reglas establecidas por el proceso de fabricación.
`signoff_fix_drc`	Este comando intenta corregir automáticamente los problemas de DRC identificados durante la verificación. La herramienta hace ajustes en el diseño para intentar cumplir con las reglas de diseño.
`signoff_create_metal_fill -mode replace -select_layers {M1 M3}`	Este comando se utiliza para añadir relleno de metal en las capas especificadas (en este caso, M1 y M3) en el diseño. El argumento '-mode replace' indica que se deben reemplazar los rellenos de metal existentes.

Cuadro 5: Comandos de *IC Validator* desde *IC Compiler II*

de que todos los datos se propaguen a través del circuito dentro de los límites de tiempo del ciclo del reloj, evitando así cualquier falla de funcionamiento causada por condiciones de carrera. En resumen, *PrimeTime* es una herramienta esencial que garantiza la fiabilidad y la eficiencia de un diseño de circuito, al asegurarse de que se cumplan todas las especificaciones de temporización [23].

6.13 *Formality*

Formality es una herramienta vital en el proceso de diseño de circuitos integrados, desempeñando una función central en la verificación formal de equivalencia. Este proceso es esencial para confirmar que dos descripciones de un circuito, comúnmente en diferentes niveles de abstracción, sean funcionalmente equivalentes. Por ejemplo, después de que el proceso de síntesis lógica y síntesis física se haya completado, generando un *netlist* de nivel de compuerta y colocando y enrutando físicamente las compuertas en el diseño, *Formality* puede comparar este *netlist* con la descripción RTL original. Así se garantiza que no se hayan introducido errores durante la síntesis y que el diseño físico es una representación fiel del diseño lógico. Si *Formality* detecta discrepancias entre las descripciones, proporciona información sobre donde ocurren, permitiendo a los ingenieros rectificar problemas antes de avanzar en el diseño [24]. *Formality* desempeña un papel esencial en el mantenimiento de la integridad del diseño en el flujo de diseño de circuitos integrados, abarcando desde la síntesis lógica hasta la síntesis física.

6.14 *TetraMAX*

TetraMAX es una herramienta primordial en el proceso de diseño de circuitos integrados, particularmente en la etapa de verificación y prueba. Su función principal es la generación automática de patrones de prueba (ATPG), una etapa crucial para garantizar la calidad y la fiabilidad de los circuitos integrados [25].

El proceso de ATPG se basa en la creación de patrones de prueba específicos que, cuando se aplican a un circuito integrado, pueden detectar y localizar potenciales defectos físicos. Estos pueden ser desde cortocircuitos y circuitos abiertos hasta otros tipos de fallas estructurales que podrían perjudicar el funcionamiento correcto del circuito.

TetraMAX genera estos patrones de prueba en base a un modelo de fallas, que es una representación de posibles defectos en el circuito. Después de que se generan estos patrones, se utilizan en la fase de prueba de fabricación, donde se aplican al circuito y se monitorean las salidas. Si la salida observada no coincide con la salida esperada para un patrón de prueba dado, entonces se sabe que hay una falla en el circuito [26].

Además, *TetraMAX* también puede realizar una técnica llamada "diagnóstico de fallas", que no solo identifica la presencia de un defecto, sino que también ayuda a localizar el defecto dentro del circuito. Esto es especialmente útil para la depuración y la mejora continua del proceso de fabricación.

Por lo tanto, *TetraMAX* juega un papel vital en el aseguramiento de la calidad durante la fabricación de circuitos integrados. Al proporcionar una metodología efectiva para probar y diagnosticar fallas en los circuitos, ayuda a garantizar que los productos de circuitos integrados sean confiables y de alta calidad, permitiendo que los defectos se identifiquen y corrijan antes de que los dispositivos lleguen a los consumidores finales.

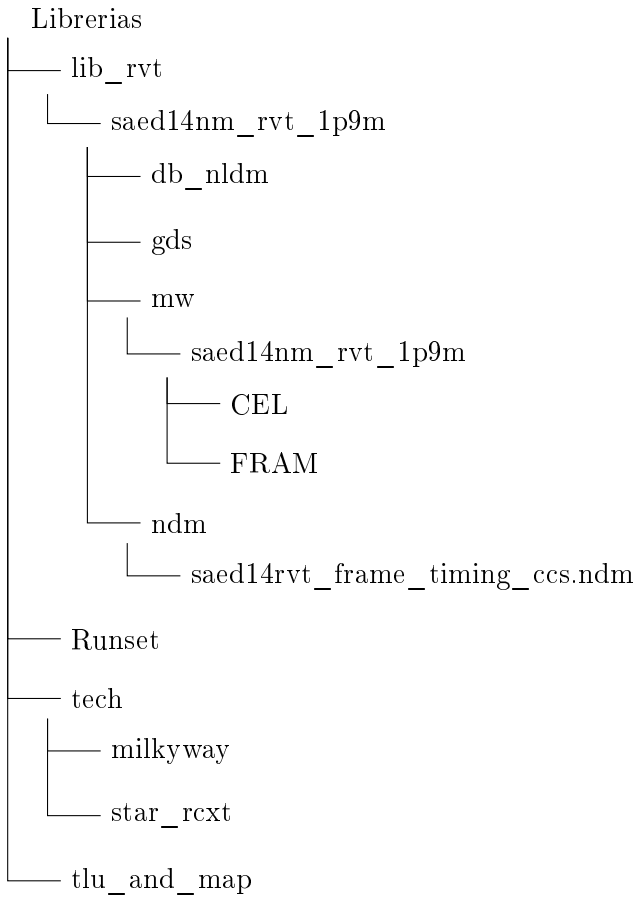
CAPÍTULO 7

Nueva jerarquía de directorios

En el flujo trabajado durante años anteriores, se obtenía de manera exitosa tanto la síntesis lógica como la síntesis física con librerías de diseño de TSMC. Sin embargo, al intentar utilizar este flujo con las librerías de diseño de *Synopsys*, se encontraron varias dificultades; estas tenían su principal raíz en el modo en el que se hacía referencia a los archivos requeridos para llevar a cabo la síntesis.

Para solucionar este problema, se propone una nueva jerarquía de directorios, en la cual todos los archivos requeridos para llevar a cabo ambas síntesis se encuentren bajo un solo directorio llamado "Librerias". Este directorio se encuentra afuera del directorio en el cual se trabaja el proyecto, de manera que si se utilizan las mismas librerías en distintos proyectos, no es necesario duplicarlo, sino que se puede referenciar múltiples veces; esto es importante debido a que las librerías de diseño pueden ocupar una cantidad de memoria en el orden de los *gigabytes*. Por lo que no es deseable copiarlas para cada proyecto.

A continuación se presenta un árbol de la jerarquía de directorios para las librerías de *Synopsys*, una estructura muy similar se utiliza para el caso de TSMC, la única diferencia se encuentra en los nombres de las librerías. En este árbol se muestran únicamente los directorios, no los archivos ya que existen directorios que contienen una gran cantidad de archivos.



En este árbol, únicamente se contiene la librería de diseño de *Synopsys* saed14rvt; sin embargo, pueden incluirse con el mismo orden las librerías saed14lvt, saed14hvt, y todas las otras librerías proveídas.

Dentro de la nueva jerarquía de directorios se tienen un directorio con el nombre del proyecto; dentro del cual existe un directorio para la síntesis física, y otro para la síntesis lógica. En trabajos anteriores tales como [8], se contaba con una jerarquía similar; sin embargo, esta resultaba confusa, ya que las referencias se hacían hacia carpetas con nombres de las personas que trabajaron distintas partes del flujo en años anteriores; y se tenían archivos duplicados dentro de estas.

La estructura con la que se contaba dificultaba significativamente entender a que se hacia referencia en los comandos; esto debido a que se habían trabajado partes de manera aislada, y la integración se realizó con fines únicamente funcionales, que no facilitaban la comprensión para personas que se unían al proyecto por primera vez.

Dado que se modificó la jerarquía de directorios de los archivos requeridos para llevar a cabo el flujo, también se modificaron las jerarquías de directorios empleados para realizar la síntesis lógica y la síntesis física; en las siguientes secciones se muestran estas.

7.1. Jerarquía de directorios de la síntesis lógica

La jerarquía de directorios de la síntesis lógica es en términos de funcionalidad igual a la realizada el año anterior, solamente se quitaron directorios que dificultaban la comprensión del flujo, así como la referencia a archivos dentro de los *scripts* utilizados; se redujo también la cantidad de *scripts* requerida para ejecutar el flujo.

A continuación se muestra la jerarquía de directorios realizada para la síntesis lógica.

```
sintesis_logica
├── DV_chip.script
├── ejecutar.bash
├── logs
├── Outputs
└── verilog
```

La nueva jerarquía cuenta con dos archivos y tres directorios. Los archivos son *scripts*, "DV_chip.script" contiene instrucciones para la herramienta *Design Vision*, y "ejecutar.bash" es el archivo que se debe correr para ejecutar el flujo completo.

Se cuenta con un directorio llamado "logs", para almacenar los *logs* generados por la herramienta, y no saturar el directorio principal; se cuenta con el directorio "Outputs", en el cual se almacenan los resultados de la síntesis lógica; y con un directorio llamado "verilog" en el cual se deben de colocar los archivos *Verilog* que se sintetizarán.

7.2. Jerarquía de directorios de la síntesis física

Dentro de la nueva jerarquía de directorios para la síntesis física se cuenta con directorios muy similares a los utilizados en el flujo con las librerías de TSMC. La principal diferencia es que se redujo la cantidad de directorios presentes; esto gracias a la unificación de todos los archivos referentes a las librerías.

A continuación se muestra la jerarquía de directorios:

```
sintesis_fisica
├── EL_GRAN_JAGUAR
├── Inputs
├── Outputs
│   ├── DRC
│   ├── Fill
│   └── IO
├── sintesis_fisica_top.tcl
└── scripts
```

En el directorio llamado "Inputs", se encuentran los archivos obtenidos a través de la síntesis lógica; en el directorio llamado "Outputs" se encuentran tres directorios más, el llamado "DRC" que contiene información referente a esta verificación, otro llamado "Fill", que contiene información referente al proceso de inserción de metal; y otro llamado "IO", que contiene las salidas de la síntesis física. En el directorio llamado "scripts" se encuentran *scripts* utilizados por el flujo. Se cuenta también con el archivo "sintesis_fisica_top.tcl", el cual se debe ejecutar para llevar a cabo todo el flujo de síntesis física.

Dentro del directorio de síntesis física se almacena el proyecto, en este caso dentro del árbol se puede observar con el nombre "EL_GRAN_JAGUAR".

CAPÍTULO 8

Síntesis lógica

8.1. Separación de la creación del archivo *Verilog* y la Síntesis Lógica

Una mejora al proceso de síntesis lógica, que aplica de manera generalizada a cualquier librería de diseño, es la separación de la generación del archivo *Verilog*, que describe el circuito a sintetizar, del flujo principal de síntesis lógica. En trabajos anteriores [9], el archivo *Verilog* se generaba dentro del mismo *script* de síntesis lógica. Esta práctica aunque es funcional, no es la más adecuada, considerando que en ocasiones se desean sintetizar circuitos distintos. Por lo tanto, es más razonable mantener el proceso de creación del archivo *Verilog* independiente de la síntesis lógica. En muchos escenarios, la creación del archivo *Verilog* no será automática, sino que será responsabilidad de un individuo en particular. En este trabajo, se emplea un *script* de *python* para generar el archivo *Verilog*, pero esto se debe exclusivamente a las características del circuito en cuestión.

Dicha separación otorga claridad al realizar pruebas de síntesis lógica. Al no regenerar el archivo *Verilog* en cada ejecución, se reduce la complejidad y el número de variables a considerar, facilitando un control más preciso sobre el proceso de síntesis. A su vez facilita sintetizar circuitos distintos, ya que no se llaman dentro del proceso de síntesis *scripts* que tengan dependencia directa con el circuito en cuestión.

La separación permite reducir la cantidad de directorios necesarios, por lo que ahora únicamente son necesarios los presentados en la jerarquía de directorios para la síntesis lógica. Esto reduce el largo de las direcciones a las que se hace referencia a través de los *scripts* y hace que estos puedan entenderse de manera más directa.

8.2. Mejoras en la síntesis lógica en librerías de TSMC

El proceso de síntesis lógica utilizando las librerías de TSMC, descrito en trabajos previos [9], proporciona resultados adecuados para la síntesis lógica. No obstante, entender este proceso no es sencillo y su modificación conlleva desafíos.

La primera mejora realizada al proceso de síntesis lógica consiste en reducir la cantidad de archivos necesarios para llevarla a cabo. Previamente existían dos *scripts* de *Design Vision* requeridos para llevar a cabo la síntesis. Se elimino uno de los archivos, esto es posible, debido a que se realizaba el proceso de síntesis dos veces, la primera siendo el circuito sin los pines de entradas y salidas, y la segunda síntesis tomando en cuenta estos. Sin embargo, es posible agregar los pines de entrada y salida, y sus respectivas conexiones, al archivo *Verilog* antes de hacer una primera síntesis.

El modo en el que se declaran los pines de entradas y salidas actualmente, consiste en declarar un módulo superior que contiene el diseño principal, los módulos de los pines y las conexiones entre estos. En trabajos previos [9] se realizó un *script* que automatiza la creación de este módulo superior para el circuito "El Gran Jaguar", la salida de este *script* era copiada al final de la primera síntesis realizada. Sin embargo es posible agregar este módulo antes de hacer algún tipo de síntesis.

El programa que automatiza la creación de este módulo superior en el que se declaran los pines, es una herramienta que facilita el proceso para el caso específico del circuito "El Gran Jaguar", sin embargo para otros circuitos, dado que la notación utilizada no será la misma, se sugiere agregarlos manualmente; este proceso es sencillo basándose en ejemplos, una explicación más detallada de esto puede encontrarse en [9]. Dado que existen distintos tipos de pines de entradas y salidas, es conveniente consultar la documentación de los pines que se desean utilizar, ya que no serán los mismos para todos los circuitos que se pueden diseñar.

Reducir la cantidad de *scripts* y número de ejecuciones de la síntesis lógica, permite a su vez reducir la cantidad de directorios requeridos, lo que facilita la comprensión del flujo. Se realizó un conteo de la cantidad de directorios requeridos para el proceso de síntesis lógica, y en el trabajo anterior [9] se contaba con 45 directorios distintos, mientras que con las mejoras implementadas se cuenta con únicamente 5.

La síntesis lógica se obtiene de manera exitosa con el nuevo flujo, la vista de esquemático puede visualizarse en la Figura 8.

Una segunda mejora introducida en la síntesis lógica se relaciona con las limitaciones de los flujos presentados en investigaciones anteriores. Estos flujos solo habían sido probados en circuitos compuestos por un único archivo, donde el módulo principal se encontraba al inicio del archivo *Verilog* y se empleaba el comando read_file. Para optimizar este proceso, se incorporó un nuevo comando, *analyze*, que identifica automáticamente el módulo superior, independientemente de su posición en el archivo. Esta adaptación ha demostrado ser efectiva al sintetizar un circuito descrito en múltiples archivos *Verilog*, confirmando que el flujo mejorado facilita una síntesis exitosa.

