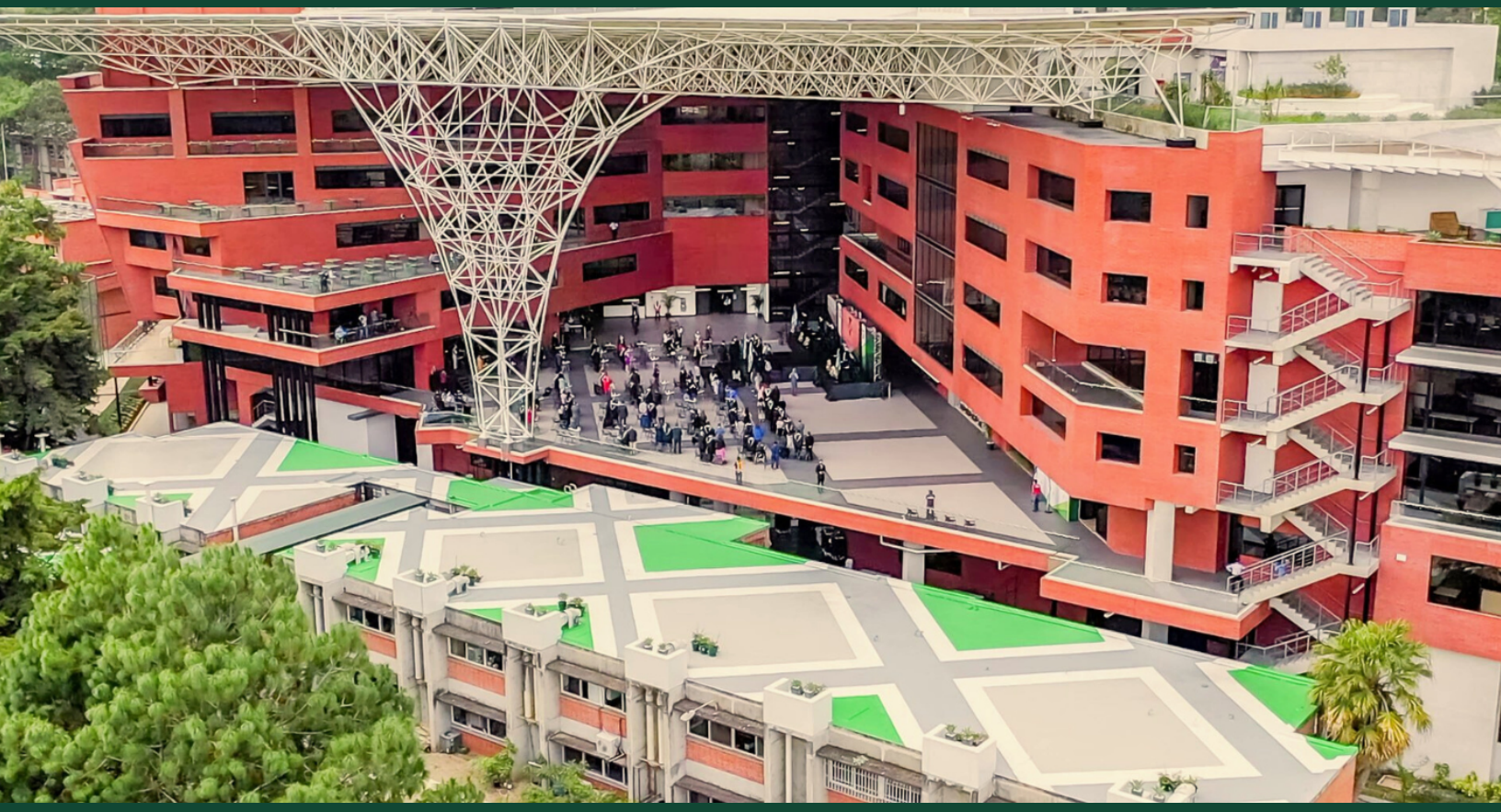
Desarrollo y documentación del proceso de síntesis física, DRC y BND para el nanochip El Gran Jaguar en tecnología de 65nm de TSMC, utilizando herramientas de Synopsys

Miguel Alberto Chacón Ortíz



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo y documentación del proceso de síntesis física, DRC y BND para el nanochip El Gran Jaguar en tecnología de 65nm de TSMC, utilizando herramientas de Synopsys

Trabajo de graduación presentado por Miguel Alberto Chacón Ortíz
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2025

Vo.Bo.:

(f) 

M.Sc. Jonathan de los Santos

(f) 

M.Sc. Carlos Esquit Hernández

Fecha de aprobación: Guatemala, 20 de noviembre de 2025.

Prefacio

Para mí, la electrónica es sinónimo de innovación y creatividad. En estos últimos años se convirtió en una parte esencial de mi vida, en la parte académica y personal. Cambió mi forma de comprender el mundo y en la que aporto soluciones en mi vida. Durante mi carrera aprendí que la ingeniería no es solo calcular, resolver circuitos y estudiar, sino también perseverar, trabajar en equipo y tener una visión hacia el futuro.

Este trabajo representa la culminación de un proceso de formación lleno de aprendizajes, desafíos y logros que no hubiera sido posible alcanzar sin el apoyo de quienes me han acompañado en este camino.

Primero, expresar mi agradecimiento a mi familia, por el apoyo y la oportunidad de poder estudiar y alcanzar mis metas, ya que en ningún momento me faltó su respaldo y confianza. A todos mis amigos, los cuales alguna vez me brindaron su ayuda, apoyo y compañía durante estos años de estudio.

Extiendo también mi gratitud a mis catedráticos de la Universidad del Valle de Guatemala, quienes con su dedicación y compromiso sembraron en mí el interés y la disciplina que requiere este área. Sus enseñanzas y orientación han sido pilares fundamentales en mi formación académica y personal.

Finalmente, a la Universidad del Valle de Guatemala, institución que me abrió las puertas y me brindó el espacio ideal para crecer académicamente y como persona, y a todos aquellos que dejaron una huella en mí en estos años.

Índice general

Prefacio I

Índice de figuras VIII

Índice de cuadros X

Resumen XI

Abstract XII

1. Introducción 1

2. Antecedentes 3

3. Justificación 5

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

5. Alcance 9

6. Marco teórico 11
6.1. El *design flow* en nanochips 11
6.2. *Frontend* 12
6.3. *Backend* 13
6.4. Verificaciones y pruebas físicas 15
6.5. Las aplicaciones de Synopsys 16
6.6. VCS 16
6.7. Design Compiler y Design Compiler NXT 16
6.8. IC Compiler II 16
6.9. IC Validator 17

6.10. Library Manager en IC Compiler II 18
6.11. Comandos utilizados en las herramientas de Synopsys 18
6.11.1. Library Manager 18
6.11.2. IC Compiler II 20
6.11.3. IC Validator 27
6.11.4. Design Compiler NXT 28
6.12. Las Reference Methodology en las herramientas de Synopsys 28

7. Jerarquía de directorios y entorno de trabajo 29
7.1. Jerarquía de directorios general 29
7.2. Jerarquía de directorios de librerías NDM y *runsets* 31
7.3. Jerarquía de directorios para la síntesis física 33
7.4. Jerarquía de directorios de metodologías de referencia 34

8. Creación de las librerías NDM con Library Manager 35
8.1. Archivos seleccionados para la creación de las librerías NDM 35
8.1.1. *Technology file* 37
8.1.2. *DB-Technology file* 38
8.1.3. LEF *files* 39
8.2. Creación de librerías 39
8.2.1. Ejecución del *script* de creación de librerías NDM 40
8.2.2. Explicación del *script* de creación de librerías NDM 40

9. Selección de *runsets* y archivos *TLU and MAP* 49
9.1. *Runset* de DRC 50
9.2. *Runset* de *antenna* 50
9.3. *Runset* de *dummy DOD/DPO fill* 50
9.4. *Runset* de *dummy metal-fill* 51
9.5. *Runset* de BND 51
9.6. Selección de *TLU and MAP files* 52
9.6.1. Archivos TLU 52
9.6.2. Archivos MAP 52

10. Modificaciones en *runsets* y archivos *TLU and MAP* 54
10.1. Modificaciones en el *runset* de DRC 54
10.2. Modificaciones en el *runset* de antena 56
10.3. Modificaciones en el *runset* de *dummy metal-fill* 56
10.4. Modificaciones en el *runset* de *dummy DOD/DPO fill* 59
10.5. Modificaciones en los archivos *TLU and MAP* 60

11. Proceso de correción de errores en síntesis física en tecnología de 65 nm 61
11.1. Correción de errores en el proceso de síntesis física en 65 nm 62
11.1.1. Primeros intentos con 6 metales 63
11.1.2. Creación del *script* de librerías y cambios en el DRC 64
11.1.3. Cambio a 9 metales y modificación del archivo MAP 64
11.1.4. Archivo mapeado de Verilog y corrección en archivo DB . . . 65

11.1.5. Cambio de *fillers* en *scripts* de síntesis física y rutas para los *runsets* . . . 65
11.1.6. Vías y *filler type* . . . 66
11.1.7. Cambios en *runset dummy metal-fill* . . . 67
11.1.8. Adición y cambios del *runset dummy DOD/DPO fill* . . . 68
11.1.9. Cursos de ICC2 y soporte de Synopsys . . . 70

12. *Scripts* del proceso de síntesis física en 65 nm **71**
12.1. *Script* `sintesis_fisica_top.tcl` . . . 71
12.1.1. Etapa 1 configuración inicial . . . 72
12.1.2. Etapa 2 síntesis del reloj . . . 72
12.1.3. Etapa 3 ruteo y verificaciones previas al *routing* . . . 73
12.1.4. Etapa 4 *routing* . . . 74
12.1.5. Etapa 5 creación de *fillers* en *core* y *I/O ring* . . . 75
12.1.6. Etapa 6 guardado, *fillers* y chequeos de DRC/LVS . . . 77
12.1.7. Etapa 7 exportación de resultados . . . 78
12.1.8. Etapa 8 finalización . . . 79
12.2. *Script* `setup.tcl` . . . 79
12.2.1. Limpieza de archivos previos . . . 79
12.2.2. Configuración de rutas y archivos base . . . 80
12.2.3. Configuración de potencia y capas . . . 81
12.2.4. Librerías de referencia . . . 81
12.2.5. Creación de la librería . . . 82
12.2.6. Lectura del diseño lógico y restricciones . . . 83
12.2.7. Configuración de parasíticos . . . 83
12.2.8. Configuración de capas y enrutamiento . . . 84
12.2.9. Variables de automatización . . . 84
12.2.10. Creación de carpetas de salida y salidas lógicas . . . 86
12.2.11. Configuración de *runsets* y rutas de ejecución . . . 86
12.3. *Script* `floorplan.tcl` . . . 87
12.3.1. Etapa 1 *floorplan* inicial . . . 88
12.3.2. Etapa 2 limpieza de estructuras previas . . . 89
12.3.3. Etapa 3 creación del anillo de *I/O* y *pads* de alimentación . . . 89
12.3.4. Etapa 4 configuración de redes de alimentación . . . 90
12.3.5. Etapa 5 creación del *power ring* (anillo de PG) . . . 91
12.3.6. Etapa 6 conexión del anillo de I/O con los *pads* de alimentación 92
12.3.7. Etapa 7 creación de la malla de *power/ground* en *core* . . . 93
12.3.8. Etapa 8 verificación y ajuste de *placement* . . . 94
12.4. Ejecución de los *scripts* . . . 95
12.5. Abrir un bloque para visualizar un diseño . . . 96

13. Circuitos sintetizados en tecnología de 65 nm **100**
13.1. *Ripple carry counter* . . . 100
13.2. Compuerta NOT . . . 103
13.3. ALU con oscilador de anillo . . . 105
13.4. Nanochip "El Gran Jaguar" . . . 106

14. Verificaciones del proceso de síntesis física de 65 nm **110**
14.1. Verificaciones de DRC 111
14.1.1. Resultados de DRC para la compuerta NOT 111
14.1.2. Resultados de DRC para la ALU 113
14.1.3. Resultados de DRC para el nanochip "El Gran Jaguar" 114
14.2. Verificaciones de antena 117
14.2.1. Resultados de antena para la compuerta NOT 118
14.2.2. Resultados de antena para la ALU 119
14.2.3. Resultados de antena para el nanochip "El Gran Jaguar" . . . 120
14.3. Verificaciones de BND 121

15. Uso de la Reference Methodology para IC Compiler II y Library Manager **123**
15.1. Encontrando la Reference Methodology para la herramienta de nuestro interés 124
15.2. RM para Library Manager de IC Compiler II 126
15.2.1. Descarga de la RM para Library Manager 126
15.2.2. Estructura de la RM para Library Manager 126
15.2.3. Uso de la RM para Library Manager 127
15.3. RM para IC Compiler II 128
15.3.1. Descarga de la RM para IC Compiler II 128
15.3.2. Estructura de la RM para IC Compiler II 129
15.3.3. Uso de la RM para IC Compiler II 129

16. Conclusiones **132**

17. Recomendaciones **134**

18. Referencias **136**

19. Anexos **140**
19.1. Replicación del flujo en 180 nm 140

20. Glosario **142**

Índice de figuras

1. Flujo de diseño para un circuito integrado 12

2. Jerarquía de directorios general del entorno de trabajo 30
3. Jerarquía de directorios para la creación de librerías NDM y *runsets* . 31
4. Jerarquía de directorios para la síntesis física 33
5. Jerarquía de directorios para el uso de las metodologías de referencia 34

6. Mensaje de finalización de creación de librerías NDM 41
7. Vista de las librerías NDM creadas 48

8. Resultado en la terminal de primeras pruebas con 6 metales en 65 nm 63
9. Compuerta NOT sintetizada de primeras pruebas con 6 metales en 65 nm 64
10. Vías faltantes en *pads* de VDD/VSS en 65 nm 66
11. Alerta de tipo de *filler* en 65 nm 66
12. Terminal con aproximadamente 28,000 errores DRC 67
13. Terminal con 20,000 errores DRC 68
14. Terminal con 10,000 errores DRC 68
15. Comando para relleno de DOD/DPO visto en el manual de ICC2 . . 69
16. Terminal con 650 errores DRC aproximadamente, después de agregar el *runset* de DOD/DPO 70

17. Directorio de síntesis física para abrir la terminal 96
18. Abrir un bloque en IC Compiler II 97
19. Seleccionar el bloque sintetizado en IC Compiler II 98
20. Seleccionar la librería de diseño en IC Compiler II 98
21. Abrir el diseño sintetizado en IC Compiler II 99

22. Diseño físico del *ripple carry counter* sintetizado en tecnología de 65 nm101
23. Anillo de entradas y salidas del *ripple carry counter* sintetizado en tecnología de 65 nm 102
24. Diseño físico del *ripple carry counter* sintetizado en tecnología de 65 nm, mostrando las celdas estándar 102

25. Diseño físico de la compuerta NOT sintetizada en tecnología de 65 nm 103
26. Anillo de entradas y salidas de la compuerta NOT sintetizada en tecnología de 65 nm . . . 104
27. Diseño físico de la compuerta NOT sintetizada en tecnología de 65 nm 104
28. Diseño físico de la ALU con oscilador de anillo sintetizada en tecnología de 65 nm . . . 105
29. Celdas y anillo de entradas y salidas de la ALU con oscilador de anillo sintetizada en tecnología de 65 nm . . . 106
30. Diseño físico del nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm . . . 107
31. Anillo de entradas y salidas del nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm . . . 108
32. Diseño físico del nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm, mostrando las celdas estándar, relleno y enrutamiento . . . 109

33. Resultados de DRC para la compuerta NOT sintetizada en tecnología de 65 nm . . . 112
34. Resumen con detalle de los errores de DRC para la compuerta NOT sintetizada en tecnología de 65 nm . . . 112
35. Resultados de DRC para la ALU con oscilador de anillo sintetizada en tecnología de 65 nm . . . 113
36. Resumen con detalle de los errores de DRC para la ALU con oscilador de anillo sintetizada en tecnología de 65 nm . . . 114
37. Resultados de DRC para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm . . . 115
38. Resumen con detalle de los errores de DRC para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm . . . 116
39. Resumen de resultados con la cantidad total de errores de DRC para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm . . . 117
40. Resultados de antena para la compuerta NOT sintetizada en tecnología de 65 nm . . . 119
41. Resultados de antena para la ALU con oscilador de anillo sintetizada en tecnología de 65 nm . . . 120
42. Resultados de antena para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm . . . 121

43. Página de SolvNet Plus para la descarga de la Reference Methodology de IC Compiler II y su versión de 2024 . . . 124
44. Descarga de la Reference Methodology desde la interfaz gráfica de IC Compiler II . . . 125
45. Selección del nodo en la versión de 2020 de IC Compiler II . . . 125
46. Estructura de la RM para Library Manager . . . 127
47. Tipos de fuentes para la creación de librerías en Library Manager . . . 128
48. Estructura de la RM para IC Compiler II . . . 129
49. Vista de la interfaz gráfica de la RM para IC Compiler II . . . 130

50. Figura de un *layout* de una compuerta NOT en tecnología de 180 nm 141

51. Figura del *layout* del nanochip en tecnología de 180 nm 141

Índice de cuadros

8.1. Significado de extensiones de archivos 36
8.2. Configuración inicial del *script* de librerías 42
8.3. Creación del Standard Workspace del *script* de librerías 43
8.4. Creación del Fillers Workspace del *script* de librerías 44
8.5. Creación del Pads Workspace del *script* de librerías 45
8.6. Creación del Corners Workspace del *script* de librerías 46
8.7. Creación del TSMC Workspace del *script* de librerías 47

10.1. *Script* modificado para mapeo, `star.map_9M` 60

12.1. Configuración inicial de `sintesis_fisica_top.tcl` 72
12.2. Etapa del reloj de `sintesis_fisica_top.tcl` 73
12.3. Etapa de ruteo y verificaciones previas al *routing* de `sintesis_fisica_top.tcl` 74
12.4. Etapa de *routing* `sintesis_fisica_top.tcl` 75
12.5. Etapa de creación de *fillers* `sintesis_fisica_top.tcl` 76
12.6. Etapa de guardado y chequeos de DRC/LVS `sintesis_fisica_top.tcl` 77
12.7. Etapa de exportación de resultados `sintesis_fisica_top.tcl` . . . 78
12.8. Etapa de finalización `sintesis_fisica_top.tcl` 79
12.9. Limpieza de archivos previos en `setup.tcl` 80
12.10. Configuración de rutas y archivos base en `setup.tcl` 80
12.11. Configuración de potencia y capas en `setup.tcl` 81
12.12. Configuración de librerías de referencia en `setup.tcl` 82
12.13. Creación de la librería en `setup.tcl` 82
12.14. Lectura del diseño lógico y restricciones en `setup.tcl` 83
12.15. Configuración de parasíticos en `setup.tcl` 84
12.16. Configuración de capas y enrutamiento en `setup.tcl` 84
12.17. Variables de automatización en `setup.tcl` 85
12.18. Creación de carpetas de salida e inputs lógicos en `setup.tcl` 86
12.19. Configuración de *runsets* y rutas de ejecución en `setup.tcl` 87
12.20. Etapa de *floorplan* inicial en `floorplan.tcl` 88
12.21. Etapa de limpieza de estructuras previas en `floorplan.tcl` 89

12.22. Etapa de creación del anillo de I/O y *pads* de alimentación en `floorplan.tcl` 90
12.23. Etapa de configuración de redes de alimentación en `floorplan.tcl` . 91
12.24. Etapa de creación del *power ring* en `floorplan.tcl` 92
12.25. Etapa de conexión del anillo IO con los pads de alimentación en `floorplan.tcl` 93
12.26. Etapa de creación de la malla de *power/ground* en `floorplan.tcl` . 94
12.27. Etapa de verificación y ajuste de *placement* en `floorplan.tcl` . . . 95

14.1. Contenido del *script* de verificación de antena `antenna.tcl` 118
14.2. Ejemplo de comandos para la verificación de BND en un proceso de tipo *wire-bond* 122

Resumen

El diseño de circuitos nanométricos es el motor que impulsa la vida moderna. Este trabajo de graduación aborda la adaptación a la tecnología de 65 nm para la síntesis física de circuitos integrados digitales, en especial del nanochip "El Gran Jaguar", un proyecto que explora una nueva área en Guatemala, el diseño VLSI.

Se enfatizó la importancia de una jerarquía en el entorno de trabajo y un cambio ordenado de 180 nm a 65 nm, estructurado por las características y fases de diseño, facilitando la creación de librerías y la síntesis física. El cuidado de la jerarquía y su entorno fue posible siguiente las recomendaciones de las herramientas y los manuales. Se abarca la selección de archivos basada en las recomendaciones de IMEC. También, la modificación de *runsets* (DRC, BEOL, FEOL, antena) y uso de *scripts* en el flujo de diseño, optimizando tiempo y recursos en la síntesis física.

El trabajo también incluyó las verificaciones físicas de los diseños con el fin de minimizar los errores la más posible. El gran desafío consistió en hacer la transferencia a la tecnología de 65 nm y adaptar el flujo al recomendado por Synopsys, ayudándonos a obtener mejores resultados. Para esto, se utilizó IC Compiler II y IC Validator, logrando una reducción de más del 98 % en los errores de DRC y resultados satisfactorios en las verificaciones de antena. Finalmente, la metodología de referencia de Synopsys para la herramienta IC Compiler II ayudó a optimizar etapas desde la creación de librerías hasta la exportación de archivos del diseño sintetizado. Explorando así una nueva alternativa para la síntesis de circuitos para futuros investigadores dentro de la universidad.

Como trabajo futuro, se recomienda seguir enfatizando la importancia del orden y la jerarquía en el entorno de trabajo, así como la documentación detallada de cada paso en el proceso de síntesis física. Además, la comunicación con el equipo de diseño, Synopsys y IMEC es crucial para corregir los errores restantes y fabricar el nanochip con éxito.

Palabras clave: *circuit design*, Synopsys, DRC, IC Compiler II.

Abstract

Nanometric circuit design is the driving force behind modern life. This graduation project addresses the adaptation to 65 nm technology for the physical synthesis of digital integrated circuits, particularly the "El Gran Jaguar" nanochip, a project that explores a new area in Guatemala: VLSI design.

The importance of a hierarchy in the work environment and an orderly change from 180 nm to 65 nm, structured by design characteristics and phases, was emphasized, facilitating the creation of libraries and physical synthesis. Caring for the hierarchy and its environment was possible by following the recommendations in the tools and manuals. It covers file selection based on IMEC recommendations. It also covers the modification of runsets and the use of scripts in the design flow, optimizing time and resources in physical synthesis.

The work also included physical verification of the designs in order to minimize errors as much as possible. The big challenge was to make the transfer to 65 nm technology and adapt the flow to that recommended by Synopsys, helping us to achieve better results. For this, IC Compiler II and IC Validator were used, achieving a reduction of more than 98 % in DRC errors and satisfactory results in antenna verifications. Finally, Synopsys' reference methodology for the IC Compiler II tool helped optimize stages from library creation to the export of synthesized design files. This explored a new alternative for circuit synthesis for future researchers within the university.

As future work, it is recommended to continue emphasizing the importance of order and hierarchy in the work environment, as well as detailed documentation of each step in the physical synthesis process. In addition, communication with the design team, Synopsys, and IMEC is crucial to correcting remaining errors and successfully manufacturing the nanochip.

Keywords: circuit design, Synopsys, DRC, IC Compiler II.

CAPÍTULO 1

Introducción

El uso de dispositivos electrónicos es cada vez más común en la vida diaria, dependemos de la constante optimización de ellos para la mejora constante de nuestras actividades. Esta optimización se logra a través del diseño e implementación de circuitos integrados que permitan cumplir con las necesidades actuales de la sociedad. En la Universidad del Valle de Guatemala, se busca formar profesionales capaces de contribuir al desarrollo tecnológico mediante la creación de soluciones innovadoras en el campo de la ingeniería electrónica, específicamente en el diseño de circuitos integrados utilizando herramientas profesionales.

El trabajo realizado corresponde a la adaptación a una nueva tecnología proporcionada por TSMC a través de IMEC, la tecnología CMOS de 65 nm, para la síntesis física de circuitos integrados digitales. Pasando de 180 nm a esta nueva tecnología, se presentaron diversos retos y desafíos que requirieron la investigación y aplicación de nuevas metodologías y herramientas. Por otro lado, la búsqueda de la optimización en los flujos de diseño y de nueva información que permitiera la correcta implementación de los circuitos integrados en esta tecnología fueron aspectos clave para el desarrollo del proyecto.

La correcta implementación de una jerarquía lógica y ordenada ayuda al entorno de trabajo, tanto para la creación de librerías como para la síntesis física de los diseños realizados, junto con la selección de los archivos y modificaciones apropiadas para cada etapa del flujo de diseño así como se explica en los capítulos 7 hasta el 10.

Entender el uso de los *scripts*, sus diferentes etapas y la correcta aplicación de los mismos en el flujo de diseño es esencial para la optimización del tiempo y recursos en la síntesis física de circuitos integrados digitales y sus resultados, como se detalla en los capítulos 12 y 13.

Las verificaciones físicas de los diseños son un aspecto crucial en el proceso de síntesis física, ya que garantizan que los circuitos cumplan con las especificaciones

y restricciones de la tecnología utilizada. En el capítulo 14, se abordan las diferentes pruebas y verificaciones realizadas, así como los resultados obtenidos y la mejora evidente en la corrección de errores de DRC y las pruebas satisfactorias en las verificaciones de antena.

Como último punto, el uso de metodologías de referencia proporcionadas por Synopsys, y específicas para la herramienta IC Compiler II, permitió una guía estructurada para la realización de las diferentes etapas del flujo de diseño VLSI, como se explica en el capítulo 15.

CAPÍTULO 2

Antecedentes

En la Universidad del Valle de Guatemala se promueve el compromiso con la innovación tecnológica y desarrollo de soluciones que tengan impacto en la sociedad. Esto se refleja en múltiples iniciativas lideradas por el Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica, donde se ha promovido activamente la formación en tecnologías de vanguardia.

Desde el año 2009, el M.Sc. Carlos Esquit, actual director de la carrera, ha impulsado cambios para modernizar y divulgar los conocimientos adquiridos en su experiencia académica en el extranjero, incorporando temáticas avanzadas como la tecnología nanométrica. En el año 2013, se introdujo el curso de y "Introducción al Diseño de Sistemas VLSI" (*very large scale integration*), dando paso al diseño de circuitos digitales en escalas micro y nanométricas. Un año más tarde, en el 2014, se formalizó un convenio académico con Synopsys, empresa líder en soluciones para diseño de semiconductores, lo que permitió a los estudiantes acceder a herramientas profesionales para simulación y síntesis, habilitando el desarrollo de diseños más complejos. Ese mismo año, se llevó a cabo el primer trabajo de graduación orientado al diseño VLSI, en el cual se diseñó en silicio un sumador/restador de 32 bits utilizando tecnología de 28 nm [1].

Siguiendo en el 2015, se realizó una actualización curricular que incluyó los cursos de Nanoelectrónica 1 y 2, permitiendo una mayor profundización y aprendizaje de este tipo de conocimientos en un área tan innovadora. Posteriormente, en 2019, se dio inicio formal al proyecto "El Gran Jaguar", con la participación de dos estudiantes que desarrollaron trabajos de graduación orientados a definir un flujo de diseño completo para la fabricación de un circuito integrado en tecnología de 180 nm. Se propuso un esquema estructurado de diseño para circuitos integrados [2], mientras que también se presentó el diseño de una máquina de estados finitos que imprime un mensaje en codificación ASCII [3]. Ambos trabajos emplearon librerías provistas por TSMC, empresa líder en manufactura de semiconductores. Además, se estableció un acuerdo

con el Interuniversity Microelectronics Centre (IMEC) para la fabricación del chip, gracias a su alianza con TSMC.

En 2020, el equipo de trabajo del proyecto se expandió, y se documentaron exhaustivamente las etapas de verificación física, incluyendo *design rule check*, una etapa esencial para verificar el correcto diseño, *Electrical Rule Check* (ERC) [4], *antenna rule check* [5] y *layout vs schematic* (LVS) [6], empleando herramientas como IC Compiler I e IC Validator, las cuales ayudaron a verificar la validez física de los *layouts*. También se dio inicio a la automatización del proceso de verificación de archivos HDL mediante la herramienta VCS [7]. En 2021, se identificó la existencia de la herramienta IC Compiler II [8] y se propuso migrar el flujo de síntesis física [9] hacia ella en la que se llevaron a cabo las verificaciones antes mencionadas [10], logrando corregir errores en las pruebas ERC y de Antena [11]. No obstante, persistieron seis errores de densidad en la verificación DRC. Por lo que, se utilizó IC Compiler para realizar el *placement* y *routing* del circuito por medio de comandos específicos [12], al mismo tiempo se ejecutó la síntesis lógica utilizando instrucciones [13] determinadas anteriormente en otro trabajo [14].

Durante el 2022, se retomó el proyecto con el objetivo de replicar y optimizar las fases avanzadas desarrolladas en ciclos anteriores. Esto incluyó la automatización de la síntesis lógica [15] y física [16], así como una reducción de los errores de densidad de seis a dos. Además, se documentaron alternativas al flujo de diseño tradicional [17] y se automatizó el proceso de verificación física, incluyendo pruebas DRC, ERC, LVS y antena [18]. En otro trabajo se logró automatizar la descarga e instalación de las herramientas de la compañía Synopsys, facilitando el proceso de configuración para futuras generaciones [19].

Continuando, en el año 2023 se realizaron mejoras al flujo de diseño, simplificando y mejorando la jerarquía de directorios utilizada tanto para la síntesis lógica como para la física [20]. También se replicó el flujo con estas mejoras, utilizando librerías proporcionadas por Synopsys. En esta etapa se trabajó intensamente en la corrección de errores de densidad restantes, que estaban asociados con la insuficiente cobertura de metal en ciertas capas. Aunque no se logró resolver del todo, una reducción significativa del error mediante un *runset* de relleno metálico proporcionado por TSMC ocurrió [20].

Llegando al año pasado, 2024, se lograron avances en las etapas de síntesis lógica al automatizarla y también los procesos de *back-end*, junto con la instalación automatizada de las herramientas de Synopsys [21]. Se logró la siguiente fase: síntesis lógica junto con las verificaciones DRC y de Antena [22]. Y se hicieron las pruebas LVS, ERC y extracción de parásitos [23]. Sin embargo, no se logró cumplir la densidad requerida del 30 por ciento para el diseño del chip "El Gran Jaguar".

CAPÍTULO 3

Justificación

A través del tiempo, se ha buscado constantemente mejorar la eficiencia, velocidad y tamaño de los dispositivos electrónicos, teniendo como eje central el diseño de circuitos integrados. Este enfoque ha hecho posible que tecnologías como los teléfonos inteligentes, los satélites y los sistemas médicos se integren de forma natural en nuestra vida diaria. En el corazón de estos avances se encuentra una serie de procesos altamente especializados que permiten transformar una idea funcional en un chip real: el diseño VLSI.

El dominio de este flujo no solo representa un reto técnico, sino también una oportunidad para el desarrollo tecnológico de países como Guatemala. Proyectos como "El Gran Jaguar", que buscan diseñar el primer nanochip guatemalteco, sientan un precedente importante en la región. A través del uso de herramientas de nivel industrial, se establece un puente entre el conocimiento académico y su aplicación práctica. De esta manera, se demuestra que el país no solo puede participar en la industria global de semiconductores, sino también liderar iniciativas que aporten al crecimiento de su soberanía tecnológica.

El presente proyecto busca profundizar en cada etapa del flujo de diseño digital: desde la síntesis lógica hasta las verificaciones físicas. Esta experiencia no solo contribuye al desarrollo de habilidades técnicas, sino que también fortalece la capacidad de análisis, resolución de problemas y trabajo colaborativo en un entorno de alta complejidad, propio de la industria de semiconductores. Añadiendo el entendimiento de las metodologías de referencia para el diseño de circuitos integrados. Realizando estas actividades tanto en tecnología de 180 nm como de 65 nm utilizando como herramienta principal ICC2 de Synopsys.

Finalmente, se elaborará una documentación estructurada y clara que recoja los aprendizajes, errores y soluciones encontradas durante el proceso. Este esfuerzo busca preservar el conocimiento adquirido y permitir que futuras generaciones puedan continuar con el proyecto sin partir desde cero. Así, no solo se garantiza la continuidad del

trabajo, sino que se contribuye activamente al crecimiento del ecosistema académico y tecnológico del país, consolidando un camino hacia una Guatemala más innovadora, autónoma y preparada para los desafíos del futuro.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Explorar y profundizar en la síntesis física para el diseño del nanochip "El Gran Jaguar" utilizando herramientas de Synopsys con tecnología de 65nm, además de documentar cada etapa del proceso para acortar la curva de aprendizaje de futuros estudiantes o investigadores que deseen replicar el proceso.

4.2. Objetivos específicos

- Replicar los procesos de diseño del nanochip de años pasados para que se aprendan los conceptos, procesos y detalles de la fases de síntesis lógica, física, y sus respectivas verificaciones físicas utilizando tecnología de 180 nm.
- Efectuar la síntesis física para el nanochip "El Gran Jaguar" en tecnología de 65 nm.
- Realizar las pruebas de DRC para el nanochip "El Gran Jaguar" y documentar los resultados de las pruebas con el fin de disminuir lo más posible los errores.
- Realizar las pruebas de antena para el nanochip "El Gran Jaguar" y documentar los resultados de las pruebas con el fin de disminuir lo más posible los errores.
- Explorar la prueba de BND para su futura implementación en el flujo de diseño del nanochip "El Gran Jaguar".
- Documentar y hacer una guía acerca de la síntesis física y verificaciones físicas para que próximas generaciones puedan replicar el proceso.

- Conocer y documentar el uso de la metodología de referencia de Synopsys para optimizar el proceso de síntesis física y el flujo de diseño.

CAPÍTULO 5

Alcance

El presente trabajo de graduación posee como alcance la ejecución de la síntesis física y las verificaciones pertinentes (DRC, *antenna* y BND) de un circuito digital, en este caso el "El Gran Jaguar" en tecnología de 65 nm de TSMC con herramientas de la empresa Synopsys, específicamente de IC Compiler II y IC Validator. Este proceso se realizará partiendo de los avances previos documentados por las generaciones anteriores. El enfoque se centra en alcanzar un diseño físicamente verificado, sin que esto implique necesariamente el cumplimiento total de los requisitos de manufacturabilidad por parte de TSMC o validaciones de IMEC.

Las actividades están limitadas a los recursos disponibles dentro del entorno académico universitario, lo que implica el acceso actual a librerías, herramientas y documentación técnica proporcionados por TSMC y Synopsys a la Universidad del Valle de Guatemala. El trabajo se llevará a cabo en un entorno de laboratorio, en el que no se contempla el recibimiento del diseño fabricado, sus verificaciones postmanufactura o algún proceso posterior en caso del envío del diseño final.

El trabajo no garantiza la eliminación total de todos los errores de diseño o violaciones que puedan surgir durante el proceso de verificación física, aunque se hará el mayor esfuerzo por minimizarlos en función del conocimiento técnico y apoyo disponible. En caso de que los requerimientos de DRC o antena y la exploración de BND no se cumplan, se documentarán el proceso de identificación, las limitaciones encontradas, intentos de corrección y recomendaciones para futuras generaciones.

Por otro lado, para las futuras generaciones se elaborará un manual que sea una guía del proceso a seguir junto con detalles acerca de los archivos, herramientas y *scripts* utilizados durante la síntesis física y validaciones. Las actividades de este trabajo de graduación se harán durante el período de enero a noviembre de 2025, bajo un enfoque práctico que prioriza la documentación, replicabilidad y la transferencia de conocimientos, antes que la manufactura física del diseño. Posibles retrasos en la comunicación, y transferencia de archivos por parte de IMEC pueden afectar al

trabajo.

CAPÍTULO 6

Marco teórico

6.1. El *design flow* en nanochips

El objetivo de tener un flujo de diseño o *design flow* es tener una serie de pasos aplicables al diseño de algo en específico, para el caso de un circuito integrado existe también una serie de pasos. Esta resulta ser de gran ayuda al funcionar como una guía general para el proceso de diseño, que resulte en la fabricación del propio chip. Un circuito integrado (IC) es un dispositivo que puede llegar a tener cientos de millones de transistores y por eso se apoya en el proceso llamado *very-large-scale-integration* (VLSI) el cual de por sí tiene un flujo de diseño que resulta en la creación del IC, empezando con el diseño, pruebas, simulaciones, verificaciones y por último la fabricación [24].

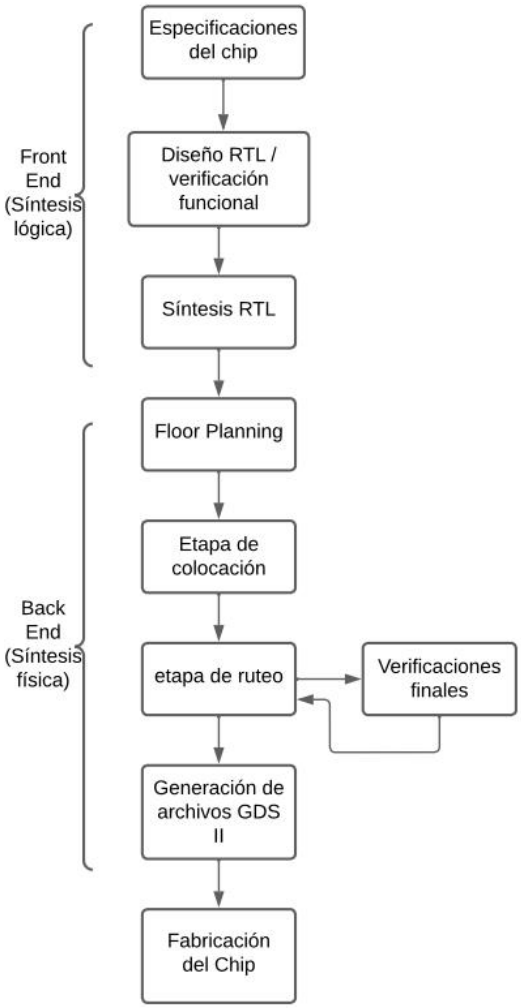
Como puede verse en la Figura 1, el flujo de diseño VLSI empieza por definir las especificaciones de nuestro dispositivo o sistema, aquí vamos a definir cuestiones como potencia, tamaño, área, tiempo o tecnología. Mediante algún HDL como Verilog o VHDL (RTL) podemos proceder al diseño del esquemático, definiendo las interconexiones del IC de forma lógica con módulos como se hace en los lenguajes descriptores de hardware. Con el diseño del circuito en un HDL podemos correr simulaciones para verificar que la fase de diseño RTL tiene el comportamiento lógico adecuado [25].

Al haber hecho estas pruebas, el diseño del circuito es efectuado. El diseño del circuito se efectúa en un proceso conocido como síntesis física o *place and route* [26] en donde se pasa la lógica a componentes físicos. Se obtiene el *layout* del circuito, el cual representa de forma física o geométrica de la lógica descrita mediante HDL. Posteriormente, se hacen verificaciones físicas mediante simulaciones, y al determinar que todas las pruebas fueron un éxito, el diseño se puede enviar al fabricante para pasar por los procesos requeridos y así tener el chip. Una vez se haga el chip en físico, se pone a prueba y se verifica su correcto funcionamiento[27].

6.2. *Frontend*

Esta es la primera parte del diseño en donde se definen los objetivos principales, detalles de la tecnología y especificaciones del sistema. Aquí se define mediante un lenguaje descriptivo de *hardware* el funcionamiento lógico del chip y se hacen pruebas lógicas. Los pasos generales son los siguientes:

Figura 1. Flujo de diseño para un circuito integrado



Nota. Este es el flujo de diseño para el diseño de un circuito integrado, dividido en sus partes de *frontend* y *backend*, obtenido de [22].

1. Especificaciones del sistema:

   Aquí se definen las características, funciones, especificaciones (tiempo, área, tecnología, entre otras) y arquitectura.

2. Diseño RTL:

En esta sección se elabora el *register transfer level*, lo que es el diseño lógico utilizando un lenguaje de HDL como verilog o VHDL, aquí se definen las conexiones entre módulos y sus funcionalidades. Puede realizarse de forma *behavioral* o *structural*.

