Verificaciones LVS y ERC del circuito integrado "El Gran Jaguar" con tecnología de 65 nanómetros de TSMC

Byron Estuardo Barrientos Pérez

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Verificaciones LVS y ERC del circuito integrado "El Gran Jaguar" con tecnología de 65 nanómetros de TSMC

Trabajo de graduación presentado por Byron Estuardo Barrientos Pérez para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2025

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Verificaciones LVS y ERC del circuito integrado "El Gran Jaguar" con tecnología de 65 nanómetros de TSMC

Trabajo de graduación presentado por Byron Estuardo Barrientos Pérez para optar al grado académico de Licenciado en Ingeniería Electrónica

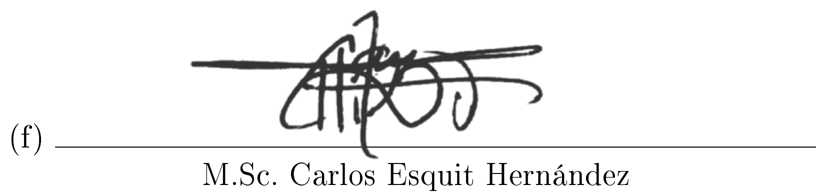
Guatemala,

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

El presente trabajo es el resultado de varios años de esfuerzo colectivo dentro de la línea de investigación en diseño de circuitos integrados de la Universidad del Valle de Guatemala. Su desarrollo no habría sido posible sin el apoyo humano, académico y técnico de diversas personas e instituciones. Me considero afortunado por haber sido parte de este proyecto, ya que no sólo representa un paso significativo en la consolidación de la capacidad local para el diseño y verificación de circuitos integrados a nanoescala, sino que también marca el culmen de mi formación como ingeniero electrónico.

En primer lugar, quiero agradecerle a mi mamá, a mi papá y a mi hermana; gracias por creer en mí y por brindarme la fortaleza, el apoyo y la determinación para superar los desafíos a los que me he enfrentado durante toda mi carrera universitaria. A mi familia y amigos, ya que su apoyo incondicional fue fundamental para mantenerme enfocado y motivado a lo largo de todo el proceso; me brindaron ánimo y comprensión durante el proyecto y celebraron conmigo cada logro alcanzado.

Extiendo mi gratitud al equipo de “El Gran Jaguar”, quienes formaron parte de este recorrido y contribuyeron con discusiones técnicas, documentación previa y apoyo en los momentos más demandantes. Este trabajo es un reflejo del esfuerzo conjunto de todo el equipo, incentivados por la visión de impulsar la investigación en tecnologías de diseño VLSI en el país.

También agradezco a mis catedráticos por su dedicación, guía constante y compromiso con el desarrollo académico de sus estudiantes. Su influencia ha sido crucial en mi formación profesional y en la realización de este proyecto. Finalmente, le agradezco a la Universidad del Valle de Guatemala por proporcionar un entorno académico propicio para la investigación y el desarrollo tecnológico. La infraestructura, los recursos y el acceso a herramientas profesionales fueron esenciales para llevar a cabo este trabajo de graduación.

Índice general

Prefacio I

Índice de figuras VI

Índice de cuadros IX

Resumen X

Abstract XI

1. Introducción 1

2. Antecedentes 4

3. Justificación 6

4. Objetivos 8
- 4.1. Objetivo general . . . 8
- 4.2. Objetivos específicos . . . 8

5. Alcance 10

6. Marco teórico 12
- 6.1. *VLSI Design* . . . 12
- 6.2. Flujo de diseño RTL a GDSII . . . 13
- 6.3. IC Compiler II . . . 14
- 6.4. IC Validator . . . 14
- 6.5. NetTran . . . 14
- 6.6. *Layout vs. Schematic* (LVS) . . . 15
- 6.7. *Electrical Rule Checking* (ERC) . . . 21

7. Creación de celdas Verilog para uso de la FPGA *Digilent Genesys* 24
- 7.1. Celda CKND2D1 - NAND de dos entradas . . . 24
- 7.2. Celda IIND4D0 - NAND con inversores en entradas A1 y A2 . . . 25

7.3. Celda IINR4D0 - NOR con inversores en entradas A1 y A2 27
7.4. Celda IND2D0 - NAND con inversor en entrada A1 28
7.5. Celda IND3D0 - NAND con inversor en entrada A1 29
7.6. Celda IND4D0 - NAND con inversor en entrada A1 30
7.7. Celda INR2D0 - NOR con inversor en entrada A1 32
7.8. Celda INR3D0 - NOR con inversor en entrada A1 33
7.9. Celda INR4D0 - NOR con inversor en entrada A1 34
7.10. Celda INVD0 - Inversor 36
7.11. Celda INVD1 - Inversor 37
7.12. Celda IOA21D0 - AND-OR-INVERT con inversores en entradas A1 y A2 . . 38
7.13. Celda MAOI22D0 - *Majority* AND-OR-INVERT 39
7.14. Celda MOAI22D0 - *Majority* OR-AND-INVERT 41
7.15. Celda ND3D0 - NAND de 3 entradas 43
7.16. Celda ND4D0 - NAND de 4 entradas 44
7.17. Celda NR2D0 - NOR de 2 entradas 45
7.18. Celda NR2XD0 - NOR de 2 entradas 46
7.19. Celda NR3D0 - NOR de 3 entradas 47
7.20. Celda NR4D0 - NOR de 4 entradas 49

8. Flujo de trabajo 51
8.1. Generación del archivo RTL 51
8.2. Generación del *netlist* a nivel de celdas estádar 52
8.3. Archivo GDSII 52
8.4. Obtención de *headers* y conversión a formato (CDL) 52
8.5. Archivo `.icv` 53
8.6. Ejecución de la prueba LVS 53
8.7. Ejecución de la prueba ERC 54

9. Pasos previos a realizar las pruebas 56
9.1. Extracción de *headers* de las librerías de celdas 56
9.2. Concatenación de los *headers* 61
9.3. Archivo SPICE (CDL) 61
9.4. Archivo ICV 62

10. Prueba *Layout vs. Schematic* (LVS) 67
10.1. Selección del *runset* 67
10.2. Configuración LVS 67
10.3. Configuración del flujo de las cajas negras 72
10.4. Bloque de comparación 87
10.5. Ejecución de la prueba LVS 98
10.6. Archivos generados 99
10.7. Réplica de la prueba LVS con tecnología de 180 nm 100
10.8. Resultados de la prueba LVS con tecnología de 65 nm 102
10.9. Análisis de resultados 105

11. Prueba *Electrical Rule Check* (ERC) **106**
11.1. Selección del *runset* . . . 106
11.2. Configuración de la prueba ERC . . . 106
11.3. Bloque de verificación . . . 109
11.4. Ejecución de la prueba ERC . . . 115
11.5. Archivos generados . . . 115
11.6. Réplica de la prueba ERC con tecnología de 180 nm . . . 116
11.7. Resultados de la prueba ERC con tecnología de 65 nm . . . 118
11.8. Análisis de resultados . . . 122
11.9. Trabajo Futuro . . . 122

12. Conclusiones **124**

13. Recomendaciones **126**

14. Referencias **127**

15. Glosario **130**

Índice de figuras

1. Compuerta lógica que conforma la celda CKND2D1 25
2. Compuertas lógicas que conforman la celda IIND4D0 26
3. Compuertas lógicas que conforman la celda IINR4D0 28
4. Compuertas lógicas que conforman la celda IND2D0 29
5. Compuertas lógicas que conforman la celda IND3D0 30
6. Compuertas lógicas que conforman la celda IND4D0 32
7. Compuertas lógicas que conforman la celda INR2D0 33
8. Compuertas lógicas que conforman la celda INR3D0 34
9. Compuertas lógicas que conforman la celda INR4D0 36
10. Compuertas lógicas que conforman la celda INVD0 37
11. Compuertas lógicas que conforman la celda INVD1 38
12. Compuertas lógicas que conforman la celda IOA21D0 39
13. Compuertas lógicas que conforman la celda MAOI22D0 41
14. Compuertas lógicas que conforman la celda MOAI22D0 42
15. Compuertas lógicas que conforman la celda ND3D0 44
16. Compuertas lógicas que conforman la celda ND4D0 45
17. Compuertas lógicas que conforman la celda NR2D0 46
18. Compuertas lógicas que conforman la celda NR2XD0 47
19. Compuertas lógicas que conforman la celda NR3D0 48
20. Compuertas lógicas que conforman la celda NR4D0 50

21. Carpeta dedicada a LVS con los archivos necesarios para la prueba 100
22. Terminal con el comando introducido para correr la prueba LVS 101
23. Archivos generados después de correr la prueba LVS 101
24. Archivo `topcell.LVS_ERRORS` de la prueba LVS con tecnología de 180 nm . . 102
25. Archivo `IO_NOT.LVS_ERRORS` de la prueba LVS con tecnología de 65 nm . . . 103
26. Archivo `IO_ALU.LVS_ERRORS` de la prueba LVS con tecnología de 65 nm . . . 104
27. Archivo `IO_nanochip.LVS_ERRORS` de la prueba LVS con tecnología de 65 nm 105