Para probar esta segunda mejora, se realizó la síntesis lógica de un circuito que permite

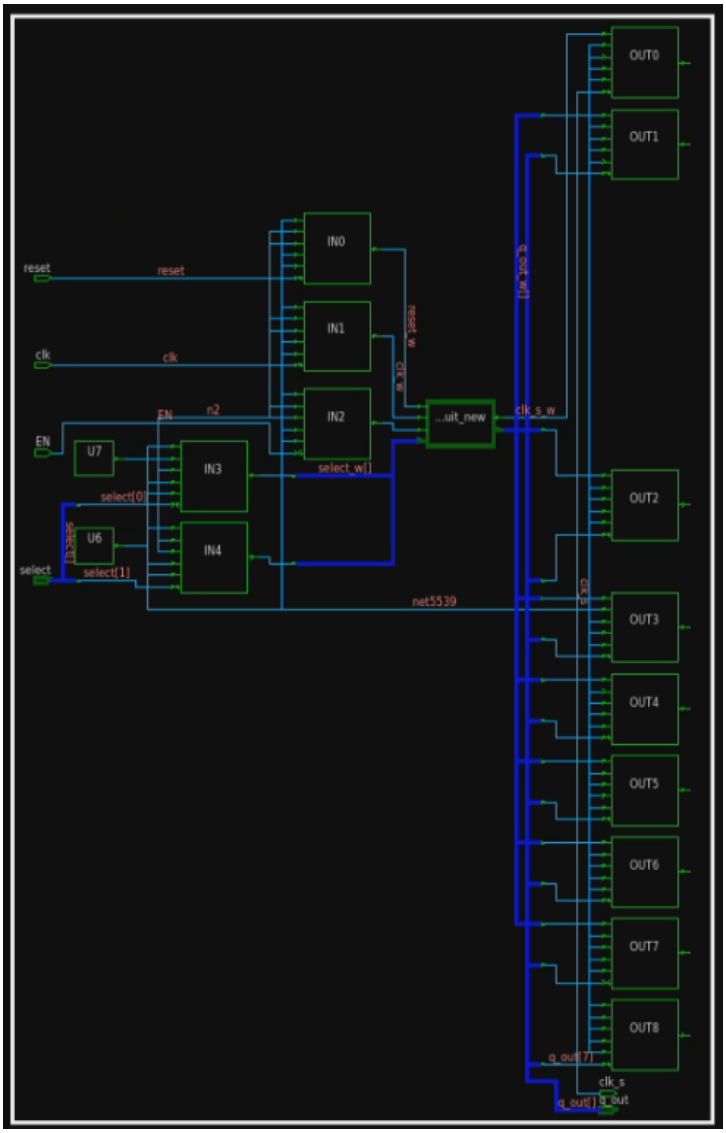


Figura 8: Síntesis lógica en librerías de TSMC del circuito "El Gran Jaguar"

jugar el juego *Pong* en una matriz de *Neo Pixel 8x8*; se optó por utilizar este circuito, debido a que esta declarado en varios archivos *Verilog* y ha sido sintetizado exitosamente en un *CPLD*. La síntesis lógica fue exitosa, el resultado se puede observar en la Figura 9

Tras las mejoras implementadas el proceso de síntesis lógica en librerías de TSMC, se realiza de manera automática al ejecutar un *script* llamado *ejecutar.bash*. Este *script* a su vez ejecuta otro *script* llamado *DV_chip.script*. Es necesario realizar múltiples *scripts* debido a que *ejecutar.bash* copia archivos hacia otros directorios, mientras el otro contiene instrucciones que son ejecutadas por la herramienta *Design Vision*.

Luego de ejecutar el *script* principal se obtienen en el directorio llamado "Salidas" tres archivos distintos, siendo estos:

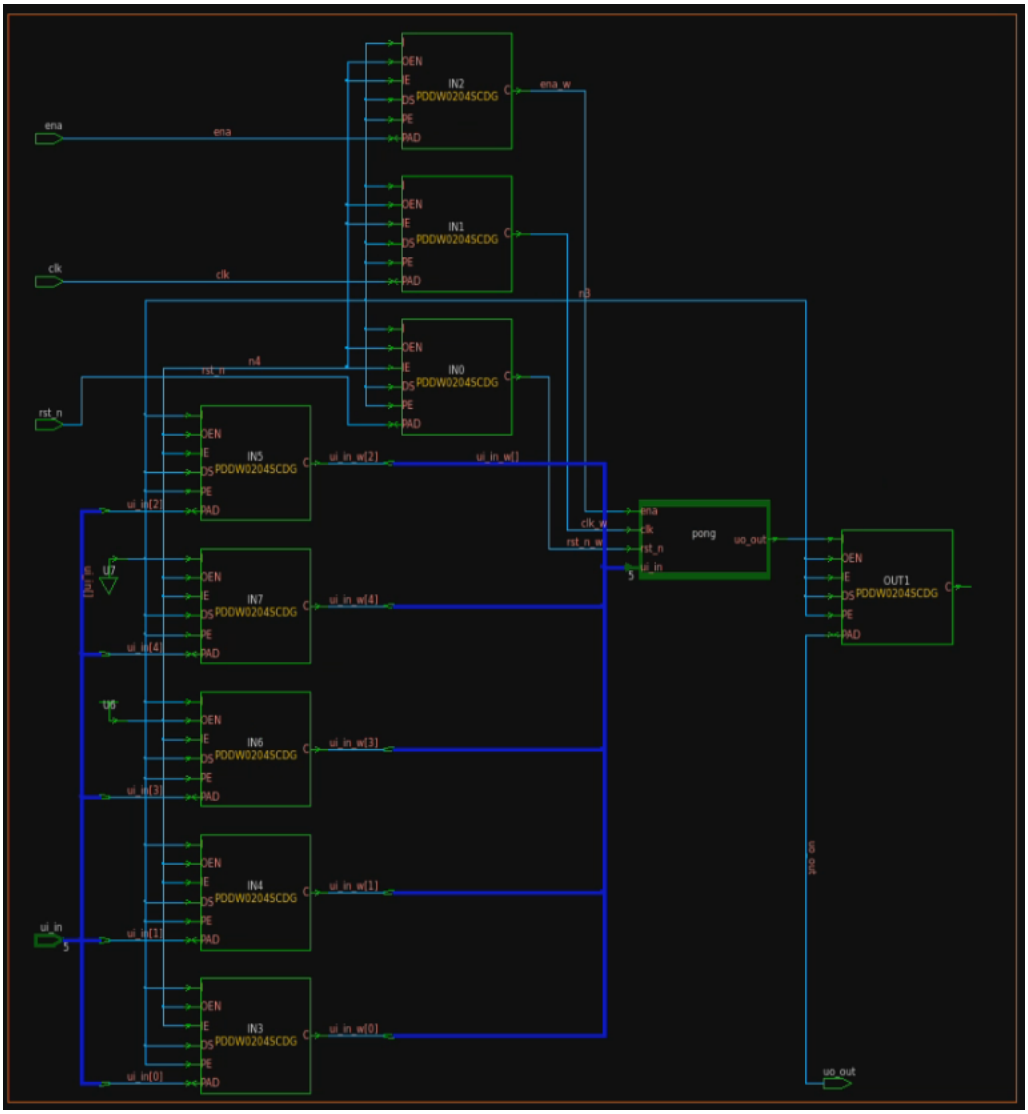


Figura 9: Síntesis lógica en librerías de TSMC de circuito para juego *Pong*

```
outchip.v
outchip.sdc
outchip.ddc
```

Estos archivos son copiados de manera automática al directorio llamado "Inputs", de la carpeta de síntesis física; estas son las salidas necesarias de la síntesis lógica para poder llevar a cabo la síntesis física.

8.3. Síntesis lógica en librerías de *Synopsys*

Todas las mejoras implementadas para la síntesis lógica en librerías de TSMC fueron adoptadas para el flujo que se utiliza en las librerías de *Synopsys*. El único cambio requerido

para que el flujo funcione con las librerías de *Synopsys* se encuentra en el archivo *DV_chip.script*, en el cual es necesario modificar el nombre de las librerías a las que se hace referencia. Esto se consigue a través de los siguientes comandos; en los cuales se coloca el nombre de los archivos .db que contienen información acerca de las celdas estándar propias de la librería.

```
set link_library " * saed14rvt_tt0p8v25c.db"
set target_library  "saed14rvt_tt0p8v25c.db"
```

Al modificar las librerías, fue posible ejecutar el flujo y obtener una síntesis lógica exitosa, esta se puede observar en las figuras 10 y 11.

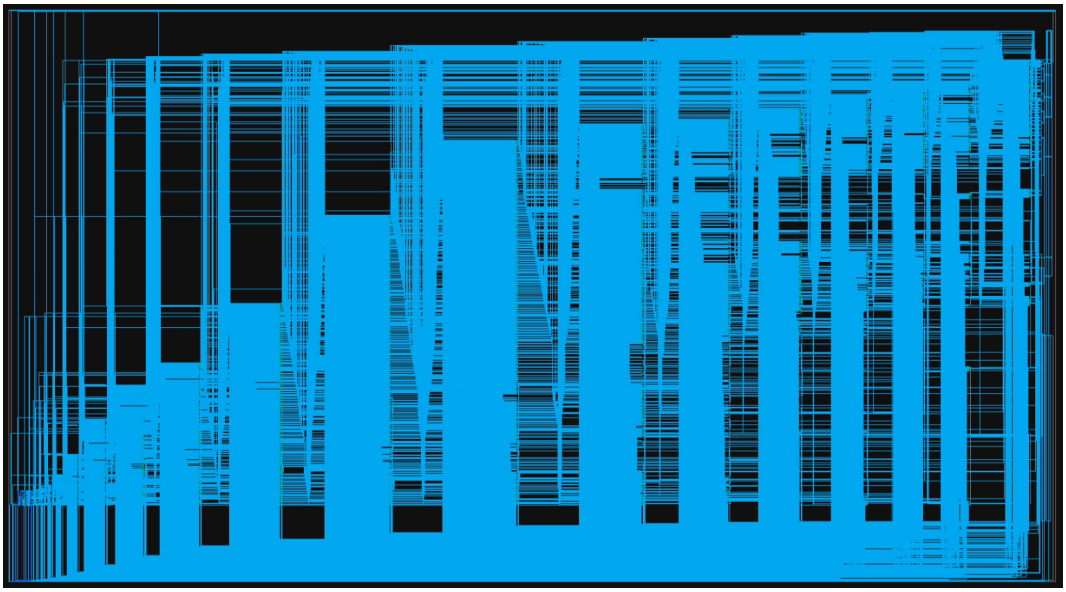


Figura 10: Síntesis lógica en librerías de *Synopsys*

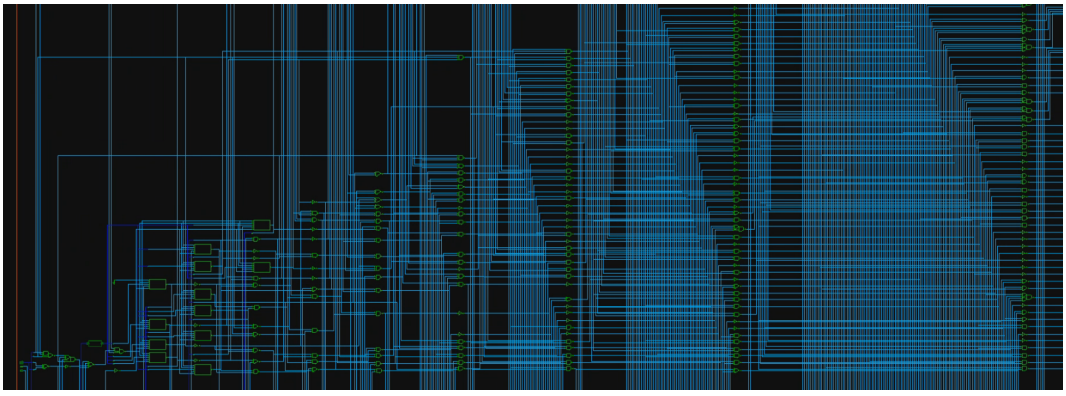


Figura 11: Síntesis lógica en librerías de *Synopsys*, vista acercada

Pines de entradas y salidas en librerías de *Synopsys*

Adicionalmente en lo que corresponde a la etapa de síntesis lógica en las librerías de *Synopsys* se adaptó el programa de *python* creado en el trabajo [9], que colocaba los pines

de entradas y salidas al circuito “El Gran Jaguar”. Este programa se llama “lectura.py”, y se encarga de leer el archivo *Verilog* sin las celdas de los pines, y a partir de la lectura agregarlos. Fue necesario realizar una adaptación, debido a que los pines de las librerías de *Synopsys* difieren respecto a los pines de las librerías de TSMC.

La adaptación usa la misma lógica que el programa original, sin embargo, para este caso se determinó que era más sencillo utilizar plantillas según el tipo de pin que se quisiera agregar, el uso de plantillas facilita la legibilidad y comprensión del programa. A continuación se presenta un ejemplo de las plantillas creadas:

```
B4ISH1025_template_input_channels = '''(
    .DOUT({i}_w[{j}]),
    .EN(1'b0),
    .DIN(1'bx),
    .R_EN(1'b1),
    .PULL_UP(1'b0),
    .PULL_DOWN(1'b0),
    .PADIO({i}[{j}]));'''
```

La plantilla mostrada es la que se utiliza para generar el texto de los pines de entrada que cuentan con varios bits. Es posible observar todo el programa modificado en la sección de anexos.

Se acudió a la documentación de la librería, para poder realizar la conexión de manera correcta. Se encontraron pines de entradas y salidas con propiedades muy similares a los utilizados en las librerías de TSMC. En este caso se utilizaron pines FINFET no inversores bi-direccionales, de `4mA`, con driver tri estado, *pull-up* y *pull-down*, y *buffer* de entrada *Schmitt triggered*. Es posible observar el diagrama de estas celdas en la Figura 12

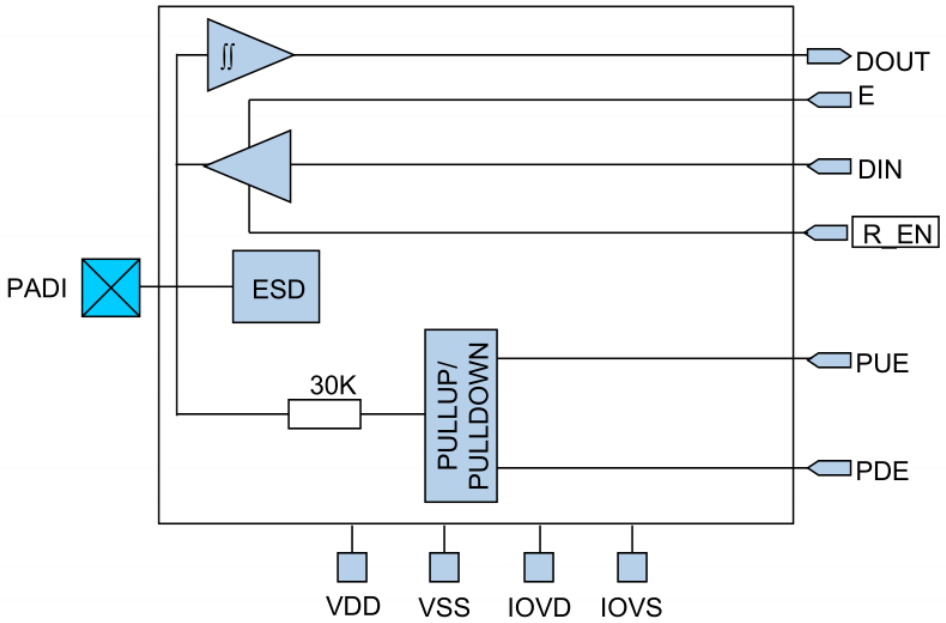


Figura 12: Diagrama de la celda de los pines de entradas y salidas en librerías de *Synopsys*

Tras agregar los pines de entradas y salidas al programa, y realizar la síntesis lógica, el resultado era el esperado. Al igual que en el caso de la síntesis lógica en las librerías de TSMC, para ejecutar la síntesis lógica únicamente es necesario ejecutar el script llamado *ejecutar.bash*.