3. Sintesís lógica y diseño:

   Utilizando herramientas especializadas se puede traducir el circuito elaborado en el HDL a una *netlist*, el cual sirve como una representación en forma de texto que nos indica sus conexiones en términos de compuertas lógicas. Con esto podemos obtener una descripción específica de su estructura empleando componentes de las librerías directas de la tecnología y fabricante que se tienen a disposición [21].

4. Verificaciones:

   Por último, las verificaciones nos confirman la equivalencia entre el *netlist* y al sistema que se hizo por diseño RTL, verificando su funcionamiento con herramientas como VCS o Formality [22].

6.3. *Backend*

El *backend* es la segunda parte antes de la fabricación del chip como se ve en la figura 1, después de hacer la descripción de *hardware* de alto nivel del diseño y obtener el *netlist* se verifica su correcto funcionamiento el cual nos permite transformar los *outputs* obtenidos en un *layout* que será enviado para su respectiva fabricación. Este es un proceso que consiste de varios pasos los cuáles son los siguientes [28]:

1. *Partitioning:*

   El *partitioning* en el diseño VLSI de nanochips es esencial, este consiste en segmentar el circuito o *netlist* en sub-bloques o particiones. El objetivo principal es minimizar el número de interconexiones entre estas particiones, lo que permite reducir la complejidad del ruteo y facilitar la optimización en etapas posteriores, como la colocación (*placement*) y el análisis de temporización.

2. *Floor planning:*

   Es un proceso que consiste en definir y denominar la distribución de los componentes de un nanochip durante la etapa de diseño. El objetivo principal es poder establecer las dimensiones del *die*, así facilitando la asignación de espacios para los bloques, red de voltajes de poder y cumplimiento de restricciones físicas (área máxima, temporización y potencia). Esto después de la determinación y colocación eficiente de elementos como celdas estándar, macros, *pads* I/O, entre otros.

   Es un diseño que desde que se logró automatizar por *software* ya no se realiza de forma manual, ya que es muy trabajoso y extenso. Existen algoritmos que

ayudan a optimizar parámetros como la longitud de los metales (cables) necesarios para las interconexiones, obteniendo la menor cantidad de retardos posible entre módulos, disminuyendo la congestión, entre otros aspectos. Este proceso se realiza mediante el IC Compiler II de Synopsys [29].

3. *Power and ground routing:*

   El enrutamiento de *power* y *ground* toma parte en el diseño físico de nanochips y se encarga de distribuir de manera óptima las redes de alimentación (*power*) y tierra (*ground*) a lo largo del chip. Este proceso tiene como objetivo principal garantizar que todos los bloques funcionales reciban una fuente de energía estable y uniforme, minimizando caídas de tensión y evitando interferencias electromagnéticas que puedan comprometer el rendimiento y la integridad del circuito y del *die*.

4. *Placement:*

   El *placement*, es una etapa que asigna posiciones físicas específicas y es crítica en el flujo de diseño, ya que a cada una de las celdas y bloques funcionales definidos en el *netlist* se le colocará en una parte definida del *wafer*. El objetivo principal es optimizar diversos parámetros como el retardo, longitud de interconexiones, congestión y el consumo de energía. Y así asegurar el cumplimiento de las restricciones de manufactura y reglas de diseño. Este proceso se realiza mediante el ICC2 de Synopsys [29].

5. *Clock tree synthesis:*

   Esta etapa tiene un objetivo principal, el cual es distribuir el reloj de manera eficiente y equilibrada a todos los elementos que sean de tipo secuencial en un nanochip, como lo pueden ser *flip-flops* y registros. Esto se realiza mediante la minimización del *skew* (diferencia en el tiempo de llegada del reloj a distintos puntos) y la latencia, garantizando así la sincronización adecuada del sistema. Esto también ayuda a optimizar el consumo de energía y mantiene la integridad de la señal, evitando situaciones de ruido.

6. *Signal routing:*

   El *signal routing* o enrutamiento de señalas es la etapa en la que se establecen las conexiones físicas entre pines de señal de las celdas lógicas, se utilizan capas metálicas disponibles en el proceso de manufactura. Tiene que cumplir las reglas de diseño (DRC), ayuda a optimizar el rendimiento y gestiona la congestión. Este proceso se realiza mediante el ICC2 de Synopsys [29].

7. *Timing closure:*

   Es un proceso en el que se garantiza que todos los caminos de señal satisfagan las restricciones de temporización impuestas por el mismo diseño. Pueden ser tiempos de configuración, mantenimiento o el período de reloj. Ayuda a eliminar violaciones de temporización, maximizan el margen de tiempo libre conocido como *slack*; equilibra el área, potencia y rendimiento, y prepara el diseño para la fabricación.

6.4. Verificaciones y pruebas físicas

La verificación física en VLSI es el conjunto de pasos automatizados que validan la corrección eléctrica, lógica y de manufacturabilidad de un diseño físico antes de su fabricación, este diseño físico se obtiene mediante la síntesis física. Incluye algunas comprobaciones como *design rule check* (DRC), *layout versus schematic* (LVS), *electrical rule checking* (ERC), *antenna rule checking*, *parasitic extraction* [23] y *bondpad verification* (BND). Cada una de estas etapas asegura que el layout cumpla tanto con las restricciones tecnológicas del proceso como con la intención del diseño, evitando defectos costosos y garantizando un silicio funcional y fiable. Durante la verificación pueden surgir errores o *warnings* sin embargo, dependiendo del fabricante estos se pueden omitir. Esta etapa depende de herramientas de verificación como IC Validator [30]:

1. *Design rule check* (DRC):

   El *design rule check* es un proceso esencial que nos ayuda a verificar que el diseño cumple con todas las restricciones, reglas o especificaciones de diseño que el fabricante determina que es capaz de manufacturar basado en la tecnología sobre la que se trabaja. Dentro de estas reglas se encuentran: espaciamiento, capas permitidas, densidad para el pulido químico-mecánico, entre otros. Nos ayuda a determinar la violaciones realizadas en el diseño, en caso de existir, y de corregirlas para que sea fabricable.

2. *Layout versus schematic* (LVS):

   Utilizando el diseño podemos obtener un *netlist* con el cual mediante una comparación con el *netlist* original, obtenido en la fase de síntesis lógica, se determina la funcionalidad del diseño realizado. Mediante esta prueba se pueden verificar errores como cortocircuitos, abiertos o desajustes en los parámetros de dispositivos.

3. *Electrical rule checking* (ERC):

   Es una prueba que se encarga de verificar conexiones eléctricas, de alimentación, tierra, cambios en tiempos de las señales y que capacitancias que surjan estén concretamente limitadas. Con el fin de que la integridad eléctrica del diseño esté asegurada.

4. *Antenna rule checking*:

   Su objetivo es evitar que en el diseño queden posibles efectos de antena, ya que estas pueden acumular carga y dañar los *gates* de transistores. Esto puede llegar a ocurrir si algunas antenas no están conectadas al silicio [23].

5. *Bondpad verification* (BND):

   El objetivo del *bondpad verification* es asegurar la correcta definición y posicionamiento de los *bondpads*, regiones específicas metálicas del chip donde se realizan conexiones físicas entre el chip y el encapsulado. Estas verificacioes de

las conexiones realizadas con hilos metálicos determinan la integridad mecánica-eléctrica y geométrica del IC [31].

6.5. Las aplicaciones de Synopsys

Synopsis Inc. es una compañía de origen estadounidense que se especializa en la industria de semiconductores y computación, desarrolla software especializado para el diseño de circuitos integrados. El tipo de software que desarrollan se conoce como EDA *(electronic design automation)*. Esta compañía provee varias herramientas/aplicaciones que permiten el desarrollo y verificación de los diseños microelectrónicos antes de su posible fabricación. Por lo tanto, es esencial conocer los principios básicos de las herramientas de Synopsys que toman un papel importante en el proyecto [32].

6.6. VCS

Es una herramienta esencial para la simulación y verificación de la fase RTL, simulando la lógica de cada elemento descrito en Verilog, VHDL, SystemVerilog, etc. Nos ayuda a compilar el archivo de verilog en particiones de netlist con su respectivo testbench. La herramienta realiza una simulación del diseño en el que después de efectuarse puede revisarse y verificarse su correcto funcionamiento [33].

6.7. Design Compiler y Design Compiler NXT

Design Compiler y Design Compiler NXT son herramientas de síntesis lógica que permiten transformar el diseño RTL en un *netlist* optimizado a nivel de compuertas lógicas [34]. Estas herramientas nos ayudan a optimizar el diseño desde el principio para cumplir con las restricciones de área, potencia y temporización. Toman como entrada el diseño en Verilog para luego realizar la síntesis lógica y generar el *netlist* que será utilizado en las siguientes etapas del flujo de diseño VLSI.

Design Compiler NXT es una versión mejorada de Design Compiler que ofrece capacidades avanzadas de optimización y análisis. Incluye características como optimización basada en las nuevas tecnologías, para nodos más recientes en comparación con Design Compiler [35].

6.8. IC Compiler II

Esta es una herramienta que nos permite ir desde diseños básicos hasta lo más detallado posible. Partiendo de bloques hasta llegar a la misma fabricación del chip,

lo que toma de entrada es un *netlist* sintetizado (*gate-level*), restricciones de tiempo, archivos de *floorplan* y *macro placements*, librerías lógicas y físicas. Como resultado nos genera el archivo GDS con el *layout* del diseño descrito en *HDL* [29].

Los pasos clave son:

1. *Floorplanning* y *partitioning*:

   Definición de *core area*, *aspect ratio* y partición de manera jerárquica para los diseños.

2. *Power planning*:

   Generación de *power rings/mesh*, *well-taps* y asignacion de conexiones de VDD/GND en capas bajas.

3. *Placement* global y legalización:

   Minimización de *wirelength* más congestión, seguido de legalización para cumplir *pitch* y *spacing* en el diseño.

4. *Clock tree synthesis timing-driven*:

   Inserción de buffers/inversores con acceso directo al motor *PrimeTime* para optimizar *skew* e *insertion delay*.

5. Global y *detailed routing*:

   Asignación de canales y trazado exacto, con *routing timing-driven* e integración *ECO* para cambios *post-sign-off*.

6.9. IC Validator

Esta es una herramienta que nos permite realizar la verificación física de *sign-off*, comprobando que el *layout* cumple con todas las reglas de manufactura y coincide con la intención lógica del diseño. Toma como entrada los archivos de *layout* (GDS), los archivos LEF de tecnología, los *rule decks* para DRC/LVS, el *netlist* para LVS, *antenna rule checking*, *metal fill* y las restricciones de ERC. Esta prueba genera reportes detallados de violaciones y también una base de datos lista para la corrección de errores[30].

Los pasos clave son:

1. *DRC* distribuido:

   Validación de *spacing*, *width*, *density* y separación de capas.

2. *Pattern matching* y DFM:

   Detección de patrones críticos de manufactura y verificación de reglas avanzadas, con corrección automática de conflictos.

3. LVS y comparación de *netlist*:

   Extracción de la *netlist* del *layout* y comparación con el *netlist* original para asegurar equivalencia funcional, detectando cortocircuitos, circuitos abiertos o errores de parametrización.

4. ERC y *antenna checks*:

   Comprobación de integridad eléctrica (conexiones *power/ground*, *fan-out*, *slews*), verificación de cambios accidentales, y detección de problemas de carga en antenas.

5. Reporte y corrección de violaciones:

   Generación de reportes detallados y marcación de ubicaciones, para su corrección manual o automática antes de la etapa de *sign-off* final.

6.10. Library Manager en IC Compiler II

IC Compiler II es una herramienta de Synopsys que permite el diseño de circuitos integrados de manera más eficiente y automatizada, utilizando un enfoque basado en una plantilla o plantillas y reglas de diseño predefinidas. Pueden ser circuitos de tipo análogo, digital o de señal mixta. IC Compiler II posee dentro de sus herramientas a el Library Manager, el cual se enfoca en la gestión y organización de las librerías de celdas utilizadas en el diseño [29].

El gestor de librerías permite crear, editar y mantener las librerías de celdas, facilitando la integración de las celdas necesarias en el circuito. Permite también administrar versiones y organizar de forma correcta la integración en flujos digitales o analógicos más grandes.

6.11. Comandos utilizados en las herramientas de Synopsys

A continuación, se presentan algunos de los comandos más utilizados en las herramientas de Synopsys, específicamente en Library Manager, IC Compiler II e IC Validator. Estos comandos son muy importantes para el correcto funcionamiento de las herramientas y para la realización de tareas específicas en el flujo de diseño VLSI.

6.11.1. Library Manager

Los comandos son provenientes del flujo de diseño correspondiente para el Library Manager y este trabajo de graduación, se pueden encontrar en el manual específico de la herramienta [36].

`create_workspace`

Este comando se utiliza para crear un nuevo espacio de trabajo (*workspace*). Permite definir el tipo de flujo asociado al *workspace* (normal, físico o agregado) y asociarlo a los archivos de tecnología correspondientes.

```
create_workspace -flow normal -technology <techfile> <WorkspaceName>
```

`read_db`

Este comando carga archivos de librerías lógicas en formato `.db` dentro del *workspace* actual.

```
read_db { <library1.db> <library2.db> ... }
```

`read_lef`

Este comando incorpora archivos `.lef` al *workspace*.

```
read_lef <cells.lef>
```

`current_workspace`

Este comando define o consulta el *workspace* activo sobre el cual se aplicarán los siguientes comandos. Es esencial para organizar el flujo de trabajo cuando se manejan múltiples *workspaces* en paralelo.

```
current_workspace <WorkspaceName>
```

`check_workspace`

Este comando valida la integridad del *workspace* activo, verificando que todos los archivos requeridos estén correctamente cargados y que no existan inconsistencias.

```
check_workspace
```

`commit_workspace`

Este comando guarda los cambios realizados en el *workspace* activo y genera un archivo `.ndm`. Este archivo contiene toda la información consolidada del *workspace* y puede ser utilizado por otras herramientas del flujo de diseño.

```
commit_workspace -output <WorkspaceName>.ndm
```

`change_selection`

Este comando cambia la selección al *workspace* especificado, permitiendo trabajar sobre diferentes *workspaces* sin necesidad de cerrarlos o eliminarlos.

```
change_selection [get_workspaces {<WorkspaceName>}]
```

`get_workspaces`

Este comando lista o devuelve los *workspaces* disponibles en el entorno actual, lo que facilita su manejo en conjunto con otros comandos como `change_selection`.

```
get_workspaces {<WorkspaceName>}
```

`read_ndm`

Este comando importa librerías ya empaquetadas en formato `.ndm` a un *workspace* agregado, lo que permite consolidar varios *workspaces* en una sola librería.

```
read_ndm <LibraryWorkspace.ndm>
```

`open_ems_database`

Este comando abre la base de datos `EMS` asociada al *workspace*, la cual contiene información de verificación y chequeos realizados sobre las librerías.

```
open_ems_database <database.ems>
```

6.11.2. IC Compiler II

Los comandos son provenientes del flujo de diseño correspondiente para el IC Compiler II y este trabajo de graduación [29].

`set`

Este comando se utiliza para definir variables dentro del entorno de trabajo de IC Compiler II. Permite almacenar rutas, parámetros de diseño, nombres de librerías y cualquier valor necesario durante la ejecución del flujo, por lo que podemos acudir a estos valores en cualquier momento durante el diseño.

```
set VARIABLE_NAME value
```

`lappend`

Este comando agrega un nuevo valor al final de una lista previamente definida. Es utilizado comúnmente para ampliar rutas de búsqueda de librerías o archivos dentro del diseño.

```
lappend search_path <new_path>
```

`set_host_options`

Este comando configura parámetros relacionados con los recursos de cómputo que utiliza la herramienta, como el número máximo de núcleos disponibles para la ejecución del flujo.

```
set_host_options -max_cores <number>
```

`puts`

Este comando imprime mensajes en la consola o en el archivo de registro.

```
puts "Mensaje o valor de variable"
```

`sh`

Este comando ejecuta instrucciones directamente en la terminal del sistema operativo desde el entorno de la herramienta. Es usado, por ejemplo, para eliminar directorios o archivos.

```
sh rm -rf <directory>
```

`create_lib`

Este comando crea una nueva librería de diseño dentro de IC Compiler II, vinculándola con los archivos de tecnología y las librerías de referencia definidas.

```
create_lib -technology <techfile> -ref_libs <libraries> <lib_name>
```

`expr`

Este comando evalúa expresiones aritméticas y devuelve el resultado. Es útil para cálculos de dimensiones o parámetros derivados de otras variables.

```
set value [expr {$a + $b}]
```

`file`

Este comando permite gestionar archivos y directorios dentro de tcl, incluyendo creación, copia o eliminación.

```
file mkdir <directorio>
file copy -force <source_file> <destination>
```

`initialize_floorplan`

Define la geometría inicial del *floorplan*, incluyendo dimensiones del núcleo, offsets y filas de colocación.

```
initialize_floorplan -site_def <site> -use_site_row \
                     -side_length {<x> <y>} -core_offset <value>
```

`create_io_ring`

Crea un anillo de IO alrededor del núcleo del chip. Este comando define la estructura básica para ubicar pads de entrada/salida.

```
create_io_ring -name <ring_name> -corner_height <value>
```

`create_cell`

Permite instanciar celdas específicas dentro de la librería del diseño, como esquinas, pads de alimentación u otros bloques macros.

```
create_cell <cell_name> <cell_master>
```

`resolve_pg_nets`

Analiza y prepara las redes de potencia y tierra (VDD, VSS) para su correcta identificación y conexión dentro del diseño.

```
resolve_pg_nets
```

`create_net`

Crea redes lógicas o físicas en el diseño. En el caso de *power nets*, se emplea para declarar explícitamente las conexiones de VDD y VSS.

```
create_net -power <net_name>
create_net -ground <net_name>
```

`connect_pg_net`

Asocia las redes de potencia y tierra a los pines de celdas o macros. Puede ejecutarse de forma manual o automática.

```
connect_pg_net -net <net_name> [get_pins <pins>]
connect_pg_net -automatic
```

`report_cells`

Genera un reporte de las celdas instanciadas, filtrando según atributos como consumo de potencia o tipo de celda.

```
report_cells -power
```

`set_ignored_layers`

Define capas de enrutamiento a excluir durante el flujo de diseño. Se puede restringir el uso de ciertas capas metálicas según reglas de fabricación.

```
set_ignored_layers -min_routing_layer <layer> \
                   -max_routing_layer <layer>
```

`place_pins`

Realiza la colocación de los pines de entrada/salida en el *floorplan*. Puede ser automática o guiada mediante reglas definidas.

```
place_pins -self
```

`add_to_io_guide` y `place_io`

Permiten guiar y ubicar celdas de IO en las regiones asignadas. Con estos comandos se distribuyen pads de entrada/salida en el anillo de IO.

```
add_to_io_guide [get_io_guides <side>] <pad_name>
place_io
```

`create_pg_ring_pattern` / `set_pg_strategy` / `compile_pg`

Estos comandos definen y compilan patrones de distribución de potencia (*power/ground rings*) alrededor del núcleo. Se especifican capas, anchos, espaciados y estrategias de conexión.

```
create_pg_ring_pattern <pattern_name> -horizontal_layer <layer> ...
set_pg_strategy <strategy_name> -pattern { ... } -core
compile_pg -strategies <strategy_name>
```

create_pg_macro_conn_pattern

Crea un patrón de conexión de potencia para macros específicos, asegurando que sus pines de VDD/VSS se conecten correctamente a las redes globales.

```
create_pg_macro_conn_pattern <pattern_name> \
    -pin_conn_type <type> -layers {<layers>} -nets {<nets>}
```

create_pg_mesh_pattern

Define un patrón de malla de potencia (*power mesh*) para el núcleo del diseño, especificando capas, *pitch* y espaciamiento.

```
create_pg_mesh_pattern <pattern_name> -layers { ... }
```

merge_pg_mesh

Fusiona diferentes estrategias de distribución de potencia, como *rings* y *meshes*, en una sola red unificada.

```
merge_pg_mesh -nets {<power_nets>} -types {ring mesh}
```

create_placement y legalize_placement

Ejecutan la colocación inicial de celdas estándar y macros dentro del *floorplan*. Posteriormente, el comando de legalización ajusta la colocación para cumplir con reglas de diseño.

```
create_placement -floorplan -congestion -effort high
legalize_placement
```

source

Importa y ejecuta otro archivo tcl dentro del flujo actual. Permite organizar scripts en módulos reutilizables.

```
source <script_file>
```

`read_verilog`

Lee un archivo Verilog y lo carga en el bloque de diseño actual para posteriores pasos de síntesis física y el diseño.

```
read_verilog <verilog_file>
```

`read_parasitic_tech`

Carga información de parasitismo y mapas de capas que afectan la temporización y el enrutamiento del diseño.

```
read_parasitic_tech -tlup <file> -layermap <file>
```

`read_sdc`

Importa restricciones de temporización en formato SDC Synopsys (*design constraints*) para el diseño.

```
read_sdc <sdc_file> -echo
```

`remove_pg_strategies`

Elimina estrategias de distribución de potencia previamente definidas, permitiendo redefinirlas o actualizarlas.

```
remove_pg_strategies -all
```

`create_clock` / `synthesize_clock_trees`

Define relojes y sintetiza árboles de reloj, ajustando rutas y temporización de señal para cumplir restricciones.

```
create_clock -period <value> -name <clk_name> [get_nets ...]
synthesize_clock_trees -clocks <clk_name> -postroute
clock_opt -list_only
check_clock_trees -clocks <clk_name>
```

`check_design`

Ejecuta verificaciones de integridad de diseño, detectando errores de temporización, DRC o preparación de rutas.

```
check_design -checks <stage>
```

`check_routability` / `route_auto`

Evalúa la capacidad de enrutar el diseño y ejecuta el enrutamiento automático siguiendo restricciones de diseño.

```
check_routability -check_pg_blocked_ports true
route_auto
```

`create_io_filler_cells` / `create_stdcell_fillers`

Genera celdas de relleno para mantener densidad física, completar huecos en el núcleo o anillo IO, y asegurar la integridad del enrutamiento de potencia.

```
create_io_filler_cells -io_guides <guides> -reference_cells <cells>
create_stdcell_fillers -lib_cells <cells>
remove_stdcell_fillers_with_violation
```

`connect_pg_net`

Conecta automáticamente todas las redes de potencia a los pines correspondientes después de agregar celdas de relleno.

```
connect_pg_net -automatic
```

`set_app_options`

Configura parámetros de la herramienta.

```
set_app_options -list {signoff.<option> <value>}
```

`save_block`

Guarda el bloque de diseño actual, incluyendo celdas, rutas y atributos físicos.

```
save_block <library>:<block_name>
```

`write_verilog` / `write_gds`

Genera archivos de salida del diseño, como Verilog equivalente o GDSII, para simulación, *signoff* o fabricación.

```
write_verilog -include all <output_file>
write_gds -library <lib> -design <design> -view <view> -hierarchy all
    -lib_cell_view <frame> <output_file>
```

6.11.3. IC Validator

Los comandos son provenientes del flujo de diseño correspondiente para el IC Validator y las verificaciones que se hacen dentro del flujo de diseño VLSI [30].

`source`

Importa y ejecuta un archivo tcl dentro del flujo de IC Validator, permitiendo reutilizar reglas, *runsets* o *scripts* de signoff.

```
source <script_file>
```

`set_app_options`

Configura parámetros de la herramienta, como rutas de ejecución de DRC, LVS, *metal fill* o *runsets* personalizados.

```
set_app_options -list {<option_name> <value>}
```

`signoff_check_drc`

Verifica que el diseño cumpla con todas las reglas de diseño físico (DRC). Detecta errores de geometría, espaciamiento y capas.

```
signoff_check_drc
```

`signoff_fix_drc`

Aplica correcciones automáticas a errores detectados durante el DRC, incluyendo ajustes de geometría o relleno de metal.

```
signoff_fix_drc
```

`signoff_create_metal_fill`

Agrega relleno de metal para cumplir con densidad mínima requerida y mejorar integridad de fabricación.

```
signoff_create_metal_fill -mode <add|check>
```

`check_lvs`

Compara la geometría física del diseño con el *netlist* lógico, asegurando que ambas representaciones coincidan.

```
check_lvs
```

6.11.4. Design Compiler NXT

Los comandos que se utilizaron para la herramienta Design Compiler NXT [37] son:

`create_lib`

Crea una nueva biblioteca de celdas estándar a partir de un archivo de tecnología y bibliotecas de referencia.

```
create_lib -technology <techfile> -ref_libs <libraries> <lib_name>
```

6.12. Las Reference Methodology en las herramientas de Synopsys

Para acceder a las Reference Methodology se debe contar con acceso al portal de clientes de Synopsys SolvNet Plus, o alternativamente desde la propia herramienta [29]. Una vez dentro del portal, llamado SolvNet, se puede buscar la sección de documentación o recursos técnicos, donde se encuentran las metodologías de referencia disponibles para las diferentes herramientas de Synopsys [38].

Estas metodologías nos dan guías más detalladas y específicas sobre cómo se pueden utilizar las herramientas de Synopsys en diferentes flujos de diseño, incluyendo ejemplos prácticos, mejores prácticas y recomendaciones para optimizar el rendimiento y la eficiencia del diseño [38]. Cada metodología se adapta dependiendo de la herramienta y el tipo de diseño que se esté realizando, tecnología, nivel de abstracción, entre otros factores [39].

CAPÍTULO 7

Jerarquía de directorios y entorno de trabajo

7.1. Jerarquía de directorios general

En la Figura 2 se puede observar la jerarquía de directorios general que se separa empezando en el `folder_de_trabajo` y dentro de este se encuentra la carpeta `TSMC_-65nm` esto dividido en el diseño de 6 metales y el otro de 9 metales que contiene los archivos esenciales para la síntesis física y lógica. Es importante resaltar que la razón por la cual existen los dos directorios de metales es porque en 180 nm el diseño se hacía con 6 capas de metal, por lo que se decidió mantener la misma estructura para facilitar la transición. Sin embargo, basándonos en la documentación proporcionada y en base a la comunicación con IMEC, se decidió utilizar el diseño de 9 metales con 9 *tracks* para el desarrollo del chip, como se recomienda en el documento `TSMC_-28nm_40nm_65nm_mini@sic_manual_ver_Jul_2025` [40], ubicado en el directorio de `TSMC/65/doc`.

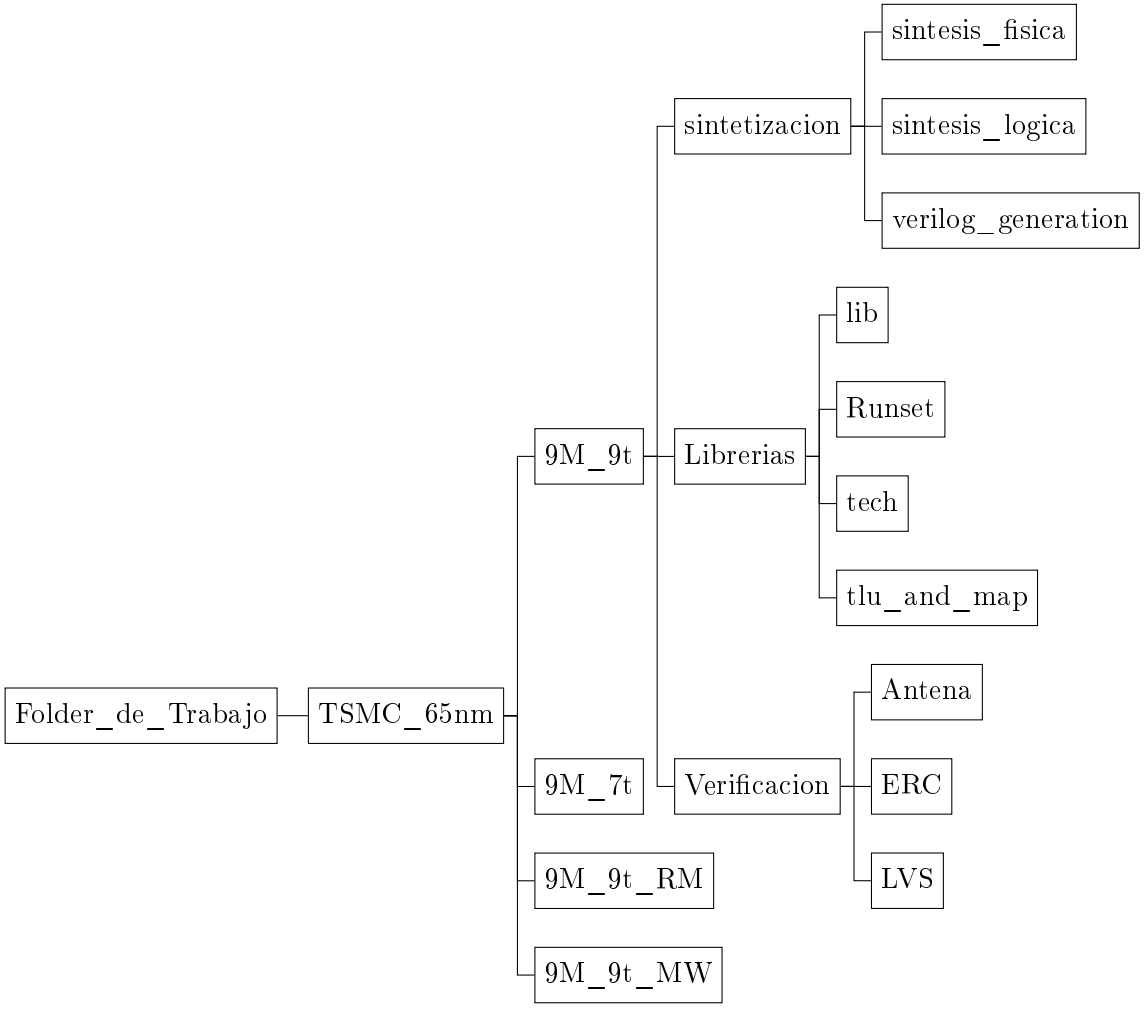
Los archivos de librerías que nos sirven para poder realizar la creación de las librerías NDM (*new data model*) los podemos encontrar en la carpeta `Librerias`. El directorio utilizado para el diseño fue el de 9 metales, aquí podremos encontrar los archivos de tecnología, base de datos de las celdas, mapeo, entre otros.

Dentro de la carpeta de sintetización se encuentran las carpetas de `sintesis_-fisica` y `sintesis_logica` que contienen los *scripts* necesarios para realizar la síntesis física y lógica respectivamente, entre otros archivos. En el caso de síntesis lógica tenemos dos carpetas pertinentes, una es la de `verilog_generation` y la otra es la de `sintesis_logica`. Y la carpeta de `Verificacion` que contiene los resultados de las verificaciones de diseño.

Cada una de estas carpetas cuenta con sus *scripts* propios para realizar las tareas necesarias en cada etapa del diseño del chip. En el caso de la síntesis física, podremos encontrar los *scripts* para la configuración inicial (*setup*), el *floorplan*, y el *script* principal de síntesis física que tiene el nombre en este formato. `nombre_circuito_top`.

Es de notar que se pueden tener varias iteraciones de diseños en 9 capas de metal o 6 capas de metal. Esto es porque hay otras consideraciones que tomar, como el número de *tracks* y el grosor de las capas de metal, que pueden variar dependiendo de las necesidades del diseño, hasta incluso las librerías que utilizaremos para sintetizar el circuito. Por lo tanto, se decidió mantener la estructura de 9 capas de metal con 9 *tracks* para el desarrollo del chip para mantener una congruencia y consistencia en el diseño.

Figura 2. Jerarquía de directorios general del entorno de trabajo

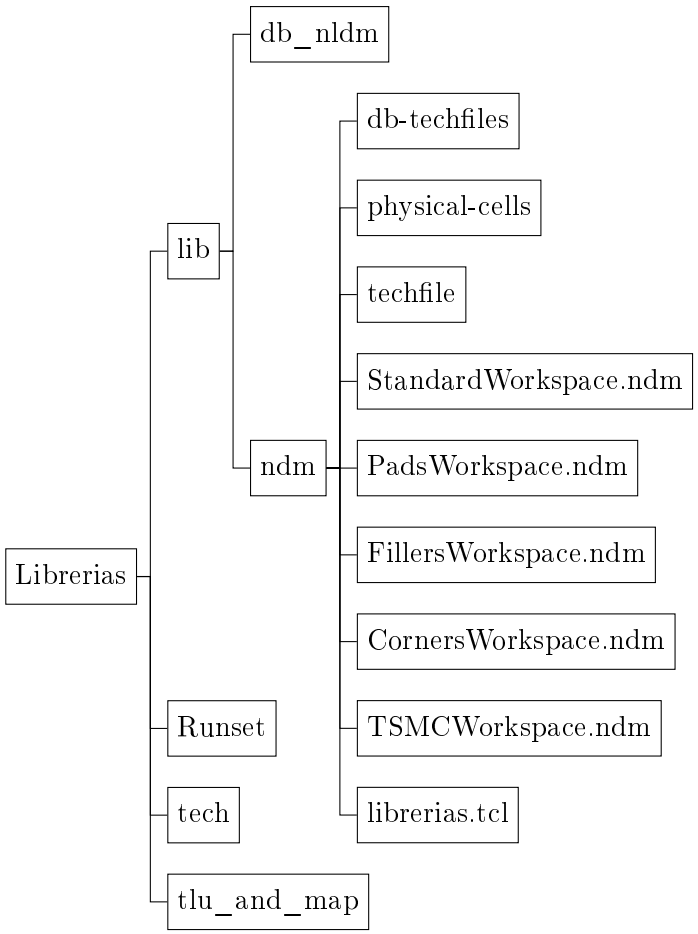


Nota. Esta es la jerarquía de directorios general del entorno de trabajo.

También es de notar otros tres directorios importantes dentro de la carpeta de `TSMC_65nm`, que son `9M_7t`, `9M_9t_RM` y `9M_9t_MW`, estos directorios son para futuras iteraciones que se deseen hacer, ya sea con 7 *tracks*, opción que se desechó por las recomendaciones de IMEC, o también trabajar con las *reference methodologies* (RM) proporcionadas por Synopsys o bien, con las librerías de tipo Milkyway. Este trabajo abarca el diseño con 9 capas de metal y 9 *tracks* y también el trabajo realizado en la exploración de las metodologías de referencia. Si se desea encontrar a detalle lo que se trabajó en el directorio `9M_9t_MW` y la creación de las respectivas librerías de tipo Milkyway, se puede consultar el trabajo de graduación de Carranza [41].

7.2. Jerarquía de directorios de librerías NDM y *runsets*

Figura 3. Jerarquía de directorios para la creación de librerías NDM y *runsets*



Nota. Esta es la jerarquía de directorios para las librerías NDM y *Runsets*.

En la Figura 3 se puede observar la jerarquía de directorios para la creación de las librerías NDM y directorio donde se guardan los *runsets* para diferentes aplicaciones

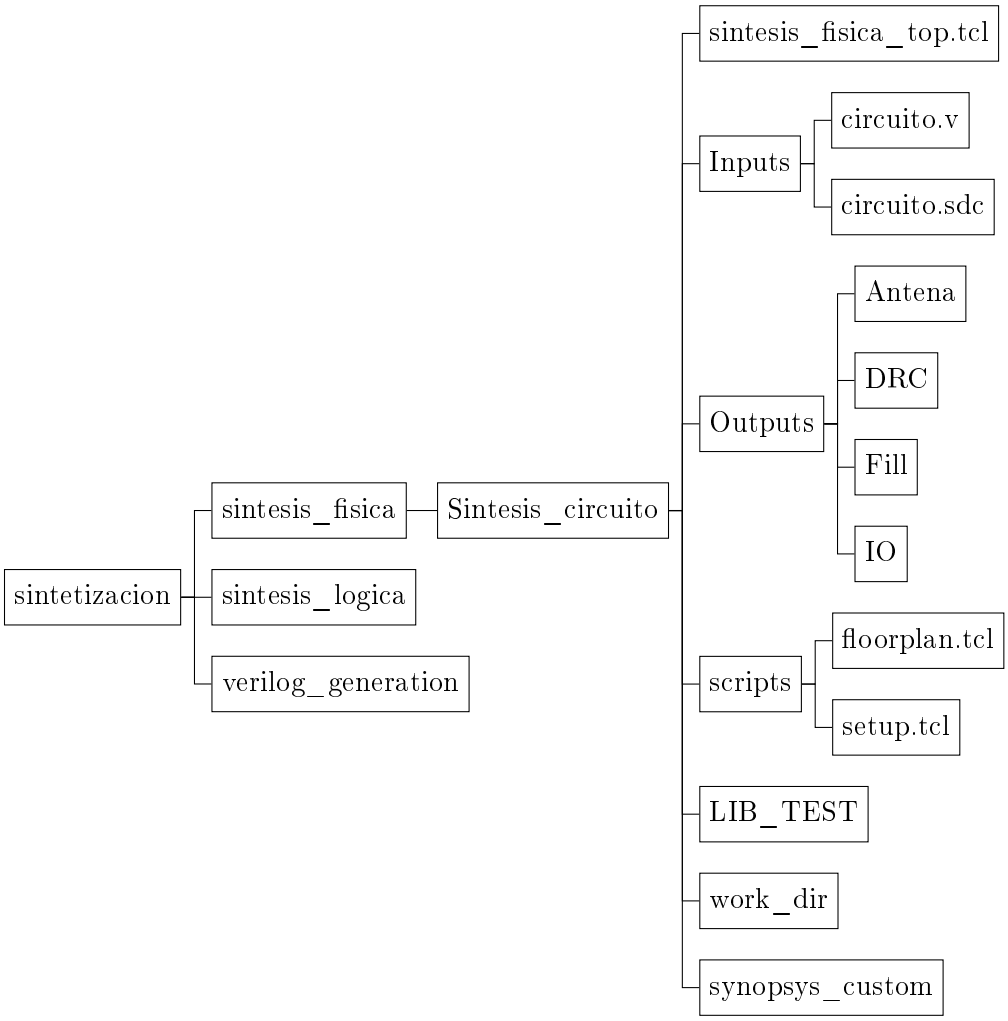
antena, DRC, relleno de difusiones y *dummy metal-fill*, los nombres de los archivos se mencionan en la sección 9.

En la carpeta de `db_nldm` se encuentran los archivos de base de datos de las celdas estándar y *pads* I/O manufacturables por el fabricante, los archivos se encuentran explicados en la sección 8.

En la carpeta de `ndm` se encuentran varias carpetas y archivos ndm esenciales, estos se explican a detalle en el capítulo 8 donde se detalla la creación de librerías y los archivos seleccionados. En la carpeta de `tech` se encuentra el archivo de tecnología, sus detalles en la sección 8.1.1. En la carpeta de `tlu_and_map` se encuentran archivos necesarios para la extracción de parásitos y mapeo, explicados en la sección 9.6.

7.3. Jerarquía de directorios para la síntesis física

Figura 4. Jerarquía de directorios para la síntesis física



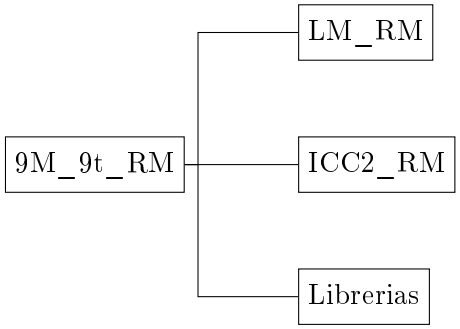
Nota. Esta es la jerarquía de directorios para la síntesis física.

En la Figura 4 se puede observar la jerarquía de directorios para la síntesis física, empezando en la carpeta de `sintetizacion`, en donde encontramos las carpetas de `sintesis_fisica`, `sintesis_logica` y `verilog_generation`. La carpeta importante para este trabajo es la de `sintesis_fisica`, en donde se encuentran los*scripts* necesarios para realizar la síntesis física, así como las carpetas de `Inputs` y `Outputs` que contienen los archivos esenciales para la síntesis física y verificaciones. Aparte se encontrarán carpetas para cada uno de los circuitos sintetizados, de manera símbolica se menciona la carpeta `Sintesis_circuito` que demuestra la estructura de cada uno de los circuitos sintetizados. Los detalles de los archivos y *scripts* se explican en el capítulo 8 y para los *scripts* usados en el flujo en el capítulo 12.