28. Carpeta dedicada a ERC con los archivos necesarios para la prueba 116
29. Terminal con el comando introducido para correr la prueba ERC 117
30. Archivos generados después de correr la prueba ERC 117
31. Archivo `topcell.RESULTS` de la prueba ERC con tecnología de 180 nm 118

32. Archivo topcell.LAYOUT_ERRORS de la prueba ERC con tecnología de 180 nm 118
33. Archivo IO_NOT.RESULTS de la prueba ERC con tecnología de 65 nm 119
34. Archivo IO_NOT.LAYOUT_ERRORS de la prueba ERC con tecnología de 65 nm . 119
35. Archivo IO_ALU.RESULTS de la prueba ERC con tecnología de 65 nm 120
36. Archivo IO_ALU.LAYOUT_ERRORS de la prueba ERC con tecnología de 65 nm . 120
37. Archivo IO_nanochip.RESULTS de la prueba ERC con tecnología de 65 nm . . 121
38. Archivo IO_nanochip.LAYOUT_ERRORS de la prueba ERC con tecnología de 65 nm 121

Índice de cuadros

1. Ejemplo de la función `lvs_black_box_options()` . . . 17
2. Archivo `ADD4.sp` del Caso 1 . . . 17
3. Archivo `add4.net` del Caso 1 . . . 18
4. Archivo `ADD4.sp` del Caso 2 . . . 18
5. Archivo `add4.net` del Caso 2 . . . 19
6. Archivo `add4_lvs.log` del Caso 2 . . . 19
7. Corrección del error del Caso 2 . . . 19
8. Archivo `ADD4.sp` del Caso 3 . . . 20
9. Archivo `add4.net` del Caso 3 . . . 20
10. Archivo `add4_lvs.log` del Caso 3 . . . 21
11. Corrección del error del Caso 3 . . . 21

12. Celda CKND2D1 . . . 24
13. Tabla de verdad de la celda CKND2D1 . . . 25
14. Celda IIND4D0 . . . 25
15. Tabla de verdad de la celda IIND4D0 . . . 26
16. Celda IINR4D0 . . . 27
17. Tabla de verdad de la celda IINR4D0 . . . 27
18. Celda IND2D0 . . . 28
19. Tabla de verdad de la celda IND2D0 . . . 29
20. Celda IND3D0 . . . 29
21. Tabla de verdad de la celda IND3D0 . . . 30
22. Celda IND4D0 . . . 31
23. Tabla de verdad de la celda IND4D0 . . . 31
24. Celda INR2D0 . . . 32
25. Tabla de verdad de la celda INR2D0 . . . 33
26. Celda INR3D0 . . . 33
27. Tabla de verdad de la celda INR3D0 . . . 34
28. Celda INR4D0 . . . 35
29. Tabla de verdad de la celda INR4D0 . . . 35
30. Celda INVD0 . . . 36
31. Tabla de verdad de la celda INVD0 . . . 36
32. Celda INVD1 . . . 37

33. Tabla de verdad de la celda INVD1 . . . 37
34. Celda IOA21D0 . . . 38
35. Tabla de verdad de la celda IOA21D0 . . . 39
36. Celda MAOI22D0 . . . 40
37. Tabla de verdad de la celda MAOI22D0 . . . 40
38. Celda MOAI22D0 . . . 41
39. Tabla de verdad de la celda MOAI22D0 . . . 42
40. Celda ND3D0 . . . 43
41. Tabla de verdad de la celda ND3D0 . . . 43
42. Celda ND4D0 . . . 44
43. Tabla de verdad de la celda ND4D0 . . . 45
44. Celda NR2D0 . . . 46
45. Tabla de verdad de la celda NR2D0 . . . 46
46. Celda NR2XD0 . . . 47
47. Tabla de verdad de la celda NR2XD0 . . . 47
48. Celda NR3D0 . . . 48
49. Tabla de verdad de la celda NR3D0 . . . 48
50. Celda NR4D0 . . . 49
51. Tabla de verdad de la celda NR4D0 . . . 49

52. *Script* `extraer_headers.py` para extraer los *headers* de las librerías de celdas estándar . . . 58
53. Celda IIND4D0 original . . . 60
54. Celda IIND4D0 después de aplicar el *script* de extracción de *headers* . . . 61
55. Celda IIND4D0 en Verilog . . . 62
56. Celda IIND4D0 en formato SPICE (CDL) . . . 62
57. *Script* `limpiar_netlist.py` para eliminar instancias no reconocidas por IC Validator . . . 64
58. Celda IIND4D0 en Verilog estructural . . . 65
59. Celda IIND4D0 en formato SPICE . . . 66
60. Celda IIND4D0 en formato ICV . . . 66

61. *Environment Setup* del *runset* sin ninguna modificación . . . 68
62. *Environment Setup* del *runset* modificado . . . 70
63. Función `compare()` sin ninguna modificación . . . 71
64. Función `compare()` modificada . . . 72
65. Configuración de LVS para cajas negras . . . 73
66. *Script* `crear_blackboxes.py` para generar las definiciones de las cajas negras . . . 75
67. Definición de las cajas negras en el *runset* utilizadas para "El Gran Jaguar" . . . 77
68. *Compare Settings* del *runset* . . . 88
69. Función `match()` del *runset* . . . 89
70. Función *compare()* del *runset* . . . 91
71. Archivo `icv_compare.rh` . . . 93
72. Bloque SPICE del *runset* . . . 97

73. Sección del *Environment Setup* sin modificaciones para correr ERC . . . 107
74. *Environment Setup* del *runset* modificado para correr ERC . . . 108
75. Bloque de verificación de *Well to P/G Check* . . . 110

76. Bloque de verificación de *Gate to P/G Check* 111
77. Bloque de verificación de *Path Check* 112
78. Bloque de verificación de *D/S to P/G Check* 113
79. Bloque de verificación de *Floating Well Check* 114

Resumen

El presente trabajo aborda la verificación física y eléctrica del circuito integrado "El Gran Jaguar", diseñado con tecnología de 65 nm utilizando librerías de TSMC. El objetivo principal es la búsqueda de la equivalencia eléctrica entre el diseño lógico y su implementación física, así como asegurar la integridad de las conexiones y la confiabilidad del circuito antes de un posible proceso de fabricación.

Se ejecutaron dos verificaciones fundamentales dentro del flujo *RTL-to-GDSII*: la prueba de *Layout Versus Schematic* (LVS) y *Electrical Rule Check* (ERC). Para ello, se emplearon herramientas de Synopsys como IC Compiler II, IC Validator y NetTran, además de la configuración de runsets específicos para tecnología de 65 nm. Debido a que el diseño incluye celdas estándar cuyo back-end no es accesible, se implementó un esquema de verificación tipo *black box* para validar únicamente las conexiones externas de dichas celdas. Además, serán incluidas etapas de preparación del entorno, réplica de diseños previos, análisis de resultados y documentación detallada del proceso, con el fin de que la replicabilidad del proyecto sea garantizada.

Adicionalmente, se desarrolló una fracción representativa de las celdas estándar en Verilog para su verificación funcional mediante una FPGA Digilent Genesys, reforzando la consistencia del trabajo realizado durante la etapa de síntesis física.

Los resultados obtenidos contribuirán a validar la funcionalidad del nanochip y establecerán un precedente para futuros desarrollos en microelectrónica avanzada en la región. Este trabajo no solo contribuye al avance tecnológico local, sino que también fortalece las capacidades de diseño y verificación de circuitos integrados en tecnologías nanométricas, abriendo nuevas oportunidades en campos como dispositivos médicos, IoT y telecomunicaciones.

Palabras clave: nanochip, LVS, ERC, TSMC, verificación física.

Abstract

This paper addresses the physical and electrical verification of the "El Gran Jaguar" integrated circuit, designed with 65 nm technology using TSMC libraries. The main objective is to find electrical equivalence between the logical design and its physical implementation, as well as to ensure the integrity of the connections and the reliability of the circuit before a possible manufacturing process.

Two fundamental verifications were performed within the *RTL-to-GDSII* flow: the *Layout Versus Schematic* (LVS) verification and the *Electrical Rule Check* (ERC). To do this, Synopsys tools such as IC Compiler II, IC Validator, and NetTran were used, in addition to the configuration of specific runsets for 65 nm technology. Because the design includes standard cells whose back-end is not accessible, a *black box* verification scheme was implemented to validate only the external connections of these cells. In addition, stages for environment preparation, replication of previous designs, analysis of results, and detailed documentation of the process will be included in order to guarantee the replicability of the project.

In addition, a representative fraction of the standard cells was developed in Verilog for functional verification using a Digilent Genesys FPGA, reinforcing the consistency of the work carried out during the physical synthesis stage.