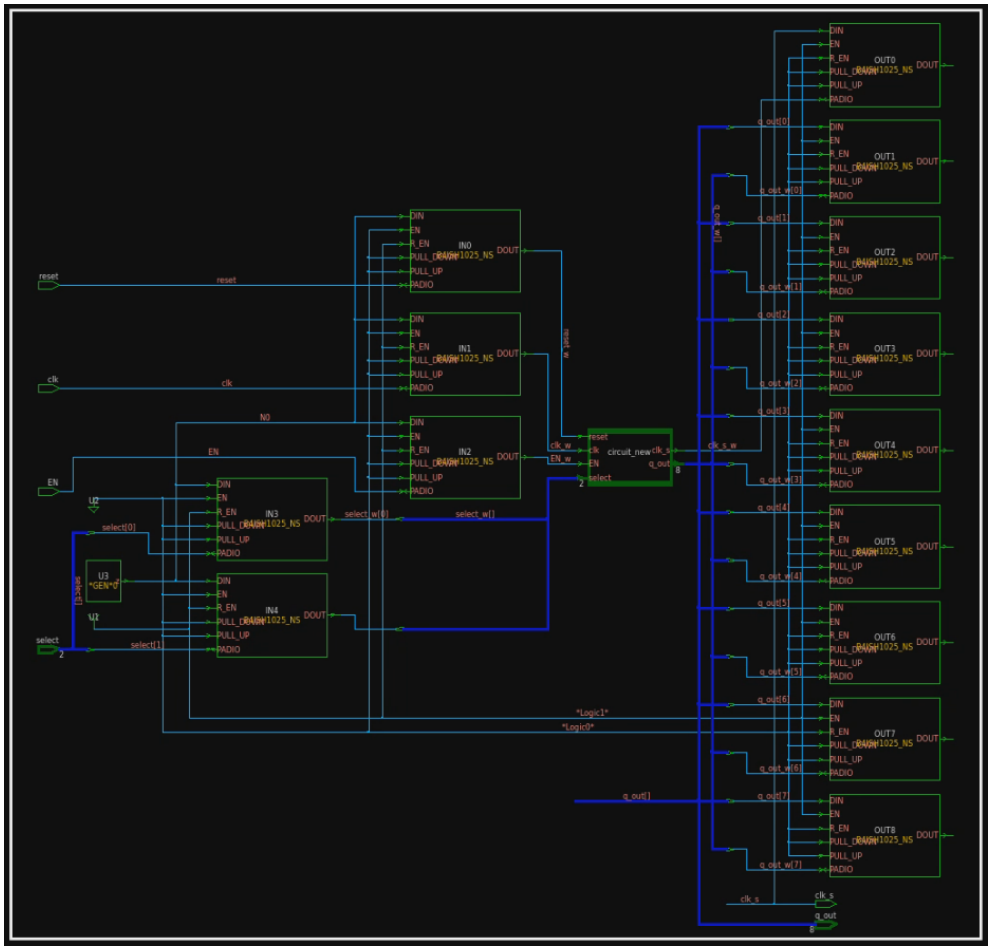


Figura 13: Vista de esquemático de la síntesis lógica en librerías de *Synopsys* con pines de entradas y salidas

CAPÍTULO 9

Síntesis física

9.1. Mejoras en la síntesis física en librerías de TSMC

El flujo propuesto se estructura aprovechando la nueva jerarquía de directorios, lo que clarifica el orden de los archivos referenciados. Para la ejecución de este flujo, se emplean tres archivos con extensión `.tcl`:

- `sintesis_fisica_top.tcl`: Es el archivo a través del cual se inicia el flujo de la síntesis física. Este archivo llama a otros *scripts* que residen en la carpeta `scripts`, específicamente `setup.tcl` y `floorplan.tcl`.
- `setup.tcl`: Dentro de este *script*, se configuran las variables de entorno asociadas a todos los archivos necesarios para la síntesis física. Esta configuración facilita etapas posteriores, permitiendo hacer referencia a los archivos de manera más intuitiva. Además, se crea la librería donde se guarda el diseño.
- `floorplan.tcl`: En este archivo, se lleva a cabo la inicialización del *floorplan*, la creación de la red de alimentación y el *placement* inicial de las celdas estándar del circuito.

El desarrollo restante del flujo se realiza en el archivo principal. Esta organización facilita la comprensión del proceso al desglosar las tareas principales en diferentes *scripts*. Para que la síntesis física se realice únicamente es necesario ejecutar el *script sintesis_fisica_top.tcl*.

La nueva jerarquía de directorios permite reducir significativamente la cantidad de directorios empleados en el proceso de síntesis física; en la versión anterior del flujo [8], se contaba con un total de 195 directorios correspondientes al proceso de síntesis física; en la nueva versión se cuenta solamente con 69. Es necesario mencionar que la gran mayoría de estos directorios corresponden a la estructura en la cual se almacena el proyecto.

Los resultados de la síntesis física pueden visualizarse en el capítulo de *DRC* y en el de la modificación del *power grid*. Se colocan dentro de estos capítulos debido a que dentro de estos se explica porque se realizaron modificaciones sustanciales en la síntesis.

9.2. Síntesis física en librerías de *Synopsys*

En el trabajo [11], se realizó la síntesis física de diversos circuitos con las librerías de diseño de *Synopsys*; dentro de ese trabajo se utilizó un flujo que daba *Synopsys* como ejemplo. En este trabajo se realizó la síntesis física del circuito "El Gran Jaguar" a través de un flujo basado en el utilizado en las librerías de TSMC.

A pesar de que el flujo utilizado es basado en el de las librerías de TSMC; fue necesario realizar modificaciones significativas. Esto se debe a que no fue posible realizar la síntesis física tomando en cuenta los pines de entradas, salidas y alimentación. La síntesis lógica tomando en cuenta los pines fue exitosa, sin embargo, al intentar realizar la síntesis física, se obtenía un error relacionado con los pines; el error indicaba que las celdas estándar que corresponden a los pines no son del tipo `PAD`, por lo que no son colocables dentro de un anillo. En consecuencia se optó por realizar la síntesis física sin los pines. Este problema es propio de las celdas de las librerías `ndm` de *Synopsys*, ya que el flujo avanzaba de la manera esperada hasta llegar a ese punto.

Como consecuencia de este problema, se optó por realizar la síntesis física sin tomar en cuenta los pines I/O; el resto del flujo es similar al utilizado con las librerías de TSMC. En la Figura 14 es posible visualizar la síntesis física obtenida.

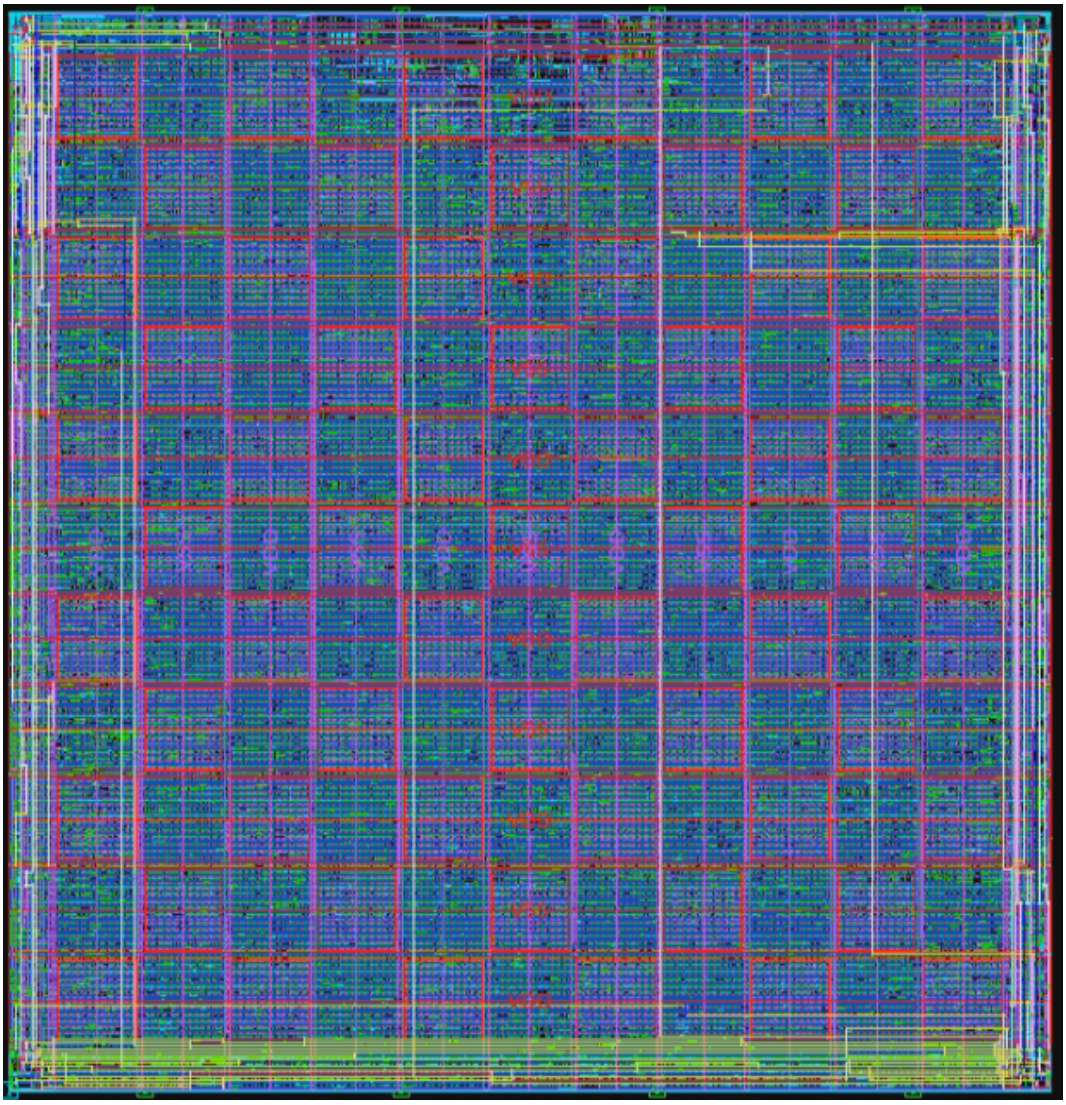


Figura 14: Síntesis física en librerías de *Synopsys*

CAPÍTULO 10

Design Rule Check

10.1. Mejoras en *Design Rule Check* en librerías de TSMC

En el trabajo [8], se llevó a cabo la verificación *DRC*, durante la cual se identificaron errores de densidad. Comúnmente, para abordar estos errores se emplea un *runset* con instrucciones para desarrollar la inserción de metal. Sin embargo, en el momento de ese trabajo, no se disponía de un *runset* compatible con *IC Validator*. Por ello, se procedió a desarrollar una traducción del *runset* para garantizar su compatibilidad con la herramienta en cuestión. A lo largo de este año, se adquirió el *runset* oficial de TSMC, diseñado para aplicar el *metal fill* usando *IC Validator* en las librerías de diseño actuales. Dicho *runset* lleva por nombre `Dummy_Metal_ICV_0.18um.215a`.

A pesar de contar con el *runset* adecuado para la generación del *metal fill*, la ejecución del comando `signoff_create_metall_fill` no produjo un circuito libre de errores de densidad. Se detectó un error específico en la capa de metal 1. Para resolver este inconveniente, se recurrió a los comandos `spread_wires` y `widen_wires`. Tras aplicar estos comandos, se obtuvo un circuito libre de errores. Es posible visualizar el *layout* de este en la Figura 15 y el resultado de la verficación *DRC* en la Figura 16.

El comando `spread_wires` se utiliza para distribuir los cables en el diseño actual con el objetivo de mejorar el área crítica total de cortocircuitos. Este comando ajusta la colocación de los cables dentro del diseño del circuito integrado para reducir la probabilidad de que ocurran cortocircuitos debido a defectos en el proceso de fabricación. Al esparcir los cables, se aumenta la distancia entre ellos, disminuyendo así el área donde las partículas contaminantes podrían causar conexiones no deseadas. Esto es esencial para mejorar el rendimiento y la fiabilidad del chip, además de optimizar el rendimiento del proceso de manufactura [27].

El comando `widen_wires` se emplea para ensanchar los cables en el diseño actual, con

el fin de reducir el área crítica de interrupciones (área donde se podrían producir cortes en las conexiones) y así mejorar el rendimiento de producción. Este comando modifica el grosor de los cables dentro del diseño del circuito integrado, aumentando su ancho para disminuir la probabilidad de que los defectos de fabricación causen cortes o interrupciones en las conexiones. Al ensanchar los cables, se refuerza la robustez de las conexiones y se mejora la confiabilidad del dispositivo, lo que resulta crucial para optimizar la producción y asegurar la calidad del chip. Esto puede mejorar la densidad de metal en cada capa del circuito [27].

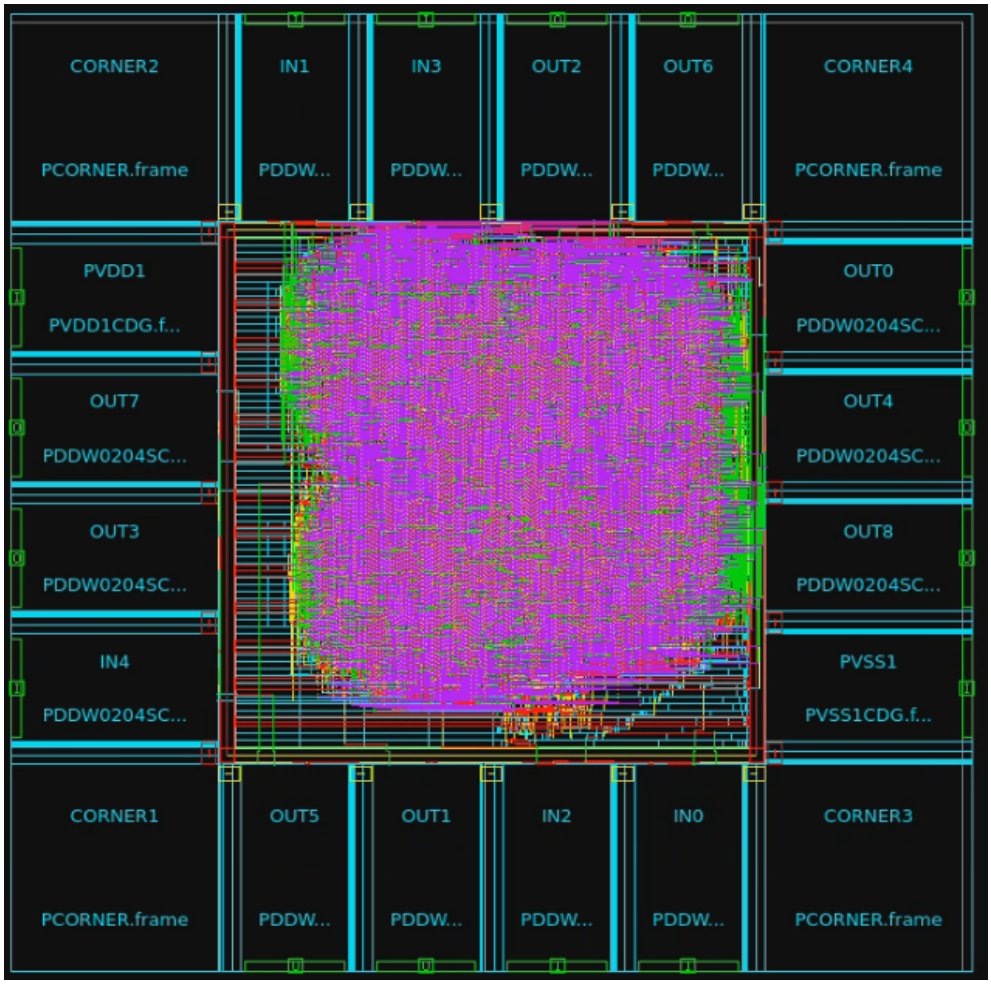


Figura 15: Síntesis física densidad

Una mejora adicional en el proceso de inserción de *metal fill* fue la capacidad de visualizarlo, algo que no se había logrado en iteraciones anteriores. Para ver el *metal fill* insertado mediante el *runset*, es necesario activar esta función en el menú `View Settings`. Para ello, se debe asignar un valor superior a 0 en la opción `Level`. Posteriormente, en la pestaña de configuración adyacente, se selecciona `Fill Cell` dentro del menú `Expanding Cell Types`. En la Figura 17 se puede visualizar este menú.