7.4. Jerarquía de directorios de metodologías de referencia

La jerarquía para los directorios de las metodologías de referencia es bastante simple en comparación con las otras jerarquías. Ya que nos podemos asegurar que esta estructura ya viene hecha por Synopsys al descargar las metodologías de referencia. Y solo debemos de preocuparnos por colocar los archivos de las librerías previamente hechas y sus rutas correctas en los *scripts* de las metodologías de referencia.

Figura 5. Jerarquía de directorios para el uso de las metodologías de referencia



Nota. Esta es la jerarquía de directorios general para el uso de las metodologías de referencia, aquí podemos encontrar las metodologías para la creación de librerías, ICC2 y sus librerías respectivas.

CAPÍTULO 8

Creación de las librerías NDM con Library Manager

Para la síntesis física se requiere de una serie de archivos necesarios para la creación de librerías NDM (*new data model*). Estás librerías utilizadas dentro de la herramienta de síntesis física IC Compiler II, son las que contienen la información de las celdas o macros que son esenciales para un diseño físico de un IC. Los archivos son proporcionados por el fabricante de la tecnología, en este caso TSMC. La selección de los archivos fue realizada basándose en la selección previa de los mismos hecha con la tecnología de 180 nm, por lo tanto, se buscó que los nombres de los archivos fueran similares a los de la tecnología de 180 nm, o en su caso, que fueran archivos que contuvieran la información necesaria para la creación de las librerías NDM. También la selección se basó en el manual desarrollado por IMEC [40]. Para la síntesis física se debe de tener una comunicación estrecha con IMEC y el equipo de sintesis lógica, para evitar errores y utilizar los archivos correctos.

8.1. Archivos seleccionados para la creación de las librerías NDM

La selección de archivos como se mencionó anteriormente, se realizó tomando en cuenta la de 180 nm. La manera de encontrar correctamente los archivos dentro de la carpeta `/TSMC`, es basándose en el directorio de *querio* y las terminaciones de los archivos. En el Cuadro 8.1 se pueden observar las claves para determinar el signficado de los archivos. Dentro del directorio de *querio*, al buscar los archivos LEF se puede encontrar un archivo el cual describe lo mencionado, su nombre es `DESIGNKIT.INFO`. Para seleccionar correctamente los archivos es necesario conocer el significado de cada extensión, así como la ubicación. Un ejemplo de donde

encontrar los archivos para las celdas de tipo estándar es el siguiente directorio: `TSMC/65/CMOS/LP/stclib/9-track/tcbn65lp-set /tcbn65lp_220a_FE/tcbn65lp_220a_nldm/`

Cuadro 8.1. Significado de extensiones de archivos

```
The <kit> initials are mapped according to following table:

  rln = release note
  gds = GDSII layout view
  gdf = GDSII phantom view
  spi = Spice/lvs netlist
  apf = Apollo FRAM (phantom) view
  apt = Apollo FRAM and CEL (layout) view
  sef = Silicon Ensemble LEF (phantom) view
  syn = Synopsys
  vlg = Verilog simulation models
  vit = Vital (VHDL) simulaion models
  mdt = Mentor DFT (Fastscan, DFT advisor)
  doc = Documents (Data sheets)
  ibs = IBIS models
  lpe = LPE Spice netlist
  vts = Voltage strom model
  ctc = Celtic model
  sgs = Signal storm model
  pdb = Physical compiler view
  ccs = Composite Current Source (CCS) view
  nldm= Synopsys Liberty view
  vcn = Magma Volcano view
  cdk = Cadence DFII view
```

Nota. Los significados de las extensiones de los archivos encontrados en el archivo llamado `DESIGNKIT.INFO`.

Otra cuestión importante que se definió para la creación de las librerías NDM, fue la selección de los archivos tomando como base en el número de *tracks* mínimos, resultando en 9 *tracks*. Por lo tanto, se debe de notar que todos los archivos para la creación de las librerías NDM, son encontrados dentro del directorio: `/TSMC/65/CMOS/LP/stclib`. En el cual se encuentran los archivos para la creación de un layout de 9 *tracks*. Un *track* es la unidad mínima de espacio que se utiliza para definir la altura de una celda estándar.

El número de capas de metal utilizados en la tecnología de 65 nm es de 9 capas, por lo tanto, se debe de notar que los archivos posean dentro de su nombre el texto `9lm`, lo cual indica que son archivos para la creación de un layout de 9 capas de metal, esto también recomendado por IMEC [40].

IMEC indica en el manual [40] que el principal método de empaquetado es *wire-bond* (con el cual se pretende que se realice la manufactura del nanochip), pero que

también se puede utilizar el método de *flip-chip* si se cumplen ciertas condiciones, y se pide de manera especial. Como referencia, se investigó que si el método de empaquetado fuera *flip-chip*, se deben de utilizar mínimo 2 capas de metal "Mz" para el diseño del chip. Esto se encuentra buscando en la documentación de TSMC, específicamente el archivo: `T-000-BP-DR-017_0_8.PDF`. Encontrado en el directorio de *querio*: `TSMC/doc`. En este documento se puede encontrar en la sección "*applicable technologies for this document*" que si se usa el N65, se deben de usar mínimo dos capas de "Mz" para *flip-chip*. Lo anterior se refiere al grosor del metal en una dirección específica, cómo se verá hay varias opciones de grosor de acuerdo a las diferentes capas existentes, el objetivo es elegir el archivo adecuado.

Se eligió el archivo que cumpla con el grosor de "2z" que es el mínimo requerido para el empaquetado *flip-chip* y así garantizar que el chip pueda ser empaquetado correctamente con cualquiera de los dos métodos. Es de notar también que IMEC requiere analizar y dar un permiso especial para el empaquetado *flip-chip* y que se debe de cumplir con las reglas de densidad del chip, también se recomienda tener a la mano el *design rule manual* (DRM) de TSMC, el cual se puede encontrar en el directorio:

```
TSMC/65/CMOS/doc
```

Y se llama:

```
T-N65-CL-DR-001_2_6_2.pdf
```

Este documento es esencial para entender las reglas de diseño y restricciones que se deben seguir al momento de realizar el diseño físico del chip. Aquí se encuentran más detalles acerca de los errores de DRC que se generen al ejecutar la síntesis física.

8.1.1. *Technology file*

El archivo de tecnología es el que contiene la información de la tecnología utilizada, nombres de las capas y sus características, así como reglas y restricciones de diseño. La selección se basó en las 9 capas de metal necesarias y el grosor de "2z" mínimo, dentro del archivo de tecnología se pueden encontrar las especificaciones. El archivo seleccionado para la creación de las librerías NDM fue el siguiente:

- Nombre del archivo:
  - `tsmcn65_9lmT2.tf`
- Ubicación:
  - `TSMC/65/CMOS/LP/stclib/9-track/tcbn65lp-set /techfile/tcbn65lp_200a_apf/TSMCHOME/digital/Back_End /milkyway/tcbn65lp_200a/techfiles/`

8.1.2. *DB-Technology file*

El archivo de base de datos de tecnología es el que contiene la información de las celdas estándar y *pads* I/O manufacturables por el fabricante. Contienen características de área, consumo, tamaño o voltaje. Aparte que este es de los archivos de más importancia, ya que el archivo "`.db`" que se usó en la etapa de síntesis lógica debe de encontrarse dentro de nuestra selección al hacer las librerías en nuestro flujo de síntesis física. La información del archivo se puede encontrar en el archivo del mismo nombre pero con la extensión "`.lib`". Es importante notar que los archivos se encuentran dentro de un archivo comprimido con la extensión:

```
.nldm
```

Lo que signfica es que se encuentra en un archivo de "Synopsys *liberty view*" como se puede ver en la Figura 8.1. Estos archivos son de vital importancia ya que deben de coincidir con el que se utilizó para la realización de la síntesis lógica.

Standard cells

- Nombre de los archivos:
  - `tcbn65lpbc.db`
  - `tcbn65lplt.db`
  - `tcbn65lpml.db`
  - `tcbn65lptc.db`
  - `tcbn65lpwc.db`
  - `tcbn65lpwcl.db`
  - `tcbn65lpwcl0d9.db`
- Ubicación:
  - `TSMC/65/CMOS/LP/stclib/9-track/tcbn65lp-set/tcbn65lp_220a_FE/tcbn65lp_220a_nldm/TSMCHOME/digital/Front_End/timing_power_noise/NLDM/tcbn65lp_220a`

Pads I/O

- Nombre de los archivos:
  - `tpdn65lpnv2od3tc.db`
- Ubicación:
  - `TSMC/65/CMOS/LP/IO2.5V/iolib/LINEAR/tpdn65lpnv2od3_200a_FE/tpdn65lpnv2od3_200a_nldm/TSMCHOME/digital/Front_End/timing_power_noise/NLDM/`

8.1.3. LEF *files*

Los archivos LEF se caracterizan por ser archivos de intercambio de información física de las celdas estándar y macros, utilizado especialmente en la etapa de *place and route* ya que contiene características y nombres de capas metálicas, reglas de enrutamiento e información de las vías, nombres de celdas, pines o posibles restricciones. El archivo LEF se encuentra dentro de un archivo comprimido con la extensión: `sef`, esto signfica según el Cuadro 8.1 que se encuentra en un archivo de *silicon ensemble LEF (phantom) view*, es decir, un archivo designado para usar con herramientas de *place and route* de Cadence (Compañía de herramientas EDA, como Synopsys), pero que puede ser utilizado en Synopsys IC Compiler II debido a que son archivos estándar dentro de la industria. Sin embargo, es conocido que a veces generan *warnings* que pueden afectar o ser ignoradas a lo largo del proceso de diseño.

Standard cells

- Nombre de los archivos:
  - `tcbn65lp_9lmT2.lef`
- Ubicación:
  - `TSMC/65/CMOS/LP/stclib/9-track/tcbn65lp-set/tcbn65lp_220a_FE/tcbn65lp_200a_sef/TSMCHOME/digital/Back_End/lef/tcbn65lp_200a/lef/`

Pads I/O

- Nombre de los archivos:
  - `tpdn65lpnv2od3_9lm.lef.lef`
- Ubicación:
  - `TSMC/65/CMOS/LP/IO2.5V/iolib/LINEAR/tpdn65lpnv2od3_200a_FE/tpdn65lpnv2od3_140b_sefu9lm/TSMCHOME/digital/Back_End/lef/tpdn65lpnv2od3_140b/mt_2/9lm/lef/`

8.2. Creación de librerías

Para la creación de las librerías NDM, se utilizó la herramienta Library Manager, la cual permite crear las librerías NDM a partir de los archivos seleccionados anteriormente. Esta herramienta se puede utilizar tanto por medio de comandos como por medio de una interfaz gráfica.

Se recomienda que se utilicen los comandos, ya que es la manera más rápida y eficiente de crear las librerías, si se quiere consultar la forma de realizar los comandos de manera gráfica, consultar el manual de realización de librerías realizado en el trabajo de Ruiz [22].

Para no confundirse con cuál es el *script* correcto, se recomienda que el nombre contenga las capas de metales y el número de *tracks*. La creación de las librerías NDM se realizó con el *script* creado en lenguaje TCL llamado `librerias_9m_9t.tcl`, el cual se creó en el directorio: `Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/lib/ndm/`

8.2.1. Ejecución del *script* de creación de librerías NDM

Para ejecutar el *script* de creación de librerías NDM, se debe de abrir una terminal, de preferencia en el directorio donde se encuentra el *script*, y ejecutar el comando:

```
lm_shell
```

Esto abrirá la herramienta de *IC Compiler II Library Manager Tool*, una vez dentro de la herramienta para ejecutar el *script* y crear las librerías NDM se debe de ejecutar el comando:

```
source librerias_9m_9t.tcl
```

Esto creará todas las librerías respectivas, las cuales se encuentran en el directorio: `Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/lib/ndm/` y finalizan con la extensión `.ndm`. Esto ya nos permite realizar el diseño físico del chip utilizando las herramientas de Synopsys, debido a que no se cuenta con un PDK diseñado para Synopsys, se debe de crear las librerías NDM desde cero.

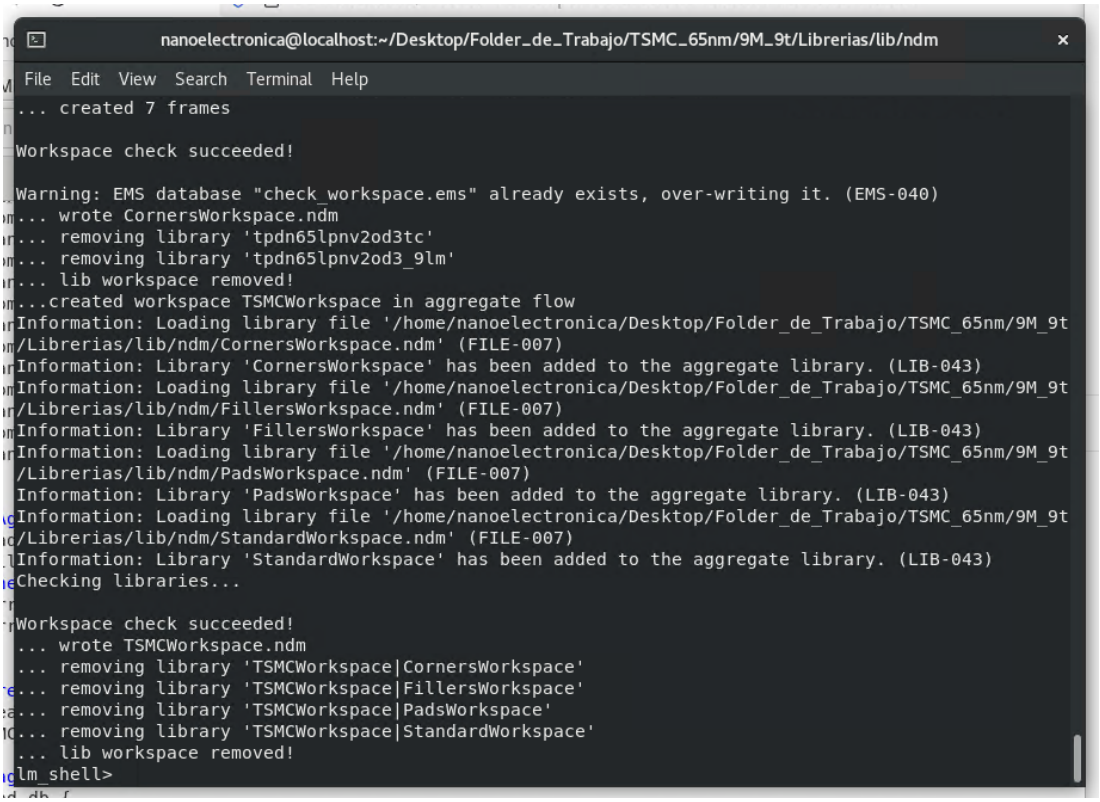
Así se verá el mensaje que obtendremos en la terminal al finalizar la creación de las librerías NDM:

Por otro lado, también podemos verificar la correcta creación de las librerías NDM viendo que se hayan creado los respectivos *workspaces* dentro del directorio donde se ejecuta el *script* de las librerías. Este directorio será clave para la utilización correcta de los siguientes *scripts* de síntesis física.

8.2.2. Explicación del *script* de creación de librerías NDM

El *script* de creación de librerías NDM, se divide en varias secciones, cada una de ellas se encarga de crear una librería específica, ya sea de celdas estándar, *pads* I/O, *fillers* o *corners*. Para todos los *workspaces* creados, se utiliza el mismo archivo de tecnología `tsmcn65_9lmT2.tf`, pero por su parte, los `lef` o los `db` pueden cambiar. El orden de creación de los *workspaces* es independiente de los demás, sin embargo, el *workspace* general `TSMCWorkspace.ndm` debe de ser el último en crearse, ya que este contiene a los demás.

Figura 6. Mensaje de finalización de creación de librerías NDM



Nota. Mensaje de finalización exitosa para la creación de las librerías NDM en la terminal de la VM. Se puede observar que no hay errores y que se crearon las librerías respectivas, aparte también se debe siempre de verificar en el directorio respectivo.

El script empieza con la configuración inicial, en donde borramos todas las librerías existentes para la creación de las nuevas, y otros archivos antiguos.

Cuadro 8.2. Configuración inicial del *script* de librerías

```
# ============================================================
# 1. Preparacion del Entorno (Limpieza de Archivos)
# ============================================================
# Borramos las librerias y archivos anteriores para crear los
    nuevos.
sh rm -rf command.log
sh rm -rf FillersWorkspace.ndm
sh rm -rf StandardWorkspace.ndm
sh rm -rf CornersWorkspace.ndm
sh rm -rf PadsWorkspace.ndm
sh rm -rf TSMCWorkspace.ndm
sh rm -rf check_workspace.ems
sh rm -rf PreFrameCheck

# Existen 4 workspaces, y 1 workspace general.
# Standard workspace (standard cells) y fillers workspace (fillers
    para core)
# Pads workspace (pads io) y corners worksapce ( fillers de corner o
    de ring io)
```

Nota. Configuración inicial del *script* de creación de librerías NDM, `librerias_9m_9t.tcl`.

Después se procede a crear las librerías de las celdas estándar, esta librería se conoce como `StandardWorkspace.ndm`, y contiene las celdas estándar seleccionadas en la etapa de síntesis lógica, como se mencionó anteriormente para ser congruente con el diseño físico. Se empieza agregando el archivo de tecnología, después el archivo de celdas estándar, los LEF y finalmente se guarda la librería, a continuación se muestra en el Cuadro 8.3.

Cuadro 8.3. Creación del Standard Workspace del *script* de librerías

```
# ============================================================
# 2. Creacion y Configuracion: Standard Workspace
#    (Flujo 'normal' para celdas funcionales)
# ============================================================
# Creando Standard Workspace
   ////////////////////////////////////////
create_workspace -flow normal -technology /home/nanoelectronica/
   Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/lib/ndm/
   techfile/tsmcn65_9lmT2.tf StandardWorkspace

# Agregar db-techfile standard cells
read_db {
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpbc.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lplt.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpml.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lptc.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpwc.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpwcl.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpwc0d9.db
}

# Agregar lef standard cells
read_lef /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9
   M_9t/Librerias/lib/ndm/physical-cells/standard-cells/
   tcbn65lp_9lmT2.lef

# check y commit de Standard workspace
current_workspace; check_workspace
current_workspace StandardWorkspace; commit_workspace  -output
   StandardWorkspace.ndm
```

Nota. Creación del Standard Workspace en el *script* de creación de librerías NDM, `librerias_9m_9t.tcl`.

Luego se debe de proceder a crear la librería o *workspace* de los *fillers*, la cual se conoce como `FillersWorkspace.ndm`, esta librería contiene las celdas que usaremos para el relleno de metal dentro del *core*, corresponde a el uso de las celdas estándar, como se muestra en el Cuadro 8.4.

Cuadro 8.4. Creación del Fillers Workspace del *script* de librerías

```
# ============================================================
# 3. Creacion y Configuracion: Fillers Workspace
#    (Flujo 'physical_only' para celdas de relleno)
# ============================================================
# Creando Fillers Workspace
   ////////////////////////////////////////
create_workspace -flow physical_only -technology /home/
   nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/techfile/tsmcn65_9lmT2.tf PhysicalOnlyWorkspace

# agregar db-techfiles
read_db {
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpbc.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lplt.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpml.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lptc.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpwc.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpwcl.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/standard-cells/tcbn65lpwc0d9.db
}

# agregar lef modificado con class fill
read_lef /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9
   M_9t/Librerias/lib/ndm/physical-cells/standard-cells/
   tcbn65lp_9lmT2.lef

# check y commit de Fillers workspace
current_workspace; check_workspace
current_workspace PhysicalOnlyWorkspace; commit_workspace  -output
   FillersWorkspace.ndm
```

Nota. Creación del Fillers Workspace en el *script* de creación de librerías NDM, `librerias_9m_9t.tcl`

Lo importante es saber seleccionar el tipo de flujo correcto, para las celdas estándar se debe de utilizar el flujo normal, y para los *fillers* se debe de utilizar el flujo *physical only*, ya que no se requiere de información de temporización o consumo, solo la información física de las celdas.

Después se procede a crear el *workspace* de los *Pads I/O*, la cual se conoce como `PadsWorkspace.ndm`, esta librería contiene las celdas de entrada y salida del chip, las cuales son esenciales para la conexión del chip con el mundo exterior. La porción del

script para la creación de este *workspace* se muestra en el Cuadro 8.5.

Cuadro 8.5. Creación del Pads Workspace del *script* de librerías

```
# ============================================================
# 4. Creacion y Configuracion: Pads Workspace
#    (Flujo 'normal' para celdas de E/S o I/O)
# ============================================================
# Creando Pads Workspace
   ////////////////////////////////////////////
create_workspace -flow normal -technology /home/nanoelectronica/
   Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/lib/ndm/
   techfile/tsmcn65_9lmT2.tf PadsWorkspace

# Agregar db-techfile i/o
read_db {
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/IO-cells/tpdn65lpnv2od3tc.db
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/IO-cells/tpdn65lpnv2od3wc.db
}

# Agregar lef i/o
read_lef /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9
   M_9t/Librerias/lib/ndm/physical-cells/IO-cells/
   tpdn65lpnv2od3_9lm.lef

# check y commit de Pads workspace
current_workspace; check_workspace
current_workspace PadsWorkspace; commit_workspace  -output
   PadsWorkspace.ndm
```

Nota. Creación del Pads Workspace en el *script* de creación de librerías NDM, `librerias_9m_9t.tcl`.

También tenemos el *workspace* de los *corners*, el cual es `CornersWorkspace.ndm`, este *workspace* contiene los *fillers* especiales para las esquinas del *core* del chip y esta zona en donde se ubican las entradas y salidas del chip. La porción del *script* para la creación de este *workspace* se muestra en el Cuadro 8.6.

Cuadro 8.6. Creación del Corners Workspace del *script* de librerías

```
# ============================================================
# 5. Creacion y Configuracion: Corners Workspace
#    (Flujo 'physical_only' para fillers de corner/ring)
# ============================================================
# Creando Corners Workspace
   ////////////////////////////////////////////
create_workspace -flow physical_only -technology /home/
   nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/techfile/tsmcn65_9lmT2.tf PhysicalOnlyWorkspace

# agregar db-techfile I/O
read_db {
/home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/
   Librerias/lib/ndm/db-techfiles/IO-cells/tpdn65lpnv2od3tc.db
}

# agregar lef I/O
read_lef /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9
   M_9t/Librerias/lib/ndm/physical-cells/IO-cells/
   tpdn65lpnv2od3_9lm.lef

# check y commit de Corners workspace
change_selection [get_workspaces {PhysicalOnlyWorkspace}]
current_workspace; check_workspace
change_selection [get_workspaces {PhysicalOnlyWorkspace}]
current_workspace PhysicalOnlyWorkspace; commit_workspace  -output
   CornersWorkspace.ndm
```

Nota. Creación del Corners Workspace en el *script* de creación de librerías NDM, `librerias_9m_9t.tcl`.

En estos dos *workspaces* también es importante seleccionar el tipo de flujo correcto, para los *Pads I/O* se debe de utilizar el flujo normal, y para los *corners* se debe de utilizar el flujo *physical only*, ya que no se requiere de información de temporización o consumo, como se menciona anteriormente.

Finalmente, se procede a la creación de un *workspace* general, el cual contiene a todos los demás *workspaces*, este se conoce como `TSMCWorkspace.ndm`, y es el que se utilizará en la etapa de síntesis física. En este caso el *workspace* general va a ser el último en crearse y debe ser con el tipo de flujo agregado. La porción del *script* para la creación de este *workspace* se muestra en el Cuadro 8.7.

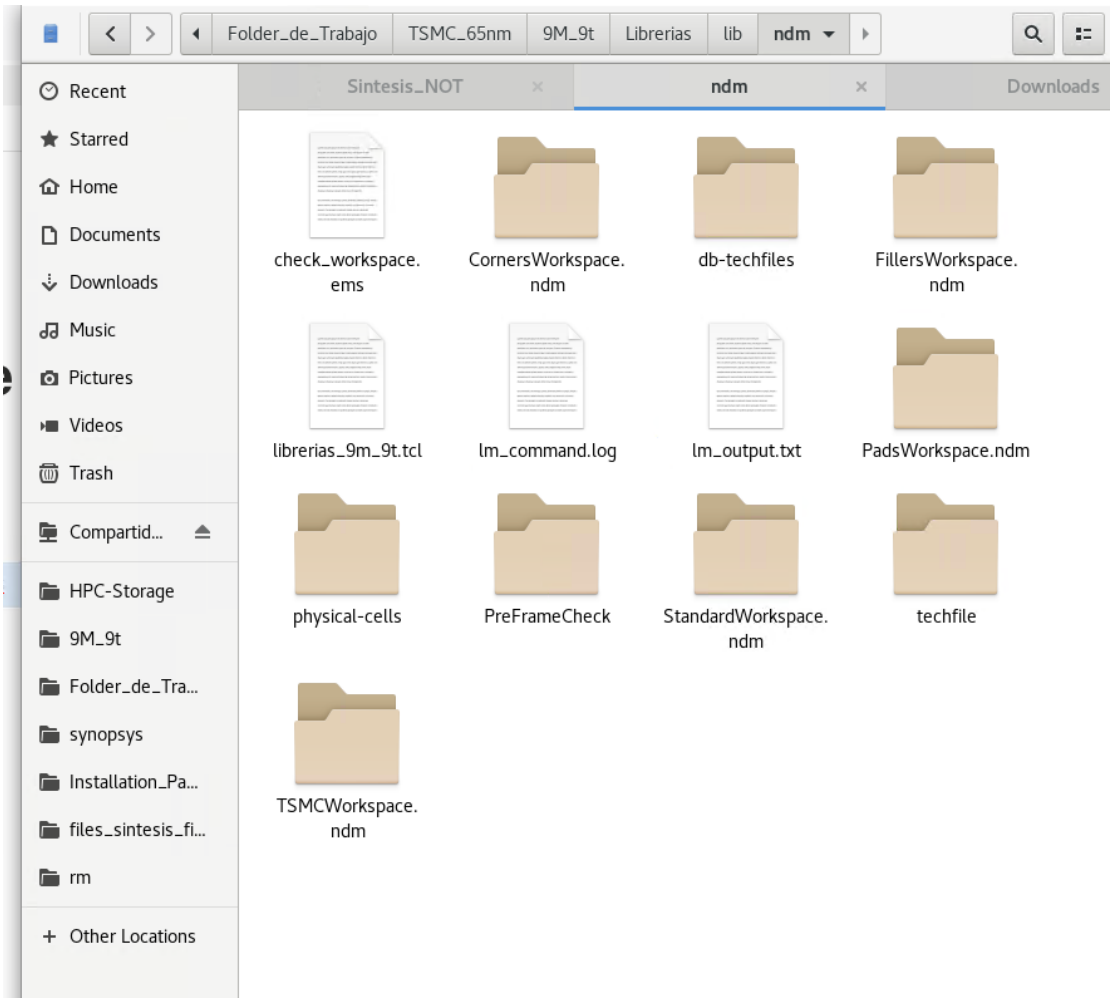
Cuadro 8.7. Creación del TSMC Workspace del *script* de librerías

```
# ============================================================
# 6. Consolidacion: TSMC Workspace (Aggregate)
#    (Workspace general que une todos los anteriores)
# ============================================================
# Creando TSMC workspace ---------------------------
create_workspace -flow aggregate TSMCWorkspace

# Leer librerias ndm
read_ndm /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/lib/ndm/CornersWorkspace.ndm
read_ndm /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/lib/ndm/FillersWorkspace.ndm
read_ndm /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/lib/ndm/PadsWorkspace.ndm
read_ndm /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/lib/ndm/StandardWorkspace.ndm

change_selection [get_workspaces {TSMCWorkspace}]
current_workspace; check_workspace
open_ems_database check_workspace.ems

change_selection [get_workspaces {TSMCWorkspace}]
current_workspace TSMCWorkspace; commit_workspace  -output TSMCWorkspace.ndm
```

Nota. Creación del TSMC Workspace en el *script* de creación de librerías NDM, `librerias_9m_9t.tcl`.

Cuando se termine la creación de las librerías NDM, se verá un mensaje en la terminal que nos indica que la creación fue exitosa, y que no hubo errores en la verificación. Esto lo podemos ver en la Figura 6. Y también podremos ver los archivos creados en el directorio de librerías NDM, como se muestra en la Figura 7:

Figura 7. Vista de las librerías NDM creadas



Nota. Vista de las librerías NDM creadas en el directorio correspondiente.

CAPÍTULO 9

Selección de *runsets* y archivos *TLU and MAP*

Un *runset* es aquel archivo que cuenta con un conjunto de reglas y restricciones que se de ser ideal el diseño físico del chip, se deberían de cumplir. Para el diseño de un chip estos *runsets* son esenciales ya que permiten a las herramientas de síntesis física realizar las verificaciones necesarias para garantizar que el diseño cumpla con las debidas reglas de diseño y restricciones. En nuestro caso la herramienta encargada es IC Validator. Estos *runsets* son proporcionados por el fabricante de la tecnología, en este caso TSMC. Dentro de ellos se pueden encontrar también ciertas opciones de configuración que determinan el tipo de reglas que se deben de cumplir respectivas a cada *runset*.

Los archivos *TLU and MAP* son archivos necesarios para la extracción de parásitos y mapeo. Los archivos TLU contienen información necesaria para la extracción de parásitos, y datos que ayudan a la herramienta a identificar y generar un diseño físico correcto. Y los MAP son archivos que contienen los nombres de las capas del diseño y su respectivo mapeo a las capas de la tecnología utilizada.

La elección de los *runsets* y archivos *TLU and MAP* se realiza basándose en el acuerdo que se llegue entre todo el equipo de diseño, tanto de síntesis lógica como física, y también tomando en cuanto la documentación proporcionada por IMEC y TSMC. Además, es de importancia notar que algunas requieren cambios especiales para adecuarse al diseño, estos cambios se describen en el capítulo 10.

La forma en la que se pueden modificar las opciones que se nos dan en los *runsets*, se realiza en el capítulo 10 y también la documentación respectiva dada por TSMC y IMEC en los directorios correspondientes.

9.1. *Runset* de DRC

El *runset* de DRC es el que contiene las reglas de diseño y restricciones que se deben cumplir cuando diseñamos el chip.El *runset* seleccionado fue el siguiente:

- Nombre del archivo:
  - `ICVLN65S_9M_6X2Z.26_2a`
- Ubicación:
  - `TSMC/65/CMOS/util/T-N65-CL-DR-001-J1_2_6_2A/DRC_ICV_65nm_V26_-2a/MAIN_DRC_TopMz/`

9.2. *Runset* de *antenna*

El *runset* de *antenna* es el que contiene las reglas que se deben cumplir para evitar los efectos de antena. Se eligió el *runset* basándose en el número de capas de metal.

- Nombre del archivo:
  - `ICVLN65S_9M_ANT.26_2a`
- Ubicación:
  - `TSMC/65/CMOS/util/T-N65-CL-DR-001-J1_2_6_2A/DRC_ICV_65nm_V26_2a/ANTENNA_DRC/`

9.3. *Runset* de *dummy DOD/DPO fill*

El *runset* de *dummy DOD/DPO fill*, también conocido como *dummy fill* FEOL, es el que contiene las reglas a seguir a ejecutar para el relleno dedicado a las difusiones de óxido y difusiones de *poly* que se deben cumplir para evitar errores de densidad. *FEOL* corresponde a *front end of line*. El *runset* es el siguiente:

- Nombre del archivo:
  - `Dummy_FEOL_ICV_65nm.26_1a`
- Ubicación:
  - `TSMC/65/CMOS/util/T-N65-CL-DR-001-J2_2_6_1A/Dummy_FEOL_ICV_65nm_V26_1a/`

9.4. *Runset* de *dummy metal-fill*

El *runset* de *dummy metal-fill*, también conocido como *dummy fill* BEOL, es el que contiene las reglas que se deben cumplir para evitar las violaciones de densidad del chip hablando en términos de relleno de metal, las cuales son celdas que nos ayudan a cubrir el área que no se utiliza dentro del chip con metales que no tienen una función, así garantizando la fabricación. BEOL corresponde a *back end of line*. El *runset* es el siguiente:

- Nombre del archivo:
  - `Dummy_BEOL_ICV_65nm.26_2a`
- Ubicación:
  - `TSMC/65/CMOS/util/T-N65-CL-DR-001-J3_2_6_2A/Dummy_BEOL_ICV_65nm_V26_2a/`

9.5. *Runset* de BND

La selección del *runset* de BND es esencial para la verificación de las conexiones eléctricas del chip, este *runset* contiene las reglas que se deben cumplir para garantizar que todas las conexiones eléctricas se hagan en la forma que está determinada para dicho empaquetado. Como se menciona en el capítulo 8, el empaquetado predeterminado es *wire bonding process*. En español se le conoce como soldadura por hilo, este es un proceso prominente en el mundo de la electrónica para la realización de las interconexiones eléctricas en la fase de encapsulación del chip con diferentes materiales, como cobre, oro o aluminio [42].

El proceso para la selección del *runset* de BND implica revisar las especificaciones del diseño y empaquetado del chip, en este trabajo se abordó la búsqueda de los archivos posibles a implementar en un futuro diseño y las variantes entre ellos. La carpeta donde podemos encontrar los *runsets* para el proceso de *wire bonding* es la siguiente:

- Nombre de la carpeta:
  - `wire_bond_DRC`
- Ubicación:
  - `TSMC/util/T-000-CL-DR-002-J1_1_9_2A/DRC_ICV_000_AL_WB_FC_V19_2a/`

Dentro de la carpeta encontraremos varios archivos, no se definió cual sería el más adecuado para el diseño, por lo que las modificaciones y verificaciones no se realizaron en este trabajo de graduación utilizando el *runset* de BND. Sin embargo, para futuros trabajos se recomienda revisar los diferentes archivos y seleccionar el que mejor se adecúe al diseño del chip.

9.6. Selección de *TLU and MAP files*

Los archivos *TLU and MAP* son esenciales para la extracción de parásitos y mapeo, la extracción de parásitos es un proceso que se realiza con otra herramienta de Synopsys llamada Star RC, esta ejecuta un script que utiliza los datos ya generados mediante el diseño físico para extraer todos los parásitos, esto no se hará en la etapa de síntesis física, pero es importante tener los archivos seleccionados y saber el lugar en el que se encontraron ya que IMEC requiere esta información a la hora de ya tener el diseño físico finalizado.

9.6.1. Archivos TLU

Los archivos TLU son aquellos que contienen la información necesaria para la extracción de parásitos, y datos que ayudan a la herramienta a identificar y generar un diseño con propiedades físicas correctas. El archivo TLU seleccionado fue el siguiente:

- Nombre del archivo:
  - `cln65lp_1p09m+alrdl_typical_top2.tluplus`
- Ubicación:
  - `TSMC/65/CMOS/LP/stclib/9-track/tcbn65lp-set/techfile/tcbn65lp_200a_apf/TSMCHOME/digital/Back_End/milkyway/tcbn65lp_200a/techfiles/tluplus`

9.6.2. Archivos MAP

Los archivos MAP son aquellos que contienen los nombres de las capas del diseño y su respectivo mapeo a las capas que se encuentran en el *techfile* de la tecnología utilizada. El archivo MAP seleccionado fue el siguiente:

- Nombre del archivo:
  - `star.map_9M`
- Ubicación:

- TSMC/65/CMOS/LP/stclib/9-track/tcbn65lp-set/techfile/tcbn65lp_200a_apf/TSMCHOME/digital/Back_End/milkyway/tcbn65lp_200a/techfiles/tluplus

CAPÍTULO 10

Modificaciones en *runsets* y archivos *TLU and MAP*

Las modificaciones que se realizaron en los *runsets* y archivos *TLU and MAP* fueron necesarias para adecuarse al diseño del chip, estas modificaciones se realizaron basándose en la documentación proporcionada por TSMC e IMEC, así como en la experiencia del equipo de diseño. Y siguiendo los manuales de diseño de Synopsys. Una vez definidos los *runsets* y archivos *TLU and MAP*, se procedió a realizar las modificaciones necesarias para adecuarse al diseño del chip. Se recomienda también efectuar pruebas con las diferentes configuraciones posibles, si se da la opción, para así encontrar la mejor configuración posible. Por simpleza y por temas de confidencialidad, solo se listarán las modificaciones que se habilitaron.

10.1. Modificaciones en el *runset* de DRC

Para modificar el DRC solo se deben de descomentar las opciones que están definidas en el siguiente formato: `#define opcion`. Solo se colocan las opciones hasta la sección de *select by DFM grouping*, ya que luego se abordan opciones específicas de DFM que se pueden activar o desactivar en la sección de configuración agrupada antes mencionada. Estas opciones las encontraremos al inicio del archivo, y se presentan en orden. Las modificaciones que se hicieron en el *runset* de DRC fueron las siguientes:

- Habilitar `ICV_REPORT_DENSITY`
- Habilitar `SNPSINDESIGN`
- Habilitar `SELECTABLE_VIOLATION_NAMES`

- Habilitar SELECTABLE_VIOLATION_COMMENTS
- Deshabilitar MERGE_DENSITY_ERRORS
- Habilitar GUIDELINE_RES
- Deshabilitar DISCONNECT_ALL_RESISTOR
- Deshabilitar CONNECT_ALL_RESISTOR
- Habilitar DEFINE_PAD_BY_TEXT
- Habilitar GUIDELINE_LUP
- Habilitar GUIDELINE_ESD
- Habilitar NW_SUGGESTED
- Habilitar DATATYPE_WARNING
- Habilitar MIXED_SCHEME
- Habilitar CHECK_LOW_DENSITY
- Habilitar FRONT_END
- Habilitar BACK_END
- Habilitar FULL_CHIP
- Deshabilitar WLCSP_2_MASK
- Deshabilitar GP
- Deshabilitar LPG
- Habilitar LP
- Deshabilitar HALF_NODE
- Deshabilitar _28K_AP
- Deshabilitar WLCSP_SEALRING
- Habilitar ChipWindowUsed
- Habilitar DFM
- Deshabilitar DFM_ONLY
- Habilitar Required
- Habilitar Recommended
- Deshabilitar Analog
- Habilitar Guideline

- Deshabilitar `First_priority`
- Deshabilitar `Systematic`
- Deshabilitar `Defect`
- Deshabilitar `d_SPICE`

10.2. Modificaciones en el *runset* de antena

El *runset* de antena también necesito unas cuantas modificaciones, este es el único *runset* que se utiliza afuera de la *shell* de IC Compiler II. La manera en la que se utiliza el script se puede encontrar en el capítulo 14. Se deben descomentar las opciones que están definidas en el siguiente formato: `#define opcion`; este archivo se utiliza con la herramienta IC Validator. Las modificaciones que se hicieron en el *runset* de antena fueron las siguientes:

- Habilitar `SELECTABLE_VIOLATION_NAMES`
- Habilitar `SELECTABLE_VIOLATION_COMMENTS`
- Deshabilitar `PRINT_ANT_RATIO`
- Deshabilitar `_28K_AP`
- Deshabilitar `DTM`

10.3. Modificaciones en el *runset* de *dummy metal-fill*

El *runset* de *dummy metal-fill* también necesitó unas cuantas modificaciones, este es el *runset* que se encarga de realizar el relleno de metal dentro del *core* del chip. Este *runset* tiene tres secciones que modificar, las opciones que están definidas en el formato `#define opcion`. Para este *runset* es recomendable revisar la documentación encontrada en el mismo directorio donde se puede encontrar originalmente el archivo, y también se puede ver el capítulo 10 para entender las consecuencias de los cambios a este *runset*. Es importante entender la razón por las que se modifica cada opción por lo que leer los manuales de IMEC y TSMC es esencial. Por otro lado, solo es recomendable modificar las opciones que se encuentran al inicio del archivo, ya que las demás opciones son parámetros propios del *runset* si se desean modificar estas opciones se debería de consultar con IMEC para entender las consecuencias de estos cambios.