The results obtained will contribute to validating the functionality of the nanochip and set a precedent for future developments in advanced microelectronics in the region. This work not only contributes to local technological advancement, but also strengthens the design and verification capabilities of integrated circuits in nanometric technologies, opening up new opportunities in fields such as medical devices, IoT, and telecommunications.

Keywords: nanochip, LVS, ERC, TSMC, physical verification.

CAPÍTULO 1

Introducción

El desarrollo de circuitos integrados en tecnologías nanométricas se ha consolidado como un pilar fundamental de la microelectrónica moderna, permitiendo integrar millones o incluso miles de millones de transistores en un único chip de silicio. Esta capacidad ha habilitado aplicaciones de alta complejidad en ámbitos como telecomunicaciones, dispositivos médicos, sistemas embebidos e Internet de las Cosas (IoT), donde se requieren soluciones con alto desempeño, bajo consumo de potencia y elevada confiabilidad. Dentro de este contexto surge el proyecto "El Gran Jaguar", el primer nanochip desarrollado en Guatemala y Centroamérica, concebido como un esfuerzo académico y tecnológico para establecer un flujo completo de diseño y verificación de circuitos integrados en tecnología de 65 nm con librerías de TSMC, gestionadas a través de IMEC como organización intermediaria entre la Universidad del Valle de Guatemala y la fundición.

La consolidación de "El Gran Jaguar" es el resultado de múltiples trabajos de graduación previos realizados en la Universidad del Valle de Guatemala, que han abarcado desde la síntesis lógica de circuitos digitales, la verificación funcional mediante simulación, el diseño físico y la generación del archivo GDSII, hasta la realización de pruebas de diseño como DRC, LVS y ERC, así como la implementación de entornos de verificación en FPGA. Estos antecedentes han permitido establecer un flujo *RTL-to-GDSII* funcional y replicable, sobre el cual se construye el presente trabajo, enfocado específicamente en la verificación física y eléctrica del nanochip utilizando herramientas profesionales de Synopsys.

En particular, este trabajo aborda la verificación física y eléctrica del circuito integrado "El Gran Jaguar" en tecnología de 65 nm, con el objetivo principal de garantizar la equivalencia eléctrica entre el diseño lógico y su implementación física, así como la integridad de las conexiones antes de un eventual proceso de fabricación. Para ello se ejecutan dos verificaciones fundamentales dentro del flujo *RTL-to-GDSII*: la prueba de LVS (*Layout Versus Schematic*) y la prueba de ERC (*Electrical Rule Check*), utilizando herramientas como IC Compiler II, IC Validator y NetTran, junto con *runsets* específicos para la tecnología empleada.

La verificación LVS tiene como finalidad comprobar que el *netlist* extraído del *layout*

físico coincide con el *netlist* de referencia generado durante la síntesis lógica, de modo que cada dispositivo y cada red estén correctamente mapeados entre ambas representaciones. Un resultado LVS *clean* implica que el *layout* implementado en silicio es eléctricamente equivalente al esquemático, reduciendo el riesgo de errores como nodos desconectados, redes intercambiadas o dispositivos omitidos que podrían provocar fallas irrecuperables después de la fabricación del chip.

Por su parte, la verificación ERC se enfoca en asegurar la integridad eléctrica del circuito, verificando la correcta polarización de pozos, la ausencia de nodos flotantes, la inexistencia de conexiones indebidas hacia potencia o tierra y la presencia de estructuras de protección frente a fenómenos como descarga electrostática (ESD) y *latch-up*. Un resultado de ERC libre de violaciones no solo avala la funcionalidad lógica del diseño, sino también su confiabilidad a largo plazo y su capacidad para operar de manera segura en condiciones reales de servicio.

En el caso particular de "El Gran Jaguar", el uso de librerías comerciales de TSMC, introduce el reto adicional de no contar con el *back-end* de las celdas estándar. Ante esta limitación, el trabajo implementa un esquema de verificación tipo *black box* para dichas celdas, de forma que IC Validator valide exclusivamente las conexiones externas sin requerir la descripción interna de los dispositivos. Este enfoque exige una preparación cuidadosa de los *netlists* de entrada, la configuración de los *runsets* y la definición explícita de las *black boxes* en el entorno de verificación.

Además de las verificaciones LVS y ERC, el proyecto incorpora la creación de una fracción representativa de las celdas estándar en *Verilog* y su posterior implementación en una FPGA Digilent Genesys. Esta validación funcional en *hardware* programable contribuye a incrementar la confianza en el diseño, al demostrar que con las celdas seleccionadas el nanochip es funcional y cumple su propósito.

Metodológicamente, el trabajo se desarrolla sobre una máquina virtual con sistema operativo Rocky Linux y hace uso exclusivo de herramientas de Synopsys, empleando los *decks* y *runsets* proporcionados por TSMC para la tecnología de 65 nm. El flujo propuesto incluye la obtención del *netlist* a nivel de celdas estándar, la obtención del archivo GDSII, la extracción y concatenación de *headers*, la conversión a formato SPICE (CDL) y a formato ICV mediante NetTran, la configuración de las opciones de LVS con *black boxes* y las verificaciones ERC correspondientes. Todo este proceso se encuentra documentado de manera detallada para facilitar su replicación en futuros proyectos.

El impacto esperado de las verificaciones llevadas a cabo es doble. En primer lugar, se busca validar la correcta implementación física de "El Gran Jaguar", asegurando su funcionalidad antes de una posible fabricación y estableciendo una referencia de calidad para futuros diseños en tecnologías nanométricas. En segundo lugar, la sistematización de los procedimientos y *scripts* desarrollados pretende reducir la curva de aprendizaje para nuevos estudiantes e investigadores, fortaleciendo las capacidades locales en verificación física VLSI y contribuyendo a cerrar la brecha tecnológica en microelectrónica avanzada en la región.

Finalmente, se concluye que el flujo desarrollado permitió llevar a cabo de forma exitosa las verificaciones LVS y ERC del circuito integrado "El Gran Jaguar" en tecnología de 65 nm, estableciendo una relación clara entre el diseño lógico y su implementación física. El análisis de LVS confirmó la equivalencia eléctrica entre el *netlist* y el *layout*, validando la

correcta integración jerárquica y de interconexiones del chip. Sin embargo, la verificación ERC evidenció la presencia de sustratos tipo P flotantes en el anillo de entradas y salidas, lo cual representa una condición eléctrica inestable que debe corregirse antes de avanzar hacia una etapa de fabricación. Además, la necesidad de utilizar un enfoque de *black-box* destacó la importancia de las estrategias industriales para proteger la propiedad intelectual de celdas estándar.

CAPÍTULO 2

Antecedentes

El presente trabajo se fundamenta en una serie de investigaciones previas realizadas en la Universidad del Valle de Guatemala que han sentado las bases para el desarrollo del primer nanochip guatemalteco, denominado "El Gran Jaguar". Este proyecto, iniciado en 2019, representa un hito en la historia de la ingeniería electrónica centroamericana, al establecer por primera vez un flujo completo de diseño para circuitos integrados en tecnologías nanométricas.

Los primeros antecedentes se remontan al trabajo de Jonathan de los Santos [1], quien exploró el diseño de circuitos digitales a escala nanométrica mediante herramientas de Synopsys. Su investigación sobre un sumador/restador de 32 bits utilizando tecnología CMOS de 28 nm demostró la viabilidad de implementar diseños complejos con las librerías proporcionadas por Synopsys, y creó una documentación detallada sobre la instalación y configuración de las herramientas utilizadas. Este trabajo pionero sentó las bases metodológicas para el uso de herramientas EDA *(Electronic Design Automation)* en el contexto universitario guatemalteco.

Un avance significativo ocurrió en 2019 con las investigaciones de Luis Nájera [2] y Steven Rubio [3], quienes establecieron los primeros pasos del flujo de diseño para circuitos nanométricos. Sus contribuciones se centraron en la síntesis lógica a partir de descripciones HDL, utilizando Design Vision con librerías de 90 nm. Desarrollaron metodologías para la verificación funcional con VCS y establecieron protocolos para la generación de *netlists* a nivel de compuertas. Estos avances permitieron a futuros investigadores acelerar su curva de aprendizaje.

El año 2020 marcó un punto de inflexión con tres trabajos clave. Matthias Sibrian [4] se enfocó en la prueba *Design Rule Check* (DRC) para el circuito integrado, utilizando Custom Compiler y *runsets* proporcionados por TSMC. Su investigación permitió validar el *layout* físico del diseño contra las reglas de fabricación, generando un archivo GDSII limpio. Paralelamente, Ricardo Girón [5] desarrolló la metodología para *Layout Versus Schematic* (LVS), implementando un flujo de verificación física que incluía la extracción de *netlists* y su comparación mediante IC Validator. Por su parte, Marvin Flores [6] abordó el diseño

del anillo de entradas y salidas *(I/O ring)*, incluyendo pruebas de antena y *Electrical Rule Check* (ERC), aspectos críticos para garantizar la integridad del chip durante la fabricación.