En la Figura 18 es posible visualizar la inserción de metal en la capa metal 1, y en la Figura 19 es posible observar la inserción de metal en la capa de metal 5, en este caso es posible observar la inserción de metal alrededor de un área que no cuenta con metal en

Figura 16: Resultado de verificación DRC

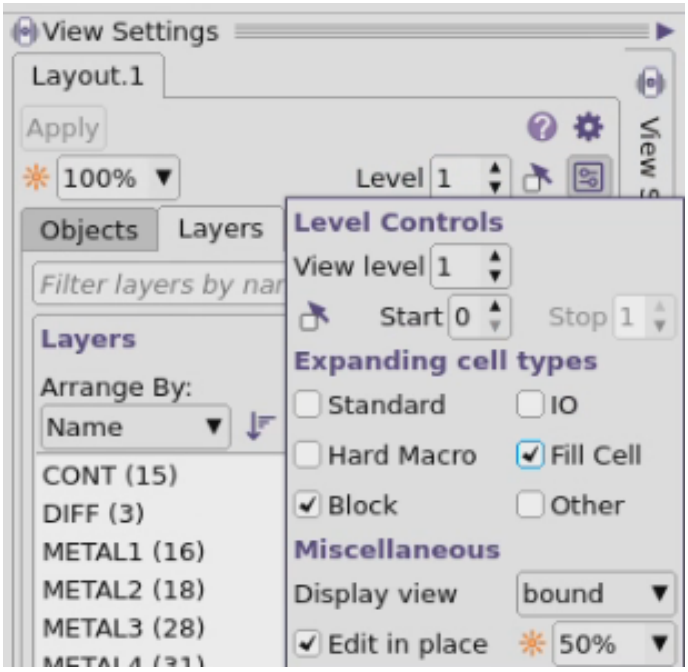


Figura 17: Menú para habilitar la visualización del *metal fill*

esa capa. La inserción es similar en las otras capas. En la Figura 20 es posible observar la visualización del *metal fill* habilitada para todas las capas de metal.

Tras corregir los errores de densidad, se ejecutó el comando *check_lvs*, para verificar que no existan cortos o *nets* desconectadas. El resultado de este comando indicaba que la alimentación y tierra del circuito no estaban conectadas. El resultado puede observarse a continuación:

```
Information: Detected floating route violation for Net VDD.
BBox: (125.0000 367.0000)(409.4800 373.0000). (RT-587)
Information: Detected floating route violation for Net VSS.
BBox: (261.3900 418.2400)(273.0900 426.2400). (RT-587)
```

No se colocan todos los mensajes obtenidos, dado que se trata del mismo mensaje en diferentes coordenadas del circuito.

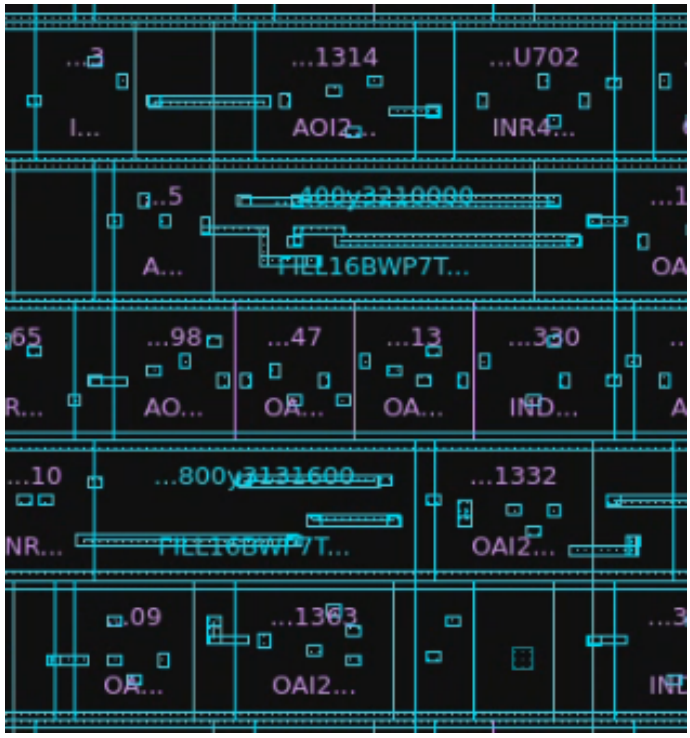


Figura 18: Visualización del *metal fill* en la capa metal 1, en librerías de TSMC

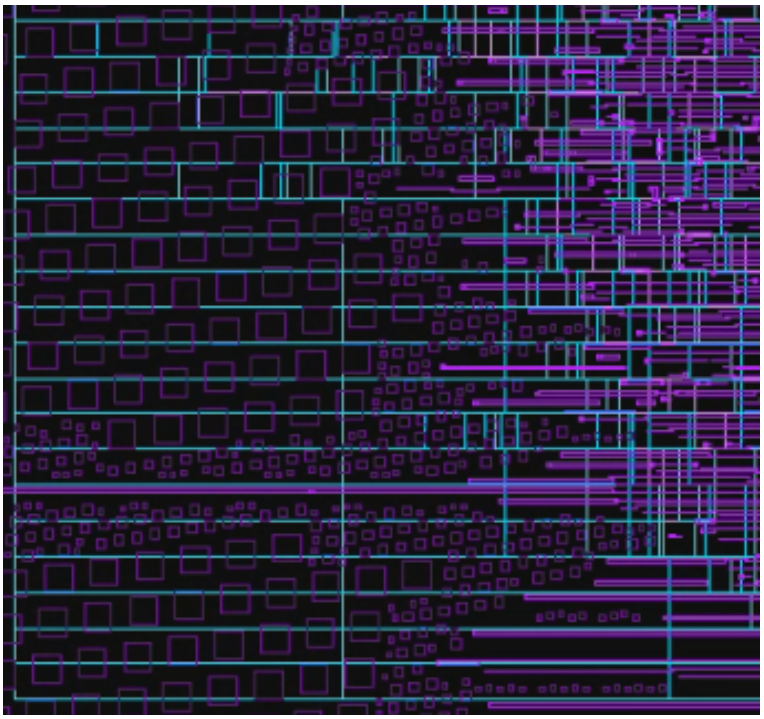


Figura 19: Visualización del *metal fill* en la capa metal 5, en librerías de TSMC

Adicionalmente se presentaban también errores de corto circuito en las celdas *filler* de los pines de entradas y salidas. Los errores de corto circuito desparacían al remover estas

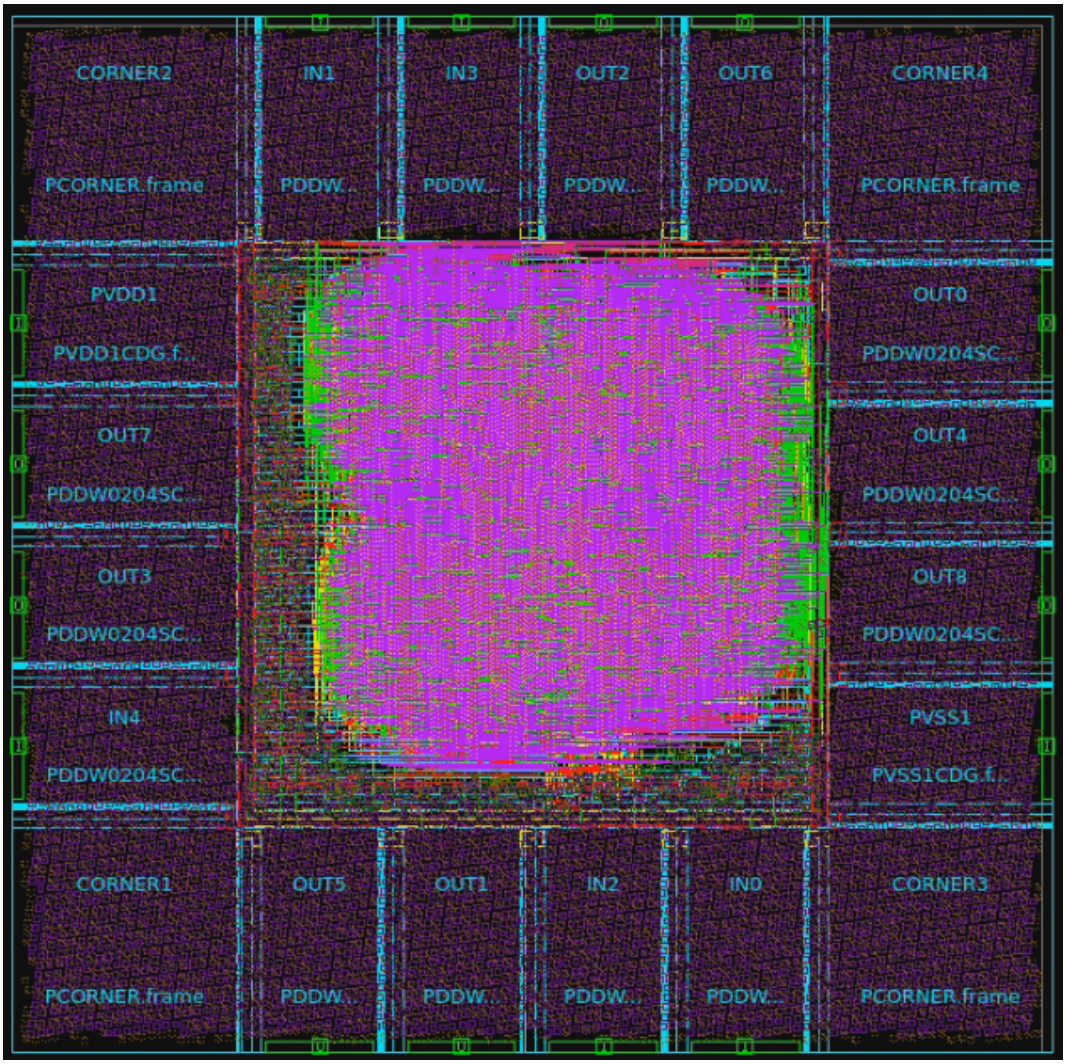


Figura 20: Síntesis física sin errores de densidad con la visualización del *metal fill* habilitada para todas las capas, en librerías de TSMC

celdas, la naturaleza de este error no ha sido contemplada previamente y debe ser consultada directamente con el *foundry* debido a que se trata de un problema intrínseco a este tipo de celdas. A continuación se muestran estos errores:

```
Information: Detected short violation.
Cell1: __added_filler_instance_2590.
Cell2: __added_filler_instance_2591.
BBox: (115.8840 -0.0010)(115.8860 115.0010).
Layer: METAL4. (RT-586)
Information: Detected short violation.
Cell1: __added_filler_instance_2590.
Cell2: __added_filler_instance_2591.
BBox: (115.8840 -0.0010)(115.8860 115.0010).
Layer: METAL3. (RT-586)
```

Al igual que para el caso anterior, no se colocan todos los mensajes obtenidos, dado que se trata del mismo mensaje en diferentes coordenadas del circuito.

El resultado obtenido fue inesperado, dado que se asumía que el flujo de diseño empleado estaba exento de errores de este tipo. No obstante, una vez detectados, fue sencillo identificar la causa. Estos errores se originan debido a la falta de conexión entre los pines de alimentación y el anillo, así como la ausencia de conexión entre el anillo y la red de alimentación interna del circuito. Esto puede observarse en la Figura 21.

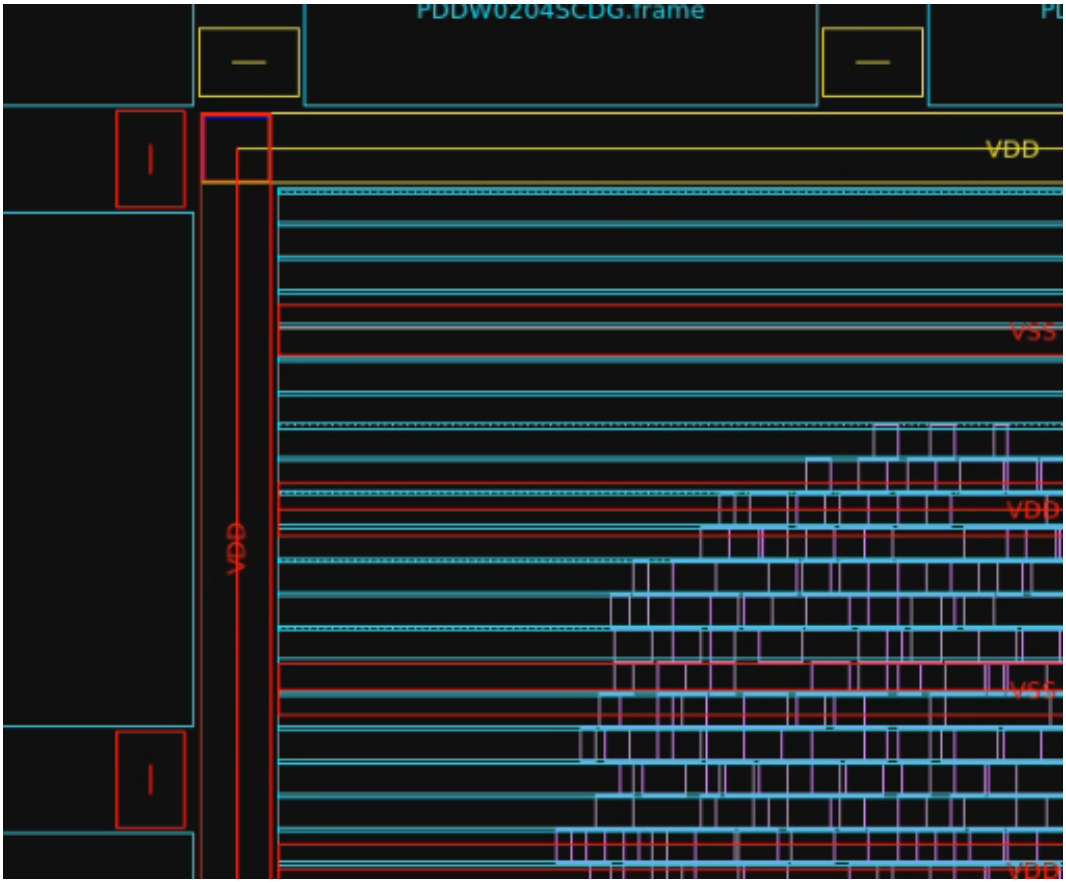


Figura 21: Redes de alimentación desconectadas en síntesis física con librerías de TSMC

Como consecuencia de estos resultados, surge la necesidad de modificar la red de alimentación del circuito. Esto se menciona en el capítulo 12.

10.2. *Design Rule Check* en librerías de *Synopsys*

Se llevó a cabo la verificación *Design Rule Check* en las librerías de 14nm de *Synopsys* para el diseño del circuito denominado "El Gran Jaguar" utilizando un *runset* compatible con la herramienta *IC Validator*, proporcionado dentro de los archivos de las librerías. Desafortunadamente, no se logró obtener un resultado de verificación libre de errores, y la cantidad de errores que presentaba el diseño lo hacían no reparable, ni a través del comando *signoff_fix_drc*, ni manualmente.