La primera es la sección de opciones generales, esta sección empieza en la línea 20 del archivo. La segunda sección (empezando en la línea 282) es relacionada a las esquinas y área del chip. Y la tercera (empezando en la línea 287) a las capas de

metal y el esquema de metal. Las modificaciones que se hicieron en el *runset* de *dummy metal-fill* de la primera sección fueron las siguientes:

- Deshabilitar `UseprBoundary`
- Habilitar `ChipWindowUsed`
- Deshabilitar `dmOnCorner`
- Habilitar `WithSealring`
- Habilitar `MIXED_SCHEME`
- Deshabilitar `FILL_IN_SLOT`
- Habilitar `FILL_DMx`
- Deshabilitar `FILL_indDMx`
- Deshabilitar `FILL_OPCDMx`
- Deshabilitar `_5K_THICK_My`
- Habilitar `_9K_THICK_Mz`
- Deshabilitar `_12K_THICK_Mr`
- Deshabilitar `_34K_THICK_Mu`
- Deshabilitar `CBM_OVER_Mx`
- Deshabilitar `FILL_AP`
- Deshabilitar `N55HV`
- Deshabilitar `N65_SIPH`
- Deshabilitar `N65_SIPH_COUPE_EC`
- Deshabilitar `FILL_DAP_C`
- Deshabilitar `SNPSINDESIGN`
- Deshabilitar `FILL_IN_MW_BLOCKAGE_LAYER`
- Deshabilitar `EXTENDED_LAYERS`
- Habilitar `USE_ICC2`
- Deshabilitar `MERGE_ORIGINAL_DESIGN`
- Deshabilitar `AUTO_FILL_COMPRESSION`
- Habilitar `OUTPUT_GDS`

Las opciones de la segunda sección que se modificaron fueron las siguientes:

- Habilitar `WithSealring`
- Deshabilitar `dmOnCorner`
- Deshabilitar `FILL_DAP_C`

Y en las opciones de la tercera sección se listan solo las opciones habilitadas:

- Habilitar `FILL_DM1`
- Habilitar `FILL_indDM1`
- Habilitar `FILL_OPCDM1`
- Habilitar `FILL_DM2`
- Habilitar `FILL_indDM2`
- Habilitar `FILL_OPCDM2`
- Habilitar `FILL_DM3`
- Habilitar `FILL_indDM3`
- Habilitar `FILL_OPCDM3`
- Habilitar `FILL_DM4`
- Habilitar `FILL_indDM4`
- Habilitar `FILL_OPCDM4`
- Habilitar `FILL_DM5`
- Habilitar `FILL_indDM5`
- Habilitar `FILL_OPCDM5`
- Habilitar `FILL_DM6`
- Habilitar `FILL_indDM6`
- Habilitar `FILL_OPCDM6`
- Habilitar `FILL_DM7`
- Habilitar `FILL_indDM7`
- Habilitar `FILL_OPCDM7`
- Habilitar `FILL_DM8`

- Habilitar `FILL_indDM8`
- Habilitar `FILL_OPCDM8`
- Habilitar `FILL_DM9`
- Habilitar `FILL_indDM9`
- Habilitar `_9K_THICK_M9`
- Habilitar `_9K_THICK_M10`
- Habilitar `_9K_THICK_M11`

10.4. Modificaciones en el *runset* de *dummy DOD/DPO fill*

En archivo FEOL o también conocido como *dummy DOD/DPO fill* es necesario realizar modificaciones. Este archivo al igual que el BEOL o también conocido como *dummy metal-fill* tiene documentación que se puede encontrar en el mismo directorio donde se encuentra el archivo. El formato de las modificaciones es el mismo que en los otros *runsets*, las opciones que están definidas en el formato `#define opcion`. Al igual que con el *runset* BEOL, es importante entender la razón por las que se modifica cada opción por lo que leer los manuales de IMEC y TSMC es de suma importancia. Por otro lado, solo es recomendable modificar las opciones que se encuentran al inicio, ya que las demás opciones son parámetros del *runset*, se recomienda consultar con IMEC de igual manera si se desea modificar alguno de estos parámetros.

También se puede ver el capítulo 11 para entender las consecuencias de los cambios a este *runset*. Este archivo basta con ser modificado en la primera sección, la cual empieza en la línea 12 del archivo. Las modificaciones que se hicieron en el *runset* de *dummy DOD/DPO fill* fueron las siguientes:

- Deshabilitar `UseprBoundary`
- Habilitar `ChipWindowUsed`
- Deshabilitar `dmOnCorner`
- Habilitar `WithSealring`
- Deshabilitar `FILL_TCD_PATTERN`
- Habilitar `FILL_DOD_DPO`
- Habilitar `AUTO_FILL_COMPRESSION`
- Habilitar `OUTPUT_GDS`
- Deshabilitar `SNPSINDESIGN`

- Deshabilitar `FILL_IN_MW_BLOCKAGE_LAYER`
- Deshabilitar `EXTENDED_LAYERS`
- Deshabilitar `MERGE_ORIGINAL_DESIGN`
- Habilitar `USE_ICC2`

10.5. Modificaciones en los archivos *TLU and MAP*

Los archivos *TLU and MAP* frecuentemente se agrupan debido a que ambos archivos tienen que ver para la información de mapeo y extracción de parásitos. En este caso no se realizaron modificaciones en el archivo de TLU, pero si en el archivo MAP. Si estas modificaciones no son efectuadas pueden ocurrir errores y obtendremos alertas durante el proceso de síntesis física en ICC2. En el archivo MAP se deben de cambiar los nombres de las capas que se encuentran en el archivo por los nombres de las capas que se encuentran en el *techfile* de la tecnología utilizada. El archivo modificado fue `star.map_9M`, y el archivo final fue el siguiente:

Cuadro 10.1. *Script* modificado para mapeo, `star.map_9M`

```
*****AstroTF_maskName VS ITF_layerName*****
conducting_layers
PO     poly
M1       metal1
M2       metal2
M3       metal3
M4       metal4
M5       metal5
M6       metal6
M7       metal7
M8       metal8
M9       metal9
AP      metal10
via_layers
CO polyCont
VIA1       via1
VIA2       via2
VIA3       via3
VIA4       via4
VIA5       via5
VIA6       via6
VIA7       via7
VIA8       via8
RV        via9
```

Nota. Archivo modificado para mapeo, `star.map_9M`.

CAPÍTULO 11

Proceso de correción de errores en síntesis física en tecnología de 65 nm

El cambio de tecnología de 180 nm a 65 nm trajo consigo varios retos y dificultades que se tuvieron que superar durante el proceso de síntesis física. La reducción en el tamaño de los transistores y la complejidad del diseño requirieron una adaptación en las herramientas y metodologías utilizadas. Además, diferencias en los archivos de librerías y reglas de diseño entre las dos tecnologías presentaron desafíos nuevos que debieron de ser considerados cuidadosamente.

Inicialmente, se profundizó leyendo y entendiendo el proceso previamente ejecutado en 180 nm, para así tener una base sólida sobre la cual trabajar. Después de conseguir esa base, se procedió a leer la documentación proporcionada por IMEC y TSMC para la tecnología de 65 nm [40]. Esta documentación es un resumen general de todas las consideraciones que se deben de tener en cuenta al momento de empezar un diseño en esta tecnología. Se hace referencia a las librerías, reglas de diseño, archivos necesarios, verificaciones respectivas y otros aspectos importantes que se deben de tomar. Al igual que recomendaciones para hacer el diseño en 9 metales, estrategias de enrutamiento, temas de colocación y el abordaje de errores comunes junto con sus soluciones.

El proceso de síntesis física en 65 nm se llevó a cabo utilizando la herramienta IC Compiler II de Synopsys. Esta herramienta es una solución integral para el diseño físico de circuitos integrados, que permite realizar todas las etapas del proceso de síntesis física, desde la colocación y enrutamiento hasta la verificación y optimización del diseño. La herramienta ofrece una amplia gama de funcionalidades y opciones de configuración que permiten a los diseñadores adaptar el proceso a las necesidades, archivos y restricciones específicas del proyecto.

En las sección 11.1 se abordan los cambios realizados para la correción de errores,

así como los archivos que se utilizaron en esas etapas. Resultando en los archivos para librerías y también los *scripts* utilizados en el proceso de síntesis física. Estos archivos de librerías se encuentran descritos en el capítulo 8 y los *scripts* en el capítulo 12.

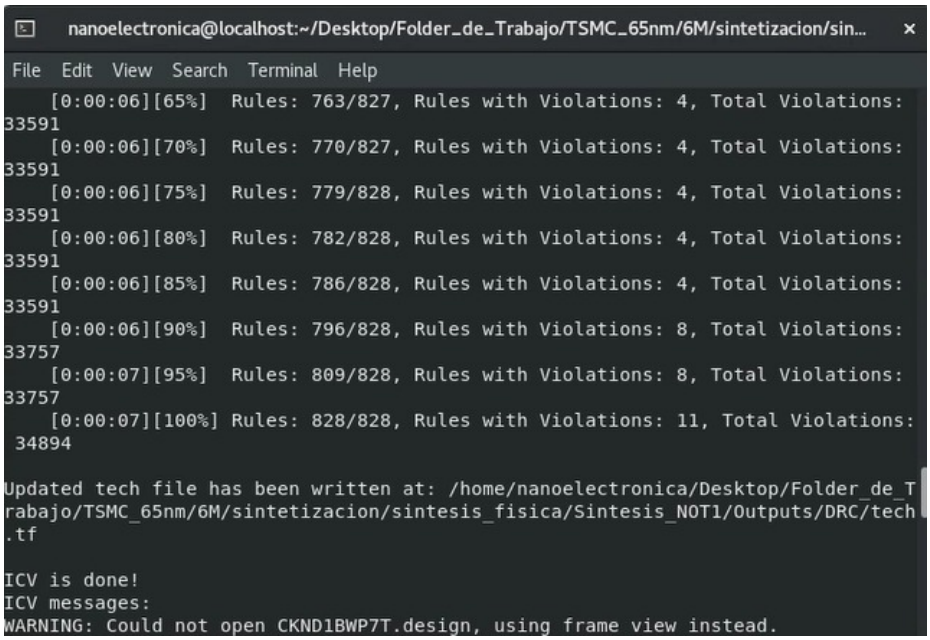
11.1. Correción de errores en el proceso de síntesis física en 65 nm

Durante el proceso de síntesis física en tecnología de 65 nm, se presentaron varios errores desde el momento inicial que se abordó con lo previamente ejecutado en años anteriores. Esto se debió a varios factores, entre ellos las diferencias iniciales en las librerías y reglas de diseño, así como la complejidad añadida de trabajar con una tecnología más avanzada. Aunque se hayan realizado varias iteraciones de los diseños en diferentes tipos de configuraciones, como 6 metales y 9 metales, se decidió solo mantener como referencia los archivos, *scripts* y configuraciones que dieran los menos errores posibles.

Como se explicará a continuación, este es un proceso iterativo que requirió de varios intentos y ajustes para lograr un diseño funcional y con los menos errores posibles. A continuación, se describen los cambios realizados para corregir los errores encontrados durante el proceso de síntesis física en 65 nm, desde empezar basándose en un diseño de 6 metales, hasta llegar a un diseño final de 9 metales con los menos errores posibles.

11.1.1. Primeros intentos con 6 metales

Figura 8. Resultado en la terminal de primeras pruebas con 6 metales en 65 nm



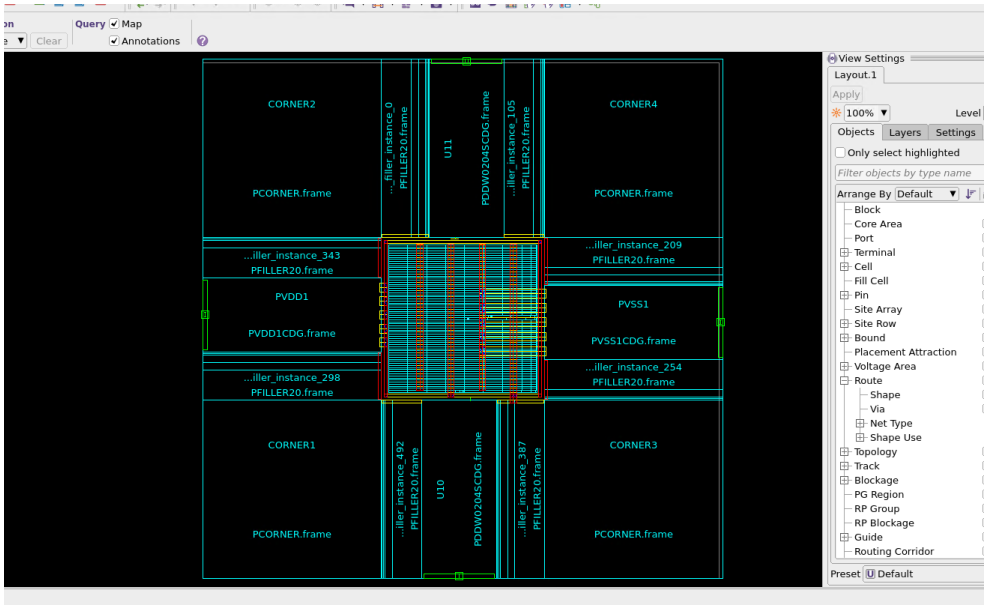
Nota. Resultado de la terminal después de la síntesis de un circuito con una compuerta NOT utilizando las librerías de 6 metales en 65 nm.

En los primeros intentos de síntesis física en 65 nm, se decidió basarse en el diseño previamente realizado en 180 nm, el cual utilizaba 6 metales. Se utilizaron archivos de 6 metales para las librerías, los cuales se pueden encontrar en la máquina virtual del proyecto. Se usaron los *scripts* originales del proyecto de 180 nm, adaptándolos para la tecnología de 65 nm. Por consiguiente, hacer cambios de rutas, modificaciones básicas de DRC y crear nuevas librerías fue lo único que se realizó.

Al correr estos primeros intentos, se encontraron más de 34,000 errores en la etapa de DRC, las cuales incluían errores de densidad, violaciones de reglas de diseño y otros problemas relacionados con la tecnología de 65 nm. En este intento se utilizó el diseño de una compuerta NOT mapeada a la tecnología de 65 nm en 6 metales, el resultado de la terminal se muestra en la Figura 8.

Y en la Figura 9 podemos observar el diseño ya sintetizado en la tecnología de 65 nm en 6 metales.

Figura 9. Compuerta NOT sintetizada de primeras pruebas con 6 metales en 65 nm



Nota. Resultado de la síntesis de un circuito con una compuerta NOT utilizando las librerías de 6 metales en 65 nm visto en ICC2.

11.1.2. Creación del *script* de librerías y cambios en el DRC

Después de los primeros intentos con 6 metales, se decidió crear un *script* para automatizar la creación de las librerías necesarias para la síntesis física en 65 nm. Esto porque en términos de tiempo se notó que crear las librerías mediante la interfaz gráfica de Library Manager era un proceso tedioso y propenso a errores.

Por lo que, la creación del *script* permitió generar las librerías de manera más rápida y consistente, asegurando que todas las librerías necesarias estuvieran disponibles para el proceso de síntesis física. Este *script* de librerías se encuentra descrito en la sección 8.2.

Además, se realizaron cambios extensivos en el *runset* de DRC para adaptarlo a la tecnología de 65 nm. Estos cambios incluyeron la modificación de las reglas de diseño, y algunos otros cambios para adaptarlo a las herramientas.

11.1.3. Cambio a 9 metales y modificación del archivo MAP

Después de los intentos iniciales con 6 metales y la creación del *script* de librerías, se decidió cambiar a un diseño de 9 metales para aprovechar las ventajas que esta configuración ofrece y su flexibilidad y también por la recomendación de IMEC para esta tecnología, la cual está indicada en la documentación proporcionada [40]. Este cambio implicó la utilización de archivos de librerías de 9 metales, los cuales también

se encuentran en la máquina virtual del proyecto. Se hizo tanto para 9 metales y 9 tracks, como para 7 tracks, para así tener más opciones.

Otro cambio importante que se realizó fue la modificación del archivo `star.map_9M`, el cual es esencial para el mapeo de las capas del diseño a las capas del *techfile* de la tecnología utilizada. Este archivo fue modificado para asegurar que todas las capas estuvieran correctamente mapeadas, evitando así posibles errores en el proceso de síntesis física. El archivo modificado se encuentra descrito en la sección 10. Originalmente aparecía un mensaje para varias capas y vías en la terminal de ICC2, como el siguiente:

```
Warning: Layer mapping file warning. Tech layer 'via2' mapped in layer
    mapping file is mask name of 'VIA2'. (TLUP-011)
```

11.1.4. Archivo mapeado de Verilog y corrección en archivo DB

Ahora se utilizó ya un archivo de Verilog mapeado a la tecnología de 65 nm en 9 metales y 9 tracks. Este archivo fue generado en la etapa de síntesis lógica utilizando la herramienta Design Compiler de Synopsys. También se realizó la corrección de incluir dentro de la creación de las librerías el archivo `.db` correspondiente al que se usó en la etapa de síntesis lógica. Esto para evitar discrepancias entre las librerías utilizadas en la síntesis lógica y física.

11.1.5. Cambio de *fillers* en *scripts* de síntesis física y rutas para los *runsets*

Un cambio importante para la congruencia en el diseño fue el cambio de los *fillers* utilizados en los *scripts* de síntesis física. Los *fillers* son celdas que se utilizan para llenar espacios vacíos en el diseño, asegurando que se cumplan las reglas de densidad y otras consideraciones de diseño. Se aseguró que los *fillers* utilizados en la sección de I/O y *core* del chip fueran los mismos que se encuentran dentro de los LEF. Esto para evitar inconsistencias y posibles errores durante el proceso de síntesis física.

Se decidió cambiar algunas rutas de los *runsets* utilizados en el proceso de síntesis física. Y se agregó a las opciones de la aplicación de IC Compiler II la ruta para el *runset* de antena. Esto se puede encontrar en el capítulo 12.

11.1.6. Vías y *filler type*

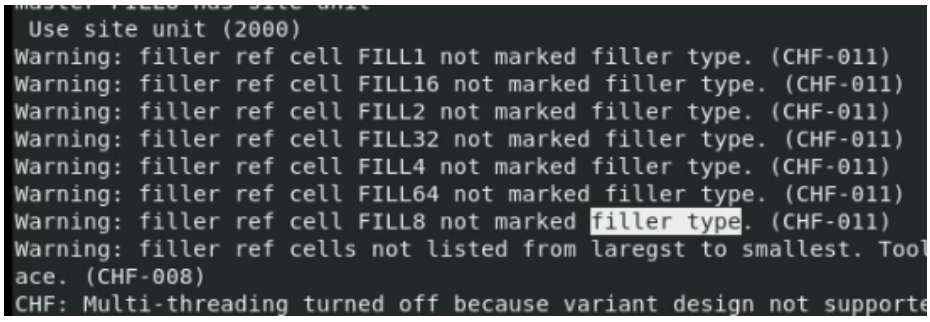
Figura 10. Vías faltantes en *pads* de VDD/VSS en 65 nm

Nota. En esta imagen se observa que no hay vías conectando el metal a los *pads* de VDD/VSS.

Se notó que a diferencia de los diseños generados en 180 nm, en 65 nm se genera el metal que conecta a los *pads* de VDD/VSS pero sin vías, las vías siendo la parte azul que se puede ver en donde se intersecta el rectangulo amarillo con el rojo. Esto se puede ver en la Figura 10, donde se observa que no hay vías conectando el metal a los *pads*.

También se observó que el tipo de *filler* utilizado en los *scripts* de síntesis física no contiene en el archivo LEF el tipo de *filler*, por lo que la herramienta IC Compiler II genera alertas al momento de leer los *fillers* de 9 *tracks*. Esto se puede ver en la Figura 11, donde se observa la alerta generada por la herramienta en la terminal.

Figura 11. Alerta de tipo de *filler* en 65 nm

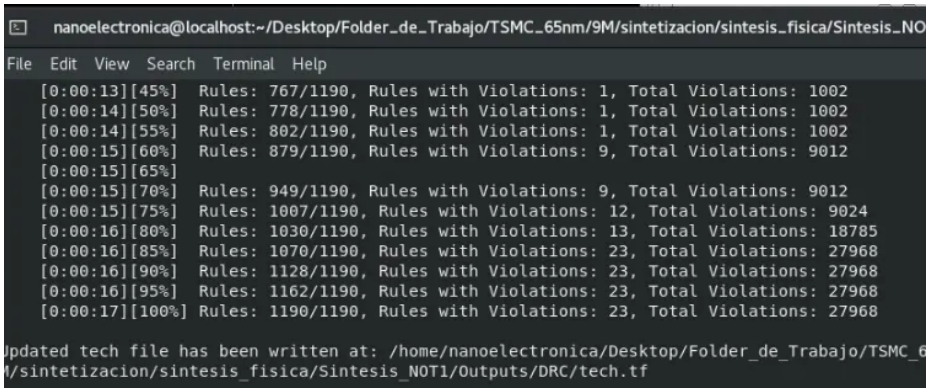


Nota. En esta imagen se observa la alerta generada por la herramienta en la terminal para fillers en 9t.

Y en esta fase se estaba obteniendo un total de aproximadamente 28,000 errores aproximadamente en la etapa de chequeo de DRC, los cuales incluían errores de den-

sidad, violaciones de reglas de diseño y otros problemas relacionados con la tecnología de 65 nm. Esto se puede ver también en la Figura 12, donde se observa el resultado de la terminal después de la síntesis de un circuito con una compuerta NOT utilizando las librerías de 9 metales en 65 nm.

Figura 12. Terminal con aproximadamente 28,000 errores DRC



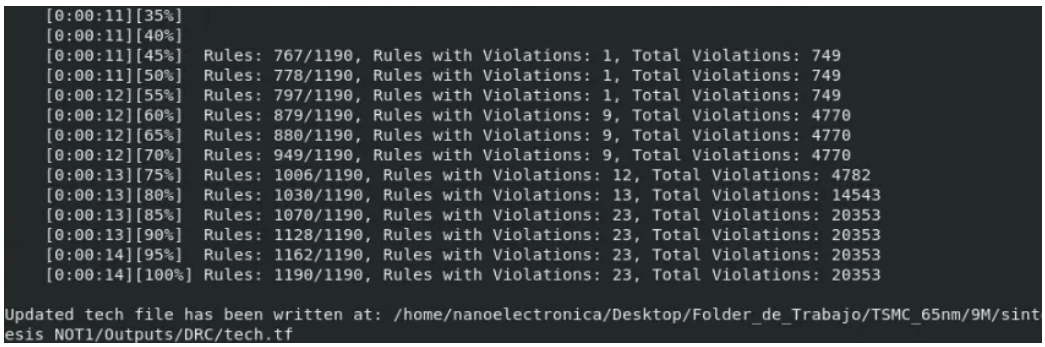
Nota. En esta imagen se observa la terminal con los errores DRC.

11.1.7. Cambios en *runset dummy metal-fill*

Investigando a fondo la documentación de los *runsets* proporcionada por IMEC y TSMC [40], se realizaron varios cambios en el *runset dummy metal-fill* para adaptarlo a las necesidades del diseño en 65 nm. Estos cambios incluyeron la modificación de las opciones relacionadas con el relleno de metal, y los márgenes del diseño. El primer cambio que se hizo se habilitó la opción relacionada a los *fillers* en los corners sin la opción de *with sealring* de manera activada.

El resultado fue una reducción a 20,000 errores aproximadamente en la etapa de chequeo de DRC. Esto se puede ver en la Figura 13, donde se observa el resultado de la terminal después de la síntesis de un circuito con una compuerta NOT utilizando las librerías de 9 metales en 65 nm. Los Cambios finales para este *runset* se encuentran descritos en el capítulo 10.

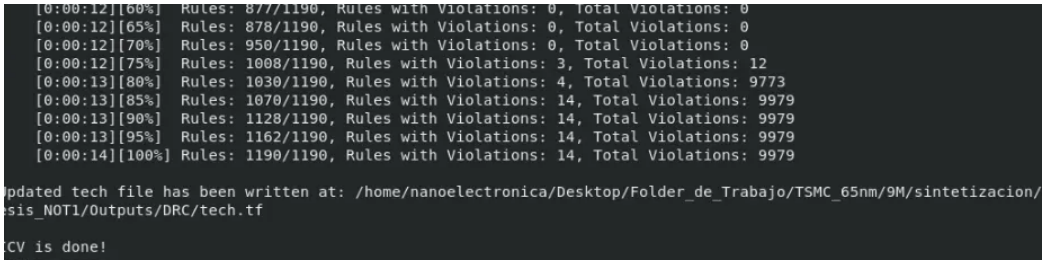
Figura 13. Terminal con 20,000 errores DRC



Nota. En esta imagen se observa la terminal con los errores DRC, después de habilitar la opción de *fillers* en corners sin *with sealring*.

Haciendo el cambio a tener activada la opción de *with sealring* y deshabilitando la opción de *fillers* en corners, se logró una reducción adicional en los errores DRC, llegando a aproximadamente 10,000 errores, como se puede ver en la Figura 14.

Figura 14. Terminal con 10,000 errores DRC



Nota. En esta imagen se observa la terminal con los errores DRC, después de habilitar la opción de *With sealring* y deshabilitar *fillers* en corners.

Haciendo también el cambio respectivo en el comando de IC Compiler II para realizar el relleno de metal con el *runset* modificado. Si se quiere consultar cualquier comando relacionado a BEOL se puede consultar el capítulo 12. El comando modificado fue:

```
signoff_create_metal_fill -mode add -foundry_fill_type beol
```

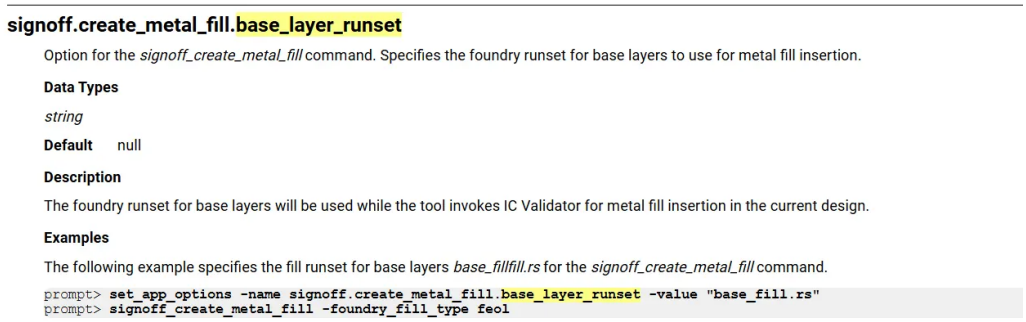
11.1.8. Adición y cambios del *runset dummy DOD/DPO fill*

Después de los cambios realizados en el *runset dummy metal-fill*, se siguió investigando en la línea de arreglar los errores DRC de densidad. Se encontró mediante la comunicación con IMEC que era necesario agregar el *runset dummy DOD/DPO fill* para así cumplir con las reglas de densidad de DOD/DPO, lo cual corresponde a

rellenar el diseño en sus capas respectivas de difusiones y polisilicio para cumplir con los requerimientos de densidad.

Después de encontrar y agregar este *runset* a nuestros directorios, se realizó la investigación de cuáles eran los nuevos comandos necesitados por parte de la herramienta para utilizar este *runset*. Se realizaron los cambios necesarios en el *script* de síntesis física para agregar este *runset* y se hicieron las modificaciones necesarias en el mismo, los cambios finales para la utilización de este *runset* se encuentran descritos en el capítulo 12. El comando agregado para realizar el relleno de DOD/DPO fue basándonos en la Figura 15, donde se puede apreciar en el manual de ICC2 la sintaxis del comando [26].

Figura 15. Comando para relleno de DOD/DPO visto en el manual de ICC2



signoff.create_metal_fill.base_layer_runset

Option for the *signoff_create_metal_fill* command. Specifies the foundry runset for base layers to use for metal fill insertion.

Data Types

string

Default null

Description

The foundry runset for base layers will be used while the tool invokes IC Validator for metal fill insertion in the current design.

Examples

The following example specifies the fill runset for base layers *base_fillfill.rs* for the *signoff_create_metal_fill* command.

```
prompt> set_app_options -name signoff.create_metal_fill.base_layer_runset -value "base_fill.rs"
prompt> signoff_create_metal_fill -foundry_fill_type feol
```

Nota. Se puede observar el comando para relleno de DOD/DPO en el manual de ICC2, y su sintaxis respectiva.

Después de agregar este *runset* y hacer los cambios necesarios, basándonos en la documentación proporcionada encontrada en el mismo directorio del *runset*, se logró una reducción significativa en los errores DRC, llegando a aproximadamente 650 errores en el diseño de la NOT. Como se puede ver en la Figura 16. Las opciones clave para llegar a esta reducción de errores fueron activar la opción de *with sealring* y deshabilitar *dmOnCorner* en el *runset* de DOD/DPO, además de hacer los cambios respectivos en el comando de ICC2 para utilizar este *runset*.

Figura 16. Terminal con 650 errores DRC aproximadamente, después de agregar el *runset* de DOD/DPO

Nota. En esta imagen se observa la terminal con los errores DRC, después de agregar el *runset* de DOD/DPO y hacer sus cambios respectivos.

11.1.9. Cursos de ICC2 y soporte de Synopsys

Los últimos ajustes y correcciones de algunos errores se lograron mediante la realización de cursos oficiales de Synopsys sobre la herramienta IC Compiler II. Estos se pueden encontrar en la página oficial de Synopsys SolvNet Plus. Estos cursos proporcionaron una comprensión más profunda de las funcionalidades de la herramienta y las mejores prácticas para su uso. Además, se contó con el soporte de Synopsys para resolver dudas como artículos o la opción de abrir casos de soporte técnico.

Los resultados conllevaron a un diseño con menos errores, y una forma más eficiente de abordar los problemas para el futuro en conjunto con *scripts* mejor estructurados y documentados. Esto fue un gran avance, ya que permite la optimización del proceso de síntesis física en 65 nm y la reducción de errores en futuros diseños.

CAPÍTULO 12

Scripts del proceso de síntesis física en 65 nm

Los *scripts* utilizados en el proceso de síntesis física en 65 nm son fundamentales para automatizar y estandarizar las diferentes etapas del diseño. Estos *scripts* contienen comandos específicos que guían a la herramienta IC Compiler II a través de las diferentes fases del proceso de síntesis física, desde la configuración inicial hasta la generación del diseño final. A continuación, se describen los principales *scripts* utilizados en este proceso, junto con sus funciones y características clave.

Estos son los *scripts* en sus versiones finales en sus versiones genéricas, por lo que no necesariamente son los mismos que se utilizaron en cada diseño específico. Esto se debe a que no se quieren causar confusiones al momento de querer reutilizar estos *scripts* en futuros diseños. Y para mayor entendimiento, no se menciona el nombre de un circuito específico en los *scripts*, sino que se deja como una variable a ser definida por el usuario.

Primero se explicarán en qué consiste cada *script* junto con los fragmentos de código respectivos para cada etapa. Luego en la sección 12.4 se incluirán los fragmentos de código específicos.

12.1. *Script* `sintesis_fisica_top.tcl`

Este *script* es la fuente para todos los demás *scripts* utilizados en el proceso de síntesis física en 65 nm. Este se separa en diferentes etapas, desde una configuración previa en donde invocamos a otros *scripts*, hasta la generación del diseño final. Y su visualización en la interfaz gráfica de IC Compiler II.

12.1.1. Etapa 1 configuración inicial

En esta etapa se realiza una configuración inicial del entorno de diseño, en donde invocamos al *script* `setup.tcl` para cargar las configuraciones generales del entorno y para el proceso de síntesis física. También se invoca al *script* `floorplan.tcl` para realizar el *floorplanning* del diseño. Además, se eliminan cualquier estrategia de *power gating* previa para limpiar el entorno de diseño.

Cuadro 12.1. Configuración inicial de `sintesis_fisica_top.tcl`

```
##################################################
# ETAPA 1: CONFIGURACION INICIAL
##################################################

# Carga el entorno de configuracion general
source scripts/setup.tcl

# Carga el script de floorplan
source scripts/floorplan.tcl -echo

# Elimina cualquier estrategia de PG previa para limpiar el entorno
remove_pg_strategies -all
```

Nota. Acá se ven las configuraciones iniciales de `sintesis_fisica_top.tcl`, en donde se carga el entorno de configuración general, el script de floorplan y se eliminan estrategias de PG previas.

12.1.2. Etapa 2 síntesis del reloj

En esta etapa se realiza la síntesis del reloj, donde se optimiza la red de reloj del diseño para cumplir con los requisitos de rendimiento y consumo de energía. Se crea el reloj con un período definido por el diseñador. Luego se verifica la integridad de la red de reloj y se optimiza para mejorar la calidad de la señal de reloj en todo el diseño.

Después se procede a la síntesis del árbol de relojes, lo cual es una etapa crucial en todo diseño que utiliza reloj, garantizando que esta señal llegue de manera eficiente y oportuna a todos los elementos del diseño. Finalmente, se realiza una optimización del reloj y un chequeo del CTS, conocido como *clock tree synthesis*.

Cuadro 12.2. Etapa del reloj de `sintesis_fisica_top.tcl`

```
################################################
# ETAPA 2: Sintesis reloj
################################################
if { $CLOCK_SYNTHESIS == "TRUE" } {
    puts "Sintetizando relojes..."
    create_clock -period <periodo> -name clk [get_nets -design [
    current_block] {clk}]
    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
} else {
    puts "Clock synthesis deshabilitada."
}
```

Nota. En este cuadro se pueden ver los comandos utilizados en la etapa de síntesis del reloj en `sintesis_fisica_top.tcl`.

12.1.3. Etapa 3 ruteo y verificaciones previas al *routing*

La etapa 3 consiste en la verificación de la capacidad de ruteo del diseño, asegurando que todas las conexiones puedan ser realizadas sin violar las reglas de diseño. Se verifica el ruteo, incluyendo puertos bloqueados por *power gating* (PG), estos puertos bloqueados pueden llegar a causar problemas si se encuentran bloqueados durante el ruteo.

Después se realiza una verificación general del diseño antes del ruteo, asegurando que el diseño esté listo para la siguiente etapa del proceso de síntesis física. Se ajusta el nivel de verbosidad durante el ruteo para obtener información detallada sobre el proceso y se realizan las verificaciones necesarias.

Finalmente, se puede encontrar un archivo comentado de reglas de antena, el cual puede ser descomentado si se desea utilizar reglas específicas para la verificación de antena durante el proceso de síntesis física, sin embargo este es un archivo designado para ICC, este es un archivo que se tiene de referencia por lo que se debe de buscar un archivo adecuado para ICC2 si se desea utilizar esta funcionalidad.

Cuadro 12.3. Etapa de ruteo y verificaciones previas al *routing* de `sintesis_fisica_top.tcl`

```
################################################
# ETAPA 3: RUTEO Y VERIFICACIONES PREVIAS AL ROUTING
################################################

# Verifica la capacidad de ruteo, incluyendo puertos bloqueados por PG
check_routability -check_pg_blocked_ports true

# Verificacion general del diseno antes del ruteo
check_design -checks pre_route_stage

# Ajusta el nivel de verbosidad durante el ruteo
set_app_options -name route.common.verbose_level -value 1
check_design -checks pre_route_stage
set_app_options -name route.common.verbose_level -value 0

# Archivo opcional de reglas de antena
#source -echo ../../../Librerias/Runset/antennaRule_n65_9lm.tcl
# report_app_options route.detail.*antenna*
```

Nota. En esta etapa se verifican aspectos importantes para garantizar las rutas y otras cuestiones importantes al momento de realizar el *routing*.

12.1.4. Etapa 4 *routing*

En la etapa de ruteo ya se hacen las conexiones físicas entre los diferentes bloques del diseño. Primero se realiza un ruteo automático global, el cual establece las rutas principales para las conexiones del diseño. Luego se verifica las rutas generadas para asegurar que cumplan con las reglas de diseño y restricciones establecidas, esto es esencial para evitar problemas en el diseño.

Consecuentemente, se agregan vías redundantes para mejorar la confiabilidad del diseño, estas vías adicionales nos ayudan a garantizar que las conexiones sean robustas y puedan manejar posibles errores o fallas dentro del propio diseño. Finalmente, se realiza un ruteo detallado incremental con DRC inicial, lo cual permite optimizar las rutas y asegurar que el diseño cumpla con todas las reglas de diseño y restricciones establecidas.

Cuadro 12.4. Etapa de *routing* `sintesis_fisica_top.tcl`

```
################################################
# ETAPA 4: ROUTING (RUTEO DEL DISENO)
################################################

# Ruteo automatico global
route_auto

# Verifica rutas generadas
check_routes

# Agrega vias redundantes para mejorar confiabilidad
add_redundant_vias -effort high

# Ruteo detallado incremental con DRC inicial
route_detail -incremental true -initial_drc_from_input true
```

Nota. En esta etapa se realiza el *routing* de forma automática por la herramienta, chequeos, agregar vías y un ruteo detallado.

12.1.5. Etapa 5 creación de *fillers* en *core* y *I/O ring*

En esta etapa se realiza la creación de *fillers* en el *core* y el *I/O ring* del diseño. Primero se debe de verificar la legalidad de la colocación antes de empezar a insertar los *fillers*, asegurando que el diseño cumpla con las reglas de diseño, espacios disponibles y otras restricciones.

Se agrega una sección de comandos comentados que sirven como ejemplos de cómo obtener *fillers* disponibles en las librerías, esto para que el usuario pueda personalizar los *fillers* a utilizar según sus necesidades.

Luego se empieza por la creación de *fillers* en el *I/O ring*, utilizando celdas de referencia específicas para este propósito. Este comando utiliza como referencia las guías de I/O definidas previamente en el *floorplan*. Y se indican los *fillers* a utilizar. Después, se procede a la creación de *fillers* en la región de celdas estándar (core), utilizando celdas de referencia específicas para este propósito. Se pueden utilizar diferentes tipos de *fillers* según el propósito de diseño, y se incluyen ejemplos alternativos comentados para que el usuario pueda personalizar los *fillers* a utilizar según sus necesidades.

Continuando con la conexión de los fillers a redes de *power/ground* de manera automática. La eliminación de *fillers* que presenten violaciones físicas, y un chequeo de legalidad después de las diferentes inserciones de *fillers* para asegurar que el diseño cumpla con las reglas de diseño.

Cuadro 12.5. Etapa de creación de *fillers* `sintesis_fisica_top.tcl`

```
##################################################
# ETAPA 5: CREACION DE FILLERS EN CORE Y IO RING
##################################################

# Verifica la legalidad de la colocacion antes de insertar fillers
check_legality

# Los comandos siguientes son ejemplos de como obtener fillers
    disponibles:
# get_lib_cells LIB_TEST/* -filter "design_type==core"
# get_lib_cells FillersWorkspace/* -filter "design_type==filler"
# get_lib_cells TSMCWorkspace|StandardWorkspace/* -filter "
    design_type==filler"

# Crea fillers en el IO ring
create_io_filler_cells \
  -io_guides [get_io_guides {ANILLO_IO.top ANILLO_IO.right
    ANILLO_IO.left ANILLO_IO.bottom}] \
  -reference_cells {PFILLER0005 PFILLER05 PFILLER1 PFILLER5
    PFILLER10 PFILLER20}

# Crea fillers en la region de celdas estandar (core)
# Fillers standard o core
create_stdcell_fillers -lib_cells [get_lib_cells {TSMCWorkspace|
    FillersWorkspace/FILL1 TSMCWorkspace|FillersWorkspace/FILL16
    TSMCWorkspace|FillersWorkspace/FILL2 TSMCWorkspace|
    FillersWorkspace/FILL32 TSMCWorkspace|FillersWorkspace/FILL4
    TSMCWorkspace|FillersWorkspace/FILL64 TSMCWorkspace|
    FillersWorkspace/FILL8}]

# Ejemplo alternativo (comentado):
# create_stdcell_fillers -lib_cells [get_lib_cells {TSMCWorkspace|
    StandardWorkspace/GFILL TSMCWorkspace|StandardWorkspace/GFILL2
    TSMCWorkspace|StandardWorkspace/GFILL3 }]

# Conecta los fillers a las redes de power/ground
connect_pg_net -automatic

# Elimina fillers con violaciones fisicas
remove_stdcell_fillers_with_violation

# Revision final de legalidad tras la insercion de fillers
check_legality
```

Nota. La etapa de la adición de *fillers* es crucial para asegurar que el diseño cumpla con las reglas de densidad y otras consideraciones de diseño. Estos se añaden tanto en la parte del *core* como en el *I/O ring*.