En 2021, Sophia Cardona [7] y Elmer Torres [8] profundizaron en la etapa de síntesis lógica. Sophia Cardona optimizó los *scripts* de síntesis en Design Vision, realizando pruebas exhaustivas con diversos circuitos digitales, desde compuertas básicas hasta una ALU completa. Elmer Torres, por su parte, implementó un entorno de verificación funcional mediante *testbenches* en VCS, permitiendo validar el comportamiento de los diseños antes de proceder a la implementación física. Ambos trabajos contribuyeron a robustecer la primera etapa del flujo de diseño.

La evolución continuó en 2022 con tres contribuciones fundamentales. Carlos Letona [9] exploró el uso avanzado de StarRC para la extracción de parásitos, desarrollando metodologías para generar modelos precisos de resistencias y capacitancias parásitas. Estuardo Mancio [10] implementó un entorno de verificación en FPGA utilizando placas Genesys de Xilinx, permitiendo validar funcionalmente el diseño del nanochip. Finalmente, Luis Gómez [11] desarrolló un sistema completo de demostración que incluía un FPGA Genesys Board, un microcontrolador Tiva C para comunicación UART y una interfaz en Python con síntesis de voz, mostrando las capacidades del diseño final.

Finalmente, en 2024 se dieron las contribuciones más recientes a esta trayectoria de trabajo por Diana Alvarado [12], quien retomó y amplió las contribuciones previas en LVS, ERC y extracción de parásitos, elaborando un flujo de diseño con el propósito de mejorar el proceso de desarrollo llevado a cabo en años anteriores, abarcando desde el posicionamiento e interconexión de los componentes hasta la corrección de errores y problemas encontrados como consecuencia de realizar las verificaciones propuestas en varios tipos de circuitos, y finalmente en “El Gran Jaguar”.

Estos antecedentes demuestran cómo, a lo largo de años de investigación continua, se ha establecido un flujo de diseño completo que abarca desde la descripción HDL inicial hasta la verificación física final. Cada trabajo ha contribuido a resolver desafíos específicos, acumulando un conocimiento invaluable sobre el diseño de circuitos integrados en tecnologías nanométricas. El presente proyecto se basa en esta sólida base para avanzar en las etapas de verificación física utilizando tecnología de 65 nm.

CAPÍTULO 3

Justificación

El desarrollo del presente trabajo de graduación adquiere relevancia tanto técnica como estratégica al abordar una de las etapas más críticas en el diseño de circuitos integrados modernos: la verificación física y eléctrica de un chip en tecnología nanométrica.

Desde el punto de vista académico, esta investigación contribuye a consolidar el conocimiento en microelectrónica avanzada dentro de la región, demostrando que es posible dominar metodologías de verificación profesional como lo son LVS (*Layout Versus Schematic*) y ERC (*Electrical Rule Check*), tradicionalmente reservadas a centros de investigación con mayores recursos.

En este contexto, las pruebas de *Layout Versus Schematic* y (*Electrical Rule Check*) no solo representan pasos formales dentro del flujo de diseño, sino mecanismos de aseguramiento de calidad que certifican la equivalencia, confiabilidad y robustez del chip. Un resultado LVS clean garantiza que el *layout* implementado en silicio representa eléctricamente el mismo circuito descrito lógicamente durante la síntesis, evitando errores como nodos desconectados, redes intercambiadas, dispositivos omitidos o modificaciones no intencionales introducidas durante la fase física. La ausencia de correspondencia entre *layout* y esquemático puede conducir a fallas funcionales irreversibles, que únicamente se manifiestan después de fabricar el circuito, generando costos económicos sustanciales y pérdida de esfuerzos de diseño.

Por su parte, un ERC clean asegura la integridad eléctrica del circuito al verificar la correcta polarización de pozos, la ausencia de nodos flotantes, conexiones indebidas hacia potencia o tierra, y la protección del chip contra fenómenos como descarga electrostática (ESD) y *latch-up*. Si estas verificaciones no se superan, el circuito puede presentar fallas letales para el circuitp, degradación acelerada, comportamiento inestable o destrucción inmediata al ser energizado. Por tanto, ERC no se limita a evaluar funcionalidad lógica, sino fiabilidad a largo plazo y supervivencia física del chip en operación real.

De esta forma, el éxito de las verificaciones LVS y ERC tiene implicaciones directas en la viabilidad del proyecto "El Gran Jaguar" como tecnología funcional. Superarlas con un estado clean establece la base necesaria para avanzar hacia etapas como extracción de parásitos, y

eventualmente, una posible fabricación. La documentación generada servirá como referencia para proyectos posteriores, reduciendo la curva de aprendizaje y fomentando la formación de nuevos especialistas en diseño VLSI en la región. Así, el presente trabajo no solo valida la calidad del diseño actual, sino que consolida un procedimiento replicable para futuros desarrollos de circuitos integrados en la región.

En el contexto del proyecto “El Gran Jaguar”, el primer nanochip desarrollado en Centroamérica, la metodología desarrollada acerca a Guatemala a la posibilidad de participar en la cadena de valor de la microelectrónica global, demostrando competencias en flujos de diseño compatibles con estándares internacionales. La colaboración con instituciones como IMEC y el uso de herramientas profesionales de Synopsys validan el enfoque adoptado y abren puertas para futuras alianzas tecnológicas.

Este proyecto representa un paso concreto del dominio tecnológico del país, ya que los resultados obtenidos no solo validarán el funcionamiento de “El Gran Jaguar”, sino que establecerán un precedente para el desarrollo de futuros chips con aplicaciones en áreas prioritarias para el desarrollo nacional.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Evaluar la confiabilidad e integridad del circuito integrado "El Gran Jaguar", d iseñado con tecnología de 65 nm de librerías de TSMC, mediante el análisis exhaustivo de los resultados de las verificaciones de *Layout vs. Schematic* (LVS) y *Electrical Rule Check* (ERC), para así caracterizar la calidad del diseño tanto en el núcleo como en el anillo de entradas y salidas, documentando y comprendiendo las discrepancias identificadas para fundamentar las decisiones de ajuste y mejora.

4.2. Objetivos específicos

- Contextualizar los fundamentos teóricos y el funcionamiento de las pruebas de *Layout vs. Schematic* (LVS) y *Electrical Rule Check* (ERC), explicando su propósito dentro del flujo de diseño de circuitos integrados, los tipos de errores que detectan y su importancia para garantizar la funcionalidad y confiabilidad del chip.

- Ejecutar y analizar las pruebas de *Layout vs. Schematic* (LVS) en el núcleo del circuito integrado y en el anillo de entradas y salidas (I/O) para identificar y documentar las discrepancias entre el diseño físico y el esquemático. El análisis se centrará en interpretar el reporte LVS, la documentación de errores y la relevancia de los hallazgos según las reglas de TSMC.

- Aplicar y examinar las pruebas de *Electrical Rule Check* (ERC) al núcleo del circuito integrado con el fin de identificar violaciones a las reglas eléctricas y realizar un análisis documentado que describa la naturaleza de las incidencias encontradas y su impacto potencial según los estándares de TSMC.

- Elaborar una guía detallada con recursos multimedia que documente el flujo de trabajo, la configuración de las pruebas LVS y ERC, y el proceso de análisis de los reportes

generados para facilitar la replicabilidad del proyecto y sentar una base para futuras investigaciones.

- Crear un tercio de las celdas del circuito integrado “El Gran Jaguar” en Verilog, para luego utilizarlas en la FPGA Diligent Genesys como parte del proceso de verificación funcional previa a la implementación física del diseño.

CAPÍTULO 5

Alcance

El presente trabajo de graduación se enfoca en la verificación física y eléctrica del circuito integrado "El Gran Jaguar", diseñado con tecnología de 65 nm utilizando las librerías de TSMC. El alcance del proyecto abarca las siguientes dimensiones:

1. Alcance técnico

   - Creación de celdas en Verilog
     - Se desarrolló una fracción representativa de las celdas estándar utilizadas en el diseño del circuito integrado "El Gran Jaguar" en el lenguaje de descripción de hardware Verilog.
     - Estas celdas fueron sintetizadas y probadas en la FPGA *Diligent Genesys* para validar su funcionalidad antes de la implementación física del diseño.
   - Verificación LVS (*Layout Versus Schematic*)
     - Se realizó una comparación exhaustiva entre el *netlist* extraído del layout físico y el *netlist* de referencia generado durante la síntesis lógica.
     - Se aplicó un enfoque de *black box LVS* para las celdas estándar de TSMC, verificando únicamente las conexiones externas.
     - Se incluyó la depuración de errores comunes como discrepancias en conexiones, dispositivos y jerarquías.
   - Verificación ERC (*Electrical Rule Checking*)

     Se ejecutaron pruebas para garantizar el cumplimiento de las reglas eléctricas fundamentales, incluyendo:
     - Integridad estructural (como nodos flotantes o cortocircuitos).
     - Protección contra eventos extremos (como descargas electrostáticas o sobrevoltajes).
     - Consistencia eléctrica (como polaridades, niveles de voltaje, relaciones de ancho y longitud de terminales críticas).