Al intentar verificar diseños más simples, como un *full adder* de 4 bits, los resultados aún presentaban errores, lo que sugiere posibles problemas con el *runset* utilizado. Es posible observar una imagen de la síntesis obtenida para el *full adder* de 4 bits en la Figura 22. A continuación se presentan los errores obtenidos tras correr el comando *signoff_fix_drc*.

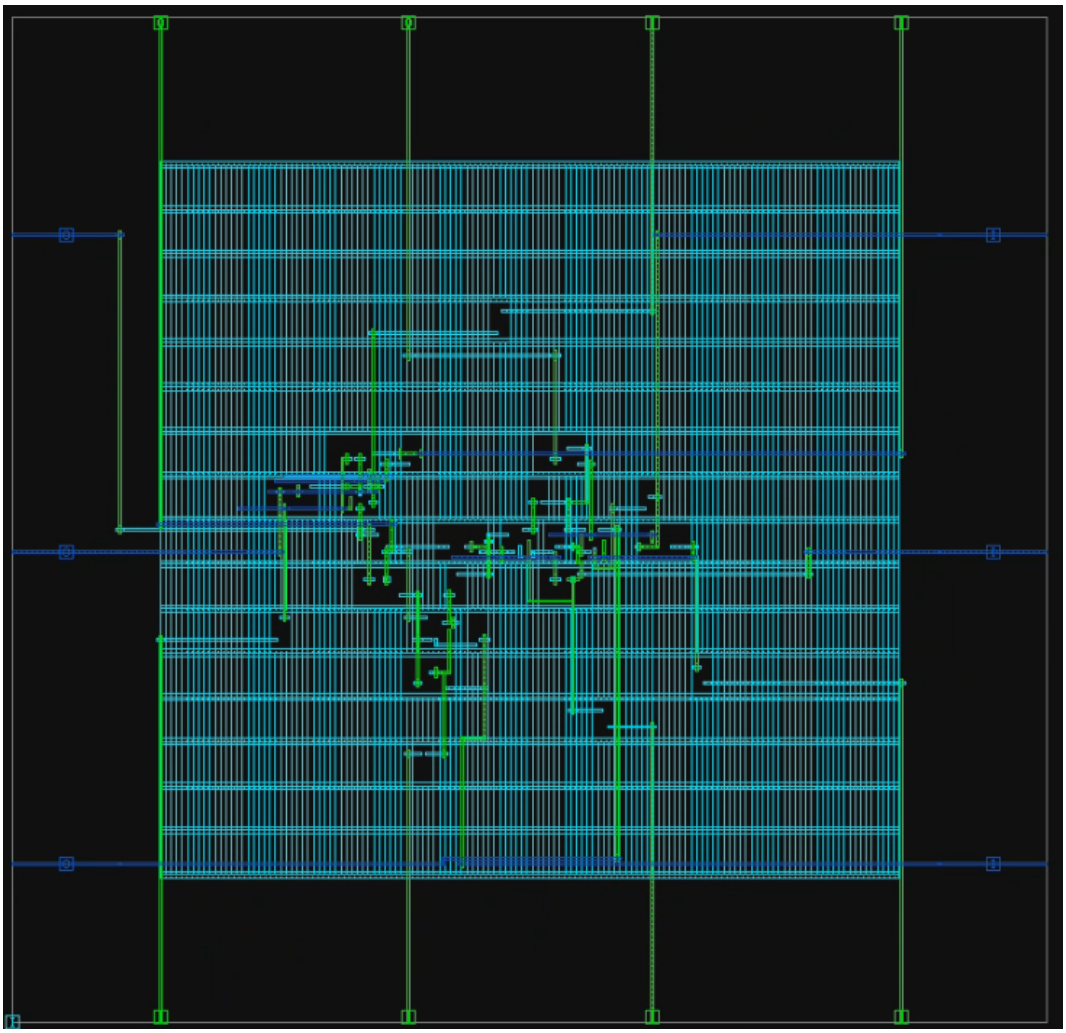


Figura 22: Síntesis física en librerías de *Synopsys* de un sumador de 4 bits

Se probaron circuito aún más simples para comprobar los resultados obtenidos a través de la verificación *DRC*. Se probó un sumador *full adder* de dos bits. La síntesis se puede observar en la Figura 23.

A pesar de tratarse de un circuito que contaba con solamente seis celdas de las librerías, este presentaba errores, incluso después de correr el comando *signoff_fix_drc*. Los errores que presentaba son los siguientes:

Al observar los errores a través del observador de errores incorporado en la herramienta *IC Compiler II*, *IC Validator VUE*, no se encontraba la razón por la cual existía error en la localidad indicada. Esto se ejemplifica en la Figura 24. El error al que se hace referencia en esa figura es el segundo del cuadro anterior.

Regla	Descripción	Errores
M1.S.1	Minimum M1 spacing must be 0.026	13 errors
M1.S.2	Minimum spacing if metal width of one of lines $>$ 0.058 and their parallel run length $>$ 0.07 must be 0.05	9 errors
M1.S.3	Minimum spacing if metal width of one of lines $>$ 0.06 and their parallel run length $>$ 0.07 must be 0.055	14 errors
M1.W.1	Minimum M1 width must be 0.03	4 errors
VIA1.E.1	Minimum VIA1 enclosure by M1 and M2 must be 0.002	4 errors
VIA1.S.1	Minimum center to center spacing of VIA1SQ to VIA1SQ: 0.05	1 error
VIA1.S.2	Minimum spacing of VIA1SQ to VIA1BAR or VIA1LG must be 0.08	6 errors

Cuadro 6: Errores presentes *full adder* de 4 bits

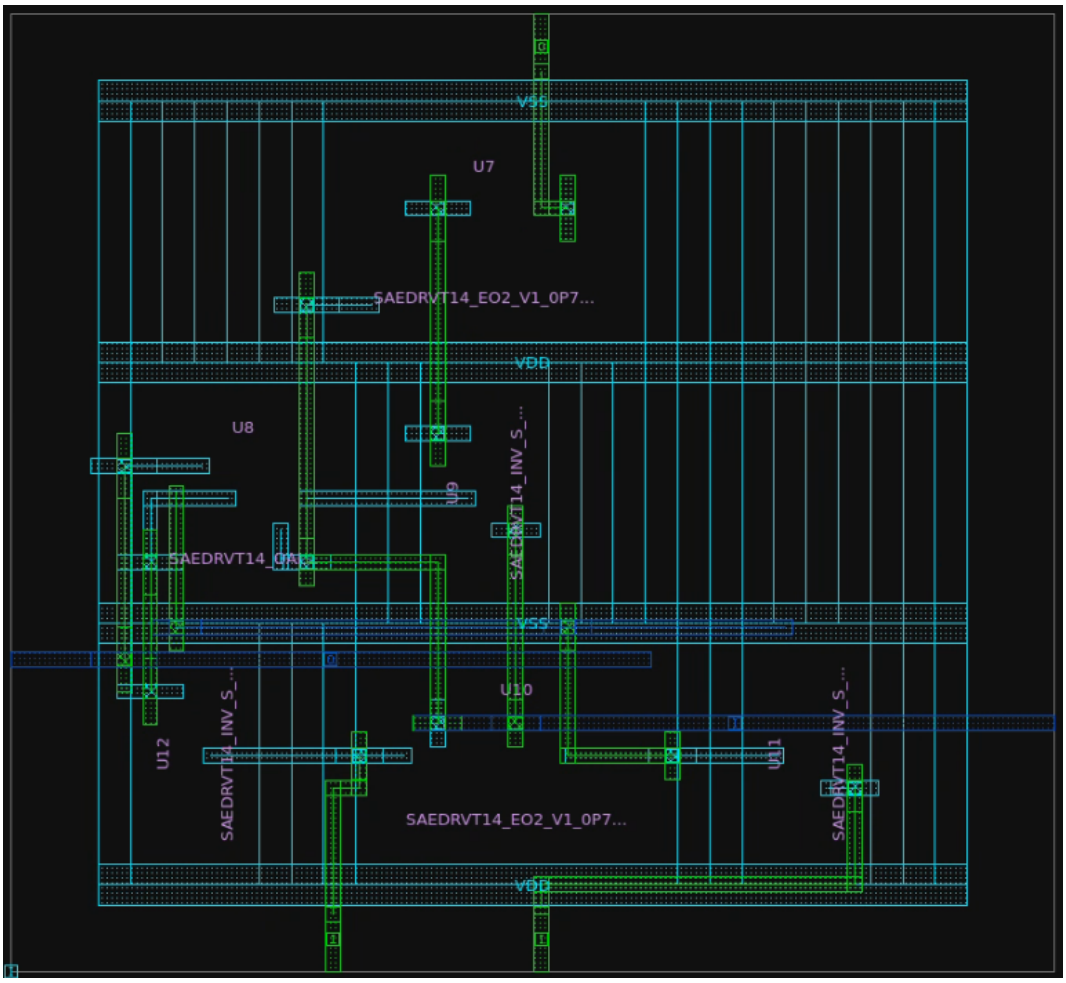


Figura 23: Síntesis física en librerías de *Synopsys* de un *full adder* de 2 bits

Se realizaron pruebas con circuito aún más simples, se generó un circuito que constaba únicamente de dos celdas, este consistía en una compuerta *AND*, una *OR* y una *XOR*. La síntesis física puede observarse en la Figura 25. Este circuito también presentaba errores incluso después de ejecutar el comando *signoff_fix_drc*. Los errores que presentaba son los

Regla	Descripción	Errores
M1.S.1	Minimum M1 spacing must be 0.026	3 errors
M1.S.3	Minimum spacing if metal width of one of lines $>$ 0.06 and their parallel run length $>$ 0.07 must be 0.055	1 error

Cuadro 7: Errores presentes en el circuito con solamente seis celdas

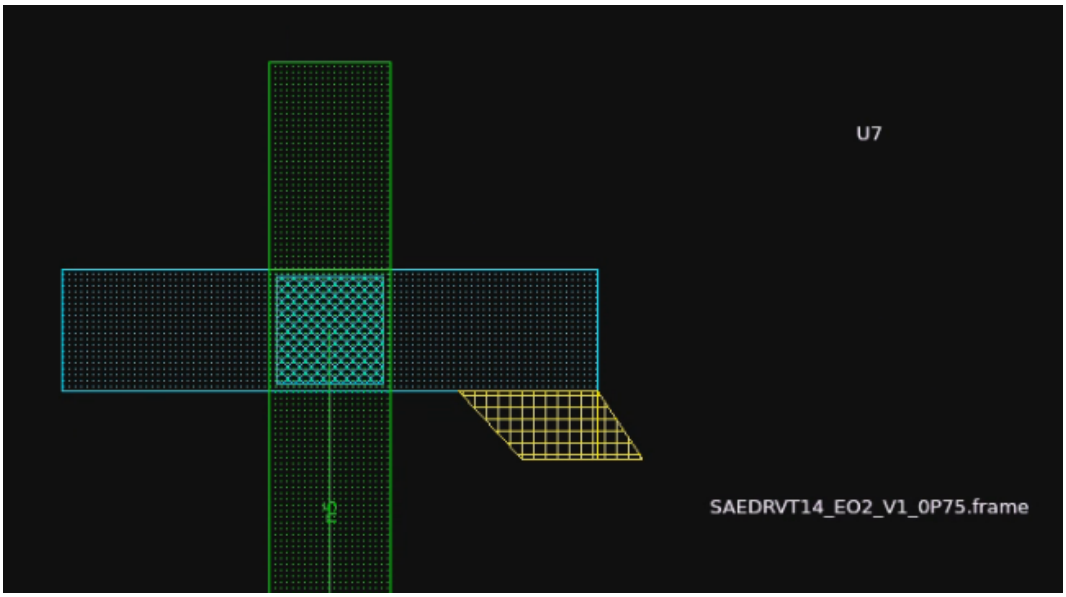


Figura 24: Error marcado por *IC Validator VUE* dentro del circuito *full adder* de 2 bits

siguientes:

Regla	Descripción	Errores
M1.S.2	Minimum spacing if metal width of one of lines $>$ 0.058 and their parallel run length $>$ 0.07 must be 0.05	1 error
M1.S.3	Minimum spacing if metal width of one of lines $>$ 0.06 and their parallel run length $>$ 0.07 must be 0.055	1 error
VIA1.S.2	Minimum spacing of VIA1SQ to VIA1BAR or VIA1LG must be 0.08	3 errors

Cuadro 8: Errores presentes en el circuito con solamente dos celdas

Dado que se encontraban errores en circuitos que constaban únicamente de dos celdas, se realizó la síntesis de un circuito que constaba de una única compuerta, se optó por una compuerta *NOT*. La síntesis física puede observarse en la Figura 26. Al realizar la verificación *DRC* esta salía sin errores. Se probaron distintos circuitos de una única compuerta, y en estos no se obtenían errores. Se realizaron pruebas con circuitos de una única compuerta debido a que se quería verificar si la herramienta no estaba detectando errores en las celdas mismas.

Esta situación motivó una investigación más detallada, que incluyó la participación en

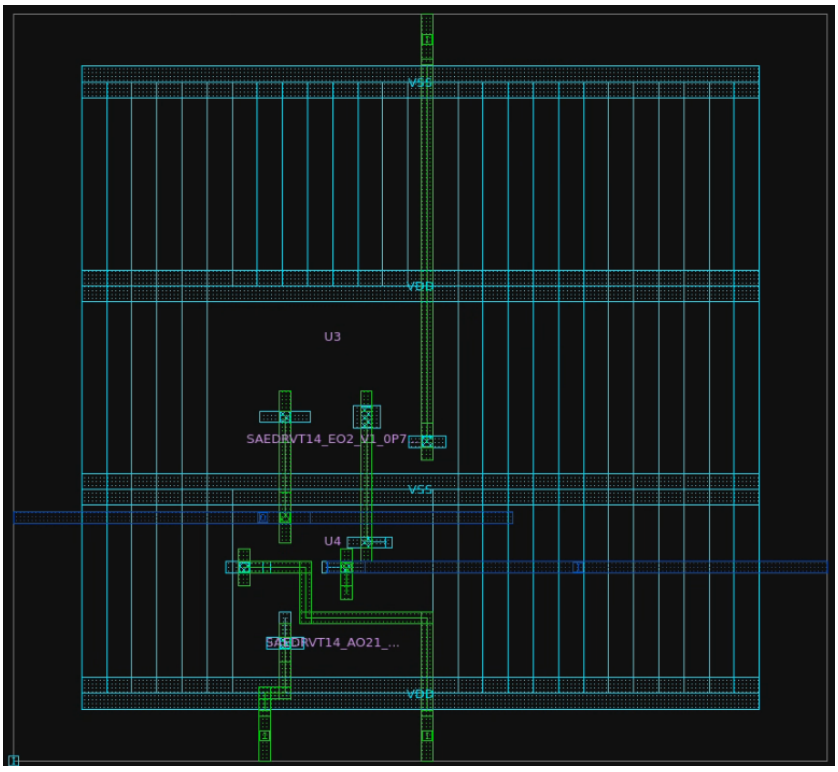


Figura 25: Síntesis física en librerías de *Synopsys* de un circuito con únicamente dos celdas

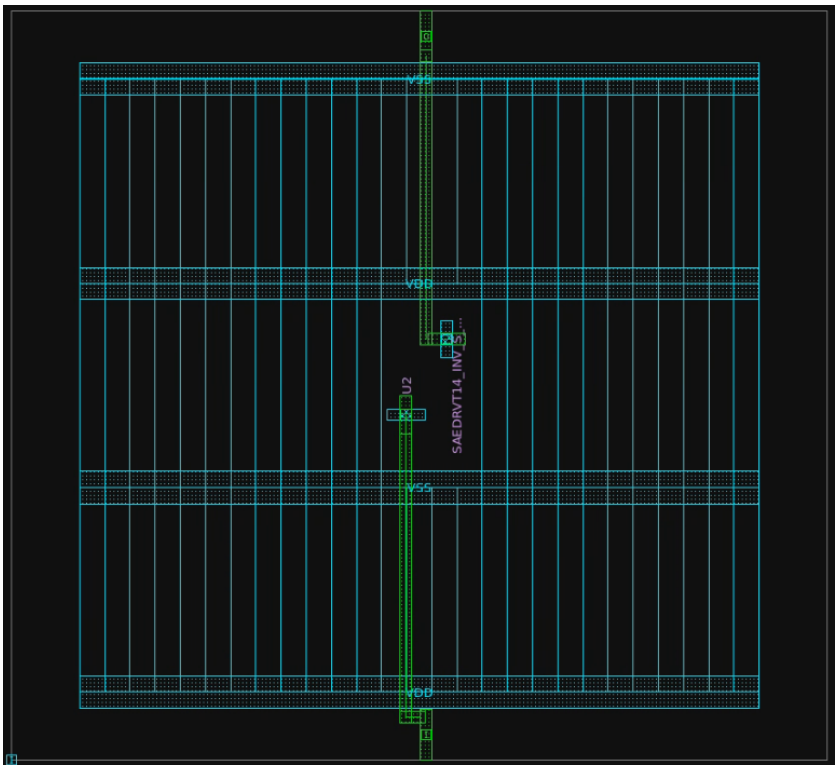


Figura 26: Síntesis física en librerías de *Synopsys* de una compuerta *NOT*

un curso de *Synopsys* titulado "Low Power Design with SAED 14nm EDK". Aunque el curso abordó diversos aspectos del enrutamiento, enfocándose en técnicas para diseños de baja potencia, no cubrió la etapa de verificación. Sin embargo, proporcionó un ejemplo para realizar la síntesis lógica y física de un microprocesador *opensource* llamado "OpenSPARC". Al ejecutar este ejemplo e integrar manualmente la etapa de verificación, se encontraron numerosos errores. Esto junto a las pruebas realizadas previamente sugiere que el *runset* proporcionado podría no ser confiable. Esta hipótesis es coherente considerando que las librerías SAED 14nm son de carácter educativo y no están destinadas a la fabricación, por lo que la verificación no sería una prioridad.