12.1.6. Etapa 6 guardado, *fillers* y chequeos de DRC/LVS

En esta etapa se guarda el bloque antes de iniciar la verificación de reglas de diseño (DRC). Luego se realiza la inserción de *metal fill* para el *back-end of line* (BEOL), lo cual ayuda a mejorar la uniformidad del diseño y cumplir con las reglas de densidad. Después se guarda nuevamente el bloque tras la inserción de BEOL. Luego se realiza la inserción de *metal fill* para el *front-end of line* (FEOL), utilizando un tipo de *filler* genérico para este propósito.

Cuadro 12.6. Etapa de guardado y chequeos de DRC/LVS
`sintesis_fisica_top.tcl`

```
##################################################
# ETAPA 6: GUARDADO, FILLERS Y CHEQUEOS DE DRC/LVS
##################################################

# Guarda el bloque antes de iniciar DRC
save_block $ndm_design_library:$CIRCUIT_NAME

# Insercion de metal fill (Back-End Of Line)
signoff_create_metal_fill -mode add -foundry_fill_type beol

# Guarda nuevamente tras la insercion BEOL
save_block $ndm_design_library:$CIRCUIT_NAME

# Insercion de metal fill (Front-End Of Line)
signoff_create_metal_fill -mode add -foundry_fill_type feol
   -foundry_for_feol_fill generic

# Corre verificacion DRC y correccion automatica
signoff_check_drc
signoff_fix_drc

# Guarda bloque actualizado tras DRC
save_block $ndm_design_library:$CIRCUIT_NAME

# Verifica LVS (Layout vs. Schematic)
check_lvs

# signoff_check_drc  ;# Ejecucion adicional de DRC si se requiere

# Desactiva mensajes especificos de GDS
set_msg GDS-028 -level off
```

Nota. Guardado de bloques, inserción de *metal fill* para BEOL y FEOL, y chequeos de DRC/LVS.

El comando de *fillers* de BEOL se modificó basándose en la documentación encontrada en el manual de ICC2 [26], y por otro lado, la adición del comando para el FEOL se hizó mediante la investigación de la documentación también de ICC2,

la opción “génerica” se selecciona debido a que en la herramienta de ICC2 no hay opciones específicas para tecnologías de 65 nm, solo para nodos más nuevos.

Como últimos comandos se ejecuta la verificación de reglas de diseño (DRC) y se corrigen hasta donde es posible los errores, esto es efectuado por la herramienta de forma iterativa. Luego se guarda el bloque actualizado después de la corrección de DRC. Finalmente, se realiza una verificación de *layout vs schematic* (LVS) para asegurar que el diseño físico coincida con el esquema original.

12.1.7. Etapa 7 exportación de resultados

En nuestra fase de exportación de resultados, generamos los archivos finales necesarios para las verificaciones, implementaciones y futuras referencias del diseño. Primero, se genera el *netlist* de Verilog del diseño físico, lo cual es esencial para la verificación funcional. También generamos el GDSII del diseño, el cual es el formato estándar para la representación física de circuitos integrados, este es el archivo clave que nos pide IMEC para poder fabricar el diseño de nuestro circuito integrado.

En el comando `write_gds` es importante identificar que se está utilizando la librería, el nombre del circuito, la vista del diseño, la vista disponible de las celdas (en nuestro caso es *frame*) y también el directorio de salida. Esto lo podemos ver en el Cuadro 12.7.

Cuadro 12.7. Etapa de exportación de resultados
`sintesis_fisica_top.tcl`

```
##################################################
# ETAPA 7: EXPORTACION DE RESULTADOS (VERILOG Y GDS)
##################################################

# Genera el netlist Verilog del diseno fisico
write_verilog -include all ./Outputs/IO/$CIRCUIT_NAME.v

# Exporta el diseno a formato GDS
write_gds \
  -library $ndm_design_library \
  -design $CIRCUIT_NAME \
  -view design \
  -hierarchy all \
  -lib_cell_view frame \
  ./Outputs/$CIRCUIT_NAME.gds
```

Nota. En esta etapa se exportan los resultados finales del diseño, incluyendo el netlist Verilog y el archivo GDSII, crucial para poder mandar a fabricar nuestro diseño en un futuro con el apoyo de IMEC en la fabrica de TSMC.

12.1.8. Etapa 8 finalización

En la etapa final del *script* `sintesis_fisica_top.tcl`, se realiza la visualización del diseño final en la interfaz gráfica de IC Compiler II. Esto permite al diseñador inspeccionar visualmente el diseño, verificar su integridad y realizar cualquier ajuste necesario antes de finalizar el proceso de síntesis física, si fuese necesario.

Cuadro 12.8. Etapa de finalización
`sintesis_fisica_top.tcl`

```
##################################################
# ETAPA 8: FINALIZACION
##################################################

echo "Finalizo la sintesis fisica!"
start_gui
```

Nota. La finalización de la etapa de la síntesis física incluye la visualización del diseño final en la interfaz gráfica de IC Compiler II.

12.2. *Script* `setup.tcl`

Este *script* se encarga de realizar la configuración inicial del entorno de diseño, incluyendo la carga de bibliotecas y la definición de variables necesarias para el proceso de síntesis física. Este es el primer *script* que se ejecuta en el proceso de síntesis física, y es fundamental para asegurar que todas las configuraciones estén correctamente establecidas antes de proceder con las etapas posteriores del diseño. Si se ve algún tipo de alerta o aviso por parte de la herramienta, se recomienda tomar nota y buscar mediante apoyo o en la documentación cómo solucionarlo, ya que puede llegar a afectar el proceso de síntesis física. A continuación se presenta y se explica el contenido del *script* de configuración inicial.

12.2.1. Limpieza de archivos previos

La primer sección del *script* de configuración se encarga de limpiar cualquier archivo previo que nos pueda llegar a causar confusión y para evitar que se acumulen archivos que ya no son necesarios para el proceso de síntesis física. Esto incluye la eliminación de archivos temporales, archivos de chequeo de diseño y otros archivos generados en ejecuciones anteriores del proceso de síntesis física.

Limpiar archivos es crucial para mantener un entorno de diseño organizado, lo que facilita la gestión del proyecto y reduce la posibilidad de errores debido a archivos obsoletos, haciéndolo tal y como se visualiza en el Cuadro 12.9.

Cuadro 12.9. Limpieza de archivos previos en `setup.tcl`

```
#================================================
# Limpieza de archivos previos
#================================================
sh clear
sh rm -f check_design*
sh rm -f default-*
sh rm -f *ems
```

Nota. Aquí limpiamos cualquier archivo previo que pueda causar confusión en el proceso de síntesis física.

12.2.2. Configuración de rutas y archivos base

Cuadro 12.10. Configuración de rutas y archivos base en `setup.tcl`

```
# ================================================
# Configuracion de rutas y archivos base
# ================================================
set command_log_file "./Logs/command.log"

set DESIGN_REF_PATH "/home/nanoelectronica/Desktop/Folder_de_Trabajo
    /TSMC_65nm/9M_9t/Librerias"

set TECH_FILE        "${DESIGN_REF_PATH}/tech/tsmcn65_9lmT2.tf"
set MAP_FILE         "${DESIGN_REF_PATH}/tlu_and_map/star.map_9M"
set TLUPLUS_FILE "${DESIGN_REF_PATH}/tlu_and_map/cln65lp_1p09m+
    alrdl_typical_top2.tluplus"

set NDM_REFERENCE_LIB_DIRS  " \
  ${DESIGN_REF_PATH}/lib/ndm/TSMCWorkspace.ndm"

set LIBRARY_FILES "${NDM_REFERENCE_LIB_DIRS}"

#Configure input paths (logic synthesis outputs) en synthesis files
file mkdir ./Inputs/
file copy -force "/ruta_archivo_mapeado/archivo_mapped.v" "./Inputs/
    "
file copy -force "/ruta_archivo_mapeado/archivo_mapped.sdc" "./
    Inputs/"
```

Nota. Aquí se configuran las rutas y archivos base necesarios para el proceso de síntesis física, aparte de la copia de los archivos de entrada necesarios del proceso de síntesis lógica.

En esta sección se configuran las rutas y archivos base necesarios para el proceso de síntesis física. Esto incluye la definición de variables como nuestro entorno o ruta de

diseño, en donde se encuentran las librerías. También se definen las rutas del archivo de tecnología, el archivo de mapeo y el archivo TLUPLUS.

Se sigue por la definición de las librerías NDM de referencia, estas creadas en la etapa de librerías con la herramienta de Library Manager. Finalmente, se copian los archivos de entrada necesarios para el proceso de síntesis física, como el netlist mapeado en Verilog y el archivo SDC, desde la ubicación de salida del proceso de síntesis lógica a la carpeta de entradas del proceso de síntesis física. Se debe de procurar que los archivos copiados sean los correctos para el diseño que se esté trabajando.

12.2.3. Configuración de potencia y capas

En esta sección se realiza la configuración de potencia y capas necesarias para el proceso de diseño, básicamente definimos los nodos de poder que son: VDD/VSS. Y también se definen como variables las capas mínimas y máximas de ruteo que se utilizarán en el diseño, se debe de procurar que estas capas coincidan con la tecnología que se esté utilizando y con parámetros que se verán luego en el *floorplan*.

Cuadro 12.11. Configuración de potencia y capas en `setup.tcl`

```
# ================================================
# Configuracion de potencia y capas
# ================================================
set MW_POWER_NET     "VDD"
set MW_POWER_PORT    "VDD"
set MW_GROUND_NET    "VSS"
set MW_GROUND_PORT   "VSS"

set MIN_ROUTING_LAYER "M1"
set MAX_ROUTING_LAYER "M9"
```

Nota. Configuración de nodos de poder y capas mínimas y máximas de ruteo en `setup.tcl`.

12.2.4. Librerías de referencia

En esta sección se carga las librerías de referencia necesarias para el proceso de síntesis física. Estas librerías tienen definiciones de celdas, y los `.db` correspondientes para nuestro diseño en la tecnología de 65 nm. Además, se configura el número máximo de núcleos que se utilizarán durante el proceso de síntesis física, esto es importante para optimizar el rendimiento y la eficiencia del proceso de diseño.

Cuadro 12.12. Configuración de librerías de referencia en `setup.tcl`

```
# ================================================
# Librerias de referencia
# ================================================
lappend search_path "${DESIGN_REF_PATH}/lib/db_nldm"

set_app_var link_library [glob ${DESIGN_REF_PATH}/lib/db_nldm/*.db]

set_host_options -max_cores 12
```

Nota. Configuración de las librerías de referencia y el número máximo de núcleos en `setup.tcl`.

12.2.5. Creación de la librería

Para continuar con el diseño debemos ejecutar la creación de una librería de diseño específica para nuestro proyecto. Esta librería contendrá todas las celdas y componentes necesarios para el diseño en la tecnología de 65 nm, y esta se asocia al archivo de tecnología previamente definido. Estas librerías como se verá en el futuro son importantes para la correcta síntesis física del diseño, y el flujo que propone Synopsys dentro de sus herramientas.

A la librería se le puede colocar el nombre que el usuario desee, en este caso se le colocó `LIB_TEST` como nombre genérico para evitar confusiones al momento de reutilizar este *script* en futuros diseños. Colocando también la librería NDM de referencia que se definió previamente.

Cuadro 12.13. Creación de la librería en `setup.tcl`

```
# ================================================
# Creacion de la libreria
# ================================================

set ndm_reference_library ${NDM_REFERENCE_LIB_DIRS}
set ndm_design_library LIB_TEST
puts "Creando libreria: $ndm_design_library"

sh rm -rf $ndm_design_library
create_lib -technology $TECH_FILE -ref_libs $LIBRARY_FILES ${
   ndm_design_library}
```

Nota. Creación de la librería de diseño en `setup.tcl`.

12.2.6. Lectura del diseño lógico y restricciones

En esta parte del *script* se realiza la lectura del diseño lógico y las restricciones necesarias para el proceso de síntesis física. Se define el nombre del archivo de entrada que contiene el diseño lógico, en este caso es `circuito_mapped`. Se coloca el nombre de esta manera para que el usuario pueda reutilizar este *script* en futuros diseños sin tener que modificar mucho el código, porque en realidad se puede sintetizar cualquier diseño siempre y cuando se coloquen los archivos correctos en la carpeta de entradas.

Cuadro 12.14. Lectura del diseño lógico y restricciones en `setup.tcl`

```
# ================================================
# Lectura del diseno logico y restricciones
# ================================================
set INPUT_name "circuito_mapped"

read_verilog ./Inputs/$INPUT_name.v
read_sdc -echo ./Inputs/$INPUT_name.sdc

link_block
report_design_mismatch -verbose
```

Nota. Lectura del diseño lógico y restricciones en `setup.tcl`, se leen archivos clave como el archivo de diseño lógico y el archivo de restricciones.

12.2.7. Configuración de parasíticos

En esta sección se realiza la configuración de los parasíticos para el proceso de síntesis física, por lo que se lee el archivo TLUPLUS y el archivo de mapeo previamente definidos. Estos archivos contienen información crucial sobre los parasíticos que afectan el rendimiento del diseño, como capacitancias, resistencias e inductancias asociadas a las conexiones físicas del circuito.

Estos archivos son importantes de incluir en una librería especialmente para que el diseño sea lo más realista posible, y se puedan hacer optimizaciones basadas en estos parasíticos. Finalmente, se genera un reporte de la librería con los parasíticos incluidos, lo cual permite al diseñador revisar y verificar la configuración de los parasíticos antes de proceder con las etapas posteriores del diseño.

Cuadro 12.15. Configuración de parasíticos en `setup.tcl`

```
# ================================================
# Configuracion de parasiticos
# ================================================
read_parasitic_tech -tlup $TLUPLUS_FILE -layermap $MAP_FILE
set_parasitic_parameters -early_spec $TLUPLUS_FILE -late_spec
    $TLUPLUS_FILE

report_lib -parasitic_tech ${ndm_design_library}
```

Nota. Configuración de parasíticos en `setup.tcl`, etapa en donde se leen archivos clave para definir los parasíticos del diseño.

12.2.8. Configuración de capas y enrutamiento

En esta sección se realiza la configuración de las capas y el enrutamiento necesarios para el proceso. Esto lo hacemos dentro de las opciones de la aplicación, definiendo parámetros para las vias redundantes y el modo de inserción de vias redundantes. Estas configuraciones son importantes para optimizar el enrutamiento del diseño, mejorando la confiabilidad y el rendimiento del circuito. Finalmente, se establece una opción para la verificación de reglas de diseño (DRC) durante el flujo.

Cuadro 12.16. Configuración de capas y enrutamiento en `setup.tcl`

```
# ================================================
# Configuracion de capas y enrutamiento
# ================================================

set_app_options -list { \
    route.common.post_detail_route_redundant_via_insertion high \
    route.common.concurrent_redundant_via_mode insert_at_high_cost \
}

set_app_options -name signoff.fix_drc.check_drc -value global
```

Nota. Configuración de capas y enrutamiento en `setup.tcl`, etapa en donde se definen las opciones de enrutamiento y verificación de diseño.

12.2.9. Variables de automatización

Para automatizar el proceso y evitar los posibles errores y confusiones al momento de reutilizar este *script* en futuros diseños, se definen variables clave que controlan aspectos importantes del diseño. Estas variables incluyen la opción de síntesis de reloj, las dimensiones de diseño (longitud del lado y desfase del *core*), las cuales se

acomodan al tamaño que nos indica IMEC, de 1 mm x 1 mm, y el desplazamiento del *core* para ajustarse a las especificaciones del diseño.

También se definen los nombres del circuito y del anillo de *I/O*, los cuales son importantes para identificar el diseño dentro de la herramienta. El nombre del circuito debe ser el mismo que el módulo con mayor jerarquía dentro del Verilog, conocido como *top module*. Finalmente, se ajustan las dimensiones del polígono basándose en las variables definidas anteriormente, asegurando que el diseño cumpla con las especificaciones requeridas.

Cuadro 12.17. Variables de automatización en `setup.tcl`

```
# ================================================
# Variables de automatizacion
# ================================================
set CLOCK_SYNTHESIS FALSE
set SIDE_LENGTH {620 620}
set CORE_OFFSET 190

set CIRCUIT_NAME "IO_circuito"
set ANILLO_NAME "ANILLO_IO"

# Ajuste de dimensiones del poligono
set x1 [expr {$CORE_OFFSET + 10}]
set y1 [expr {$CORE_OFFSET + 10}]
set adjusted_polygon_x [expr {[lindex $SIDE_LENGTH 0] + $CORE_OFFSET
    - 10}]
set adjusted_polygon_y [expr {[lindex $SIDE_LENGTH 1] + $CORE_OFFSET
    - 10}]
set POLYGON_DIMENSIONS "$adjusted_polygon_x $adjusted_polygon_y"

puts "Clock synthesis enabled: $CLOCK_SYNTHESIS"
puts "Circuit name: $CIRCUIT_NAME"
```

Nota. Variables de automatización en `setup.tcl`, etapa en donde se definen variables clave para controlar aspectos importantes del diseño.

12.2.10. Creación de carpetas de salida y salidas lógicas

Cuadro 12.18. Creación de carpetas de salida e inputs lógicos en `setup.tcl`

```
# ================================================
# Creacion de carpetas de salida e inputs logicos
# ================================================
file mkdir ./Outputs/IO/
file mkdir ./Outputs/DRC/
file mkdir ./Outputs/Fill/
file mkdir ./Outputs/Antena/
```

Nota. Creación de carpetas de salida e inputs lógicos en `setup.tcl`, etapa en donde se crean las carpetas necesarias para almacenar los resultados del diseño.

En esta sección del setup creamos las carpetas necesarias para almacenar los resultados del diseño. Estas carpetas incluyen `IO`, `DRC`, `Fill` y `Antena`, cada una destinada a almacenar diferentes tipos de resultados generados durante el proceso de síntesis física. En `DRC` podremos encontrar los resultados de la verificación de diseño. En `Fill` se almacenarán los resultados relacionados con la inserción de *fillers*, y en `Antena` se guardarán los resultados de la verificación de antena si es que sea realiza mientras se ejecute este flujo. Y como última carpeta está `IO`, en donde se almacenarán los resultados generales del diseño, como el *netlist* Verilog y el archivo GDSII, aquí podremos ver estos archivos con las celdas ya insertadas y el diseño finalizado.

12.2.11. Configuración de *runsets* y rutas de ejecución

En esta sección vamos a empezar configurando las rutas de ejecución para las diferentes etapas de chequeo del diseño, chequeo del DRC, correciones del DRC y creación de *metal fill*, asegurándonos de que los resultados se guarden en las carpetas correspondientes creadas previamente.

Cuadro 12.19. Configuración de *runsets* y rutas de ejecución en `setup.tcl`

```
# ================================================
# Configuracion de runsets y rutas de ejecucion
# ================================================
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/}}
set_app_options -list {signoff.create_metal_fill.run_dir {./Outputs/Fill/}}

set_app_options -list {signoff.check_design.runset {../../../Librerias/Runset/ICVLN65S_9M_6X2Z.26_2a}}
set_app_options -list {signoff.check_drc.runset {../../../Librerias/Runset/ICVLN65S_9M_6X2Z.26_2a}}

set_app_options -list {signoff.create_metal_fill.runset {../../../Librerias/Runset/Dummy_BEOL_ICV_65nm.26_2a}}
set_app_options -list {signoff.create_metal_fill.base_layer_runset {../../../Librerias/Runset/Dummy_FEOL_ICV_65nm.26_1a}}

set_app_options -list {signoff.antenna.custom_runset_file {../../../Librerias/Runset/ICVN65S_9M_ANT.26_2a}}

set_app_options -list {signoff.create_metal_fill.fix_density_errors true}
```

Nota. Para configurar los *runsets* y las rutas de ejecución en `setup.tcl`, se definen las rutas de ejecución para las diferentes etapas de chequeo del diseño, asegurándose de que los resultados se guarden en las carpetas correspondientes.

Continuando, vemos en el Cuadro 12.19 la configuración necesaria en las opciones de la aplicación para la definición del *runset* para chequeo y verificación de reglas. También se agregan los *runsets* para la creación de *metal fill* tanto para BEOL como para FEOL. Además, se define el *runset* de antena. Finalmente, se activa una opción para la corrección de los errores de densidad durante la creación de *metal fill*.

12.3. *Script* `floorplan.tcl`

El *script* de *floorplan* es una parte crucial del proceso de síntesis física, ya que define la estructura y organización del diseño físico del circuito integrado. Este *script* establece las dimensiones del *core*, la ubicación de los bloques de celdas estándar, la colocación de pines y otras configuraciones esenciales para el diseño físico. A continuación, se presenta y se explica el contenido del *script* de *floorplan*.

12.3.1. Etapa 1 *floorplan* inicial

Cuadro 12.20. Etapa de *floorplan* inicial en `floorplan.tcl`

```
################################################
# ETAPA 1: FLOORPLAN INICIAL
################################################
# Configura el estilo de macro (bloques grandes)
set_app_options -name plan.macro.style -value freeform

# Inicializa el floorplan con los parametros definidos
initialize_floorplan \
  -use_site_row \
  -site unit \
  -keep_all \
  -side_length $SIDE_LENGTH \
  -core_offset $CORE_OFFSET

shape_blocks
set_attribute [get_site_defs unit] symmetry {X Y}

# Configura opciones para legalizacion y colocacion de macros
set_app_options -name place.legalize.enable_advanced_legalizer
    -value true
set_app_options -name place.coarse.fix_hard_macros -value false
set_app_options -name plan.place.auto_create_blockages -value auto

# Realiza una colocacion preliminar del floorplan para evaluar
    congestion
create_placement -floorplan -congestion -effort high
    -congestion_effort high
check_legality

# Define las capas permitidas para los pines del bloque
set_block_pin_constraints -self -allowed_layers {M2 M3 M4 M5 M6}
# Coloca los pines automaticamente
place_pins -self
```

Nota. Etapa de *floorplan* inicial en `floorplan.tcl`, donde se configuran las opciones iniciales para el diseño físico del circuito integrado.

El *floorplan* inicial es esencial para tener un diseño que sea una base sólida para lo que seguirá después. Es muy probable que los ajustes más importantes se vayan a estar haciendo en la etapa de colocación y planeación, ya que a menudo, los errores pueden surgir por estas etapas. Por lo tanto, es crucial que el *floorplan* inicial esté bien definido y optimizado para evitar problemas en las etapas posteriores del diseño.

Viendo el 12.20, se empieza por configurar el estilo de los *macros* los cuales son bloques grandes dentro de un diseño [26]. Luego comienza la inicialización del *floorplan* utilizando los parámetros previamente definidos en el *script* de configuración, como

las dimensiones del *core* y el desplazamiento del *core* y el sitio del diseño. Continuando con la simetría de los bloques de celdas estándar.

Las opciones de legalización y la colocación de las macros también son configuradas para asegurar que se pueda verificar o legalizar de manera correcta que nuestro *floorplan* se haga eficientemente y de manera correcta. Procede con la colocación preliminar del *floorplan* para evaluar la congestión del diseño, lo cual es crucial para identificar posibles problemas de enrutamiento en etapas posteriores, los mismos comandos permiten la modificación de qué tanto se desea eliminar la congestión. Finalmente, se definen las capas permitidas para los pines del bloque y se colocan automáticamente los pines, asegurando que estén ubicados en las capas adecuadas para un rendimiento óptimo.

12.3.2. Etapa 2 limpieza de estructuras previas

La etapa 2 del *script* de *floorplan* se enfoca en la limpieza de estructuras previas relacionadas con la gestión de potencia y tierra. Si queremos generar nuevos diseños y evitar que se traslapen configuraciones previas, es importante eliminar cualquier configuración anterior que pueda interferir con el nuevo diseño.

Cuadro 12.21. Etapa de limpieza de estructuras previas en `floorplan.tcl`

```
################################################
# ETAPA 2: LIMPIEZA DE ESTRUCTURAS PREVIAS DE POWER/GROUND
################################################

remove_pg_strategies -all
remove_pg_patterns -all
remove_pg_regions -all
remove_pg_via_master_rules -all
remove_pg_strategy_via_rules -all
```

Nota. Etapa de limpieza de estructuras previas en `floorplan.tcl`, donde se eliminan las configuraciones previas relacionadas con la gestión de potencia y tierra.

12.3.3. Etapa 3 creación del anillo de *I/O* y *pads* de alimentación

En la etapa 3 se crea el anillo de I/O y los *pads* de alimentación necesarios para el diseño. El anillo de I/O es crucial para la conexión del circuito con el mundo exterior, mientras que los *pads* de alimentación aseguran que el circuito reciba la energía necesaria para su funcionamiento. El valor del tamaño de las esquinas se escoge de acuerdo con el tamaño del diseño y las necesidades de enrutamiento.

Creamos las celdas de esquina necesarias para el anillo de I/O, y los *pads* de alimentación para VDD y VSS, asegurando que el diseño tenga una estructura sólida

para la gestión de potencia y tierra. El nombre de los *pads* de alimentación debe coincidir con los nombres definidos en las librerías de referencia para asegurar que la herramienta coloque los que se definen en nuestro diseño.

Cuadro 12.22. Etapa de creación del anillo de I/O y *pads* de alimentación en `floorplan.tcl`

```
##################################################
# ETAPA 3: CREACION DEL ANILLO IO Y PADS DE ALIMENTACION
##################################################

# Crea el anillo IO (perimetro)
create_io_ring -name $ANILLO_NAME -corner_height 120

# Crea las celdas de esquina (corners)
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

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

Nota. Etapa de creación del anillo de *I/O* y *pads* de alimentación en `floorplan.tcl`, donde se definen las estrategias de enrutamiento y se crean los *pads* de alimentación necesarios.

12.3.4. Etapa 4 configuración de redes de alimentación

Configurar las redes de alimentación es esencial para asegurar que cada sección del circuito reciba la energía necesaria para su funcionamiento. En esta etapa, se resuelven las redes de *power/ground* del diseño, creando las nets de VDD y VSS. Definimos la capa mínima y máxima basándonos en el diseño de 9 metales.

Y agregamos los *pads* de alimentación a las guías de *I/O* del anillo, asegurando que estén correctamente ubicados para una distribución eficiente de la energía. Finalmente, se colocan los *pads* en el diseño, completando la configuración de las redes de alimentación.

Cuadro 12.23. Etapa de configuración de redes de alimentación en `floorplan.tcl`

```
################################################
# ETAPA 4: CONFIGURACION DE REDES DE ALIMENTACION
################################################

# Resuelve las redes de power/ground del diseno
resolve_pg_nets

# Crea las nets de VDD y VSS
create_net -power VDD
create_net -ground VSS

set_ignored_layers -min_routing_layer M1 -max_routing_layer M9

# Agrega los pads a las guias IO del anillo (si no tienen ubicacion
    definida)
add_to_io_guide [get_io_guides $ANILLO_NAME.left] PVDD*
add_to_io_guide [get_io_guides $ANILLO_NAME.right] PVSS*
place_io
```

Nota. Etapa de configuración de redes de alimentación en `floorplan.tcl`, donde se definen las redes de alimentación y se asignan los *pads* de alimentación a las celdas correspondientes.

12.3.5. Etapa 5 creación del *power ring* (anillo de PG)

La creación del *power ring* es una etapa crucial en el diseño físico, ya que este anillo proporciona una ruta eficiente para la distribución de energía a lo largo del circuito. En esta etapa, se define el patrón del anillo de PG, especificando las capas horizontales y verticales, así como sus anchos y espacios. Aquí se mantuvieron las capas originales con las que se trabajaba en el diseño de 180 nm, se hicieron pruebas intentando modificar las capas en las que se crea el anillo, pero no se obtuvieron resultados significativos.

Luego, se define la estrategia del anillo en el *core*, especificando el patrón y las redes asociadas (VDD y VSS), así como el desplazamiento del anillo dentro del *core*. Finalmente, se compila el anillo de PG, asegurando que esté correctamente integrado en el diseño físico.

Cuadro 12.24. Etapa de creación del *power ring* en `floorplan.tcl`

```
################################################
# ETAPA 5: CREACION DEL POWER RING (ANILLO DE PG)
################################################

# Define patron del anillo PG (Power/Ground)
create_pg_ring_pattern ring_pattern \
 -horizontal_layer M2    -horizontal_width {2} -horizontal_spacing
    {2}    -vertical_layer M3 -vertical_width {2} -vertical_spacing
    {2}

# Define la estrategia del anillo en el core
set_pg_strategy core_ring    -pattern {{name: ring_pattern}    {
    nets: {VDD VSS}} {offset: {1 1}}} -core

# Compila el anillo de PG
compile_pg -strategies core_ring
```

Nota. Etapa de creación del *power ring* en `floorplan.tcl`, donde se define el patrón del anillo de *PG* y se establece la estrategia de conexión en el *core*.

12.3.6. Etapa 6 conexión del anillo de I/O con los *pads* de alimentación

La conexión del anillo de I/O con los *pads* de alimentación es una etapa crítica para asegurar que el circuito reciba la energía necesaria de manera eficiente. En esta etapa, se define el patrón de conexión entre los *pads* y el anillo, utilizando un tipo de conexión dispersa (*scattered*) para optimizar la distribución de energía, En realidad, se debe de intentar probar diferentes configuraciones y hacer más simulaciones en la etapa de verificación para ver cuál es la que mejor se adapta a nuestro diseño. El diseño que se intenta ejectuar en este trabajo de graduación no obedece a restricciones de alto desempeño o eficiencia energética, por lo que se optó por una configuración sencilla y que funcione.

Luego, se configura una opción para tratar los *pads* como macros, lo que facilita su gestión durante el proceso de síntesis física. Se establece la estrategia de conexión de los *pads* al anillo, especificando las macros involucradas y el patrón de conexión definido previamente. Se define una regla de vía específica para esta conexión, asegurando que las vías utilizadas partan de las capas de metal adecuadas. Continuando con la compilación de la conexión entre los *pads* y el anillo, asegurando que estén correctamente integrados en el diseño físico. Finalmente, se conectan explícitamente los pines de VDD y VSS a las redes correspondientes, y se genera un reporte de las celdas de alimentación para verificar su correcta configuración.

Cuadro 12.25. Etapa de conexión del anillo IO con los pads de alimentación en `floorplan.tcl`

```
################################################
# ETAPA 6: CONEXION DEL ANILLO IO CON LOS PADS DE ALIMENTACION
################################################

# Define patron de conexion entre pads y el anillo (tipo scattered)
create_pg_macro_conn_pattern hm_pattern \
  -pin_conn_type scattered_pin \
  -layers {M2 M3} \
  -nets {VDD VSS} \
  -pin_layers {M2}

# Configura opcion para tratar pads como macros
set_app_options -name plan.pgroute.treat_pad_as_macro -value true

# Estrategia de conexion de pads a anillo
set_pg_strategy macro_conn \
  -macros [get_cells {PVDD* PVSS*}] \
  -pattern {{name: hm_pattern} {nets: {VDD VSS}}}

# Define regla de via para esta conexion
set_pg_strategy_via_rule macro_conn_via_rule \
  -via_rule { {{{strategies: macro_conn}} {{existing: all} {layers:
   M3}} {via_master: default}} {{intersection: undefined} {
   via_master: NIL}} }

# Compila la conexion pad anillo
compile_pg -strategies macro_conn -via_rule macro_conn_via_rule -tag
    test

# Conecta explicitamente pines de VDD/VSS a las redes
connect_pg_net -net VDD [get_pins -physical_context */VDD*]
connect_pg_net -net VSS [get_pins -physical_context */VSS*]
connect_pg_net -automatic

# Reporte de celdas de alimentacion
report_cells -power
```

Nota. Etapa de conexión del anillo IO con los pads de alimentación en `floorplan.tcl`, donde se define el patrón de conexión entre los pads y el anillo, y se establece la estrategia de conexión.

12.3.7. Etapa 7 creación de la malla de *power/ground* en *core*

En la etapa 7 se crea la malla de *power/ground* dentro del *core* del diseño. La malla es esencial para distribuir la energía de manera uniforme a lo largo del circuito, asegurando que todas las secciones reciban la potencia necesaria para su funcionamiento. Esta se hace de tipo *mesh* o enrejado, ya que es una de las configuraciones

más comunes y efectivas para la distribución de energía en circuitos integrados. Se colocan las coordenadas del polígono ajustadas previamente para definir el área dentro del *core* donde se creará la malla. Luego, se define la estrategia de malla dentro del *core*, especificando el patrón y las redes asociadas (VDD y VSS), así como las áreas de bloqueo para evitar interferencias con los macros.

Conectamos con las celdas estándar, después se compila la malla de *power/ground*, asegurando que esté correctamente integrada en el diseño físico. Finalmente, se une la malla con el anillo de PG.

Cuadro 12.26. Etapa de creación de la malla de *power/ground* en `floorplan.tcl`

```
################################################
# ETAPA 7: CREACION DE LA MALLA DE POWER/GROUND EN CORE
################################################

# Define patron de malla PG (mesh)
create_pg_mesh_pattern mesh_pattern -layers {{{ vertical_layer: M3}
    {width: 4.2}{ pitch: 42} {spacing: interleaving }}}

# Estrategia de malla dentro del core
set_pg_strategy mesh_strategy \
 -polygon "{190.000 190.000} {$adjusted_polygon_x
    $adjusted_polygon_y}" -pattern {{ pattern: mesh_pattern }{ nets:
    {VDD VSS }}} -blockage {macros: all}

# Conexion para celdas estandar
create_pg_std_cell_conn_pattern std_cell_pattern
set_pg_strategy std_cell_strategy -core -pattern {{pattern:
    std_cell_pattern} {nets: {VDD VSS}}}

# Compila las estrategias de malla
compile_pg

# Une (merge) la malla con el anillo PG
merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {M2 M3}
```

Nota. Etapa de creación de la malla de *power/ground* en `floorplan.tcl`, donde se define el patrón de malla y se establece la estrategia de conexión para las celdas estándar.

12.3.8. Etapa 8 verificación y ajuste de *placement*

En la etapa final del *script* de *floorplan*, se realiza la verificación y ajuste del *placement* del diseño. Esta etapa es crucial para asegurar que todos los componentes estén correctamente ubicados y que el diseño cumpla con las restricciones físicas y de diseño establecidas. Primero revisamos la consistencia del diseño antes de ejecutar la colocación, asegurándonos de que no haya errores o inconsistencias que puedan afectar la

calidad del diseño. Luego, se legaliza la colocación final, ajustando la ubicación de los componentes para cumplir con las restricciones de diseño y optimizar el rendimiento del circuito.

Cuadro 12.27. Etapa de verificación y ajuste de *placement* en `floorplan.tcl`

```
################################################
# ETAPA 8: VERIFICACION Y AJUSTE DE PLACEMENT
################################################

# Revisa consistencia del diseno antes del placement
check_design -checks pre_placement_stage
check_design -checks physical_constraints

# Legaliza la colocacion final
legalize_placement
```

Nota. Etapa de verificación y ajuste de *placement* en `floorplan.tcl`, donde se revisa y ajusta la colocación de los componentes en el diseño.

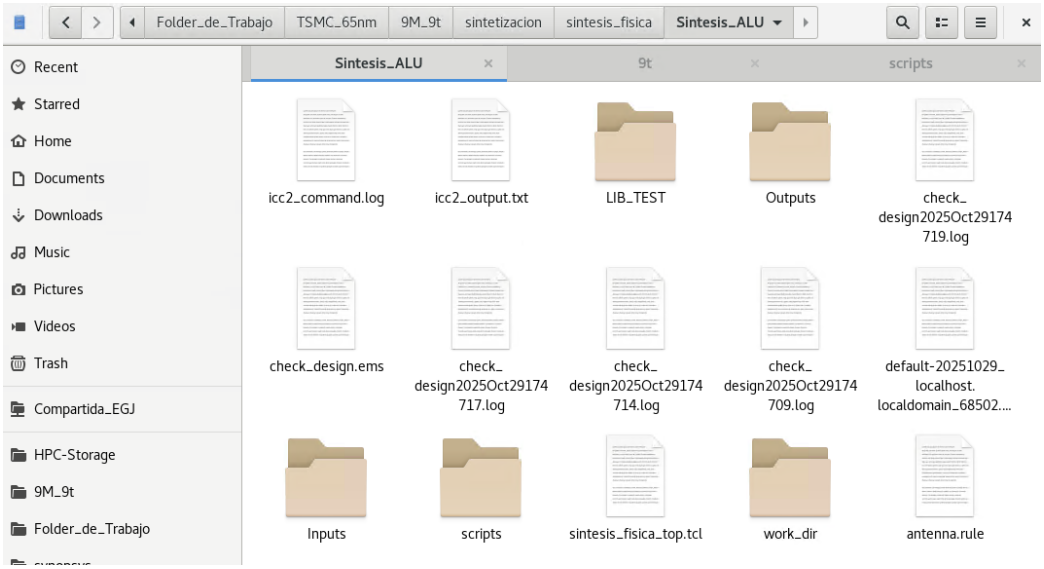
12.4. Ejecución de los *scripts*

Para ejecutar los *scripts* de síntesis física, primero debemos de asegurarnos de que todos los archivos necesarios estén en sus respectivas carpetas, y tener la carpeta destinada para el circuito que deseemos. Con esto vamos a dirigirnos a la carpeta o directorio del circuito específico que deseamos sintetizar. Por ejemplo, podemos navegar a la carpeta de `9M_9t`, esencial para este trabajo de graduación, y dentro de esta carpeta, acceder a la subcarpeta de `sintetizacion`, y luego a la subcarpeta de `sintesis_fisica`, donde se encuentran las carpetas para la síntesis de nuestro circuitos. Aquí vamos a encontrar las carpetas de los circuitos con el formato de `Sintesis_circuito`, donde `circuito` es el nombre del circuito que deseamos sintetizar. Un ejemplo de directorio se muestra a continuación:

- `/9M_9t/sintetizacion/sintesis_fisica/Sintesis_circuito`

En la Figura 17 se muestra un ejemplo de cómo se vería el directorio de síntesis física del cual queremos abrir su terminal. Tras la ejecución exitosa de los *scripts*, se generarán varios archivos de salida en las carpetas designadas, incluyendo el netlist Verilog y el archivo GDSII del diseño finalizado, carpetas como la de *outputs*, `LIB_TEST`, algunos archivos de *logs*, entre otros.

Figura 17. Directorio de síntesis física para abrir la terminal



Nota. Directorio de síntesis física, mostrando la estructura de carpetas y archivos necesarios para ejecutar los *scripts* de síntesis física.

Una vez dentro de la carpeta del circuito que deseamos sintetizar, podemos proceder a ejecutar los *scripts* de síntesis física utilizando la herramienta IC Compiler II. Para ello, abrimos la terminal y ejecutamos el siguiente comando:

```
icc2_shell
```

Esto abrirá la interfaz de línea de comandos de IC Compiler II. A continuación, cargamos y ejecutamos el *script* principal de síntesis física, escribiendo y corriendo en la terminal:

```
source sintesis_fisica_top.tcl
```

12.5. Abrir un bloque para visualizar un diseño

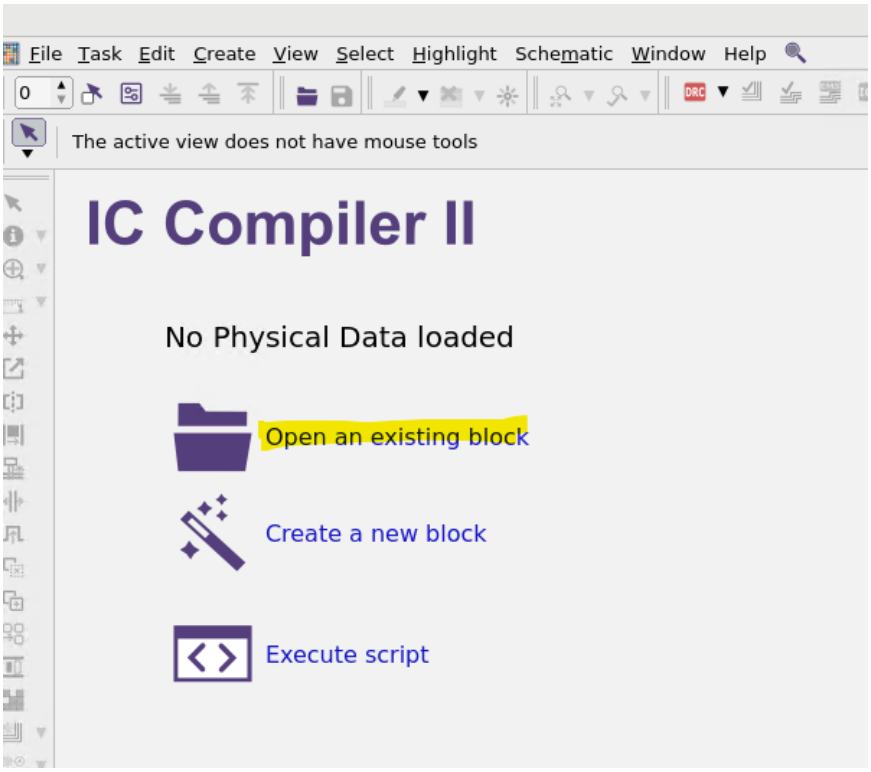
Para abrir un bloque y visualizar un diseño sintetizado en IC Compiler II, primero debemos asegurarnos de que el diseño haya sido sintetizado correctamente, estar en el directorio desde donde se ejecuta `sintesis_fisica_top.tcl` y que los archivos necesarios estén disponibles. Una vez que tenemos el diseño listo, y en el caso que ya hayamos cerrado la herramienta, podemos abrir nuevamente IC Compiler II ejecutando el siguiente comando en la terminal:

```
icc2_shell -gui
```

Esto abrirá la interfaz gráfica de IC Compiler II. A continuación, para cargar el diseño sintetizado, podemos presionar el botón de *open an existing block* tal y como

se ve en la Figura 18.

Figura 18. Abrir un bloque en IC Compiler II

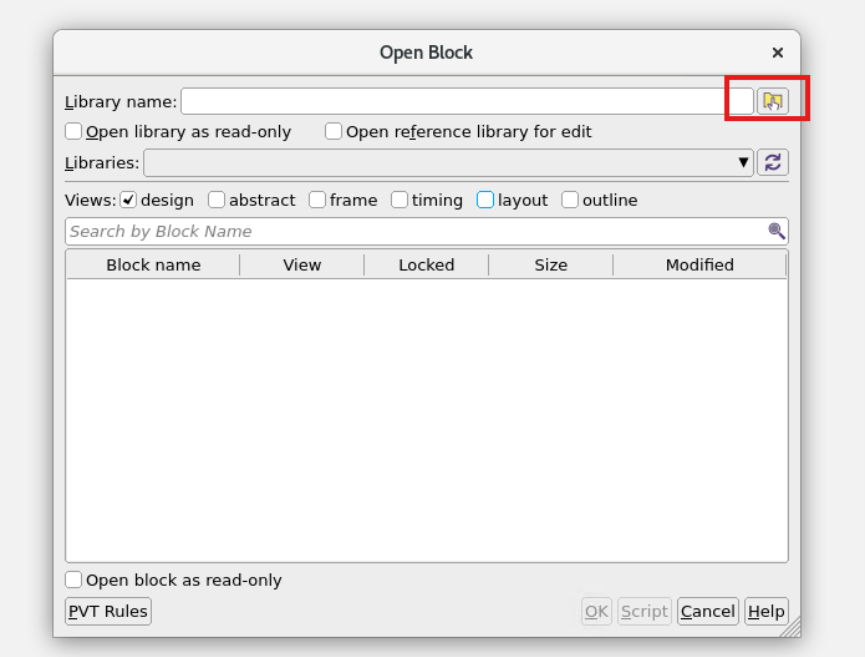


Nota. Abrir un bloque en IC Compiler II, mostrando la interfaz gráfica y las opciones para cargar y visualizar el diseño sintetizado.

A continuación, se abrirá una ventana emergente donde debemos navegar hasta la ubicación del archivo del bloque sintetizado. En la Figura 19 se muestra un ejemplo de qué botón presionar. Seleccionamos el archivo correspondiente y hacemos clic en *open* para cargar el diseño en la herramienta.

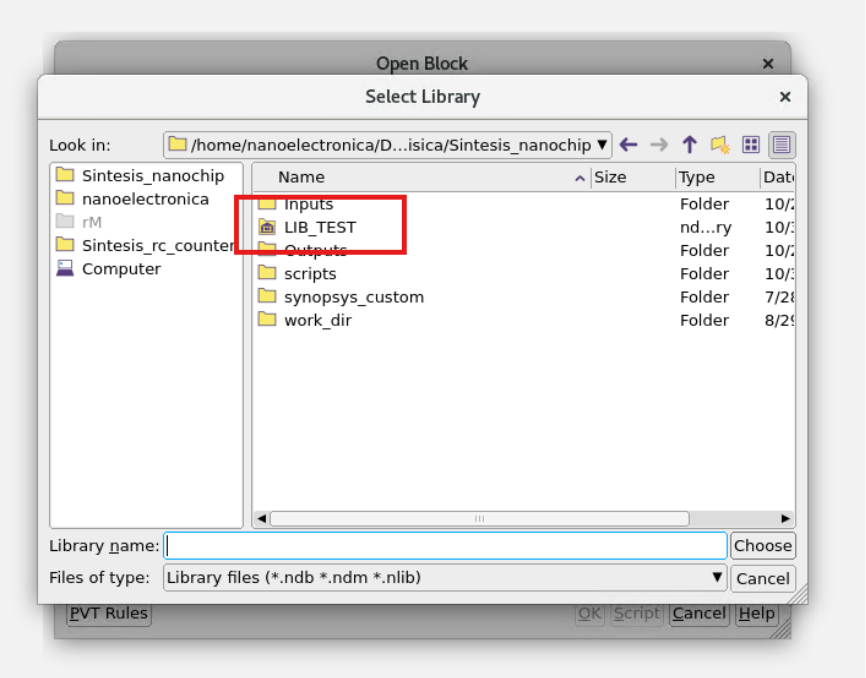
Ahora, desde la venta emergente seleccionamos `LIB_TEST`, la librería de diseño creada en el *script* de síntesis física como se ven en la Figura 20, y luego seleccionamos el bloque del circuito que deseamos visualizar.

Figura 19. Seleccionar el bloque sintetizado en IC Compiler II



Nota. Seleccionar el bloque sintetizado en IC Compiler II, mostrando la ventana emergente para navegar y seleccionar el archivo del bloque.

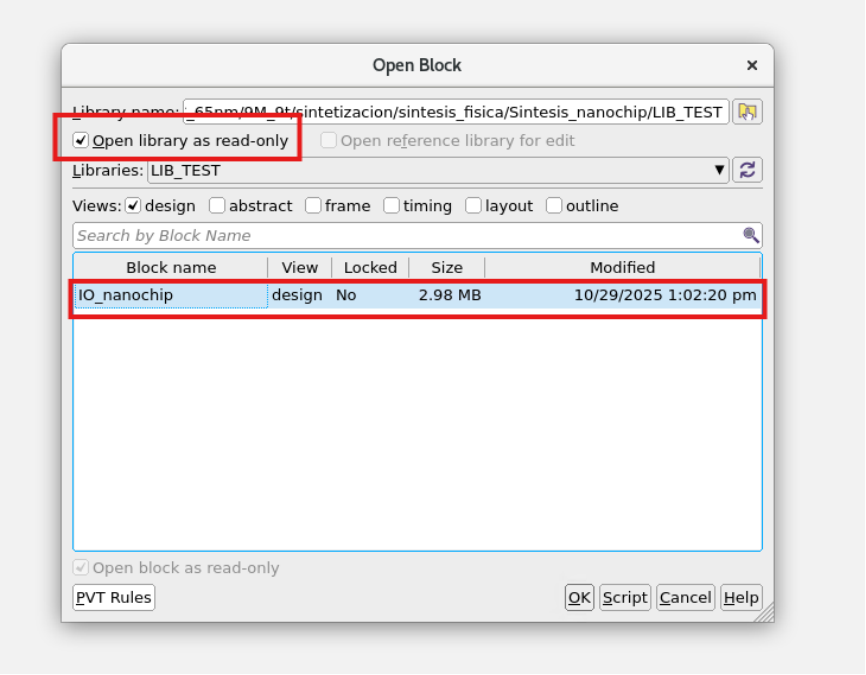
Figura 20. Seleccionar la librería de diseño en IC Compiler II



Nota. Seleccionar la librería de diseño en IC Compiler II, mostrando la ventana emergente para elegir la librería y el bloque del circuito.

Finalmente, hacemos clic en *open library as read-only* y doble clic en el nombre del bloque como se ve en la Figura 21. Y esto nos permite abrir el diseño sintetizado para poder visualizarlo y analizarlo.

Figura 21. Abrir el diseño sintetizado en IC Compiler II



Nota. Abrir el diseño sintetizado en IC Compiler II, mostrando la opción para abrir la librería y el bloque del circuito.

CAPÍTULO 13

Circuitos sintetizados en tecnología de 65 nm

En este capítulo se presentan los circuitos que fueron sintetizados utilizando la tecnología de 65 nm en y los resultados para cada uno de estos. Estos circuitos fueron seleccionados para evaluar el desempeño y la eficiencia del proceso de síntesis física implementado en esta tecnología avanzada.

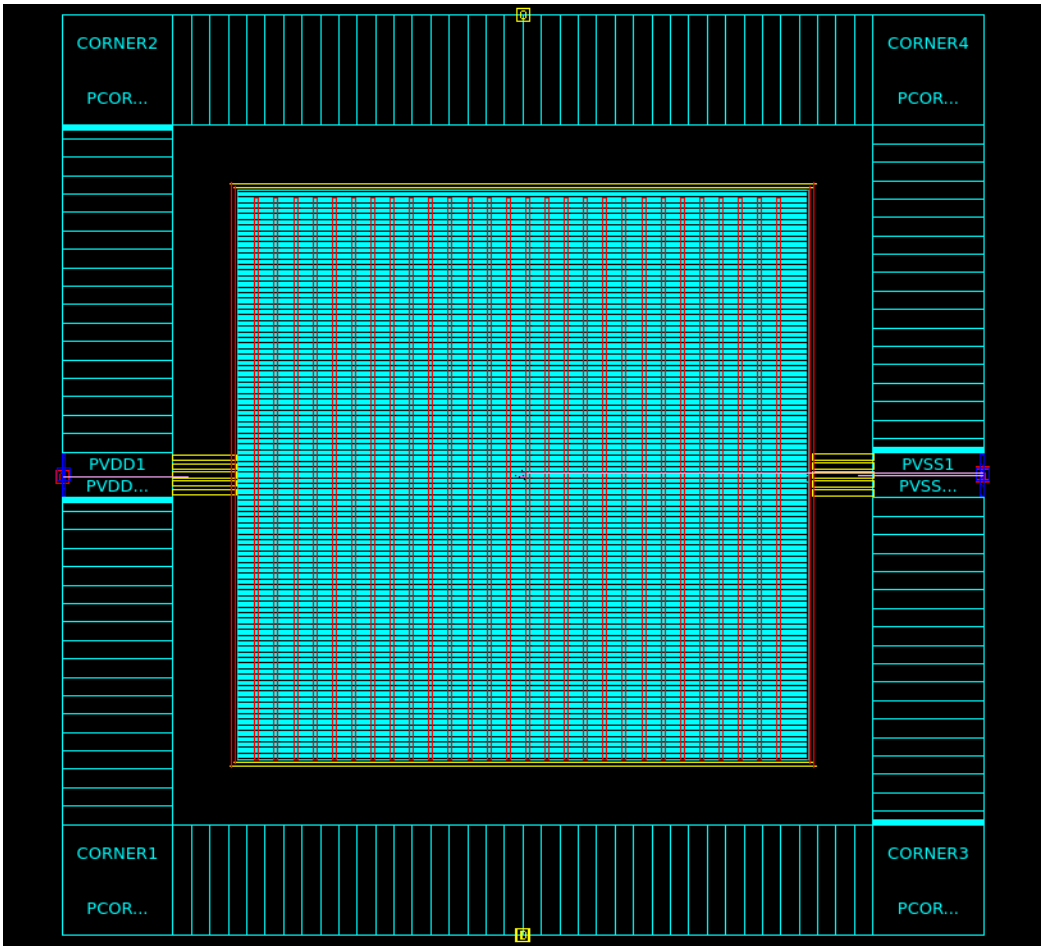
Se cuenta con tres circuitos sintetizados con *pads*, estos son: una compuerta NOT, una ALU con un oscilador de anillo y el nanochip "El Gran Jaguar". Además, se sintetizó otro circuito sin *pads*, el cual es un *ripple carry counter*. Se empezará presentando los resultados del circuito sin *pads*, seguido de los circuitos con *pads*. Esto porque en la etapa de síntesis lógica, se generaron estos cuatro circuitos para el caso de 9 metales y 9 *tracks*, y el único que no se generó con *pads* fue el *ripple carry counter*.

13.1. *Ripple carry counter*

El *ripple carry counter* es un circuito secuencial que cuenta en binario, incrementando su valor con cada pulso de reloj. Este tipo de contador es conocido por su simplicidad y facilidad de implementación, aunque puede ser menos eficiente en términos de velocidad debido a la propagación del acarreo a través de los flip-flops. En este trabajo, se sintetizó un *ripple carry counter* utilizando la tecnología de 65 nm, sin la inclusión de *pads* de entrada/salida. A continuación, se presentan los resultados obtenidos tras la síntesis física del *ripple carry counter*.

En la Figura 22 se muestra el diseño físico del *ripple carry counter* después de la síntesis física. El diseño incluye la colocación de las celdas estándar, rellenos, el enrutamiento de las conexiones y la integración de las redes de alimentación y tierra.

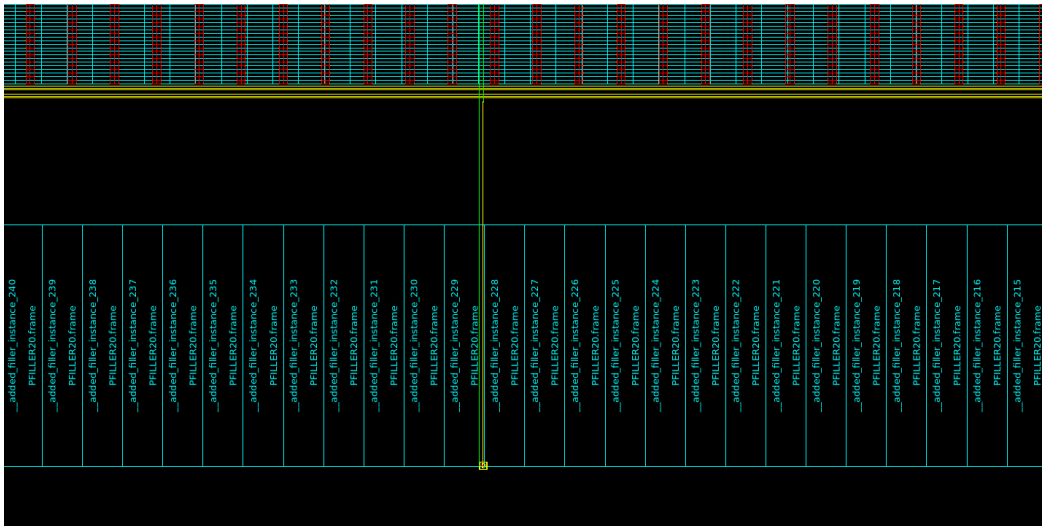
Figura 22. Diseño físico del *ripple carry counter* sintetizado en tecnología de 65 nm



Nota. Diseño físico del *ripple carry counter*, mostrando la colocación de las celdas estándar y el enrutamiento de las conexiones.

Para ver cómo se diferencia a los demás, basta con ver el que sería el anillo de entradas y salidas, acá veremos que no hay *pads* alrededor del diseño, ya que este circuito no los incluye. Si tuviésemos *pads* de I/O, estos estarían ubicados alrededor del perímetro del diseño, conectando el circuito con el mundo exterior y con nombres específicos como $OUT\ 1$, como se podrá ver en los otros circuitos los cuales sí incluyen *pads*. En la Figura 23 se muestra el anillo de entradas y salidas del circuito sintetizado para el *ripple carry counter*.

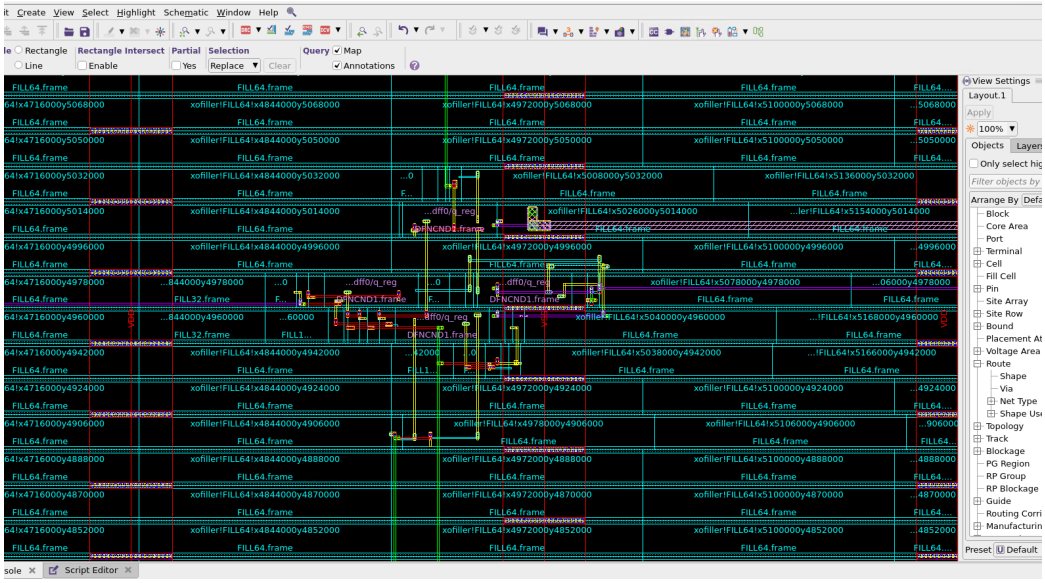
Figura 23. Anillo de entradas y salidas del *ripple carry counter* sintetizado en tecnología de 65 nm



Nota. Anillo de entradas y salidas del *ripple carry counter*, mostrando la ausencia de *pads* de I/O.

En la Figura 24 se puede ver el diseño del *ripple carry counter* desde más cerca, llegando a visualizar las celdas estándar que componen el diseño.

Figura 24. Diseño físico del *ripple carry counter* sintetizado en tecnología de 65 nm, mostrando las celdas estándar

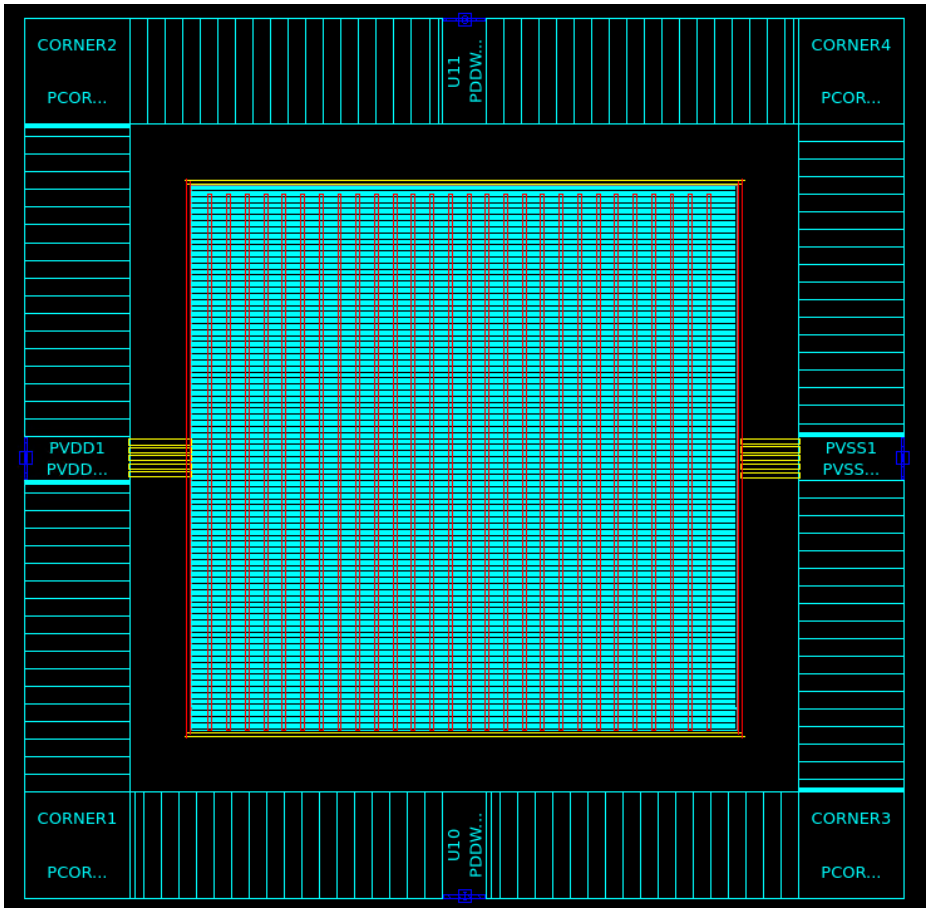


Nota. Diseño físico del *ripple carry counter*, mostrando la colocación de las celdas estándar y el enrutamiento de las conexiones.

13.2. Compuerta NOT

La compuerta NOT es un circuito lógico fundamental que invierte el valor de su entrada. Fue el circuito con el cual se inició el proceso de síntesis física en tecnología de 65 nm. Debido a su facilidad de síntesis tanto en términos de lógica como de física, se utilizó como un primer paso para familiarizarse con el flujo de trabajo y las herramientas necesarias para llevar a cabo la síntesis física en esta tecnología avanzada. A continuación, se presentan los resultados obtenidos tras la síntesis física de la compuerta NOT en la herramienta IC Compiler II.

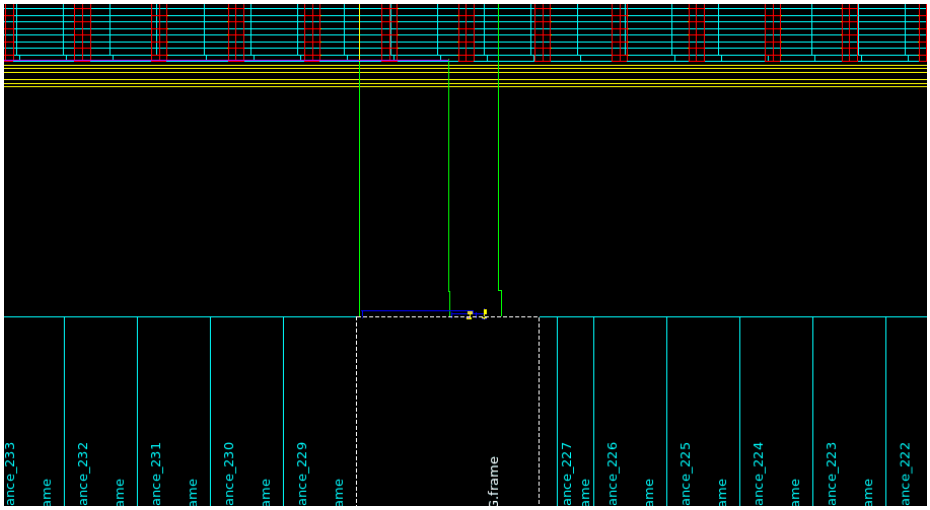
Figura 25. Diseño físico de la compuerta NOT sintetizada en tecnología de 65 nm



Nota. Diseño físico de la compuerta NOT, mostrando la colocación de las celdas estándar y el enrutamiento de las conexiones.

En la Figura 25 se muestra el diseño físico de la compuerta NOT después de la síntesis física. El diseño incluye la colocación de las celdas estándar, rellenos, el enrutamiento de las conexiones y la integración de las redes de alimentación y tierra. En la Figura 26 se muestra el anillo de entradas y salidas del circuito sintetizado para la compuerta NOT.

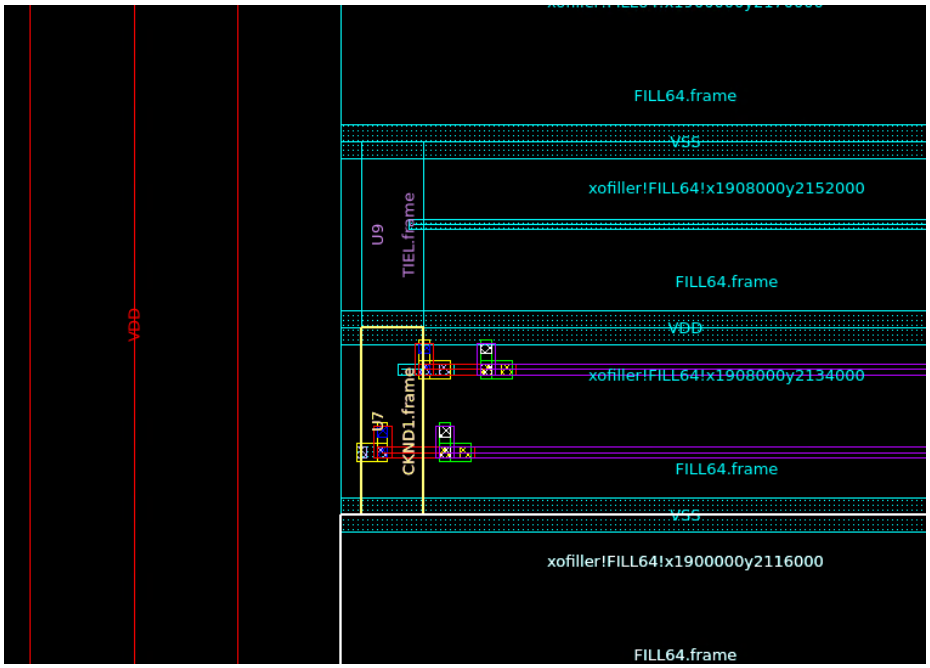
Figura 26. Anillo de entradas y salidas de la compuerta NOT sintetizada en tecnología de 65 nm



Nota. Anillo de entradas y salidas de la compuerta NOT, mostrando los *pads* de I/O.

Y en la Figura 27 se puede ver el diseño de la compuerta NOT desde más cerca, llegando a visualizar las celdas estándar que componen el diseño.

Figura 27. Diseño físico de la compuerta NOT sintetizada en tecnología de 65 nm



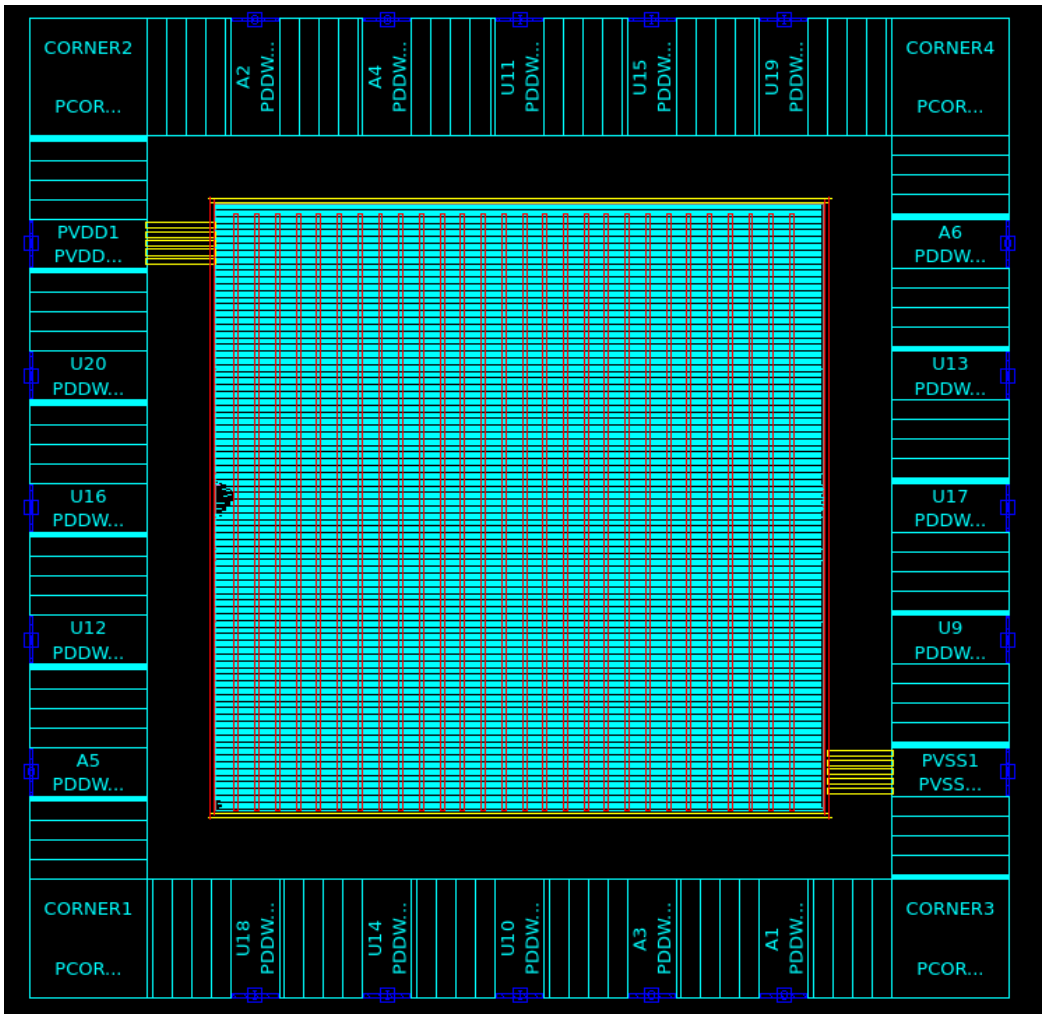
Nota. Diseño físico de la compuerta NOT, mostrando la colocación de las celdas estándar y el enrutamiento de las conexiones.

13.3. ALU con oscilador de anillo

Una unidad aritmética o lógica (ALU, por sus siglas en inglés) es un componente fundamental en los procesadores y sistemas digitales, se sintetizó una ALU que incluye un oscilador de anillo, el cual es un circuito que genera una señal de reloj mediante la retroalimentación de una serie de inversores [43]. A continuación, se presentan los resultados obtenidos tras la síntesis física de la ALU con oscilador de anillo en tecnología de 65 nm.

En la Figura 28 se muestra el diseño físico de la ALU con oscilador de anillo después de la síntesis física. El diseño incluye la colocación de las celdas estándar, rellenos, el enrutamiento de las conexiones y la integración de las redes de alimentación y tierra.

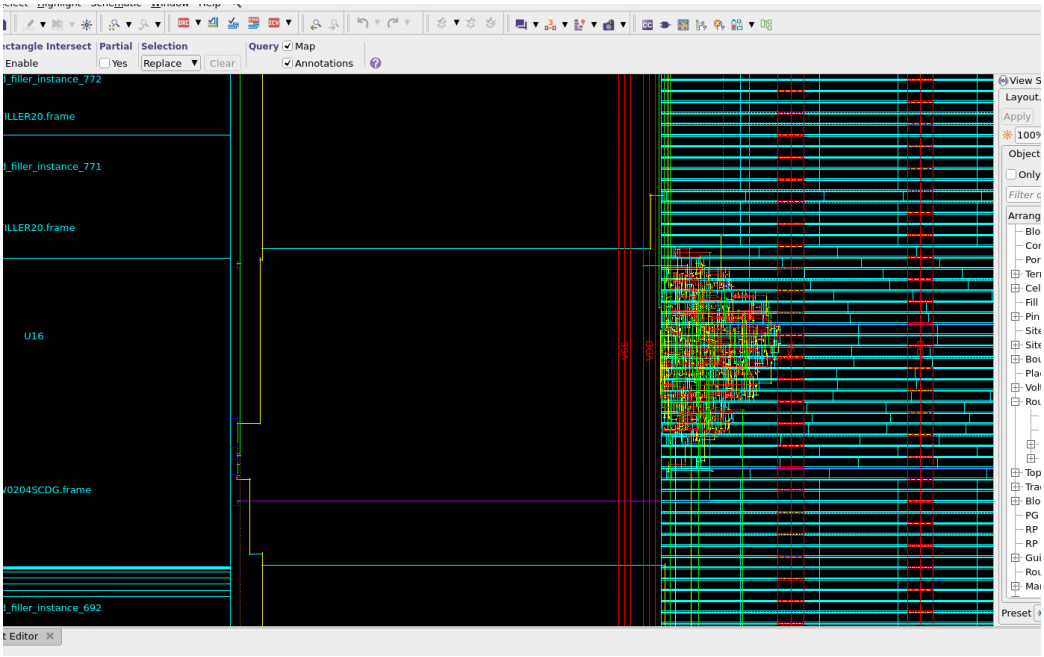
Figura 28. Diseño físico de la ALU con oscilador de anillo sintetizada en tecnología de 65 nm



Nota. Diseño físico de la ALU con oscilador de anillo, mostrando la colocación de las celdas estándar y el enrutamiento de las conexiones.

En la Figura 29 se muestra el anillo de entradas y salidas del circuito sintetizado para la ALU con oscilador de anillo, y algunas de las celdas que lo componen desde una vista más cercana.

Figura 29. Celdas y anillo de entradas y salidas de la ALU con oscilador de anillo sintetizada en tecnología de 65 nm



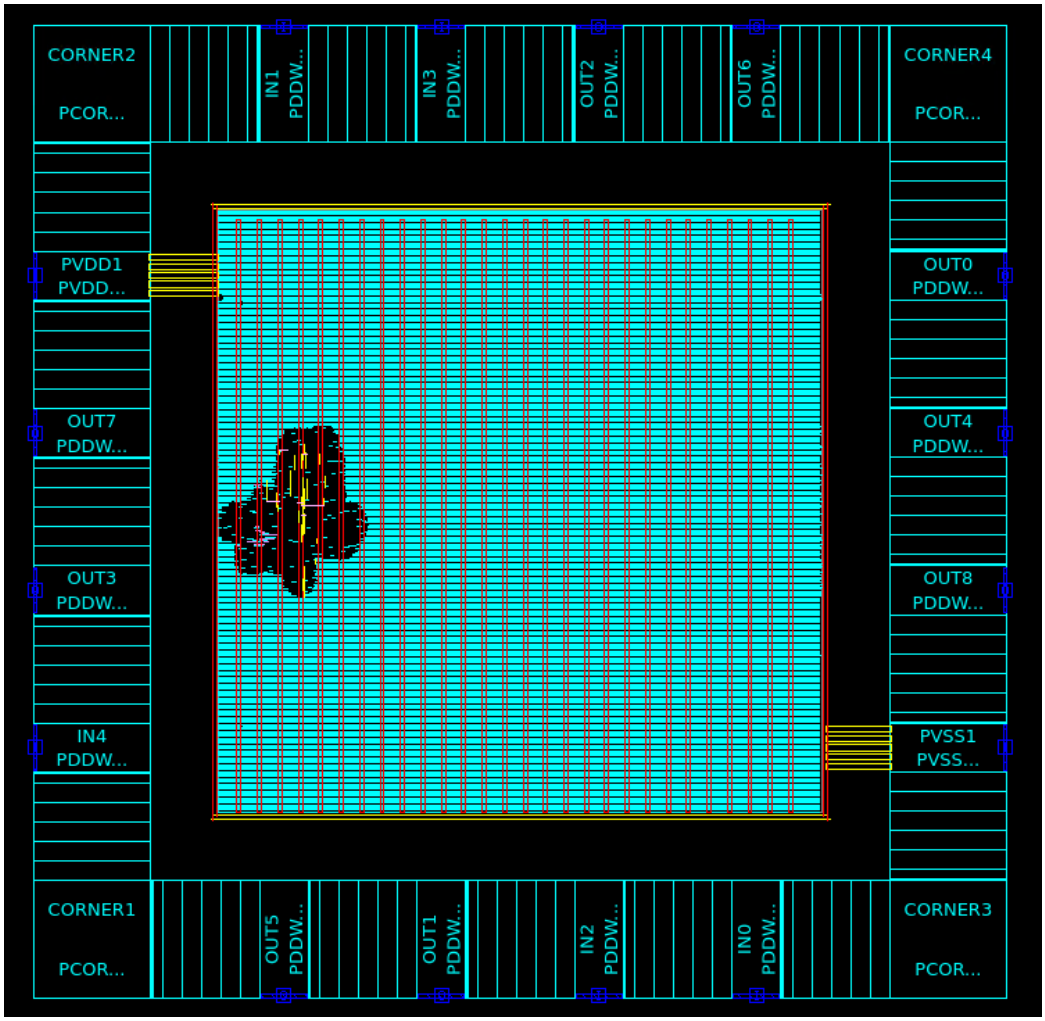
Nota. Celdas y anillo de entradas y salidas de la ALU con oscilador de anillo, mostrando las celdas estándar, rellenos y el enrutamiento de las conexiones.

13.4. Nanochip "El Gran Jaguar"

Este es el circuito principal en el cual se basa el trabajo de graduación, los circuitos anteriores son circuitos que nos permiten probar y analizar diferentes aspectos del diseño. Por otro lado, son circuitos que siempre y cuando se logren sintetizar, nos permiten demostrar que el flujo de síntesis física implementado es funcional, incluso escalando de 180 nm a tecnologías más avanzadas como la de 65 nm. A continuación, se presentan los resultados obtenidos tras la síntesis física del nanochip "El Gran Jaguar" en tecnología de 65 nm.

En la Figura 30 se muestra el diseño físico del nanochip "El Gran Jaguar" después de la síntesis física. El diseño incluye la colocación de las celdas estándar, rellenos, el enrutamiento de las conexiones y la integración de las redes de alimentación y tierra.

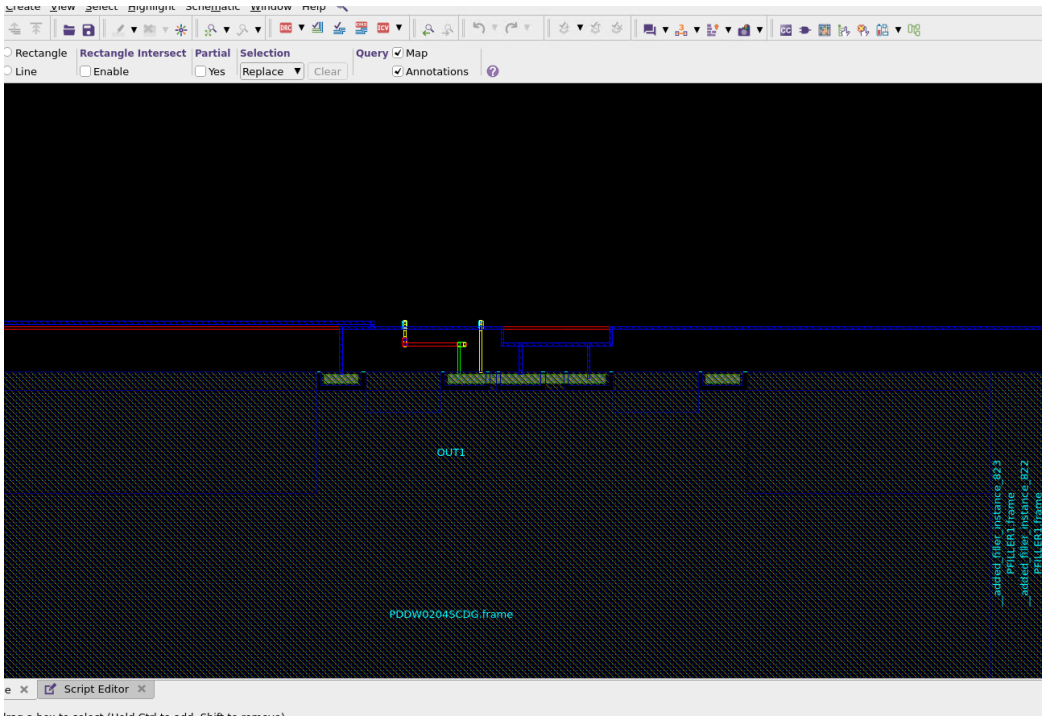
Figura 30. Diseño físico del nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm



Nota. Diseño físico del nanochip "El Gran Jaguar", mostrando la colocación de las celdas estándar, rellenos y el enrutamiento de las conexiones.

En la Figura 31 se muestra el anillo de entradas y salidas del circuito sintetizado desde una vista más cercana para el nanochip "El Gran Jaguar". Para poder visualizar la información que contiene la celda `.frame`, tenemos que aumentar el nivel de vista desde los ajustes de vista de IC Compiler II. Sin embargo, es de notar que no se posee una vista completa de las celdas, ya que las celdas estándar de tipo `.frame` solo contienen la información necesaria para enrutamiento y colocación de las celdas.

Figura 31. Anillo de entradas y salidas del nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm

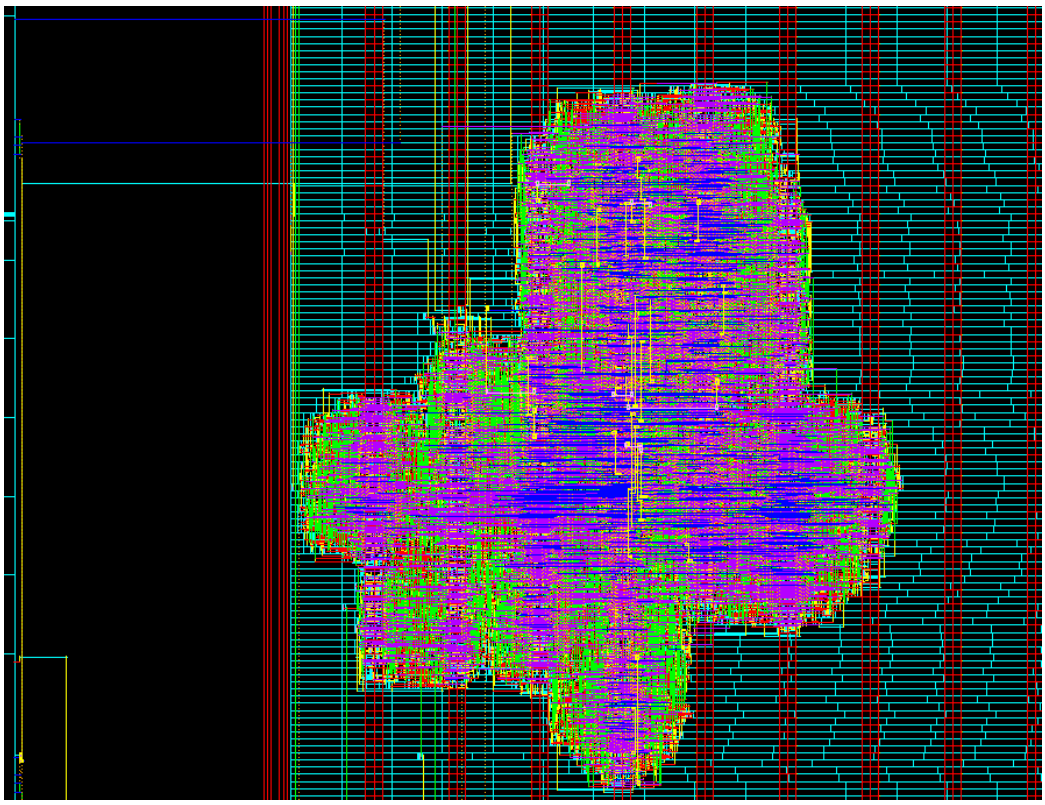


Nota. Anillo de entradas y salidas del nanochip "El Gran Jaguar" desde una vista más cercana, mostrando las celdas estándar con un nivel de vista mayor, mostrando las conexiones y el enrutamiento de las conexiones hacia las salidas.

El nanochip "El Gran Jaguar" es un circuito complejo que integra múltiples funcionalidades y componentes, lo que se refleja en su diseño físico detallado. En la Figura 32 se puede ver el diseño del nanochip "El Gran Jaguar" desde más cerca, llegando a visualizar las celdas estándar que componen el diseño.

Podemos ver a simple vista que este diseño se compone de muchas más celdas estándar que los otros circuitos sintetizados, lo que es lógico ya que este circuito es mucho más complejo que los demás, e incluso toma considerablemente más espacio en el *core* del diseño y tiempo para la herramienta de síntesis física poder completar todas las etapas del flujo de síntesis física.

Figura 32. Diseño físico del nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm, mostrando las celdas estándar, relleno y enrutamiento



Nota. Diseño físico del nanochip "El Gran Jaguar", mostrando la colocación de las celdas estándar, rellenos y el enrutamiento de las conexiones.

Para el caso del nanochip "El Gran Jaguar", al ejecutar el comando de reporte de diseño `report_design`, se obtiene la información resumida del diseño sintetizado. Obteniendo como tal los siguientes resultados:

- Área del diseño aproximada: $1000\mu m \times 1000\mu m = 1000000\mu m^2$
- Número de celdas estándar: 29,229 celdas
- Número de *pads*: 16 *pads*
- Número de vías: 68,725 vías
- Longitud total del enrutamiento: 160,313 μm
- Total de nodos: 4,633 nodos

CAPÍTULO 14

Verificaciones del proceso de síntesis física de 65 nm

Después de que se realiza el diseño es de suma importancia realizar las verificaciones necesarias para asegurarse de que el diseño cumple con las reglas de diseño y restricciones establecidas. Estas verificaciones se realizan utilizando la herramienta IC Validator de Synopsys, esta herramienta es capaz de correr en el mismo ambiente en el que se corre IC Compiler II, lo que facilita la integración del proceso de verificación dentro del flujo de síntesis física.

Sin embargo, en el caso de las verificaciones de antena o *layout vs schematic* se hacen en IC Validator puramente, saliendo del entorno de IC Compiler II. Y así tener un flujo más ordenado, para el interés de este trabajo se abarca la verificación de DRC y antena para los circuitos que cuentan con *pads* en tecnología de 65 nm con 9 capas de metal y *tracks*.

Aunque las verificaciones de DRC y de antena se lleguen a cumplir y el diseño esté libre de errores, es posible que otras pruebas como LVS, ERC o BND presenten posibles errores que comprometan el funcionamiento de nuestro diseño en un futuro. Por lo tanto, es crucial realizar todas las verificaciones necesarias para garantizar la integridad y funcionalidad del diseño antes de proceder a la fabricación del circuito integrado. Y mantener una constante comunicación con el equipo de diseño y verificación para resolver cualquier problema que pueda surgir durante el proceso de verificación.

14.1. Verificaciones de DRC

Las verificaciones de DRC son esenciales para garantizar que el diseño físico cumple con las reglas de diseño establecidas por el fabricante del proceso de fabricación. Estas reglas incluyen restricciones sobre el tamaño mínimo de las características, el espaciamiento entre ellas, la alineación y otras consideraciones críticas para asegurar la funcionalidad y la fiabilidad del circuito integrado. Cabe decir que como se menciona en capítulos anteriores, las reglas de diseño pueden variar dependiendo de la tecnología utilizada, en este caso, se trabajó con la tecnología de 65 nm y también de las opciones que se configuren tanto en la herramienta como en el *runset* que se utilice.

El proceso de resolución de errores se debe de acompañar de un análisis detallado de los errores que se generan, reivisiones de los manuales proporcionados por IMEC y TSMC, lectura de las recomendaciones de IMEC y TSMC, consulta de manuales de las herramientas, comunicación con IMEC y Synopsys, entre otros. Como se puede ver en el manual de IMEC [40], hay diferentes etapas de diseño que ellos pueden realizar por nosotros, como agregar el relleno de metal y difusiones corriendo los *runsets* si es que nuestro diseño ya solo cuenta con errores de densidad derivados del relleno.

Debido a la inclusión del *runset* y la incorporación de los comandos de chequeo en el flujo de la síntesis física dentro de IC Compiler II, se realiza de forma integrada. Para acceder a los reportes de DRC generados durante la síntesis física, se debe de acceder al siguiente directorio una vez estemos en el circuito que hayamos sintetizado:

- `/sintesis_fisica/Sintesis_circuito/Outputs/DRC/signoff_check_drc_-run/`

Dentro de esta carpeta, encontraremos varios archivos relacionados con las verificaciones de DRC, incluyendo los reportes detallados que indican si el diseño cumple o no con las reglas de diseño establecidas. Estos reportes son fundamentales para identificar y corregir cualquier violación de las reglas antes de proceder a la fabricación del circuito integrado. El archivo en donde podemos ver los resultados, errores y el detalle de los mismos posee la extensión de `.LAYOUT_ERRORS`.

14.1.1. Resultados de DRC para la compuerta NOT

Los resultados de las verificaciones de DRC para la compuerta NOT sintetizada en tecnología de 65 nm ayudaron a la correción de errores de densidad, vías y otros aspectos del diseño. A continuación, se presentan los resultados finales obtenidos tras la verificación de DRC para la compuerta NOT en la Figura 33.

Figura 33. Resultados de DRC para la compuerta NOT sintetizada en tecnología de 65 nm