    - Se utilizaron las capacidades nativas de IC Validator y *scripts* personalizados para cubrir requisitos específicos del diseño.
  - Herramientas y entorno de trabajo
    - El proyecto se desarrolló en una máquina virtual con Rocky Linux, utilizando sólamente herramientas de Synopsys.
    - Se configuraron los *decks* proporcionados por TSMC para garantizar la precisión en las verificaciones.

2. Alcance de aplicación

   El trabajo se centró en el núcleo y el anillo de entradas/salidas del circuito integrado, por lo que no incluye:

   - Modificaciones al diseño lógico o físico del chip.
   - Verificación funcional mediante simulación (ya abordada en etapas previas).

3. Alcance documental

   Se generó una guía detallada con los pasos para ejecutar las verificaciones LVS y ERC en tecnología de 65 nm, incluyendo:

   - Generación de archivos de entrada.
   - Ejecución de pruebas.
   - Interpretación de reportes de error.
   - Se elaboró materiales multimedia para facilitar la replicabilidad del proyecto.

   De igual manera, se documentó el proceso de creación de las celdas en Verilog, incluyendo una descripción de cada una de las celdas y su funcionalidad.

4. Limitaciones

   - Dependencia de las librerías y *decks* proporcionados por TSMC, sin acceso a modificaciones internas.
   - Las verificaciones LVS y ERC se limitaron a las conexiones externas de las celdas estándar en formato de (*black box*).
   - Los recursos y herramientas a utilizar condicionan la profundidad del estudio, principalmente, en que no es posible llevar a cabo el análisis de efectos parásitos al no poseer el *backend* de las celdas estándar.

5. Impacto esperado

   - Validar la correcta implementación física de "El Gran Jaguar", asegurando su funcionalidad antes de una eventual fabricación.
   - Establecer un precedente metodológico para futuros diseños de circuitos integrados en la región, especialmente en tecnologías nanométricas.
   - Fortalecer las capacidades locales en verificación física VLSI, reduciendo la brecha tecnológica en microelectrónica avanzada.

CAPÍTULO 6

Marco teórico

6.1. *VLSI Design*

Very-large-scale integration design o "Diseño de integración a muy gran escala" se refiere al proceso de creación de circuitos integrados mediante la combinación de transistores en un único chip de silicio. Este proceso incluye la metodología, las herramientas y las técnicas utilizadas para diseñar y fabricar dispositivos semiconductores complejos de gran funcionalidad y rendimiento.

Esta tecnología le ofrece al usuario una gama nueva y más compleja de circuitos, y los procesos de diseño VLSI son tales que los diseñadores de sistemas pueden diseñar fácilmente sus propios circuitos especializados sin que la complejidad de estos sea un obstáculo, lo que proporciona un nuevo grado de libertad a los diseñadores para producir avances significativos. A medida que los avances tecnológicos reducen el tamaño de los circuitos integrados en silicio, la densidad de integración aumenta rápidamente. Los avances en las tecnologías de litografía y grabado han permitido a la industria reducir las dimensiones físicas de los transistores y empaquetar más transistores en la misma superficie del chip. Estos avances, combinados con un crecimiento constante del tamaño de los chips, han dado lugar a un crecimiento exponencial del número de transistores y bits de memoria por chip. [13]

El diseño VLSI es parte de la columna vertebral del desarrollo de "El Gran Jaguar", ya que permite integrar sistemas complejos en un único chip de silicio mediante tecnologías nanométricas. Esta metodología posibilita la implementación de miles de millones de transistores en el espacio reducido de 65 nm, optimizando tanto el rendimiento como el consumo energético del nanochip. Las herramientas especializadas del flujo VLSI como IC Compiler II para la síntesis física y IC Validator para verificar la fabricabilidad del chip mediante la verificación rigurosa del diseño con las pruebas *Layout Versus Schematic* y *Electrical Rule Checking*, asegurando que el circuito cumpla con todas las especificaciones técnicas antes de su fabricación y garantizando una alta precisión en el diseño del mismo.

La adopción de tecnologías VLSI en este proyecto permite el desarrollo de este y muchos

otros circuitos integrados personalizados sin depender de soluciones comerciales externas en el país. El dominio de estas técnicas no solo hace viable a "El Gran Jaguar", sino que establece capacidades locales para futuros desarrollos en electrónica. La escalabilidad del diseño VLSI asegura que esta área pueda evolucionar hacia nodos tecnológicos más avanzados, manteniendo su competitividad a nivel internacional. [14]

6.2. Flujo de diseño RTL a GDSII

El flujo de diseño RTL a GDSII, o *RTL-to-GDSII*, define la secuencia de pasos que permiten transformar una descripción funcional de un sistema digital escrita a nivel de registro, o *Register Transfer Level* por sus siglas en inglés (RTL), en una representación física del circuito contenida en un archivo GDSII.

Formalmente, el diseño a nivel RTL se expresa mediante lenguajes de descripción de hardware como Verilog o VHDL. Estos describen el comportamiento lógico y la arquitectura estructural del circuito en términos de bloques funcionales, registros, operaciones aritméticas, condiciones y control de flujo de datos. A partir de esta descripción, el primer paso del flujo es la síntesis lógica, donde se genera una *netlist* utilizando celdas estándar de una biblioteca tecnológica específica. Esta *netlist* representa una implementación lógica del diseño optimizada en términos de área, retardo, potencia, o una combinación de estos.

Posteriormente, en la etapa de síntesis física, se asignan ubicaciones físicas a los componentes lógicos dentro del área del chip (*floorplanning* y *placement*), y se conectan físicamente mediante rutas metálicas (*routing*). A lo largo de estas etapas, se realizan análisis de temporización estática (STA), estimación de consumo de potencia, análisis de integridad de señal y otras verificaciones críticas. Finalmente, el diseño se somete a reglas de verificación física como *Design Rule Checking* (DRC) y *Layout Versus Schematic* (LVS), para asegurar que el layout es manufacturable y que representa correctamente el diseño lógico.

Evidentemente, las verificaciones LVS y ERC son fundamentales en el flujo *RTL-to-GDSII*, ya que garantizan la integridad física y eléctrica del diseño antes de su fabricación. Estos chequeos minimizan riesgos de manufactura, evitan retrabajos costosos y aseguran que el chip final cumpla con los requisitos de rendimiento y fabricación.

El resultado final del flujo es un archivo en formato GDSII (*Graphic Data System II*), el cual codifica toda la geometría y capas del *layout* del circuito, el cual contiene la información necesaria para fabricar las máscaras fotolitográficas que definirán físicamente el chip en silicio. Por tanto, el flujo *RTL-to-GDSII* constituye la espina dorsal del proceso de implementación física de circuitos integrados digitales, articulando de manera rigurosa la transición del diseño abstracto al objeto físico manufacturable. [15]

El flujo de diseño *RTL-to-GDSII* tradicional sufre ineficiencias debido a la desconexión entre sus etapas (síntesis, colocación-ruteo y verificación), lo que genera retrabajos y dificulta alcanzar los objetivos de potencia, rendimiento y área. Las herramientas de Synopsys integran todas las etapas en un flujo unificado, comparten motores de optimización entre fases, eliminan márgenes excesivos y reducen iteraciones, logrando una correlación perfecta entre diseño y verificación. Además, incorpora pruebas (DFT) desde el inicio, acelerando el

tiempo de entrega y mejorando la calidad del diseño. El resultado es un proceso más rápido, predecible y con mejores resultados. [16]

6.3. IC Compiler II

El IC Compiler II de Synopsys es una plataforma de implementación física altamente sofisticada que integra un conjunto completo de herramientas para el diseño de circuitos integrados en tecnologías avanzadas. Su arquitectura modular abarca todas las etapas críticas del flujo físico, desde la planificación inicial del diseño (tanto en enfoques planos como jerárquicos) hasta la optimización final de ruteo y el cierre para fabricación.

Esta herramienta integra innovaciones clave como planificación jerárquica, síntesis de árbol de reloj, optimización basada en congestión y enrutamiento avanzado, acelerando el cierre del diseño. Especializado para cumplir con exigentes requisitos de rendimiento, potencia y área (PPA), la plataforma emplea optimización multiobjetivo, técnicas para FinFET y aprendizaje automático (ML) para un cierre predictivo. La incorporación de análisis de caídas de voltaje, cálculo de retardos y optimizaciones de temporización en etapas finales, garantiza convergencia rápida y resultados superiores en diseños complejos. [17]

6.4. IC Validator

El IC Validator de Synopsys es una solución de verificación física de alto rendimiento que, gracias a su arquitectura distribuida, puede escalar hasta más de 4000 núcleos de CPU, permitiendo la verificación completa de chips que contienen miles de millones de transistores en plazos extraordinariamente cortos.