En la Figura 27 se muestra el resultado obtenido al hacer la síntesis lógica y física del microprocesador "OpenSPARC"; el resultado de la verificación *DRC* consta de muchos errores de diversa naturaleza y nos es reparable a través del comando *signoff_fix_drc*.

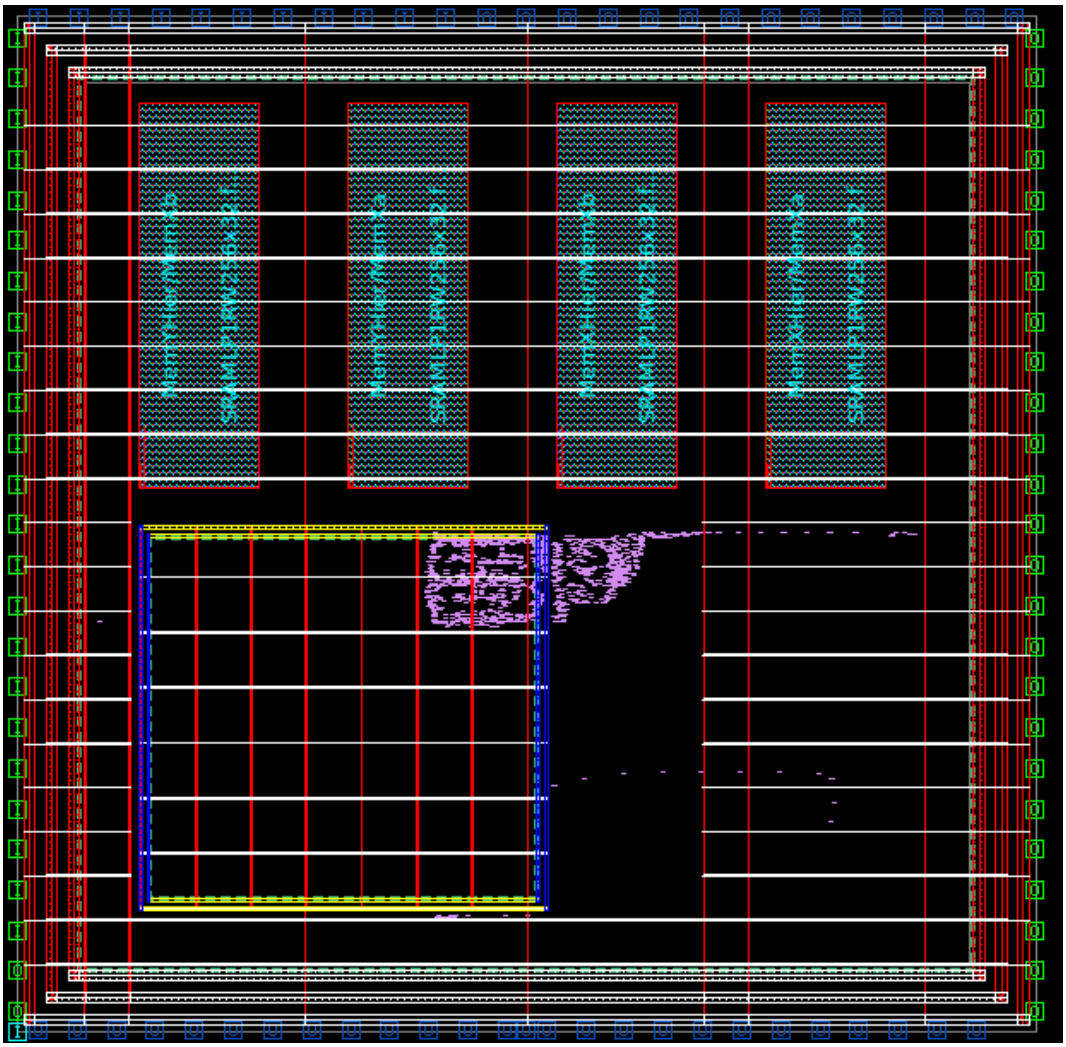


Figura 27: Síntesis física en librerías de *Synopsys* del microprocesador OpenSPARC

CAPÍTULO 11

Verificación de antena

11.1. Mejoras en verificación de antena en librerías de TSMC

Dado que la verificación de antena se realizaba de manera óptima en iteraciones pasadas no se generaron cambios significativos a esta para las librerías de *Synopsys*. En el trabajo [5] es posible encontrar una descripción detallada sobre el modo en el que se realiza la verificación.

11.2. Verificación de antena en librerías de *Synopsys*

Para realizar la verificación de antena en librerías de *Synopsys* se requiere el *runset* proveído dentro del `iPDK`, y el archivo `gds` al que se le quiere aplicar la verificación. El *runset* para realizar la verificación de antena se llama: `saed14nm_1p9m_ant_drc_rules.rs`.

La verificación de antena se ejecuta a través de la herramienta *ICV*. Para realizar la verificación se debe utilizar el siguiente comando:

```
icv -i path_hacia_el_archivo_gds -c nombre_de_top_cell \
-sf ICV -vue ../../Librerias/Runset/saed14nm_1p9m_ant_drc_rules.rs
```

Este proceso no difiere del realizado en las librerías de TSMC. Se realizó la verificación en varios circuitos, y en todos los casos se obtiene un resultado sin errores.

A continuación se presentan los resultados obtenidos para la verificación de antena de diversos circuitos:

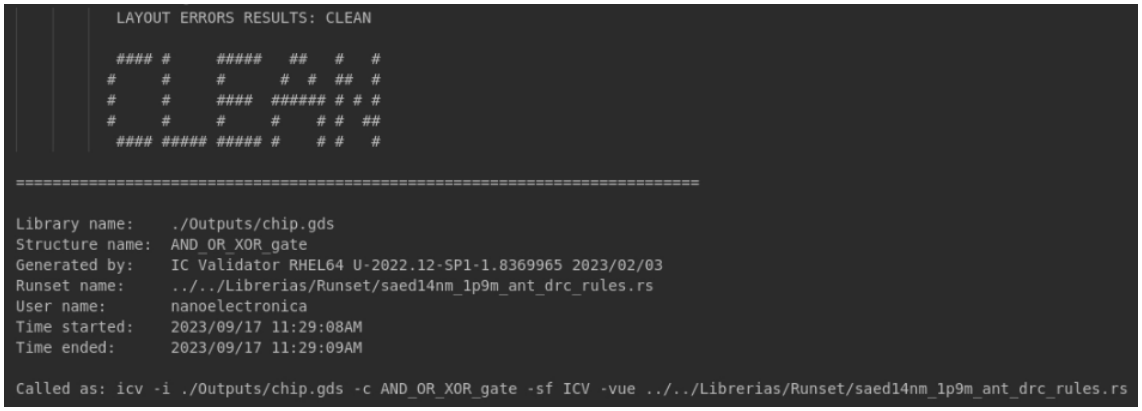


Figura 28: Verificación de antena en librerías de *Synopsys* de circuito con dos celdas

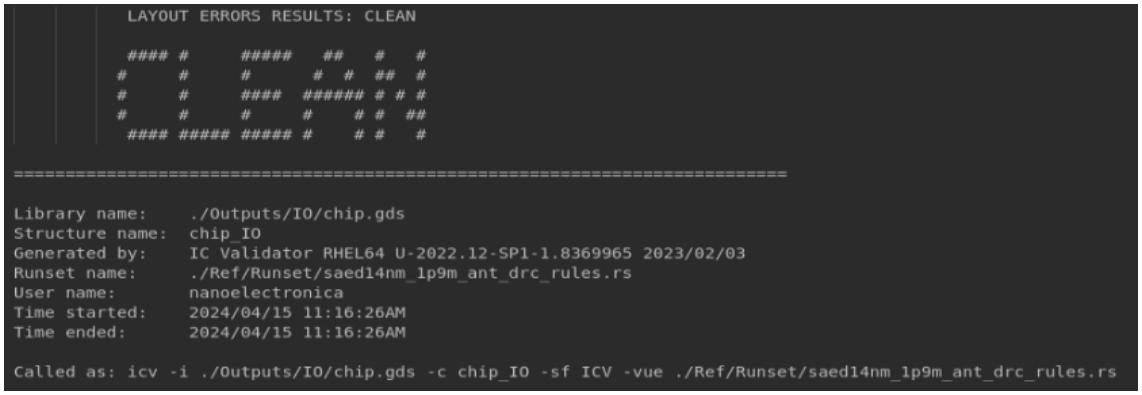


Figura 29: Verificación de antena en librerías de *Synopsys* del circuito El Gran Jaguar

CAPÍTULO 12

Modificación de *Power Grid* en librerías de TSMC

Debido a que tras realizar la verificación *DRC* al circuito "El Gran Jaguar" se obtuvo un resultado sin errores, se realizó la verificación *layout vs schematic* (LVS). A través de esta fue posible determinar que existían errores en la red de alimentación propuesta en el trabajo [8]. Se asume que estos errores no fueron contemplados, debido a que se estaba buscando solucionar los errores de densidad. Como consecuencia de esto surge la necesidad de modificar la red de alimentación.

La modificación en la red de alimentación consistió en utilizar como base la red de alimentación propuesta en el trabajo [5]. Agregando conexión para la red de alimentación en la capa metal 1, a través de un patrón de alimentación vertical en la capa metal 4.

Las modificaciones a la red consistieron en cambiar el orden de las capas del anillo de alimentación, es decir utilizar metal 2 en la capa vertical y metal 3 en la capa horizontal. Esto en consecuencia, implica modificar la conexión con el anillo, la cual se realizó utilizando conexiones verticales en la capa metal 4, estas a su vez establecen la conexión de toda la red de alimentación en metal 1, donde se conectan directamente todas las celdas estándar.

Estas modificaciones fueron realizadas, debido a que se observó que el tiempo necesario para realizar la síntesis era menor, y se obtenían menos errores en cada paso. A través de estas modificaciones se obtuvo la síntesis que se puede observar en la Figura 30.

12.0.1. Verificación DRC tras la modificación de *Power Grid*

Desafortunadamente no fue posible solucionar los errores de densidad presentes tras modificar el flujo. Sin embargo, tras realizar todos los pasos mencionados en el capítulo de la verificación DRC, se obtuvieron únicamente 2 errores. Los errores de densidad presentes en este caso se encuentran en la capa de metal 1, y en la capa de metal 2.

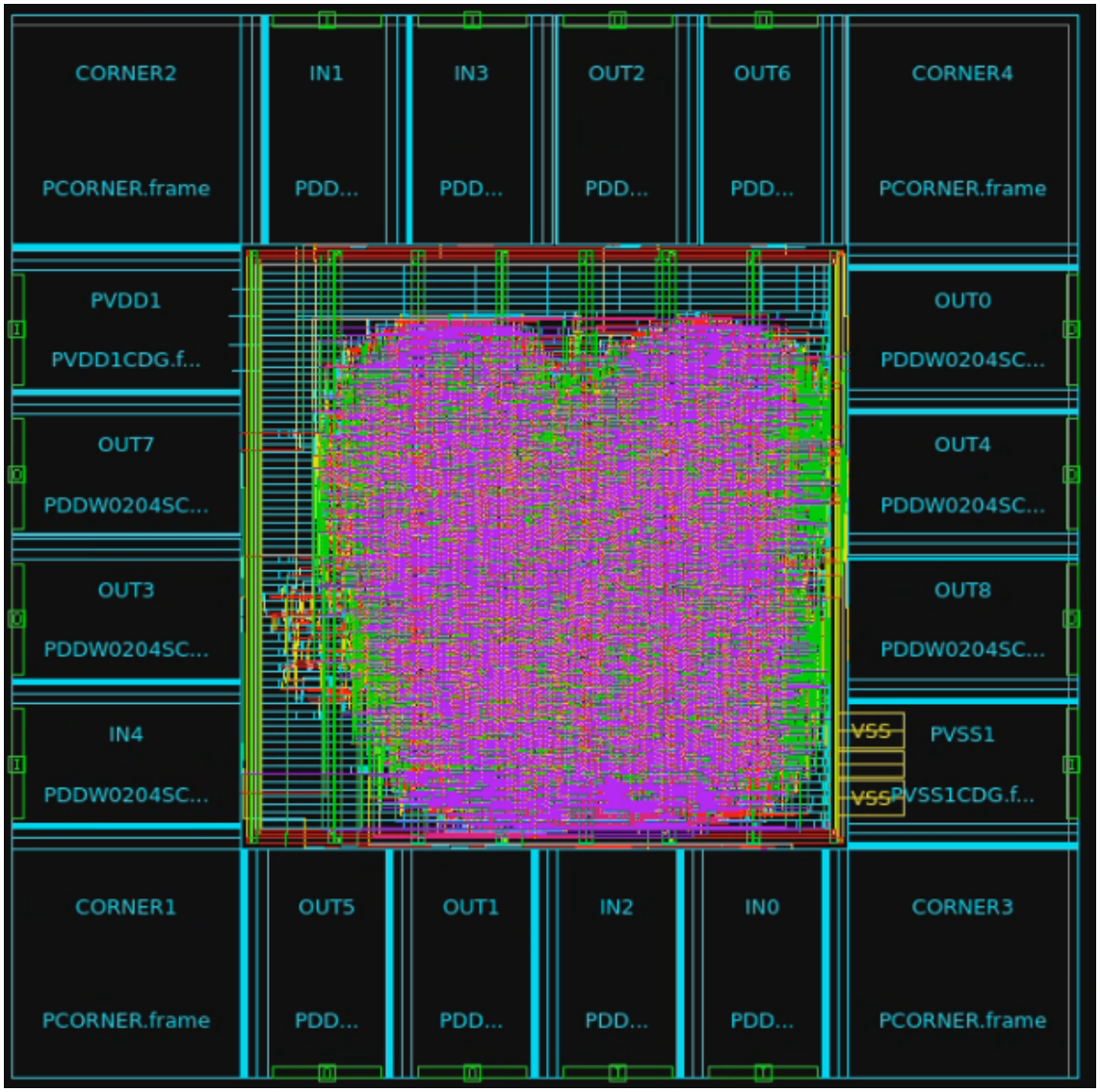


Figura 30: Síntesis obtenida tras modificar la red de alimentación

El porcentaje de densidad para la capa metal 1 en este caso es del 28.69 % y para la capa metal 2 es del 26.10 %. Dado que estos porcentajes son cercanos al valor esperado y que no se puede resolver de la manera convencional, es decir, con el *runset* de relleno de metal, el siguiente paso será recurrir a TSMC, para determinar si es posible cumplir los requisitos de algún otro modo.