```
                    LAYOUT ERRORS RESULTS: ERRORS

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

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

Library name:    LIB_TEST
Structure name:  IO_not
Generated by:    IC Validator RHEL64 W-2024.09-SP4.11451981 2025/02/26
Runset name:     /home/nanoelectronica/Desktop/Folder_de_Trabajo/TSMC_65nm/9M_9t/Librerias/Runset/
  ICVLN65S_9M_6X2Z.26_2a
User name:       nanoelectronica
Time started:    2025/10/30 11:47:30PM
Time ended:      2025/10/30 11:47:47PM
```

Nota. Visualización de los resultados de DRC para la compuerta NOT, en donde se muestra que posee errores de DRC que deben ser corregidos.

Estos errores se componen de dos tipos para el caso del circuito de la compuerta NOT, los cuales son: errores de densidad. En la Figura 34 se puede ver un resumen de los errores de DRC para la compuerta NOT, donde se observa la cantidad total de errores y su clasificación por tipo.

El primer error que se presenta es el de la ventana de densidad de las difusiones, ligado al *runset* FEOL y los comandos realizados durante la síntesis física dentro de este aspecto. Se ve que hay 169 violaciones de esta regla, en donde ventanas del tamaño definido en esta regla no cumplen con una densidad mínima del 20 %. El segundo error ahora marca mínimos de densidad menores a 0.1 % en el tamañó que indica la ventana.

Figura 34. Resumen con detalle de los errores de DRC para la compuerta NOT sintetizada en tecnología de 65 nm

```
                          ERROR SUMMARY

OD.DN.2 : Min. OD density over window 150 step 75
>= 20%
  density ..................................... 169 violations found.

PO.DN.2 : {OD OR DOD OR PO OR DPO } local density
(minimun) over window 20um x 20um stepping 10um >=
0.1%
  density ..................................... 436 violations found.
```

Nota. Se pueden ver los dos tipos de errores de DRC para la compuerta NOT, los cuales son errores de densidad en las difusiones. En total 605 errores de DRC.

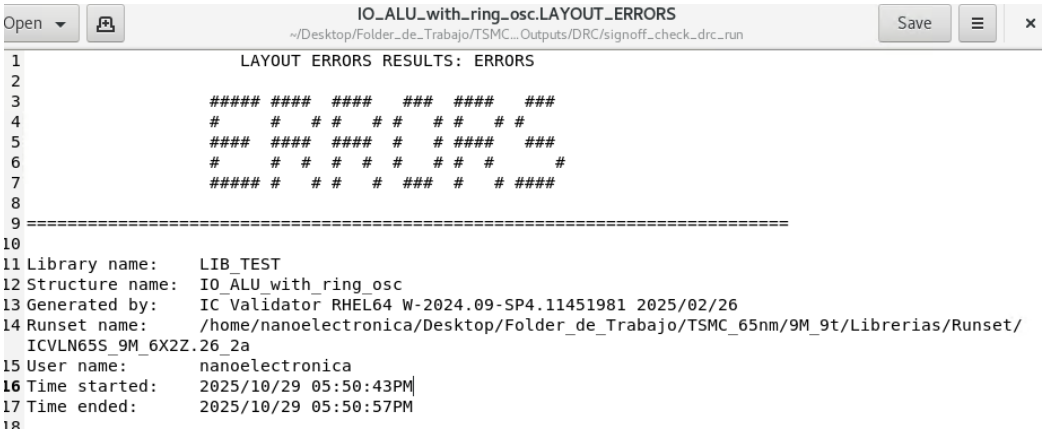
Partiendo originalmente de un diseño con 34,894 errores de DRC, se logró reducir a

605 errores de DRC tras las correcciones realizadas. Logrando una reducción de errores del 98.27 %, lo cual es un gran avance en la mejora del diseño y su cumplimiento con las reglas de diseño establecidas.

14.1.2. Resultados de DRC para la ALU

Los resultados de las verificaciones de DRC para la ALU con oscilador de anillo sintetizada en tecnología de 65 nm ayudaron a la correción de errores de densidad, vías y otros aspectos del diseño. A continuación, se presentan los resultados finales obtenidos tras la verificación de DRC para la ALU con oscilador de anillo en la Figura 35.

Figura 35. Resultados de DRC para la ALU con oscilador de anillo sintetizada en tecnología de 65 nm



Nota. Visualización de los resultados de DRC para la ALU con oscilador de anillo, en donde se muestra que posee errores de DRC que deben ser corregidos.

En la Figura 36 se puede ver un resumen de los errores de DRC para la ALU con oscilador de anillo, donde se observa la cantidad total de errores y su clasificación por tipo al igual que en todos los demás circuitos. Se obtuvieron los mismos errores y cantidad que en el caso de la compuerta NOT.

Figura 36. Resumen con detalle de los errores de DRC para la ALU con oscilador de anillo sintetizada en tecnología de 65 nm

```
                        ERROR SUMMARY

OD.DN.2 : Min. OD density over window 150 step 75
>= 20%
  density ..................................... 169 violations found.

PO.DN.2 : {OD OR DOD OR PO OR DPO } local density
(minimun) over window 20um x 20um stepping 10um >=
0.1%
  density ..................................... 436 violations found.
```

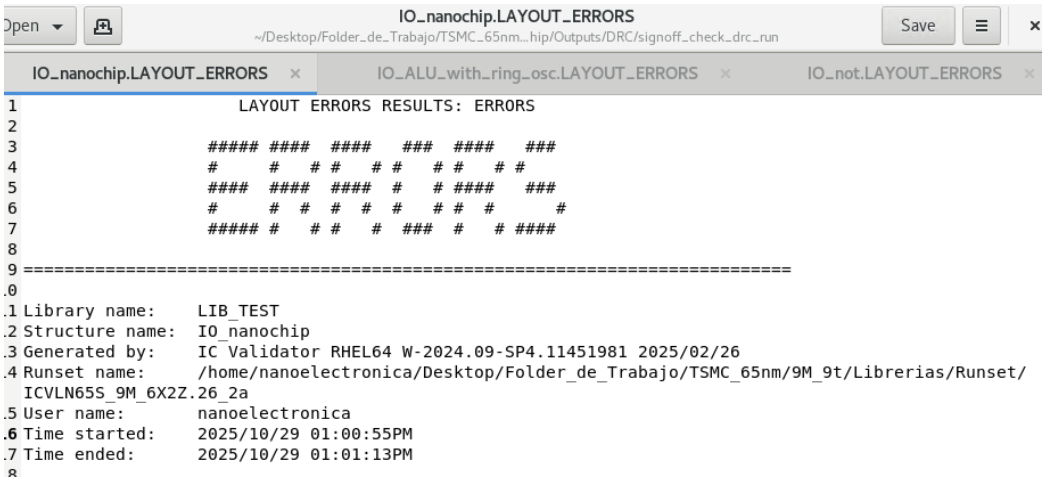
Nota. Se pueden ver los dos tipos de errores de DRC para la ALU con oscilador de anillo, los cuales son errores de densidad en las difusiones. Y en total 605 errores de DRC, al igual que la NOT.

Partiendo originalmente de un diseño con 34,894 errores de DRC, se logró reducir a 605 errores de DRC tras las correcciones realizadas para el circuito de la unidad aritmética lógica. Logrando una reducción de errores del 98.27 %, lo cual es un gran avance en la mejora del diseño y su cumplimiento con las reglas de diseño establecidas.

14.1.3. Resultados de DRC para el nanochip "El Gran Jaguar"

El nanochip sin duda es un circuito que representa un reto para la síntesis física y las verificaciones de DRC, debido a su complejidad y tamaño. Los resultados de las verificaciones de DRC para el nanochip "El Gran Jaguar" se derivan de la sintetización de los circuitos anteriores. A continuación, se presentan los resultados finales obtenidos tras la verificación de DRC para el nanochip "El Gran Jaguar" en la Figura 37.

Figura 37. Resultados de DRC para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm



Nota. Visualización de los resultados de DRC para el nanochip "El Gran Jaguar", en donde se muestra que posee errores de DRC que deben ser corregidos.

En la Figura 38 se puede ver un resumen de los errores de DRC para el nanochip "El Gran Jaguar", donde se observa la cantidad total de errores y su clasificación por tipo al igual que en todos los demás circuitos. El nanochip presentó los mismos errores y cantidad que en el caso de la compuerta NOT y la ALU con oscilador de anillo. Añadiendo otros de menor cantidad, pero que también son importantes de corregir y de otro tipo, a continuación se presenta un pequeño resumen de los errores que se reportaron por ICV:

- Área encerrada mayor o igual a 0.2 μm: 1 violación.
- Mínimo mayor o igual a 0.1 μm: 13 violaciones.
- Espacio desde el *corner* en Mx: 12 violaciones.
- Mínimo de densidad en OD en la venta mayor o igual a 0.2 μm: 169 violaciones.
- Mínimo de densidad local en la ventana mayor o igual a 0.1 μm: 436 violaciones.
- Espacio a vía en X mayor o igual a 0.13 μm: 1 violación.

Figura 38. Resumen con detalle de los errores de DRC para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm

```
21
22                        ERROR SUMMARY
23
24 M2.A.2 : Enclosed area >= 0.2 um
25   area ..................................................... 1 violation found.
26
27 M2.S.1 : Spacing >= 0.1 um
28   external1 ................................................ 13 violations found.
29
30 M2.S.5 : Space at Mx line-end (W<Q=0.120) in a
31 dense-line-end configuration: If Mx has parallel
32 run length with opposite Mx (measured with T=0.035
33 extension) along 2 adjacent edges of Mx [any one
34 edge <Q distance from the corner of the two edges],
35 then one of the space (S1 or S2) needs to be at least
36 this value (except for small jog with edge length
37 < 0.10 um (R)) >= 0.12 um.
38   external2 ................................................ 12 violations found.
39
40 OD.DN.2 : Min. OD density over window 150 step 75
41 >= 20%
42   density .................................................. 169 violations found.
43
44 PO.DN.2 : {OD OR DOD OR PO OR DPO } local density
45 (minimun) over window 20um x 20um stepping 10um >=
46 0.1%
47   density .................................................. 436 violations found.
48
49 VIA1.S.3 : Space to neighboring VIAx [different net
50 and common parallel run length > 0] >= 0.13 um
51   external1 ................................................ 1 violation found.
52
53
```

Nota. Se pueden ver los dos tipos de errores de DRC para el nanochip "El Gran Jaguar", los cuales son errores de densidad en las difusiones.

Y en la Figura 39 se puede ver el resumen de resultados con la cantidad total de errores de DRC para el nanochip "El Gran Jaguar". Para este caso los errores aumentaron en comparación con los otros dos circuitos. Ahora se obtuvieron un total de 632 errores de DRC tras las correcciones realizadas. Partiendo originalmente de un diseño con 34,894 errores de DRC, se logró reducir a 632 errores de DRC tras las correcciones realizadas. Logrando una reducción de errores del 98.18 %, lo cual es un gran avance en la mejora del diseño y su cumplimiento con las reglas de diseño establecidas y lograr la manufactura del nanochip.

Figura 39. Resumen de resultados con la cantidad total de errores de DRC para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm

```
35 ----------------------------------------|-------------
36 Results Summary
37 ----------------------------------------------------
38
39 Rule and DRC Error Summary
40
41 702 total rules were run.
42 1240 rules NOT EXECUTED.
43 6 rules have violations.
44 There are 632 total violations.
45 Refer to IO_nanochip.LAYOUT_ERRORS
46
47 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
48
49 Rule                          Violations Found
50 M2.A.2                        v = 1
51 M2.S.1                        v = 13
52 M2.S.5                        v = 12
53 OD.DN.2                       v = 169
54 PO.DN.2                       v = 436
55 VIA1.S.3                      v = 1
56
57
```

Nota. Resumen de resultados con la cantidad total de errores de DRC para el nanochip "El Gran Jaguar".

14.2. Verificaciones de antena

Las verificaciones de antena son cruciales para asegurar que el diseño físico no sufra de efectos adversos debido a la acumulación de carga eléctrica durante el proceso de fabricación [30]. Estos efectos pueden causar daños en las estructuras del circuito, afectando su rendimiento y fiabilidad. La herramienta IC Validator de Synopsys es utilizada para llevar a cabo estas verificaciones, asegurando que el diseño cumple con las reglas de antena establecidas por el fabricante del proceso de fabricación.

Para ejecutar las verificaciones de antena, se utiliza la herramienta IC Validator de Synopsys. Debemos de crear una carpeta para el circuito que deseamos verificar, en la ruta `/9M_9t/sintetizacion/sintesis_fisica/Verificaciones_antena`, y dentro de esta carpeta, crear una subcarpeta con el nombre del circuito que deseamos verificar, por ejemplo::

- `/9M_9t/Verificacion/Antena/Circuito/`

Para la ejecución de la verificación de antena, abrimos la terminal en el directorio del circuito que deseamos verificar, asegurándonos que contenga nuestro *script* `antenna.tcl`. El *script* de verificación de antena es bastante sencillo, ya que solo

contiene los comandos necesarios para cargar el diseño y ejecutar la verificación de antena. A continuación, se muestra un ejemplo del contenido del *script* `antenna.tcl` en el Cuadro 14.1.

Cuadro 14.1. Contenido del *script* de verificación de antena `antenna.tcl`

```
file delete *rdb
file delete *rep
file delete .remote*

icv -i <ruta_gds/circuito.gds> -c <modulo_top> -sf ICV -vue <
   ruta_runset>
```

Nota. Ejemplo de contenido del *script* de verificación de antena `antenna.tcl`, donde se eliminan archivos temporales y se ejecuta la herramienta IC Validator con los parámetros necesarios para la verificación de antena.

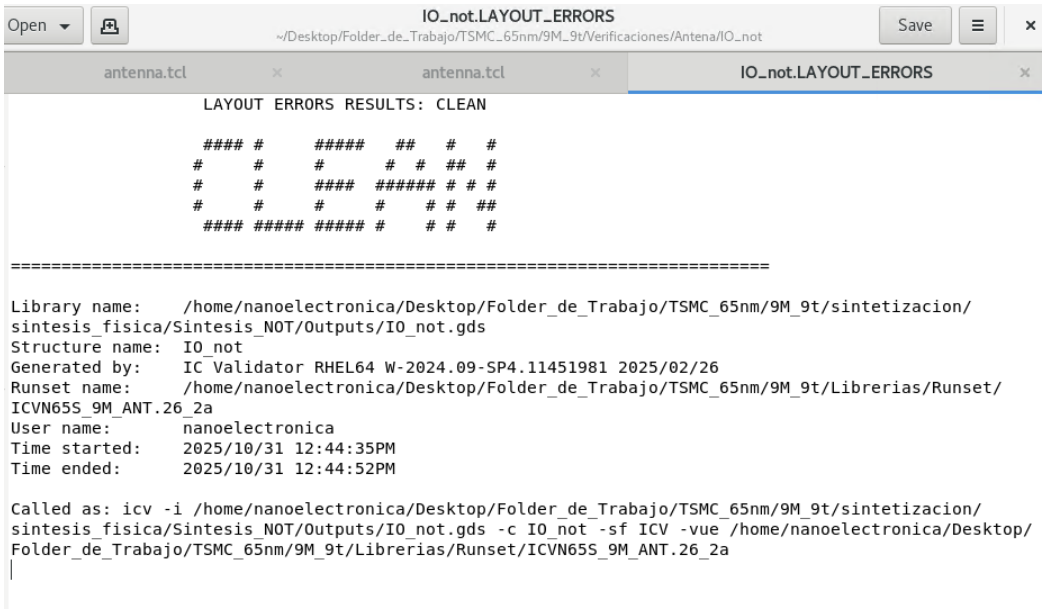
Y para ejecutar el *script*, simplemente escribimos en la terminal:

```
source antenna.tcl
```

14.2.1. Resultados de antena para la compuerta NOT

Los resultados tras la verificación de antena para la compuerta NOT sintetizada en tecnología de 65 nm mostraron que el diseño cumple con las reglas de antena establecidas. A continuación, se presentan los resultados obtenidos tras la verificación de antena para la compuerta NOT en la Figura 40.

Figura 40. Resultados de antena para la compuerta NOT sintetizada en tecnología de 65 nm

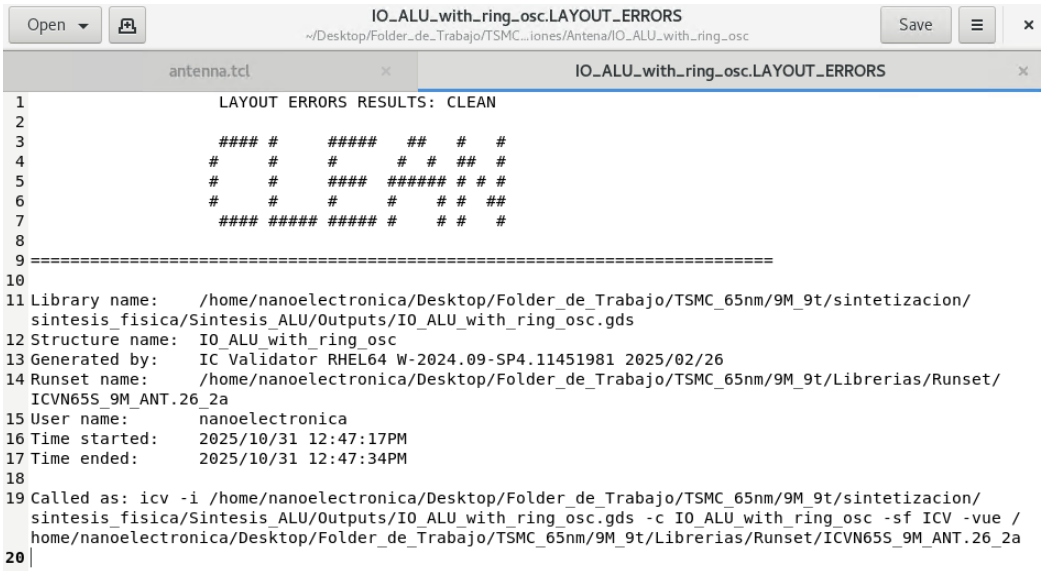


Nota. Visualización de los resultados de antena para la compuerta NOT, en donde se muestra que no posee errores de antena.

14.2.2. Resultados de antena para la ALU

Los resultados tras la verificación de antena para la ALU con oscilador de anillo sintetizada en tecnología de 65 nm mostraron que el diseño cumple con las reglas de antena establecidas. A continuación, se presentan los resultados obtenidos tras la verificación de antena para la ALU con oscilador de anillo en la Figura 41.

Figura 41. Resultados de antena para la ALU con oscilador de anillo sintetizada en tecnología de 65 nm

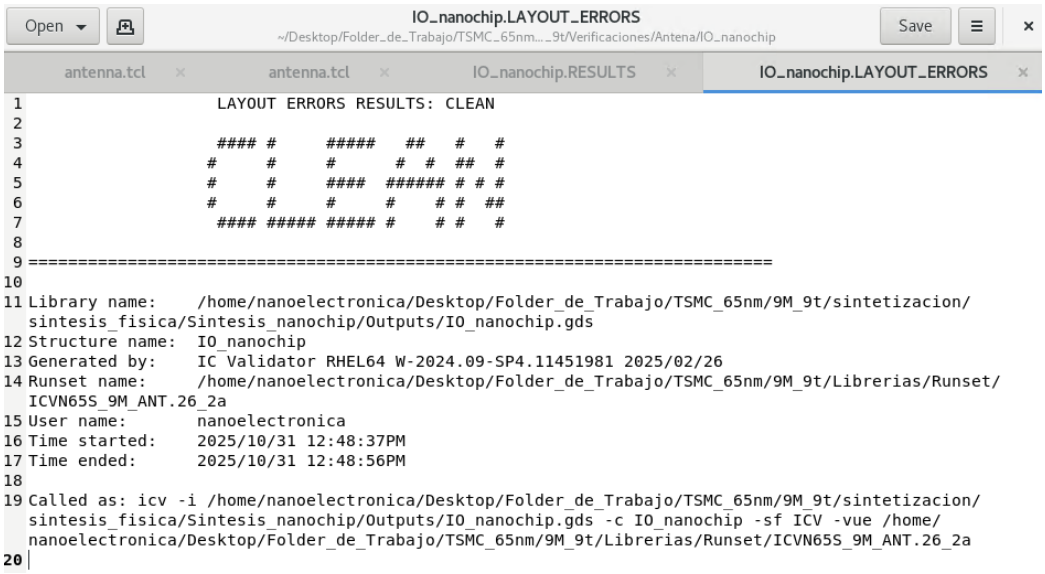


Nota. Visualización de los resultados de antena para la ALU con oscilador de anillo, en donde se muestra que no posee errores de antena.

14.2.3. Resultados de antena para el nanochip "El Gran Jaguar"

Los resultados tras la verificación de antena para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm mostraron que el diseño cumple con las reglas de antena establecidas. A continuación, se presentan los resultados obtenidos tras la verificación de antena para el nanochip "El Gran Jaguar" en la Figura 42.

Figura 42. Resultados de antena para el nanochip "El Gran Jaguar" sintetizado en tecnología de 65 nm



Nota. Visualización de los resultados de antena para el nanochip "El Gran Jaguar", en donde se muestra que no posee errores de antena.

14.3. Verificaciones de BND

Las verificaciones de BND o de conexiones del *pad* al empaquetado son esenciales para comprobar que nuestro diseño presenta validez entre el *layout* y sus conexiones eléctricas hacia el mundo exterior. Según la exploración realizada se deben de hacer varios pasos para garantizar que esas verificaciones puedan formar parte de nuestro flujo y se ejecuten de manera correcta.

Empezando se debe de añadir al flujo, junto con todos los demás *runsets* el comando siguiente con el *runset* específico de BND:

```
set_app_options -name signoff.check_design.runset -value <
    path_to_your_runset_file>
```

Esto nos asegurará que tendremos este runset agregado en nuestro flujo de síntesis física. Se debe de confirmar que se está ejecutando correctamente tanto la parte de DRC como ahora de BND, ya que para agregar estos *runsets* se accede a la misma opción dentro de la aplicación.

Luego con un chequeo de diseño podremos visualizar con la ayuda de IC Validator si las reglas que contiene el *runset* se cumplen para nuestro diseño o no. Para esto fue fundamental antes haber modificado correctamente las opciones dentro del archivo del *runset*, para que coincida con las reglas de diseño que pretendemos verificar basándonos en las propias características definidas a lo largo del flujo.

Mediante la comunicación con Synopsys se logró entender que el proceso de verificación de BND para un proceso de tipo *wire-bond* no es dependiente de crear un *bump-array*, este arreglo es conocido para procesos de tipo *flip-chip*, pero no es necesario para el proceso de *wire-bond*. Por lo tanto, se puede ejecutar un flujo del siguiente tipo como se ve en el Cuadro 14.2, basado en recomendaciones de Synopsys. Además, de estar acompañado con la correcta modificación de las opciones dentro del *runset* de BND para que coincida con las reglas de diseño que se desean verificar.

Cuadro 14.2. Ejemplo de comandos para la verificación de BND en un proceso de tipo *wire-bond*