Esta herramienta cubre todo el espectro de verificaciones físicas críticas, incluyendo *Design Rule Checking* (DRC) y *Layout Versus Schematic* (LVS), las verificaciones clave a desarrollar en el presente trabajo.

Su integración coincide con el flujo *RTL-to-GDSII* de Synopsys, particularmente con *IC Compiler* y *Fusion Compiler*, ya que no sólamente permite la detección de errores, sino también su corrección automática en muchos casos, mediante la generación de parches que pueden ser aplicados directamente en el entorno de implementación física. Esta capacidad de çorrección asistidareduce drásticamente los tiempos de cierre del diseño y mejora la productividad general del flujo. [18]

6.5. NetTran

NetTran es una herramienta de Synopsys diseñada para convertir formatos estándar de *netlists*, como SPICE o Verilog, al formato de *netlists* de IC Validator. También puede utilizar la utilidad NetTran de IC Validator para fusionar diferentes archivos de *netlists* en distintos formatos, con el fin de crear un archivo de lista de redes de nivel superior para la

ejecución de LVS. [19]

6.6. *Layout vs. Schematic* (LVS)

La comprobación de diseño frente al esquemático (LVS) compara el *netlist* extraído de la implementación física (*layout*) con el *netlist* del esquemático original del circuito para determinar si coinciden. La comprobación de comparación se considera limpia si todos los dispositivos y las redes del esquema coinciden con los dispositivos y las redes del diseño. Opcionalmente, también se pueden comparar las propiedades de los dispositivos para determinar si coinciden dentro de una tolerancia determinada. Cuando se comparan las propiedades, todas las propiedades deben coincidir también para lograr una comparación limpia.

Esta verificación representa un paso crítico de calidad en el flujo de diseño físico, asegurando que el *layout* del circuito sea eléctricamente equivalente a su representación esquemática original. Este proceso va mucho más allá de una simple comparación geométrica, involucrando una verificación exhaustiva de la conectividad eléctrica, los tipos de dispositivo y los parámetros eléctricos clave; lo que ayuda a minimizar riesgos asociados a errores de diseño y garantizar la viabilidad y calidad del diseño final del chip.

Las herramientas modernas de LVS, como las implementadas en IC Validator, pueden manejar diseños extremadamente complejos mediante técnicas inteligentes de particionamiento y procesamiento paralelo. El flujo típico de LVS comienza con la extracción de un *netlist* a partir de la base de datos del *layout*, identificando todos los dispositivos semiconductores (transistores, diodos, resistencias, etc.) y sus interconexiones. Este *netlist* extraído se compara entonces con el *netlist* de referencia generado a partir del esquemático original o de la descripción RTL. [20]

6.6.1. Flujo de trabajo

El flujo completo de verificación LVS puede descomponerse en cuatro etapas fundamentales, cada una con sus particularidades y desafíos técnicos. [21]

En un inicio, lo más común es realizar una preparación de Archivos de Entrada. Esta etapa inicial implica la recopilación y preparación de todos los archivos necesarios para la verificación. El *layout* físico típicamente se proporciona en formato GDSII, mientras que el *netlist* esquemático puede venir en formatos SPICE o Verilog. Un componente crítico en esta fase es el archivo de reglas (*rule deck*) que contiene todas las especificaciones tecnológicas y parámetros de verificación específicos para el nodo de fabricación (en este caso, 65 nm TSMC). Este archivo define cómo se deben interpretar las diferentes capas del *layout* y establece las tolerancias aceptables para la comparación.[22]

Luego, se hace uso de la herramienta LVS (como IC Validator) para analizar minuciosamente el *layout*, identificando todos los componentes activos (transistores, diodos) y pasivos (resistencias, capacitancias), así como sus interconexiones. La extracción del *netlist* físico debe considerar factores como la identificación correcta de dispositivos a través de las capas, el reconocimiento de las conexiones metálicas entre componentes y la consideración de

efectos de proximidad y patrones repetitivos. [23] [24]

Después, el *netlist* extraído se compara con el esquemático de referencia mediante algoritmos sofisticados que buscan equivalencia eléctrica. En este caso, debido a que TSMC no nos brinda acceso al *back-end* de las celdas que se utilizaron en el circuito, se realizó un *black box LVS*, donde sólo se verifican las conexiones externas a dichas celdas. [25]

6.6.2. Errores comunes

En la práctica, los ingenieros de verificación se enfrentan regularmente a un conjunto recurrente de problemas durante las verificaciones LVS.

- Errores de conexión: estos representan la mayoría de los problemas encontrados en LVS. Pueden manifestarse como: cortocircuitos, cuando dos señales que deberían estar separadas aparecen conectadas en el *layout*; circuitos abiertos, cuando conexiones que deberían existir están rotas o incompletas; o conexiones intercambiadas, cuando dos señales aparecen cruzadas.
- Discrepancias en dispositivos: estos errores ocurren cuando los componentes en el *layout* no coinciden con los del esquemático, tales como la definición de parámetros fuera de especificación o dispositivos faltantes/adicionales dentro del circuito.
- Problemas de jerarquía: comunes en diseños complejos con múltiples niveles de abstracción, presentes cuando existen instancias faltantes o duplicadas, conexiones jerárquicas incorrectas o problemas de *bus ordering* (metodología utilizada para organizar y conectar las señales de un bus en un diseño de circuitos integrados).

Comprender estos errores y sus soluciones típicas es fundamental para acelerar el proceso de depuración. [21] [24]

6.6.3. Black Box LVS

El *black box* LVS es una técnica de verificación que se utiliza cuando el diseño incluye celdas o bloques que no están completamente definidos o accesibles, como en el caso de diseños comerciales donde las celdas son propiedad intelectual y no se proporciona el *layout* detallado. En este enfoque, se verifica la conectividad externa de las celdas sin necesidad de conocer su implementación interna.

Esta verificación implica comparar el *netlist* extraído del *layout* con el *netlist* del esquemático, pero solo en las conexiones externas. Esto permite validar que las interconexiones entre las celdas cumplen con las especificaciones sin necesidad de acceder a los detalles internos de cada celda.

La función `lvs_black_box_options()` de IC Validator permite configurar el entorno y especificar las celdas que se tratarán como cajas negras y cómo deben coincidir sus puertos.

Cuadro 1. Ejemplo de la función `lvs_black_box_options()`

```
lvs_black_box_options(
    equiv_cells = {
        {schematic_cell = "invb", layout_cell = "invb"},
        {schematic_cell = "nor2b", layout_cell = "nor2b"}
    }
);
```

Nota. Este ejemplo mapea las celdas del esquemático “invb” y “nor2b” con las celdas correspondientes en el *layout*.

Algunas opciones de configuración

- `equiv_cells`: mapea nombres de las celdas en el esquemático con nombres en el layout.
- `equiv_ports`: mapea manualmente puertos con nombres diferentes.
- `remove_layout_ports` / `remove_schematic_ports`: ignora puertos adicionales en el *layout* o esquemático.
- `schematic_swappable_ports`: define pines intercambiables (para celdas simétricas).

En este trabajo, aunque sólo existe un caso de uso, es importante conocer los casos fundamentales, ya que los casos más personalizados abordan cada uno un desafío diferente:

Caso 1: Todos los puertos coinciden

En el siguiente ejemplo, todos los puertos del *layout* coinciden con los del esquemático.

Cuadro 2. Archivo `ADD4.sp` del Caso 1

```
.SUBCKT invb GND VDD A Z
M1 GND A Z GND n L=1u W=13u
M2 VDD A Z VDD p L=1u W=20.5u
.ENDS

.SUBCKT nor2b GND VDD QN A B
M1 QN B GND GND n L=1u W=13u
M2 QN A GND GND n L=1u W=13u
M3 n11 B QN VDD p L=1u W=20.5u
M4 n11 A VDD VDD p L=1u W=20.5u
.ENDS
```

Nota. El cuadro contiene el código del esquemático de las celdas “invb” y “nor2b”.

Cuadro 3. Archivo `add4.net` del Caso 1

```
{CELL invb
  {PORT VDD GND Z A}
  {PROP top=50.000000 bottom=0.000000
     left=0.000000 right=12.000000}
}

{CELL nor2b
  {PORT VDD GND QN A B}
  {PROP top=50.000000 bottom=0.000000
     left=0.000000 right=18.000000}
}
```

Nota. El cuadro contiene el código del *layout* de las celdas "invb" y "nor2b".

LVS se ejecuta correctamente hasta su finalización porque el número de puertos y los nombres coinciden. Si el número de puertos o los nombres no coinciden entre el esquema y el diseño, entonces, por defecto, la herramienta IC Validator informa de un error.

Caso 2: Los puertos no coinciden entre las *netlists*

Si el diseño cambia de tal manera que la lista de redes del diseño tiene una red con un nombre diferente al de la lista de redes del esquema, la herramienta IC Validator informa de un error. En el siguiente ejemplo, en la celda "invb", el nombre del puerto cambia de "A" a "A1".