Para visualizar los porcentajes de densidad de metal que no cumplen con las reglas de diseño, es posible ejecutar el comando *signoff_ check_ drc*; e inmediatamente se generan archivos con el nombre `MX_density.log` (donde la X se reemplaza por la capa en la cual la densidad no es suficiente). Estos archivos de generan en la ubicación donde se ejecuta el DRC. Con la configuración utilizada se define que esto se ejecute en un directorio llamado DRC, dentro de la carpeta *Outputs*, la cual se encuentra dentro del directorio de la síntesis física.

Regla	Descripción	Errores
M1.R.1	Min M1 area coverage $< 30\,\%$ density	1 violation found.
M2.R.1	Min M2 area coverage $< 30\,\%$ density	1 violation found.

Cuadro 9: Errores obtenidos tras la verificación DRC, en el circuito con *Power Grid* modificada

Tras realizar estas modificaciones, a pesar de que no fue posible solucionar por completo los errores de densidad, se ejecutó nuevamente la verificación LVS, en este caso no se obtenían errores de redes desconectadas. Descartando el error que se fue mencionado en el capítulo 10, respecto a cortos circuitos detectados en las celdas *filler* de los pines de entradas y salidas, no se obtiene ningún error en la verificación LVS. A continuación se puede observar el informe de la prueba LVS tras modificar la red de alimentación (este resultado es tras remover celdas *filler* de los pines de entradas y salidas):

```
Total number of input nets is 2799.
Total number of short violations is 0.
Total number of open nets is 0.
Total number of floating route violations is 0.
```

CAPÍTULO 13

Conclusiones

- Se realizó la síntesis lógica del circuito El Gran Jaguar en las librerías de diseño de *Synopsys*, y se generó un flujo para obtener la síntesis lógica en estas librerías a partir de cualquier archivo *verilog*.
- Se realizó la síntesis física del circuito El Gran Jaguar en las librerías de diseño de *Synopsys*, y se generó un flujo para obtener la síntesis física en estas librerías a partir del resultado de la síntesis lógica.
- Se realizaron mejoras en los flujos existentes de síntesis lógica y síntesis física en las librerías de TSMC. Las principales mejoras corresponden a una nueva jerarquía de directorios que facilita la comprensión del flujo, la implementación de la inserción de metal para resolver errores de densidad, y la modificación de la red de alimentación para solucionar errores en la verificación LVS.
- Se separó el proceso de creación del archivo *verilog* del proceso de síntesis lógica, para simplificar el flujo de la síntesis lógica.
- Se realizó la verificación física DRC en las librerías de diseño de *Synopsys*, y se determinó que el *runset* proveído dentro de estas no permite obtener un resultado sin errores.
- Se utilizó el *runset* para realizar el relleno de metal proveído por TSMC, sin embargo este no resolvió por completo los errores de densidad.
- Se utilizó el *runset* para realizar el relleno de metal proveído dentro de las librerías de diseño de *Synopsys*, y no se obtuvieron errores de esta naturaleza en la verificación DRC. Estos errores nunca fueron detectados por la verificación DRC en estas librerías.
- Se realizó la verificación de antena en las librerías de diseño de *Synopsys*; el resultado de esta verificación era el esperado para diversos circuitos. Respecto a esta verificación no se encontraron mejoras respecto a lo propuesto previamente dentro de la línea de investigación.

CAPÍTULO 14

Recomendaciones

Aislar distintas etapas de diseño del circuito, dejando claro que salidas son entradas de la siguiente etapa, ya que esto facilita el control de cada etapa, y la comprensión del flujo para nuevos integrantes de la línea de investigación.

Dado que el requisito de densidad de metal para ambas capas en las cuales no se cumple es cercano al mínimo, se recomienda contactar al *foundry* para consultar acerca de las posibilidades para abordar este problema.

Utilizar nuevas librerías de diseño *Open Source*, tales como la librería de diseño *Skywater*, para profundizar más en el diseño de circuitos integrados.

Utilizar herramientas *Open Source* para el diseño de circuitos, tales como las proporcionadas por el proyecto *OpenRoad*, con la finalidad de profundizar más en el diseño de circuitos integrados.

CAPÍTULO 15

Bibliografía

[1] J. A. de los Santos Chonay, *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.* Universidad del Valle de Guatemala, Guatemala, 2014.

[2] S. H. R. Vasquez, *Definición del Flujo de Diseño para Fabricación de un Chip con Tecnologia VLSI CMOS*, Universidad del Valle de Guatemala, Guatemala, 2019.

[3] L. A. N. Vásquez, *Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado.* Universidad del Valle de Guatemala, Guatemala, 2019.

[4] M. G. F. Espino, *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*, Universidad del Valle de Guatemala, Guatemala, 2020.

[5] J. A. A. Escobar, *Diseño de un Circuito Integrado con Tecnología de 180 nm Usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificaciones de Antena y Corrección de Errores Obtenidos*, Universidad del Valle de Guatemala, Guatemala, 2021.

[6] A. A. Hernández, *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.* Universidad del Valle de Guatemala, Guatemala, 2021.

[7] J. E. S. Jo, *Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: ejecución de la síntesis física, validación de reglas eléctricas y corrección de errores obtenidos*, Universidad del Valle de Guatemala, Guatemala, 2021.

[8] D. R. E. Pirir, *Diseño e implementación de interfaz gráfica de la automatización de las fases de diseño de un circuito integrado y uso avanzado de IC Compiler II.* Universidad del Valle de Guatemala, Guatemala, 2022.

[9] A. E. A. Chocooj, *Automatización de la etapa de síntesis lógica y creación de archivos Verilog para pruebas físicas en un FPGA Genesys Xilinx Virtex-5 LX50T y automatización de la verificación extracción de parásitos.* Universidad del Valle de Guatemala, Guatemala, 2022.

[10] L. R. G. Velásquez, *Procesamiento de las señales generadas por un circuito integrado con tecnología de 180nm usando librerías de diseño de TSMC montado en un FPGA Digilent Genesys Board demostrando el funcionamiento mediante una aplicación y sintetizador digital*, Universidad del Valle de Guatemala, Guatemala, 2022.

[11] H. A. O. Barrios, *Diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC: Implementación de un alternativo flujo de diseño proporcionado por Synopsys con las herramientas de Formality y Design Compiler*, Universidad del Valle de Guatemala, Guatemala, 2022.

[12] J. D. P. D. Cid, *Diseño de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC: Implementación de un alternativo flujo de diseño proporcionado por Synopsys con las herramientas PrimeTime y TetraMAX*, Universidad del Valle de Guatemala, Guatemala, 2022.

[13] N. H. E. Weste y D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th. Boston, MA: Addison-Wesley, 2011.

[14] I. Synopsys, *Synopsys.* dirección: `https://www.synopsys.com` (visitado 2023).

[15] I. Synopsys, *Accelerate Your Silicon Success with High-Quality IP.* dirección: `https://www.synopsys.com/designware-ip.html` (visitado 2023).

[16] I. Synopsys, *Polaris.* dirección: `https://www.synopsys.com/software-integrity.html` (visitado 2023).

[17] I. Synopsys, *ZeBu Server 5.* dirección: `https://www.synopsys.com/verification/emulation/zebu-server.html` (visitado 2023).

[18] I. Synopsys, *VCS Functional Verification Solution.* dirección: `https://www.synopsys.com/verification/simulation/vcs.html#resources` (visitado 2023).

[19] I. Synopsys, *Design Compiler.* dirección: `https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html` (visitado 2023).

[20] I. Synopsys, *Synopsys IC Compiler II.* dirección: `https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html` (visitado 2023).

[21] I. Synopsys, *Physical Verification - IC Validator.* dirección: `https://www.synopsys.com/implementation-and-signoff/physical-verification.html` (visitado 2023).

[22] I. Synopsys, *PrimeTime Static Timing Analysis.* dirección: `https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html` (visitado 2023).

[23] "PrimeTime Basics," en *Advanced ASIC Chip Synthesis Using Synopsys® Design Compiler™ Physical Compiler™ and PrimeTime®*. Boston, MA: Springer US, 2002, págs. 239-259, ISBN: 978-0-306-47507-8. DOI: `10.1007/0-306-47507-3_12`. dirección: `https://doi.org/10.1007/0-306-47507-3_12`.

[24] I. Synopsys, *Formality Equivalence Checking.* dirección: `https://www.synopsys.com/implementation-and-signoff/signoff/formality-equivalence-checking.html` (visitado 2023).

[25] I. Synopsys, *Synopsys TetraMAX Diagnostics for Rapid Yield Learning Adopted By UMC*. dirección: `https://news.synopsys.com/index.php?s=20295&item=122401` (visitado 2023).

[26] Synopsys, Inc., *TetraMAX® ATPG User Guide*, Version I-2013.12-SP4, 700 E. Middlefield Road, Mountain View, CA 9404, jun. de 2014.

[27] I. Synopsys, *Man pages - ICC2*, 2023.

CAPÍTULO 16

Anexos

16.1. Creación de archivo *verilog* con pines IO para síntesis lógica en librerías de *Synopsys*

```
1
2 from asyncore import write
3 from contextlib import ContextDecorator
4 import re
5
6 B4ISH1025_template_input = '''(
7 .DOUT({i}_w),
8 .EN(1'b0),
9 .DIN(1'bx),
10 .R_EN(1'b1),
11 .PULL_UP(1'b0),
12 .PULL_DOWN(1'b0),
13 .PADIO({i})
14 );
15 '''
16
17
18 B4ISH1025_template_input_channels = '''(
19 .DOUT({i}_w[{j}]),
20 .EN(1'b0),
21 .DIN(1'bx),
22 .R_EN(1'b1),
23 .PULL_UP(1'b0),
24 .PULL_DOWN(1'b0),
25 .PADIO({i}[{j}])
26 );
27 '''
28
29 B4ISH1025_template_output = '''(
```

```
30
31 .EN(1'b1),
32 .DIN({i}_w),
33 .R_EN(1'b0),
34 .PULL_UP(1'b0),
35 .PULL_DOWN(1'b0),
36 .PADIO({i})
37 );
38 '''
39
40 B4ISH1025_template_output_channels = '''(
41
42 .EN(1'b1),
43 .DIN({i}_w[{j}]),
44 .R_EN(1'b0),
45 .PULL_UP(1'b0),
46 .PULL_DOWN(1'b0),
47 .PADIO({i}[{j}])
48 );
49 '''
50
51
52 archivo = open("circuito.v", 'r')
53 lista=archivo.readlines()
54 archivo.close()
55 sup=lista.index('endmodule\n')
56 inputsito=[]
57 outputsito=[]
58 quitacorinput = []
59 quitacoroutput = []
60 solocorinput = []
61 solocorinput1 = []
62 solocoroutput = []
63 solocoroutput1 = []
64 sincorinput = []
65 sincoroutput = []
66
67 for a in range(sup):
68 b=lista[a]
69 if b.find("input")>=0:
70 inputsito.append(re.sub("input |;","",b.rstrip('\n')))
71 if b.find("output")>=0:
72 outputsito.append(re.sub("output |;","",b.rstrip('\n')))
73
74 for c in inputsito:
75 if c.find('[')>=0:
76 quitacorinput.append(re.sub("\[.*?\]","",c))
77 solocorinput.append(c)
78 solocorinput1.append(re.sub("\[.*?\]","",c))
79 else:
80 quitacorinput.append(c)
81 sincorinput.append(c)
82
83 for c in outputsito:
```

```
84 if c.find('[')>=0:
85 quitacoroutput.append(re.sub("\[.*?\]","",c))
86 solocoroutput.append(c)
87 solocoroutput1.append(re.sub("\[.*?\]","",c))
88 else:
89 quitacoroutput.append(c)
90 sincoroutput.append(c)
91
92 zinput2 = str('module')
93 with open("circuito.v", 'r') as file:
94 for line in file:
95 if zinput2 in line:
96 modulito = line
97 break
98 buscarini = modulito.find("module")
99 buscarfin = modulito.find("(")
100 mensaje9 = modulito
101 startLoc = buscarini
102 endLoc = buscarfin
103 mensaje9b = mensaje9[startLoc: endLoc]
104 #filtro del nombre del modulo
105 namemodulito = mensaje9b.replace('module ','')
106 file.close()
107
108 f = open('modulochipio.v','w')
109
110 #inicia el modulo
111 f.write("module circuit(" + ",".join(quitacorinput) + "," +
      ",".join(quitacoroutput) + ");")
112 f.write( '\n\n')
113
114 for i in outputsito:
115 f.write('\t' + "output " + i + ";" + '\n')
116 for i in inputsito:
117 f.write('\t' + "input " + i + ";" + '\n')
118 for i in inputsito:
119 f.write('\t' + "wire " + i + "_w;" + '\n')
120 for i in outputsito:
121 f.write('\t' + "wire " + i + "_w;" + '\n')
122
123 f.write( '\n')
124
125 #linea extra
126 extra_line = namemodulito + " circuit_new ("
127
128 for i in quitacoroutput:
129 extra_line = extra_line + "." + i + "(" + i + "_w), "
130 for i in quitacorinput:
131 extra_line = extra_line + "." + i + "(" + i + "_w), "
132 extra_line2 = extra_line + ");"
133 #filtro de la linea extra
134 lineaextra = extra_line2.replace(', )',')')
135 f.write(lineaextra + '\n')
136 f.write( '\n')
```

```
137
138
139
140 f.write("//Entradas (C):" + '\n')
141 contadorsito = 0
142
143
144 for i in sincorinput:
145 f.write("B4ISH1025_NS IN" + f"{contadorsito:000}" +
      B4ISH1025_template_input.format(i = i))
146
147 contadorsito = contadorsito + 1
148
149
150
151 e = 0
152 for c in solocorinput:
153 k = int(c[c.index(':') + 1:c.index(']')])
154 j = int(c[c.index('[') + 1:c.index(':')])
155 p = solocorinput1[e]
156 for k in range(j+1):
157 f.write("B4ISH1025_NS IN" + f"{contadorsito:000}" +
      B4ISH1025_template_input_channels.format(i = p, j = k))
158 contadorsito = contadorsito + 1
159 e = e + 1
160
161 f.write("//Salidas (I):" + '\n')
162 contadorsito2 = 0
163 for i in sincoroutput:
164 f.write("B4ISH1025_NS OUT" + f"{contadorsito2:000}" +
      B4ISH1025_template_output.format(i = i))
165 contadorsito2 = contadorsito2 + 1
166
167 e = 0
168 for c in solocoroutput:
169 k = int(c[c.index(':') + 1:c.index(']')])
170 j = int(c[c.index('[') + 1:c.index(':')])
171 p = solocoroutput1[e]
172 for k in range(j+1):
173 f.write("B4ISH1025_NS OUT" + f"{contadorsito2:000}" +
      B4ISH1025_template_output_channels.format(i = p, j = k))
174 contadorsito2 = contadorsito2 + 1
175 e = e + 1
176
177
178 f.write("//Pines de VDD y VSS")
179 f.write('\n' + "VDD_NS PVDD1(.VDD(VDD));"+'\n')
180 f.write("VSS_NS PVSS1(.VSS(VSS));"+'\n')
181
182 f.write("\n"+"endmodule")
183 f.close()
```