```
place_io
create_routing_rule wirebond_rule -widths {M1 1.0} -spacings {M1 0.5 }
set_routing_rule -rule wirebond_rule [get_nets -of_objects [get_pins -of_objects [get_cells * -filter "design_type==pad"]]]
route_eco -layers {M1}
set_app_options -name signoff.check_drc.runset -value < path_to_wirebond_runset >
signoff_check_drc
report_drc_errors
```

Nota. Ejemplo de comandos para la verificación de BND en un proceso de tipo *wire-bond*, donde se colocan las entradas y salidas, se crean las reglas de enrutamiento, se realiza el enrutamiento ECO, se establece el *runset* de DRC y se ejecuta la verificación de DRC.

CAPÍTULO 15

Uso de la Reference Methodology para IC Compiler II y Library Manager

La referencia metodológica o Reference Methodology (RM) es un conjunto de mejores prácticas y procedimientos recomendados por Synopsys para el uso de sus herramientas de diseño, en este caso para la herramienta de síntesis física IC Compiler II y su herramienta de librerías Library Manager. Estas metodologías proporcionan una guía estructurada para llevar a cabo el proceso de diseño físico de manera eficiente y efectiva, asegurando que se cumplan los estándares de calidad y se optimicen los recursos disponibles.

Estas referencias de metodología se pueden descargar desde la página oficial de Synopsys SolvNet Plus, en la sección respectiva de documentación para cada herramienta [38]. Hay diferentes versiones de la RM dependiendo de la versión de la herramienta que se esté utilizando y aparte de la herramienta que se decida utilizar. En este caso se utilizó la versión de 2024, correspondiente a la que está instalada en la máquina virtual para IC Compiler II. Y una versión de la RM para Library Manager del año 2020, ya que esa era la más nueva disponible en la página de Synopsys SolvNet Plus.

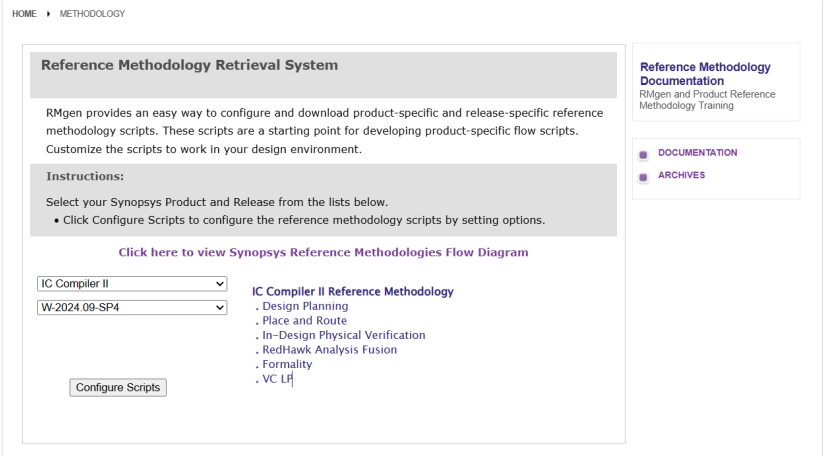
Cabe mencionar que las metodologías de referencia se quedaron en proceso de configuración y prueba, ya que, dentro del tiempo de este trabajo de graduación no se contemplaba la implementación de la RM en su totalidad. Sin embargo, se dejó la base para que en un futuro se pueda implementar y utilizar de manera correcta.

15.1. Encontrando la Reference Methodology para la herramienta de nuestro interés

Para lograr encontrar la RM adecuada, se puede buscar ya sea en la página oficial de Synopsys SolvNet Plus accediendo mediante el enlace `https://solvnet.synopsys.com/rmgen/` o yendo a la sección de *documentation* accediendo a la herramienta de nuestro interés, por ejemplo IC Compiler II, y buscando en la sección de *downloads* y haciendo clic en donde se haga mención a la RM. La otra opción es acceder a la interfaz gráfica de la herramienta de nuestro interés, y en la pestaña de *help* buscar la opción de RM.

Si se busca en la página oficial de Synopsys SolvNet Plus, se debe de iniciar sesión con una cuenta válida de *Synopsys*, y luego buscar la herramienta de nuestro interés o acceder al link antes mencionado y encontraremos una página como la que se muestra en la Figura 43, solo es de colocar el nombre de la herramienta de nuestro interés y la versión respectiva.

Figura 43. Página de SolvNet Plus para la descarga de la Reference Methodology de IC Compiler II y su versión de 2024



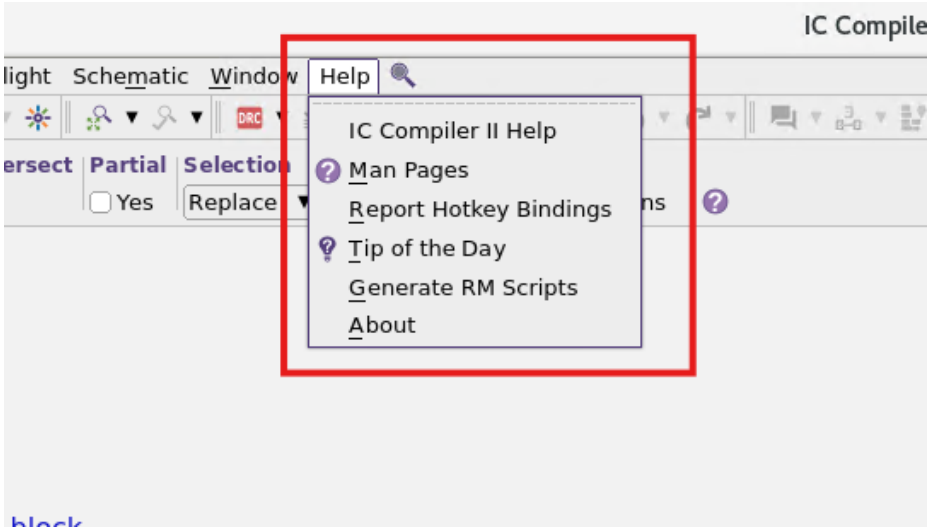
Nota. Página de SolvNet Plus en donde se encuentra la RM Gen necesaria para acceder a la metodología de referencia de la herramienta de nuestro interés.

Y mediante la interfaz gráfica de la herramienta, se puede acceder a la RM, en la pestaña de *help* como se muestra enmarcada en el cuadro rojo en la Figura 44. Para acceder a la interfaz gráfica de IC Compiler II, podemos ejecutar como comando: `icc2_shell -gui`.

Dependiendo de la versión de la herramienta que se esté utilizando, la interfaz gráfica puede variar un poco, pero la opción de RM normalmente estará en la pestaña de *help*. Por ejemplo, en la versiones de IC Compiler II del 2020, se podra encontrar la opción para elegir la RM basándonos en el nodo de tecnología que nos haya proporcionado el fabricante una vez se presione "*Generate RM scripts*", como se muestra

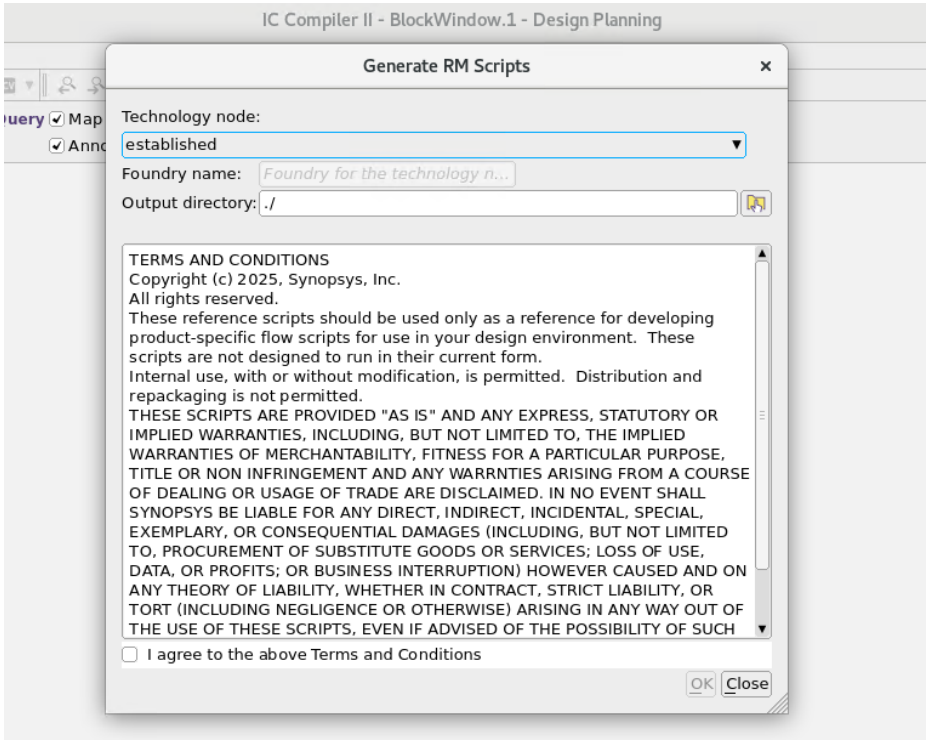
en la Figura 45.

Figura 44. Descarga de la Reference Methodology desde la interfaz gráfica de IC Compiler II



Nota. Accediendo a la RM desde la interfaz gráfica de IC Compiler II.

Figura 45. Selección del nodo en la versión de 2020 de IC Compiler II



Nota. Acá se puede seleccionar el nodo y el directorio en donde se guaradará la RM.

15.2. RM para Library Manager de IC Compiler II

Un diseño físico siempre empieza con las especificaciones y requerimientos del proyecto que son definidos por el equipo de diseño. Estos requerimientos incluyen aspectos como el tamaño del chip, la tecnología de fabricación, las restricciones de temporización, el consumo de energía, y algún que otro parámetro que el equipo considere relevante. Una vez definidos estos requerimientos, se procede a la creación de las librerías, lo que conlleva una selección de archivos específicos que son proporcionados dentro de un directorio por el fabricante de la tecnología; para nosotros TSMC.

La metodología de referencia para Library Manager proporciona una guía detallada sobre cómo preparar estas librerías para su uso en la síntesis física. Esta metodología incluye pasos específicos para configurar las librerías, validar su integridad y asegurarse de que cumplen con los estándares necesarios para el proceso de diseño físico. Y nos ayuda a generar librerías que se pueden implementar en el siguiente proceso, que es la síntesis física con IC Compiler II.

15.2.1. Descarga de la RM para Library Manager

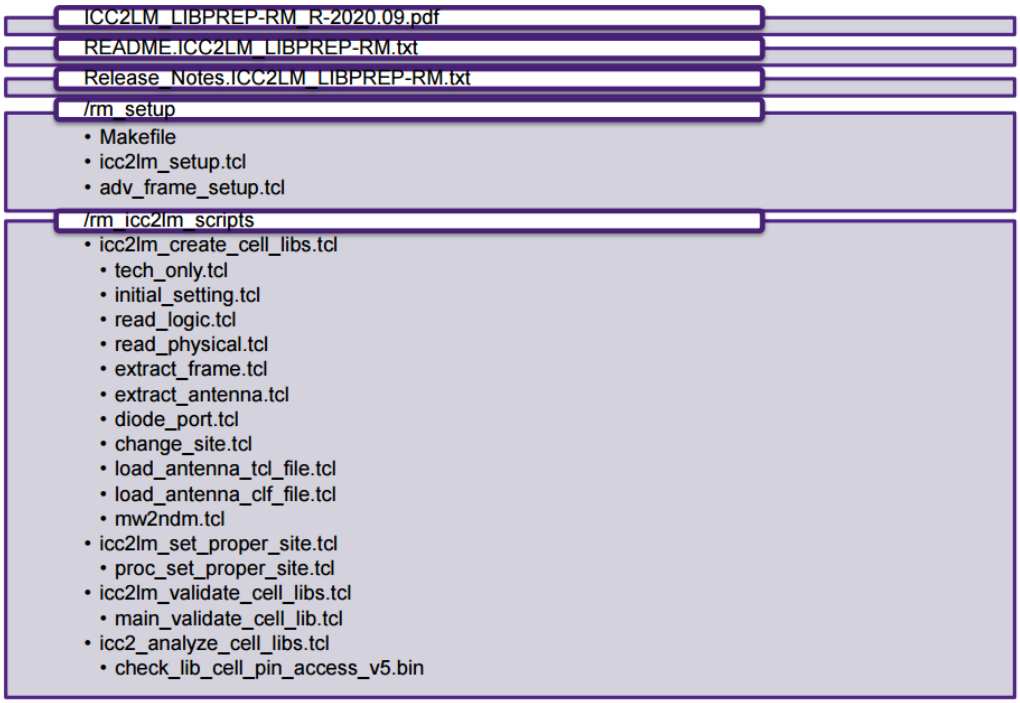
Si seguimos los pasos mencionados en la sección anterior, podremos descargar la RM para Library Manager. La versión descargada y utilizada para este caso fue la del año 2020, que es compatible con la versión de IC Compiler II instalada en la máquina virtual.

Una vez descargada la RM, se procede a descomprimir el archivo en un directorio de nuestra preferencia. Para nosotros el que se indica en jerarquía de directorios en el capítulo 7. Por lo que se encontrará en la ruta `9m_9t_RM/LM_RM`

15.2.2. Estructura de la RM para Library Manager

La estructura de la RM para Library Manager se puede ver en la Figura 46, esta proporcionada por Synopsys en la documentación de la metodología [39].

Figura 46. Estructura de la RM para Library Manager



Nota. Estructura general de la RM para Library Manager, mostrando las carpetas y archivos principales tomada desde la guía oficial de Synopsys [39].

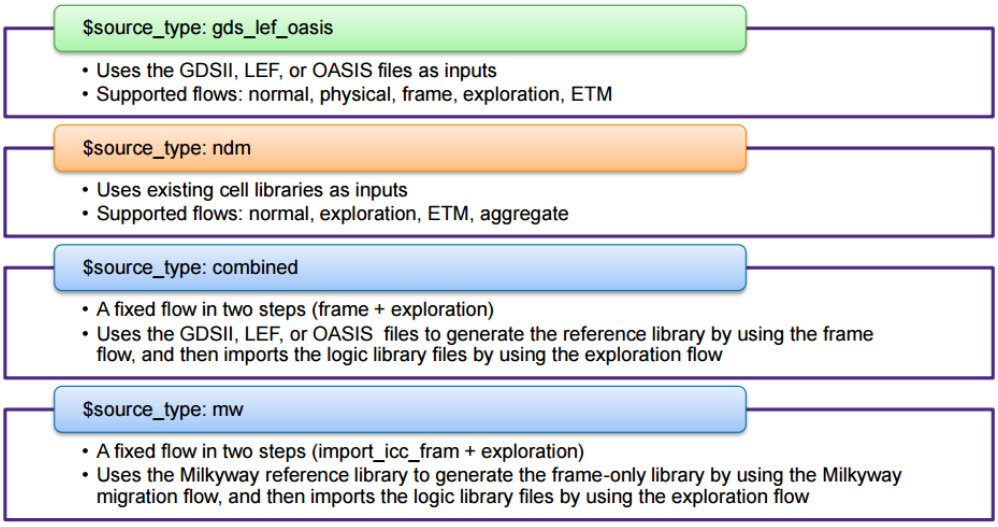
15.2.3. Uso de la RM para Library Manager

El uso de la RM para Library Manager se basa en modificar dos archivos principales que se encuentran en la carpeta `/rm_setup`, los cuales son `icc2lm_setup.tcl` y `adv_frame_setup.tcl`. El único archivo que se modificó fue `icc2lm_setup.tcl`, ya que el otro archivo no requería modificaciones para nuestro caso específico.

El archivo `icc2lm_setup.tcl` contiene una serie de comandos y configuraciones que definen cómo se deben preparar las librerías para su uso en IC Compiler II. Estos comandos incluyen la especificación de las rutas de los archivos de la librería, la configuración de las opciones de síntesis y la definición de las reglas de diseño específicas para la tecnología utilizada. También se trabajó en conjunto con Luis Carranza para explorar la RM de Library Manager de manera correcta con diferentes tipos de librerías [41].

En la Figura 47 se muestra una captura de pantalla de la guía de uso de la RM para Library Manager, donde se pueden ver los cuatro tipos principales de fuentes que se pueden usar para crear las librerías necesarias para la síntesis física en IC Compiler II. Dependiendo de qué tipo de flujo se elija o se disponga se pueden utilizar diferentes herramientas para la creación de las librerías, como IC Compiler para una fuente de tipo Milkyway.

Figura 47. Tipos de fuentes para la creación de librerías en Library Manager



Nota. Captura de pantalla de la guía de uso de la RM para Library Manager, mostrando los cuatro tipos principales de fuentes para la creación de librerías [39].

La RM para Library Manager se utiliza para preparar las librerías necesarias para la síntesis física y tiene un flujo específico que pasa por diferentes *scripts* y etapas. Empezando con elegir la librería objetivo y generar las librerías de las celdas disponibles, generar las librerías de referencia con los datos de la tecnología, preparar los sitios, validar las librerías de las celdas y finalmente analizar las mismas librerías para garantizar que se puedan utilizar en el flujo de síntesis física.

15.3. RM para IC Compiler II

La RM para IC Compiler II es una guía y plantilla detallada que proporciona mejores prácticas y procedimientos recomendados para llevar a cabo el proceso de diseño físico utilizando la herramienta IC Compiler II. Esta metodología abarca desde la configuración inicial del proyecto hasta la verificación final del diseño, asegurando que se cumplan los estándares de calidad y se optimicen los recursos disponibles.

15.3.1. Descarga de la RM para IC Compiler II

Siguiendo los pasos mencionados en la sección anterior, se procede a descargar la RM para IC Compiler II. La versión descargada y utilizada para este caso fue la misma que se dispone en la máquina virtual, la W-2024.09-SP3.

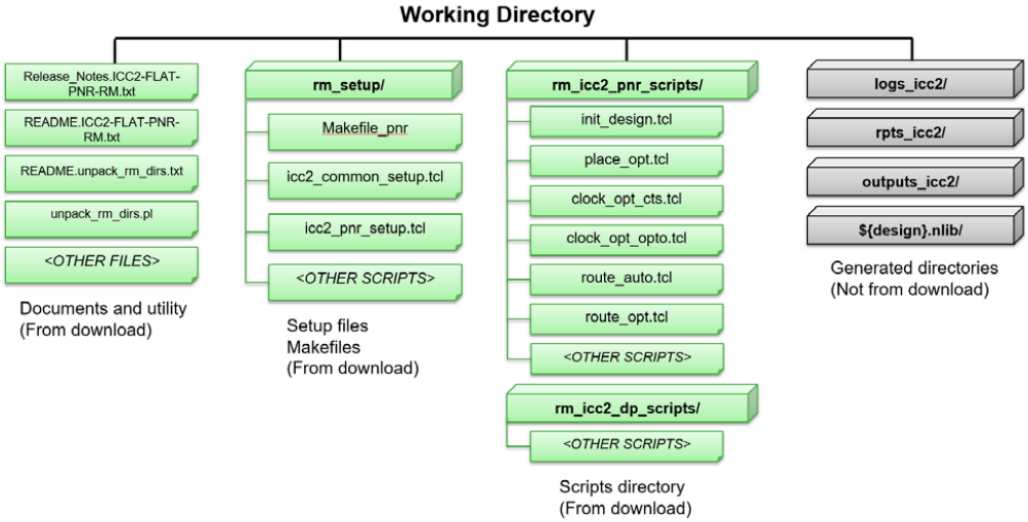
Una vez descargada la RM, se procede a descomprimir el archivo en un directorio de nuestra preferencia. Para nosotros el que se indica en jerarquía de directorios en

el capítulo 7.

15.3.2. Estructura de la RM para IC Compiler II

Para entender mejor cómo utilizar la RM para IC Compiler II, es importante conocer su estructura general. La estructura de la metodología se puede ver en la Figura 48, esta proporcionada por Synopsys en la documentación de la metodología en SolvNetPlus [37].

Figura 48. Estructura de la RM para IC Compiler II



Nota. Estructura general de la RM para IC Compiler II, mostrando las carpetas y archivos principales tomada desde la guía oficial de Synopsys [37].

15.3.3. Uso de la RM para IC Compiler II

La RM para IC Compiler II se utiliza en cada etapa del diseño físico, desde la planificación inicial hasta la verificación final. A continuación, se describen algunas de las principales aplicaciones de esta metodología:

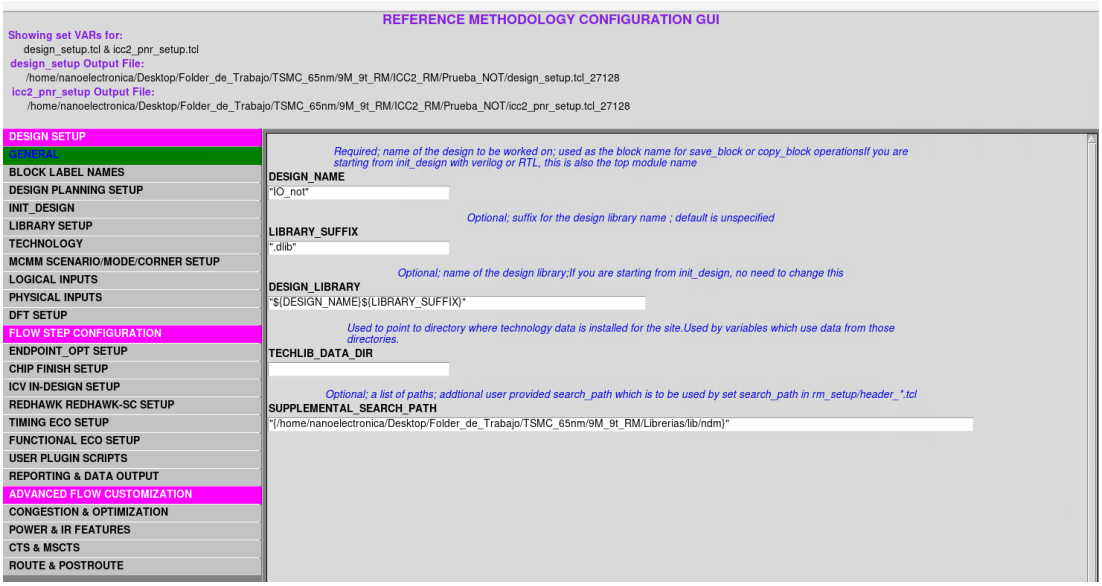
- **Configuración del entorno de diseño:** la metodología proporciona pautas para configurar el entorno de diseño, incluyendo la selección de bibliotecas, la configuración de herramientas y la definición de flujos de trabajo.
- **Optimización del diseño:** se ofrecen técnicas y estrategias para optimizar el diseño físico, incluyendo la reducción de área, la mejora del rendimiento y la minimización del consumo de energía.

- **Verificación y validación:** la metodología incluye procedimientos para verificar y validar el diseño físico, asegurando que cumpla con los requisitos especificados y funcione correctamente en el chip.

También se puede utilizar la interfaz gráfica de la RM para IC Compiler II. Esta requirió de ciertos cambios, los cuales fueron adaptarla al uso de ciertos paquetes y librerías de gráficos que requiere la interfaz, estos se pueden ver en el caso que se abrió con Synopsys entrando con el usuario correspondiente en la plataforma de SolvNetPlus para futura referencia.

Los *scripts* a modificar de la RM para IC Compiler II se encuentran en la carpeta `/rm_setup/`. Dependiendo de la versión de la herramienta que se esté utilizando, los *scripts* pueden variar un poco de nombre o lo que incluyen dentro de los mismos. La estructura sigue siendo similar por lo que puede ser comparable con la que se muestra en la Figura 48. Los archivos modificados y adaptados con las rutas, librerías, y archivos descritos durante este trabajo de graduación fueron: `design_setup.tcl`, `icc2_dp_setup.tcl` y `icc2_pnr_setup.tcl`.

Figura 49. Vista de la interfaz gráfica de la RM para IC Compiler II



Nota. Interfaz gráfica de la RM para IC Compiler II, mostrando las diferentes opciones y configuraciones disponibles para el diseño físico.

Es de importancia notar aspectos como la jerarquía del diseño, en nuestro caso no aplica, ya que todo está descrito en un mismo archivo Verilog, siendo de tipo plano o *flat*. Considerar los *inputs* de la metodología de referencia también es esencial, por lo que se debe de trabajar extensivamente para hacer funcionar correctamente la metodología de referencia para las librerías y luego pasar a hacer uso de esta otra cuando ya no tengamos ningún problema con la generación de los archivos de librerías.

También como se ve en la Figura 49, se puede observar que la interfaz gráfica de la RM para IC Compiler II cuenta con diferentes pestañas y opciones que permiten configurar y personalizar el flujo de diseño físico según las necesidades del proyecto. Podemos configurar desde qué tipo de *input* de la síntesis lógica queremos tomar como partida, opciones de librerías, verificación, diseño, y otros aspectos importantes del flujo de diseño físico.

La interfaz gráfica es una herramienta que nos permite visualizar y modificar las configuraciones de la metodología de referencia de manera más intuitiva y accesible. Sin embargo, también se aconseja familiarizarse con los *scripts* y configuraciones subyacentes para comprender completamente cómo funciona la metodología y cómo se pueden adaptar a las necesidades específicas del proyecto. Para acceder a la interfaz gráfica de la RM para IC Compiler II, podemos ejecutar el comando `./rm_utilities/configureRM` en el directorio principal de la metodología de referencia.

CAPÍTULO 16

Conclusiones

- Las mejoras en los *scripts* para creación de librerías y síntesis física permitieron la eficiencia en la ejecución de los flujos y la adaptación rápida a cambios en requerimientos o archivos pertinentes al diseño.
- La correcta identificación de los archivos y sus modificaciones respectivas para cada etapa de la síntesis física fue esencial para la reducción de errores y la optimización del trabajo.
- Se realizó la síntesis de circuitos complejos con éxito y con los menos errores posibles en una tecnología inexplorada dentro del entorno académico de la universidad.
- Las verificaciones físicas de DRC de los diseños mostraron una mejora satisfactoria con respecto al punto de partida al inicio del trabajo, reduciendo la cantidad de errores en aproximadamente 98 %.
- Los errores obtenidos en la etapa de verificación demuestran que ha habido una mejora significativa en la comprensión de las reglas de diseño y en la aplicación de las mismas durante la síntesis física.
- Las verificaciones de antena de los diseños lograron cumplir con las especificaciones de la tecnología y mostraron un resultado satisfactorio.
- Se profundizó el conocimiento en una nueva prueba de conexión de *pads* conocida como BND y se exploró su implementación en la tecnología utilizada.
- La comunicación con las instituciones y empresas del sector permitió el avance, la obtención de recursos y conocimientos necesarios para la culminación del proyecto.
- Las metodologías de referencia fueron una buena alternativa a considerar para la realización de las diferentes etapas del flujo de diseño VLSI.

- La documentación y el curso de capacitación de las herramientas utilizadas fueron de gran ayuda para la comprensión y manejo de las mismas, facilitando el desarrollo y corrección de errores en el flujo de diseño.

CAPÍTULO 17

Recomendaciones

- Se recomienda profundizar en la automatización de los flujos de diseño mediante *scripts* para mejorar la eficiencia y reducir errores humanos.
- Se sugiere explorar la inclusión de nuevos archivos y formas de hacer las librerías como partir de una librería Milkyway para optimizar el proceso de creación de librerías personalizadas.
- Se recomienda familiarizarse con los requerimientos de diseño leyendo en la etapa inicial la documentación proporcionada por IMEC, entre ellas el manual para el diseño de ASIC.
- Es recomendable implementar un sistema de control de versiones para los archivos de diseño y *scripts* para facilitar el seguimiento de cambios y colaboraciones futuras.
- Se aconseja que se realicen pruebas con el nuevo *runset* de BND desde el inicio para asegurar que su implementación fue exitosa y cambios en el flujo de diseño permitan corregir posibles errores a tiempo.
- También se aconseja tener un flujo de trabajo continuo con el equipo de verificación de LVS y ERC para asegurar que las modificaciones en el diseño no introduzcan nuevos errores.
- Se aconseja tomar los cursos proporcionados por los fabricantes de las herramientas de diseño desde el primer instante para mantenerse actualizado con las mejores prácticas y nuevas funcionalidades.
- Es recomendable profundizar en las herramientas de verificación de DRC, tomar los cursos oficiales y establecer una comunicación frecuente para resolver dudas técnicas y asegurar que las etapas de verificación se realicen correctamente.
- Se recomienda documentar detalladamente cada etapa del proceso de diseño y síntesis para facilitar futuras referencias y mejoras en los flujos de trabajo.

- Se sugiere establecer canales de comunicación con IMEC y Synopsys para resolución de dudas técnicas y obtención de soporte durante el desarrollo de proyectos similares.

CAPÍTULO 18

Referencias

[1] J. D. los Santos, «Diseño de un sumador/restador completo de 32 bits con tecnología CMOS en un proceso de 28 nanómetros usando aplicaciones de diseño de la empresa Synopsys,» Universidad Del Valle de Guatemala, Guatemala, 2014.

[2] S. Rubio, «Definición del Flujo de Diseño para Fabricación de un Chip con Tecnologia VLSI CMOS,» Universidad Del Valle de Guatemala, Guatemala, 2019.

[3] L. Nájera, «Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado.,» Universidad Del Valle de Guatemala, Guatemala, 2019.

[4] M. Illescas, «Verificación de reglas de diseño (DRC) para el desarrollo de un flujo funcional de un circuito integrado con tecnología nanométrica,» Universidad Del Valle de Guatemala, Guatemala, 2020.

[5] M. Flores, «Corrección de anillo de entradas/salidas y pruebas de antenna y ERC para la definición del flujo de diseño del primer chip con tecnología nanométrica desarrollado en Guatemala,» Universidad Del Valle de Guatemala, Guatemala, 2020.

[6] J. Girón, «Etapa de vericación física de Diseño en Silicio vs. Esquemático (LVS) en el flujo de diseño para un chip a nanoescala,» Universidad Del Valle de Guatemala, Guatemala, 2020.

[7] J. Ruano, «Definición del flujo en la herramienta VCS para la simulación de HDLs en la Fabricación de un Chip con Tecnología Nanométrica CMOS,» Universidad Del Valle de Guatemala, Guatemala, 2020.

[8] J. Ayala, «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.

[9] J. Ruiz, «Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: ejecución de la fase de verificación física Layout vs Schematic (LVS),» Universidad Del Valle de Guatemala, Guatemala, 2021.

[10] J. Shin, «Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: ejecución de la síntesis física, Verificación de reglas eléctricas y corrección de errores obtenidos,» Universidad Del Valle de Guatemala, Guatemala, 2021.

[11] A. Altuna, «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.

[12] L. Abadía, «Posicionamiento e interconexión entre componentes de un circuito sintetizado para el flujo de diseño de un circuito a escala nanométrica utilizando la herramienta de IC Compiler,» Universidad Del Valle de Guatemala, Guatemala, 2021.

[13] E. Torres, «Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: Ejecución y simulación para la etapa de síntesis lógica,» Universidad Del Valle de Guatemala, Guatemala, 2021.

[14] S. Cardona, «Mejoramiento del proceso de síntesis lógica llevada a cabo para la elaboración de un circuito integrado a escala nanométrica,» Universidad Del Valle de Guatemala, Guatemala, 2021.

[15] A. Aguilar, «Mejoramiento del proceso de síntesis lógica llevada a cabo para la elaboración de un circuito integrado a escala nanométrica,» Universidad Del Valle de Guatemala, Guatemala, 2021.

[16] D. Equité, «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.

[17] J. Ponce, «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.

[18] S. Schwendener, «Automatización de las vericaciones físicas de un circuito integrado con tecnología de 180 nm utilizando librerías de diseño de TSMC,» Universidad Del Valle de Guatemala, Guatemala, 2022.

[19] P. Mendizábal, «Automatización del proceso de instalación de software de Synopsys e implementación de mejoras utilizando contenedores,» Universidad Del Valle de Guatemala, Guatemala, 2022.

[20] N. Prado, «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, vericación DRC y antena,» Universidad Del Valle de Guatemala, Guatemala, 2023.

[21] K. Hernández, «Automatización de la etapa de síntesis lógica, optimización del proceso de instalación de las aplicaciones de Synopsys y documentación de los pasos de Front-End y Back-End para el diseñó del Circuito Integrado El Gran Jaguar,» Universidad Del Valle de Guatemala, Guatemala, 2024.

[22] L. Ruiz, «Diseño y fabricación de un circuito integrado ocn tecnología de 65nm usando librerías de de diseño de TSMC: pruebas finales de funcionamiento en HSPICE, generación y documentación de la síntesis física, verificaciones de Antena y DRC,» Universidad Del Valle de Guatemala, Guatemala, 2024.

[23] D. Alvarado, «Diseño de un circuito integrado con tecnología de 65 nm utilizando librerías de diseño de TSMC: Pruebas de LVS, ERC y extracción de parásitos,» Universidad Del Valle de Guatemala, Guatemala, 2024.

[24] Y. T. T. H. Ning, *Fundamentals of Modern VLSI Devices*, 3rd. Cambridge: Cambridge University Press, 2022.

[25] S. H. Gerez, «Circuit Modeling with Hardware Description Languages,» en *Digital VLSI Design and Simulation with Verilog*. Elsevier, 2015.

[26] I. Synopsys, *IC Compiler II Online Help X-2025.05-SP1 User Documentation*. visitado 2025. dirección: `https://spdocs.synopsys.com/dow_retrieve/latest/dg/icc2olh/olh_icc2/landing_page.html`.

[27] H. R. W. Dehaene, *Efficient Design of Variation-Resilient Ultra-Low Energy Digital Processors*. Springer, 2019.

[28] A. B. Kahng, J. Lienig, I. L. Markov y J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer, 2011, vol. 312.

[29] I. Synopsys, *Synopsys IC Compiler II*. visitado 2025. dirección: `https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html`.

[30] I. Synopsys, *Physical Verification - IC Validator*. visitado 2025. dirección: `https://www.synopsys.com/implementation-and-signoff/physical-verification.html`.

[31] Europractice IC, «ASIC Package Design Rules,» Europractice IC, inf. téc., 2020. dirección: `https://europractice-ic.com/wp-content/uploads/2020/06/ASIC-Package-Design-Rules-10062020.pdf`.

[32] I. Synopsys, *Synopsys*. visitado 2025. dirección: `https://www.synopsys.com`.

[33] I. Synopsys, *VCS Functional Verification Solution*. visitado 2025. dirección: `https://www.synopsys.com/verification/simulation/vcs.html#resources`.

[34] I. Synopsys, *Design Compiler*. visitado 2025. dirección: `https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html`.

[35] I. Synopsys, *Design Compiler NXT*. visitado 2025. dirección: `https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html`.

[36] I. Synopsys, *Library Manager User Guide*. visitado 2025. dirección: `https://spdocs.synopsys.com/dow_retrieve/latest/dg/icc2olh/lmug/preface/about_this_user_guide.html`.

[37] I. Synopsys, *SolvNet Plus*. visitado 2025. dirección: `https://solvnetplus.synopsys.com/s/`.

[38] I. Synopsys, *RM Gen*. visitado 2025. dirección: `https://solvnet.synopsys.com/rmgen/`.

[39] I. Synopsys, *IC Compiler II Library Preparation Reference Methodology*. visitado 2025. dirección: `https://solvnetplus.synopsys.com/s/article/IC-Compiler-II-Library-Preparation-Reference-Methodology-1577133842517`.

[40] I. link Tapeout Team, *TSMC 28nm 40nm 65nm mini sic manualver Jul 2025*. IMEC, 2025.

[41] L. Carranza, «Diseño de un circuito integrado con tecnología de 65 nm de TSMC: Fase de síntesis física, verificación y generación de archivos para pruebas DRC, BND y de antena,» Universidad Del Valle de Guatemala, Guatemala, 2025.

[42] H. Zhou et al., «Copper wire bonding: A review,» *Micromachines*, vol. 14, n.º 8, pág. 1612, 2023.

[43] M. K. Mandal y B. C. Sarkar, «Ring oscillators: Characteristics and applications,» *Indian journal of pure & applied physics*, vol. 48, n.º 1, págs. 136-145, 2010.

CAPÍTULO 19

Anexos

19.1. Replicación del flujo en 180 nm

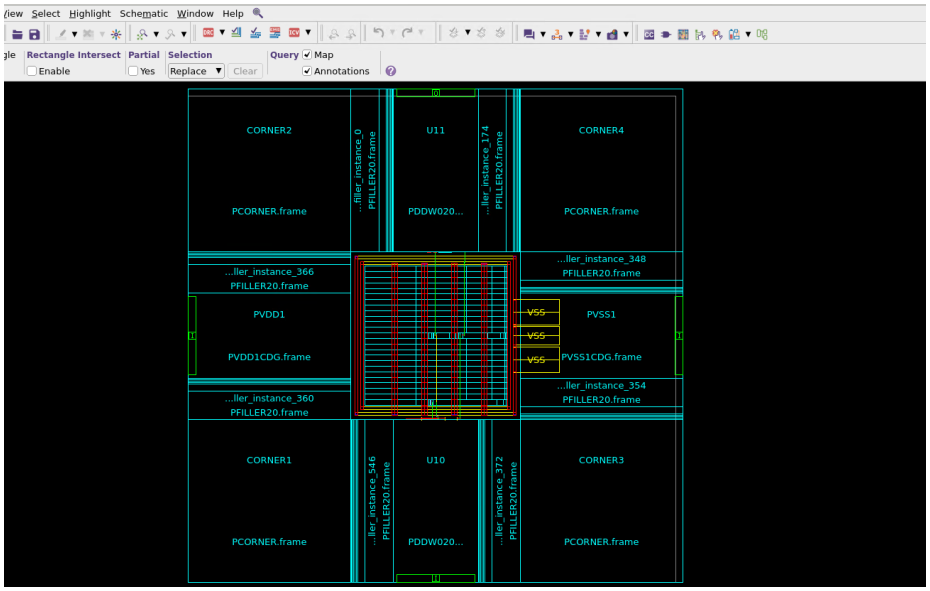
Como parte del proceso de diseño de un chip se debe de tener un proceso de acercamiento y familiarización con las herramientas y flujos de trabajo que se utilizarán. Esto incluye la comprensión de la jerarquía de directorios y el entorno de trabajo, así como la configuración de las herramientas necesarias para llevar a cabo el diseño. Por lo tanto, fue de suma importancia tener un acercamiento al flujo de diseño utilizado en la tecnología de 180 nm, para así poder replicar el flujo en la tecnología de 65 nm.

El flujo de diseño en 180 nm de la síntesis física se puede ver en el trabajo de graduación de Ruiz [22], en el cual se detalla el proceso de creación de las librerías NDM, la selección de los archivos necesarios, así como la configuración de los *runsets* y la ejecución de los *scripts* para llevar a cabo la síntesis física.

Para la replicación se acudió a hacer una copia de los archivos encontrados en la antigua máquina virtual utilizada para los trabajos realizados en la tecnología anterior. Estos archivos fueron copiados a la nueva máquina virtual, y se realizó una revisión de los mismos para asegurarse de que todos los archivos necesarios estuvieran presentes y en buen estado. Al principio hubo unos errores con rutas, sin embargo, se corrigieron y se logró ejecutar el flujo de diseño en 180 nm sin problemas.

Se hizo la ejecución del flujo de diseño de síntesis física para dos circuitos, una not y el nanochip "El Gran Jaguar". Primeramente, se puede ver en la Figura 50, algunos a detalles a notar en comparación a las versiones finales de los circuitos en la tecnología de 65 nm presentados en este trabajo de graduación, serán el tamaño del *layout*, rellenos de metal, rellenos para las difusiones, cantidad de capas de metal, la falta del metal para la conexión de los *pads* de alimentación, entre otros que se podrán observar en los siguientes capítulos.

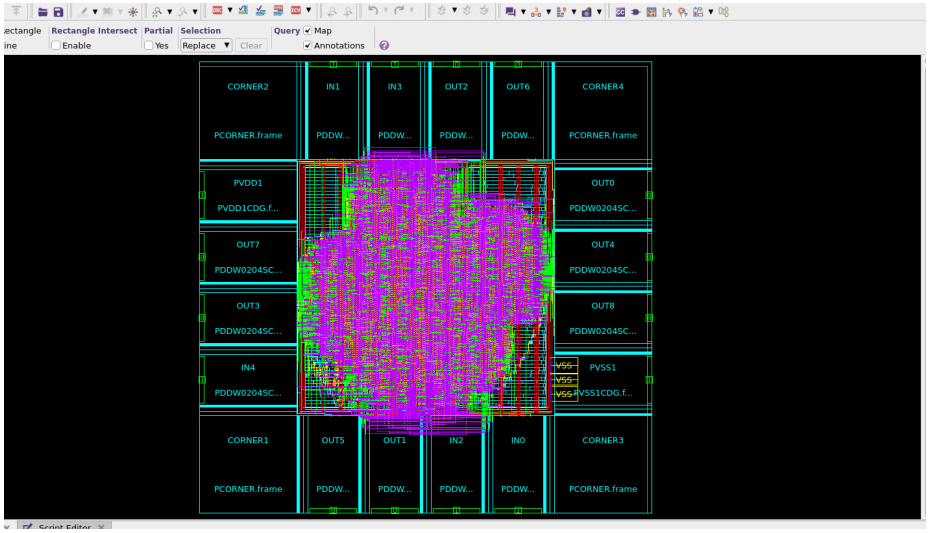
Figura 50. Figura de un *layout* de una compuerta NOT en tecnología de 180 nm



Nota. Imagen de la compuerta NOT sintetizada en tecnología de 180 nm en el trabajo de Ruiz [22].

A continuación, en la Figura 51 se puede observar el *layout* del nanochip "El Gran Jaguar" en la tecnología de 180 nm. En este *layout* podemos observar detalles similares a los del circuito NOT, como el tamaño del *layout*, y en consecuencia algunos metales se salen del área del *core*.

Figura 51. Figura del *layout* del nanochip en tecnología de 180 nm



Nota. Imagen del "El Gran Jaguar" sintetizado en tecnología de 180 nm en el trabajo de Ruiz [22].

CAPÍTULO 20

Glosario

Antenna rule check El proceso de verificación que revisa violaciones de antena en un diseño.

ASIC (application-specific integrated circuit) Circuito Integrado de Aplicación Específica.

Backend La parte del flujo de diseño que se encarga de la implementación física del circuito, también conocido como Síntesis Física.

Behavioral Un nivel de abstracción en el diseño de circuitos digitales que se centra en la funcionalidad del sistema y su comportamiento lógico sin el detalle de su implementación física.

Bondpad verification **(BND)** El proceso de verificar que los pads de unión y todas las áreas de contacto metálicas deban estar correctamente diseñadas, permitiendo una conexión confiable entre el chip y el empaquetado.

Clock tree synthesis **(CTS)** El proceso de diseñar la red de distribución de reloj en un circuito integrado.

Custom Compiler Una erramienta de Synopsys que permite la creación, editar y verificación de diseños análogicos y/o digitales a nivel transistor.

Design flow **(flujo de diseño)** El proceso de diseño para un circuito integrado, desde la especificación hasta la fabricación.

Design for manufacturability **(DFM)** Un enfoque de diseño que busca facilitar la fabricación de circuitos integrados, asegurando su fabricación de manera confiable que respeta las limitaciones y especificaciones del proceso de fabricación.

Design rule check **(DRC)** El proceso de verificar que un diseño cumple con las reglas de diseño establecidas por el fabricante.

Die La porción de un *wafer* que contiene un circuito integrado, después de ser cortado del *wafer*.

***Electrical rule check* (ERC)** El proceso de verificar que un diseño cumple con las reglas eléctricas establecidas por el fabricante.

EMS database Una base de datos utilizada para gestionar la información de diseño en un flujo de diseño de circuitos integrados.

***Engineering change order* (ECO)** Un documento que describe un cambio propuesto en el diseño de un circuito integrado de una forma controlada para corregir errores o mejorar el rendimiento.

Floorplanning El proceso de asignar áreas físicas a los bloques funcionales o celdas en un diseño de circuito integrado.

Frontend La parte del flujo de diseño que se encarga de la descripción y especificación utilizando HDL y el diseño lógico del circuito.

GDS (GDSII) Un formato de archivo utilizado para representar el diseño físico de circuitos integrados.

***Hardware description language* (HDL)** Lenguaje de Descripción de Hardware, utilizado para describir circuitos.

IC Compiler II Una herramienta de Synopsys dedicada al diseño de circuitos integrados que permite la síntesis y optimización del diseño.

IC Validator Una herramienta de Synopsys utilizada para verificar la corrección de un diseño de circuito integrado mediante *Runsets* o parámetros específicos.

***Integrated circuit* (IC)** Circuito Integrado, un conjunto de componentes electrónicos conectados en chip.

Latencia El tiempo de retraso que tarda la señal en propagarse desde la entrada hacia la salida.

Layout La representación física de un circuito integrado, incluyendo la disposición de los componentes, geometría, dimensiones, y conexiones.

***Layout versus schematic* (LVS)** El proceso de verificar que el diseño físico de un circuito coincide en cuanto a su funcionalidad con su esquema lógico.

***Library Exchange Format* (LEF)** Un formato de archivo utilizado para describir la geometría y las propiedades de las celdas estándar, macros y *fillers* en un diseño.

Library Manager Una herramienta de Synopsys utilizada para gestionar las bibliotecas de celdas estándar y macros en un diseño.

Macro Un bloque de diseño que encapsula una función completa y se puede reutilizar en diferentes diseños *VLSI*.

Metal fill El proceso de agregar material metálico en áreas específicas del diseño para mejorar y garantizar la manufacturabilidad y el rendimiento.

NDM Un formato de archivo utilizado como una base de datos para optimizar y garantizar la operabilidad entre diferentes herramientas en un flujo de diseño de circuitos integrados.

Netlist Una representación de un circuito electrónico/eléctrico en términos de sus componentes y las conexiones entre ellos.

Pads I/O Las áreas metálicas en un chip donde se conectan las señales de entrada/salida.

Partitioning El proceso de dividir un diseño en partes más pequeñas y manejables, así facilitando las etapas del flujo de diseño.

Placement El proceso de ubicar los componentes en el diseño de un circuito integrado de manera que los componentes estén optimamente ubicados para cumplir con las condiciones del diseño y restricciones.

Power and ground routing El proceso de diseñar las rutas de alimentación y tierra en un circuito integrado con el fin de alimentar correctamente al circuito.

Power rings / mesh Estructuras utilizadas para distribuir la potencia de manera eficiente en un chip, pueden hacerse de varias maneras, como anillos o mallas.

PrimeTime Una herramienta de Synopsys de análisis de *timing* utilizada en el diseño de circuitos integrados.

Register Transfer Level (RTL) Un nivel de abstracción en el diseño de circuitos digitales, que se centra en la transferencia de datos entre registros mediante operaciones lógicas definidas en un lenguaje de descripción de hardware (HDL).

Rule Deck Un conjunto de reglas utilizadas para la verificación de diseños de circuitos integrados, que se corre de forma automática utilizando herramientas de verificación como ICV.

Signal routing El proceso de diseñar las rutas de señal en un circuito integrado, asegurando la integridad de la señal y el cumplimiento de las restricciones de diseño.

Skew La diferencia de tiempo en la llegada de la señal de reloj a diferentes componentes en un chip.

Slack La cantidad de tiempo adicional que tiene una señal para llegar a su destino antes de que se considere fuera de tiempo o atrasada.

***Standard cells* (celdas estándar)** Bloques de diseño predefinidos que se utilizan para construir circuitos integrados, los cuáles están garantizados por el fabricante.

Structural Un nivel de abstracción en el diseño de circuitos digitales que se centra en la interconexión de componentes, realizado mediante un lenguaje de descripción de hardware (HDL).

Testbench Un entorno de prueba utilizado para verificar el funcionamiento de un diseño de circuito integrado, simulando su comportamiento bajo diferentes condiciones y/o estímulos.

Timing closure El proceso de asegurar que un diseño cumple con los requisitos de tiempo.

Verilog Un lenguaje de descripción de hardware utilizado para modelar sistemas electrónicos o eléctricos, ya sea de tipo digital, analógico o de señal mixta.

Very-Large-Scale Integration (VLSI) Es la tecnología que consiste en integrar una gran cantidad de transistores en un chip de silicio y así crear circuitos integrados complejos.

Wafer Un disco delgado de material semiconductor en el que se fabrican circuitos integrados.

Well-taps Estructuras utilizadas para conectar las fuentes de voltaje a las capas de transistores en un chip.

Workspace El entorno de trabajo donde se lleva a cabo el diseño de circuitos integrados.