Cuadro 4. Archivo `ADD4.sp` del Caso 2

```
.SUBCKT invb GND VDD A Z
M1 GND A Z GND n L=1u W=13u
M2 VDD A Z VDD p L=1u W=20.5u
.ENDS

.SUBCKT nor2b GND VDD QN A B
M1 QN B GND GND n L=1u W=13u
M2 QN A GND GND n L=1u W=13u
M3 n11 B QN VDD p L=1u W=20.5u
M4 n11 A VDD VDD p L=1u W=20.5u
.ENDS
```

Nota. El cuadro contiene el código del esquemático de las celdas "invb" y "nor2b".

Cuadro 5. Archivo `add4.net` del Caso 2

```
{CELL invb
  {PORT VDD GND Z A1}
  {PROP top=50.000000 bottom=0.000000
     left=0.000000 right=12.000000}
}

{CELL nor2b
  {PORT VDD GND QN A B}
  {PROP top=50.000000 bottom=0.000000
     left=0.000000 right=18.000000}
}
```

Nota. El cuadro contiene el código del *layout* de las celdas "invb" y "nor2b", pero con el puerto cambiado de "A" a "A1" de la celda "invb".

La ejecución de LVS se detiene y muestra el siguiente mensaje en el archivo de registro de LVS (`add4_lvs.log`):

Cuadro 6. Archivo `add4_lvs.log` del Caso 2

```
Processing black boxes' pins...
ERROR: Schematic port "A" does not have a corresponding port in
the layout in blackbox {invb, invb}
ERROR: Layout port "A1" does not have a corresponding port in the
schematic in blackbox {invb, invb}
```

Nota. El cuadro muestra los errores generados por el Caso 2.

Este error se puede corregir definiendo manualmente la asignación de puertos para los pines que no coinciden, utilizando la función `lvs_black_box_options()`:

Cuadro 7. Corrección del error del Caso 2

```
lvs_black_box_options(
    equiv_cells = {{"invb","invb"}},
    equiv_ports = {{"A","A1"}}
);
```

Nota. El cuadro muestra cómo corregir el error generado utilizando la función `lvs_black_box_options()`.

Modelo 3: Puertos extra en la celda de caja negra del esquemático

Cuando se trabaja con celdas sin terminar, o a las que simplemente no se tiene acceso completo, la lista de redes del diseño carece de puertos en comparación con el esquema. El diseño no está lo suficientemente completo como para tener todas las interacciones jerárquicas necesarias para crear los puertos. En el siguiente ejemplo, no se extrajo el puerto "A" de la celda "invb":

Cuadro 8. Archivo `ADD4.sp` del Caso 3

```
.SUBCKT invb GND VDD A Z
M1 GND A Z GND n L=1u W=13u
M2 VDD A Z VDD p L=1u W=20.5u
.ENDS

.SUBCKT nor2b GND VDD QN A B
M1 QN B GND GND n L=1u W=13u
M2 QN A GND GND n L=1u W=13u
M3 n11 B QN VDD p L=1u W=20.5u
M4 n11 A VDD VDD p L=1u W=20.5u
.ENDS
```

Nota. El cuadro contiene el código del esquemático de las celdas "invb" y "nor2b".

Cuadro 9. Archivo `add4.net` del Caso 3

```
{CELL invb
  {PORT VDD GND Z }
  {PROP top=50.000000 bottom=0.000000
     left=0.000000 right=12.000000}
}

{CELL nor2b
  {PORT VDD GND QN A B}
  {PROP top=50.000000 bottom=0.000000
     left=0.000000 right=18.000000}
}
```

Nota. El cuadro contiene el código del *layout* de las celdas "invb" y "nor2b", pero sin el puerto de la "A" de la celda "invb".

La ejecución de LVS se detiene y muestra el siguiente mensaje en el archivo de registro de LVS (`add4_lvs.log`):

Cuadro 10. Archivo `add4_lvs.log` del Caso 3

```
Processing black boxes' pins...
ERROR: Schematic port "A" does not have a corresponding port in
the layout in blackbox {invb, invb}
```

Nota. El cuadro muestra los errores generados por el Caso 3.

Este error se puede corregir eliminando manualmente el puerto utilizando el argumento `remove_schematic_ports` de la función `lvs_black_box_options()`.

Cuadro 11. Corrección del error del Caso 3

```
lvs_black_box_options(
    equiv_cells             = {{"invb","invb"}},
    remove_schematic_ports = {"A"}
);
```

Nota. El cuadro muestra cómo corregir el error generado utilizando la función `lvs_black_box_options()`.

Este es el caso que se presenta para la creación de las *black boxes* de "El Gran Jaguar", ya que no se tiene acceso al *back-end* de las celdas que se utilizarán en el circuito.

Estos casos de uso ilustran los desafíos enfrentados en este trabajo, sin embargo, existen muchos otros escenarios que pueden surgir al trabajar con celdas de caja negra en LVS que no han sido cubiertos aquí. Para más información, se puede consultar la documentación oficial de Synopsys, [26], donde se detallan los casos abordados en esta sección y otros adicionales.

6.7. *Electrical Rule Checking* (ERC)

La comprobación de las reglas eléctricas (ERC) valida los problemas de fiabilidad de los diseños de circuitos integrados que no pueden comprobarse con la comprobación de las reglas de diseño (DRC) o del diseño frente al esquemático (LVS). [27]

Estas comprobaciones de fiabilidad suelen estar relacionadas con la descarga electrostática (ESD), pero también pueden abarcar otras comprobaciones, como la sobrecarga eléctrica (EOS), la ruptura dieléctrica, etc. Estas normas incluyen información sobre conectividad y listas de conexiones, pero también deben permitir una personalización total de un diseño a otro. Estas comprobaciones permiten la detección de cortocircuitos, niveles de voltaje incorrectos en nodos específicos y conexiones flotantes, entre otras configuraciones que podrían comprometer la fiabilidad e integridad del funcionamiento de un chip. [28]

La comprobación programable de reglas eléctricas (PERC) de IC Validator es una solución de verificación de la fiabilidad que permite la comprobación personalizada de las reglas

EOS/ESD/ERC, y admite la comprobación de *Netlist Domain Checks* (NDC), *Mixed-Mode Checks* (MMC), también *Current Density* (CD) y *Point-to-Point Resistance* (P2P). La tecnología *IC Validator PERC* proporciona un rendimiento rápido, escalabilidad y depuración intuitiva para la verificación de la fiabilidad. [29]

En tecnologías avanzadas como la de 65 nm, ERC adquiere mayor complejidad debido al aumento de dominios de voltaje en diseños modernos, mayor sensibilidad a efectos parásitos y requerimientos más estrictos de consumo energético, además de la necesidad de protección avanzada contra ESD.

Un ERC completo debe considerar no solo las reglas genéricas de fabricación, sino también requisitos específicos del diseño, como las densidades máximas de corriente permitidas, la resistencia punto a punto en rutas críticas y los requerimientos especiales para señales sensibles (relojes, resets). [24]

6.7.1. Verificaciones a realizar

Verificación de *taps* conectados de manera incorrecta *(WELL to Power/Ground Check)*

Los *taps* o contactos de pozo (también denominados *substrate ties*) constituyen las conexiones físicas que vinculan los pozos de los transistores con sus respectivas fuentes de potencial. En tecnologías CMOS, los *p-taps* deben conectarse a GND (para el substrato P), mientras que los *n-taps* deben conectarse a VDD (para los pozos N); si sucede lo contrario, se genera una condición de error que puede comprometer la funcionalidad del circuito. [22]

Estas verificaciones son particularmente importantes en diseños de alta densidad, donde la proximidad de múltiples dispositivos puede amplificar los efectos del acoplamiento entre pozos.

Verificación de compuertas conectadas a potencia o tierra *(Gate to Power/-Ground Check)*

Este conjunto de verificaciones fundamentales se orienta a identificar las compuertas de transistores MOS que se encuentran directamente conectadas a los nodos de potencia o tierra. [22] En un diseño funcional correcto, la compuerta debe recibir una señal lógica proveniente de una red de control o de un nodo de interconexión, pero no debe fijarse de manera permanente a un potencial estático.

Cuando una compuerta está conectada directamente a VDD o VSS, se pierde su capacidad de control dinámico y el transistor permanece permanentemente encendido o apagado. Además, este tipo de conexión puede inducir fallas de confiabilidad a largo plazo, debido a diferencias de potencial no controladas.

En consecuencia, este tipo de verificación contribuye tanto a la integridad eléctrica como a la robustez del diseño frente a fenómenos de degradación o daño eléctrico.