16.2. Síntesis lógica en librerías de Synopsys

DV_chip.script

```
lappend search_path cd ../../Librerias/lib_rvt/
    saed14nm_rvt_1p9m/db_nldm
lappend search_path cd ../../Librerias/io_std/db_nldm

set link_library " * saed14rvt_tt0p8v25c.db"
set target_library  "saed14rvt_tt0p8v25c.db"

read_file -format verilog {./verilog/not_gate.v}

reset_design
set_max_area 0
link
uniquify
create_clock clk -period 40 -waveform {0 20}

check_design

compile -map_effort high -area_effort high

report_area
report_design
report_power

write -format ddc -h -o ./Outputs/outchip.ddc
write -hierarchy -format verilog -output ./Outputs/outchip.v
write_sdc ./Outputs/outchip.sdc

exit
```

16.3. Síntesis lógica en librerías de TSMC

DV_chip.script

```
lappend search_path cd ../../Librerias/lib/db_nldm
set link_library " * tcb018gbwp7ttc.db tcb018gbwp7twc.db
    tcb018gbwp7tbc.db tpd018nvtc.db"
set target_library "tcb018gbwp7ttc.db"

read_file -format verilog {verilog/circuito.v}

#read_file -format verilog {verilog/pong/pong_neopixel.v
    verilog/pong/debounce.v verilog/pong/slow_clock.v verilog/
    pong/wave_shape.v}
reset_design
set_max_area 0
link
uniquify
```

```
create_clock clk -period 40 -waveform {0 20}

check_design

compile -map_effort high -area_effort high

report_area
report_design
report_power

write -format ddc -h -o ./Outputs/outchip.ddc
write -hierarchy -format verilog -output ./Outputs/outchip.v
write_sdc ./Outputs/outchip.sdc

exit
```

16.4. Síntesis física en librerías de Synopsys

sintesis_fisica_top.tcl

```
# Script para librerias SAED 14nm
source scripts/setup.tcl -echo

read_verilog Inputs/outchip.v
read_sdc Inputs/outchip.sdc

remove_pg_strategies -all
# Scenario Setup
create_corner "worst"
read_parasitic_tech  -tlup ${TLUPLUS_MAX_FILE} -layermap ${
    MAP_FILE} -name maxTLU
set_parasitic_parameters -early_spec maxTLU -late_spec maxTLU
     -corner "worst"

set_process_label "nominal"
redirect -file ./connect_pg.rpt {connect_pg_net -auto -
    verbose}
create_mode functional
create_scenario -mode "functional" -name "functional_worst" -
    corner worst
report_pvt

set_msg -limit 1 CSTR-021
read_sdc Inputs/outchip.sdc

source scripts/floorplan.tcl
#check_physical_constraints
#save_block -as dp_done
#return
#.3
set_app_options -name place.coarse.
    continue_on_missing_scandef -value true
```

```
28 place_opt
29 report_timing > logs/top.timing.placement.rpt
30 report_congestion -mode summary
31
32 #save_block -as post_place_opt
33 #return
34 # remove_clock_uncertainty [all_clocks]
35 #clock_opt
36 #save_block -as post_CTS
37 #return
38
39 route_auto -save_after_global_route true -
       save_after_track_assignment true -save_after_detail_route
       true
40 optimize_routes -reroute_all_shapes_in_nets true
41 #save_block -as post_route_opt
42
43 #.4a
44 create_stdcell_fillers -lib_cells [get_lib_cells *_FILL_*]
45
46 connect_pg_net -automatic
47 remove_stdcell_fillers_with_violation
48 check_legality
49
50 # Configuramos el DRC runset file
51 set_app_options -list {signoff.check_design.run_dir {./
       Outputs/DRC/}}
52 set_app_options -list {signoff.check_design.runset {../../
       Librerias/Runset/saed14nm_1p9m_drc_rules.rs}}
53
54 set_app_options -list {signoff.check_drc.run_dir {./Outputs/
       DRC/}}
55 set_app_options -list {signoff.check_drc.runset {../../
       Librerias/Runset/saed14nm_1p9m_drc_rules.rs}}
56
57 set_app_options -list {signoff.fix_drc.run_dir {./Outputs/DRC
       /}}
58
59
60 set_app_options -list {signoff.create_metal_fill.run_dir {./
       Outputs/Fill/}}
61 set_app_options -list {signoff.create_metal_fill.runset
       {../../Librerias/Runset/saed14nm_1p9m_mfill_rules.rs}}
62
63 #Guardamos el bloque y corremos DRC y su Fix
64 save_block EL_GRAN_JAGUAR:chip_SP
65
66 signoff_check_drc
67 signoff_fix_drc
68
69 save_block El_GRAN_JAGUAR:full_adder
70
71 signoff_check_drc
72 signoff_fix_drc
```

```
signoff_create_metal_fill
signoff_fix_drc
signoff_check_drc

check_lvs

save_block El_GRAN_JAGUAR:full_adder

#Creamos el verilog equivalente de la sintesis fisica
write_verilog -include all ../Outputs/IO/EL_GRAN_JAGUAR.v
#gds
write_gds -library EL_GRAN_JAGUAR -design AND_OR_XOR_gate -
    view design -hierarchy all -lib_cell_view frame ./Outputs/
    chip.gds
#exit


icv -i chip.gds -c ../../../../Librerias/Runset/
    saed14nm_1p9m_ant_drc_rules.rs
icv -i chip.gds -c chip_IO -sf ICV -vue ../../../../Librerias
    /Runset/saed14nm_1p9m_ant_drc_rules.rs
```

floorplan.tcl

```
initialize_floorplan -site_def unit -use_site_row -keep_all -
    side_length {2 2} -core_offset {0.2}
#initialize_floorplan -site_def unit -use_site_row -keep_all
    -side_length {285 285} -core_offset {125}

place_pins  -self

resolve_pg_nets

create_pg_std_cell_conn_pattern std_pattern_1 -layers {M1}

set_pg_strategy  pg_std_cell -core  -pattern {{pattern:
    std_pattern_1 }{nets: {VSS VDD}} }

set_app_options -name plan.pgroute.ignore_signal_route -value
     true
compile_pg

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
```

setup.tcl

```
set DESIGN_REF_PATH "../../Librerias"
```

```
2
3
4 set LIBRARY_FILES " \
5 saed14rvt_tt0p8v25c.db \
6 "
7
8 set MW_REFERENCE_LIB_DIRS  " \
9 ${DESIGN_REF_PATH}/lib_rvt/saed14nm_rvt_1p9m/mw/
      saed14nm_rvt_1p9m
10 "
11
12 set NDM_REFERENCE_LIB_DIRS  " \
13 ${DESIGN_REF_PATH}/lib_rvt/saed14nm_rvt_1p9m/ndm/
      saed14rvt_frame_timing_ccs.ndm"
14 set TECH_FILE        "${DESIGN_REF_PATH}/tech/milkyway/
      saed14nm_1p9m_mw.tf"
15 set MAP_FILE         "${DESIGN_REF_PATH}/tech/star_rcxt/
      saed14nm_tf_itf_tluplus.map"
16 set TLUPLUS_MAX_FILE "${DESIGN_REF_PATH}/tech/star_rcxt/
      saed14nm_1p9m_Cmax.tluplus"
17 set TLUPLUS_MIN_FILE "${DESIGN_REF_PATH}/tech/star_rcxt/
      saed14nm_1p9m_Cmin.tluplus"
18
19 set MW_POWER_NET                "VDD" ;#
20 set MW_POWER_PORT               "VDD" ;#
21 set MW_GROUND_NET               "VSS" ;#
22 set MW_GROUND_PORT              "VSS" ;#
23
24 set MIN_ROUTING_LAYER            "M2"   ;# Min routing layer
25 set MAX_ROUTING_LAYER            "M8"   ;# Max routing layer
26
27 set LIBRARY_FILES "${NDM_REFERENCE_LIB_DIRS}"
28 lappend search_path "${DESIGN_REF_PATH}/lib_rvt/
      saed14nm_rvt_1p9m/db_nldm"
29
30 set_host_options -max_cores 8
31 set ndm_reference_library ${NDM_REFERENCE_LIB_DIRS}
32 set ndm_design_library EL_GRAN_JAGUAR
33 set command_log_file "./logs/command.log"
34 sh rm -rf $ndm_design_library
35 create_lib -technology $TECH_FILE -ref_libs $LIBRARY_FILES ${
      ndm_design_library}
```

16.5. Síntesis física en librerías de TSMC

sintesis_fisica_top.tcl

```
1 source scripts/setup.tcl
2
3 #Abrimos el verilog file sintetizado
4
5 read_verilog ./Inputs/outchip.v
```

```
read_sdc -echo ./Inputs/outchip.sdc

# read_verilog ./Inputs/granjaguar.v
# read_sdc -echo ./Inputs/granjaguar.sdc

read_parasitic_tech -tlup $TLUPLUS_FILE -layermap $MAP_FILE

remove_pg_strategies -all

#antenna
source -echo -verbose "../../Librerias/Runset/
    antennaRule_018_6lm.tcl"

source scripts/floorplan.tcl

#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 {PFILLER0005 PFILLER1 PFILLER5
    PFILLER05PFILLER10 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 {./
    Outputs/DRC/}}
set_app_options -list {signoff.check_drc.run_dir {./Outputs/
    DRC/}}
set_app_options -list {signoff.fix_drc.run_dir {./Outputs/DRC
    /}}
```

```
45 set_app_options -list {signoff.create_metal_fill.run_dir {./
       Outputs/Fill/}}
46
47 set_app_options -list {signoff.check_design.runset {../../
       Librerias/Runset/ICVLM18_LM16_LM152_6M.215a_pre041518}}
48 set_app_options -list {signoff.check_drc.runset {../../
       Librerias/Runset/ICVLM18_LM16_LM152_6M.215a_pre041518}}
49
50 set_app_options -list {signoff.create_metal_fill.runset
       {../../Librerias/Runset/Dummy_Metal_ICV_0.18um.215a}}
51
52 set_app_options -list {signoff.create_metal_fill.
       fix_density_errors true}
53
54
55 #Guardamos el bloque y corremos DRC y su Fix
56 save_block EL_GRAN_JAGUAR.ndm:chip_IO
57 #write_gds -library EL_GRAN_JAGUAR.ndm -design chip_IO -view
       design -hierarchy all -lib_cell_view frame ../Outputs/IO/
       chip.gds
58 #signoff_create_metal_fill
59 signoff_check_drc
60 signoff_fix_drc
61 signoff_create_metal_fill -
       timing_preserve_setup_slack_threshold 0.01
62
63 #signoff_create_metal_fill -mode add
64 save_block EL_GRAN_JAGUAR.ndm:chip_IO
65 signoff_check_drc
66 signoff_create_metal_fill
67 signoff_fix_drc
68 signoff_check_drc
69 check_lvs
70 save_block EL_GRAN_JAGUAR.ndm:chip_IO
71
72 #Creamos el verilog equivalente de la sintesis fisica
73 write_verilog -include all ../Outputs/IO/EL_GRAN_JAGUAR.v
74
75 #ver errores gui show error data
76 #gds
77 write_gds -library EL_GRAN_JAGUAR.ndm -design chip_IO -view
       design -hierarchy all -lib_cell_view frame ./Outputs/IO/
       chip.gds
78 #exit
```

setup.tcl

```
1 file delete -force  EL_GRAN_JAGUAR.ndm
2 set command_log_file "./Logs/command.log"
3
4 set DESIGN_REF_PATH "../../Librerias"
5
6 set NDM_REFERENCE_LIB_DIRS  " \
```

```
${DESIGN_REF_PATH}/lib/ndm/TSMCWorkspace.ndm
"
set TECH_FILE        "${DESIGN_REF_PATH}/tech/tsmc018_6lm.tf"
set MAP_FILE         "${DESIGN_REF_PATH}/tlu_and_map/star.
    map_6M"
set TLUPLUS_FILE "${DESIGN_REF_PATH}/tlu_and_map/
    t018lo_1p6m_typical.tluplus"

set MW_POWER_NET                "VDD" ;#
set MW_POWER_PORT               "VDD" ;#
set MW_GROUND_NET               "VSS" ;#
set MW_GROUND_PORT              "VSS" ;#

set MIN_ROUTING_LAYER            "METAL1"   ;# Min routing
    layer
set MAX_ROUTING_LAYER            "METAL6"   ;# Max routing
    layer

set LIBRARY_FILES "${NDM_REFERENCE_LIB_DIRS}"
lappend search_path "${DESIGN_REF_PATH}/lib/db_nldm"

set_host_options -max_cores 12
set ndm_reference_library ${NDM_REFERENCE_LIB_DIRS}
set ndm_design_library EL_GRAN_JAGUAR
sh rm -rf $ndm_design_library
create_lib -technology $TECH_FILE -ref_libs $LIBRARY_FILES ${
    ndm_design_library}
```

floorplan.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_JAGUAR -corner_height 115
```

```
23
24 #Coloca los pines de entradas y salidas (Pads) en un lugar
       arbitrario de no ser especificado en el floorplan
25 add_to_io_guide [get_io_guides anillo_IO_JAGUAR.left] PVDD*
26 add_to_io_guide [get_io_guides anillo_IO_JAGUAR.right] PVSS*
27 place_io
28
29 #Creacion del anillo de PG
30 create_pg_ring_pattern ring_pattern -horizontal_layer METAL3
         -horizontal_width {2} -horizontal_spacing {2}    -
       vertical_layer METAL2 -vertical_width {2} -
       vertical_spacing {2}
31 set_pg_strategy core_ring    -pattern {{name: ring_pattern}
        {nets: {VDD VSS}} {offset: {1 1}}} -core
32 compile_pg -strategies core_ring
33
34 # Conexion del anillo IO con los pads de alimentacion
35 # La conexion no se esta realizando exitosamente con VDD por
       lo que se realiza de manera manual
36 create_pg_macro_conn_pattern hm_pattern -pin_conn_type
       scattered_pin -layers {METAL2 METAL3} -nets {VDD VSS} -
       pin_layers {METAL2}
37 set_app_options -name plan.pgroute.treat_pad_as_macro -value
       true
38 set_pg_strategy macro_conn -macros [get_cells {PVDD* PVSS*}]
       -pattern {{name: hm_pattern} {nets: {VDD VSS}}}
39 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}}}
40 compile_pg -strategies macro_conn -via_rule
       macro_conn_via_rule -tag test
41
42
43
44 connect_pg_net -automatic
45 create_pg_mesh_pattern mesh_pattern -layers {{{vertical_layer
       : METAL4} {width: 4} {spacing: minimum} {pitch: 42} {trim:
        true} }} -via_rule {}
46
47 # create_pg_mesh_pattern mesh_pattern -layers {{{
       vertical_layer: METAL4} {width: 3} {spacing: minimum} {
       pitch: 42} {trim: true} }} -via_rule {}
48
49 set_pg_strategy mesh_strategy -polygon {{117.98 117.98}
       {416.480 414.240 }} -pattern {{ pattern: mesh_pattern }{
       nets: {VDD VSS}}} -blockage {macros: all}
50
51 create_pg_std_cell_conn_pattern std_cell_pattern
52 set_pg_strategy std_cell_strategy -core -pattern {{pattern:
       std_cell_pattern}{nets: {VDD VSS}}}
53 compile_pg
54
55 # 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 -congestion -effort medium -
    congestion_effort high
#create_placement -floorplan -timing_driven -congestion -
    effort high -congestion_effort high
legalize_placement
```