Verificación de rutas eléctricas o conexiones *(Path Check)*

Este grupo de verificaciones se centra en la integridad global de la red de interconexión eléctrica del circuito. Estas verificaciones, denominadas *path checks*, aseguran que todas las redes del diseño posean una ruta válida hacia los nodos de potencia, tierra, o hacia una red etiquetada dentro del *netlist*. [22] Para este caso, hay cuatro variantes de este tipo de error que se pueden detectar:

- `PATH1`: nodos que solo tienen conexión hacia potencia, pero no hacia tierra.
- `PATH2`: nodos conectados únicamente a tierra.
- `PATH3`: nodos completamente aislados, sin ruta ni a VDD ni a VSS.
- `PATH4`: nodos sin conexión a ningún red redactada o etiqueta identificable.

[27]

Estos errores suelen originarse por desconexiones accidentales, omisión de etiquetas, o errores de enrutamiento durante la edición del *layout*. Las consecuencias pueden ser significativas: un nodo flotante puede mantener un nivel lógico indefinido, afectar el consumo dinámico o generar oscilaciones ilegítimas. Por ello, las verificaciones de tipo *path check* constituyen la última capa de validación de la conectividad eléctrica completa del diseño.

Verificación de conexiones de drenaje/fuente a potencia y tierra *(Drain/Source to Power/Ground Check)*

Esta categoría de verificaciones ERC tiene como objetivo identificar transistores MOS (tanto NMOS como PMOS) que su terminal de drenaje se encuentra conectada a un nodo de potencia (VDD) y su terminal de fuente está conectada a tierra (VSS). [22] Este tipo de condición representa un error crítico en el diseño, ya que establece un camino de conducción directa entre los dos potenciales de referencia principales del circuito.

Verificación de pozos flotantes *(Floating Well Check)*

Esta última categoría corresponde a las verificaciones de pozos flotantes. En los procesos CMOS, los transistores PMOS se implementan sobre pozos tipo N (*n-well*), mientras que los transistores NMOS se construyen directamente sobre el sustrato tipo P (*p-substrate*). Cada uno de estos pozos o regiones debe estar conectado al potencial adecuado, ya sea alimentación positiva (VDD) o a tierra (VSS). [22]

Cuando un pozo o región del substrato queda flotante, puede adquirir un potencial indeterminado debido a cargas parásitas, inducción o acoplamientos capacitivos, generando inestabilidad en los dispositivos activos. Además, un pozo mal conectado incrementa la susceptibilidad del circuito al fenómeno de *latch-up*, que consiste en la activación inadvertida de estructuras parásitas PNPN internas al silicio, capaces de provocar un cortocircuito persistente entre VDD y GND.

CAPÍTULO 7

Creación de celdas Verilog para uso de la FPGA *Digilent Genesys*

Para poder comprobar el funcionamiento de "El Gran Jaguar" utilizando la FPGA *Digilent Genesys*, fue necesario crear una serie de celdas Verilog que simularan el comportamiento de las celdas lógicas que conforman el circuito integrado. En este trabajo, se encuentra la documentación de un tercio del total de las celdas creadas, incluyendo su módulo Verilog, tabla de verdad y esquemático.

7.1. Celda CKND2D1 - NAND de dos entradas

Compuerta NAND estándar de dos entradas.

7.1.1. Módulo Verilog

Cuadro 12. Celda CKND2D1

```
module CKND2D1 (
    input  A1, input A2,
    output ZN
);
    assign ZN = ~(A1 & A2);
endmodule
```

Nota. El cuadro muestra el código de la celda CKND2D1 en Verilog.

7.1.2. Tabla de verdad

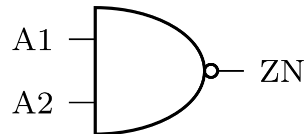
Cuadro 13. Tabla de verdad de la celda CKND2D1

A1	A2	ZN
0	0	1
0	1	1
1	0	1
1	1	0

Nota. El cuadro muestra el comportamiento de la celda CKND2D1.

7.1.3. Esquemático

Figura 1. Compuerta lógica que conforma la celda CKND2D1



Nota. La imagen muestra el esquemático de la celda CKND2D1, que incluye una compuerta NAND de dos entradas.

7.2. Celda IIND4D0 - NAND con inversores en entradas A1 y A2

Esta celda implementa una función NAND de 4 entradas donde las señales A1 y A2 son invertidas antes de ingresar a la compuerta NAND.

7.2.1. Módulo Verilog

Cuadro 14. Celda IIND4D0

```
module IIND4D0 (A1, A2, B1, B2, ZN);
    input A1, A2, B1, B2;
    output ZN;
        assign ZN = ~(B1 & B2 & ~A1 & ~A2);
endmodule
```

Nota. El cuadro muestra el código de la celda IIND4D0 en Verilog.

7.2.2. Tabla de verdad

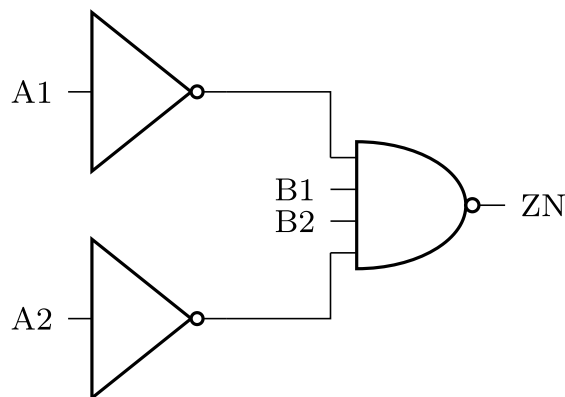
Cuadro 15. Tabla de verdad de la celda IIND4D0

A1	A2	B1	B2	ZN
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Nota. El cuadro muestra el comportamiento de la celda IIND4D0.

7.2.3. Esquemático

Figura 2. Compuertas lógicas que conforman la celda IIND4D0



Nota. La imagen muestra el esquemático de la celda IIND4D0, que incluye dos inversores y una compuerta NAND de cuatro entradas.

7.3. Celda IINR4D0 - NOR con inversores en entradas A1 y A2

Similar a IIND4D0, pero utilizando una compuerta NOR en lugar de NAND. Las entradas A1 y A2 son invertidas antes de la operación NOR.

7.3.1. Módulo Verilog

Cuadro 16. Celda IINR4D0

```
module IINR4D0 (A1, A2, B1, B2, ZN);
    input A1, A2, B1, B2;
    output ZN;
        assign ZN = ~(B1 | B2 | ~A1 | ~A2);
endmodule
```

Nota. El cuadro muestra el código de la celda IINR4D0 en Verilog.

7.3.2. Tabla de verdad

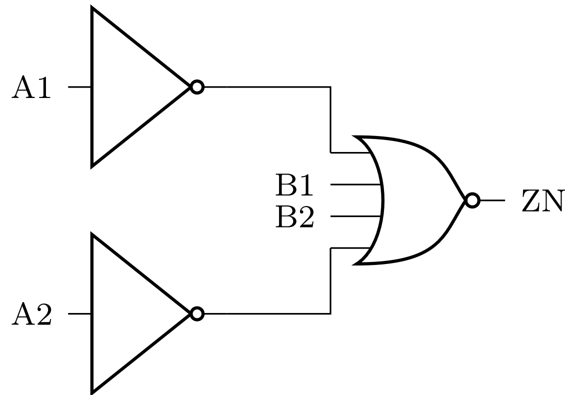
Cuadro 17. Tabla de verdad de la celda IINR4D0

A1	A2	B1	B2	ZN
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda IINR4D0.

7.3.3. Esquemático

Figura 3. Compuertas lógicas que conforman la celda IINR4D0



Nota. La imagen muestra el esquemático de la celda IINR4D0, que incluye dos inversores y una compuerta NOR de cuatro entradas.

7.4. Celda IND2D0 - NAND con inversor en entrada A1

Compuerta NAND de 2 entradas donde la entrada A1 es invertida antes de la operación.

7.4.1. Módulo Verilog

Cuadro 18. Celda IND2D0

```
module IND2D0 (A1, B1, ZN);
    input A1, B1;
    output ZN;
        assign ZN = ~(B1 & ~A1);
endmodule
```

Nota. El cuadro muestra el código de la celda IND2D0 en Verilog.

7.4.2. Tabla de verdad

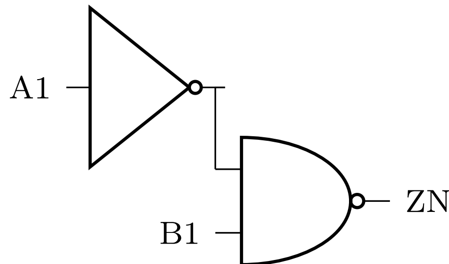
Cuadro 19. Tabla de verdad de la celda IND2D0

A1	**B1**	**ZN**
0	0	1
0	1	0
1	0	1
1	1	1

Nota. El cuadro muestra el comportamiento de la celda IND2D0.

7.4.3. Esquemático

Figura 4. Compuertas lógicas que conforman la celda IND2D0



Nota. La imagen muestra el esquemático de la celda IND2D0, que incluye un inversor y una compuerta NAND de dos entradas.

7.5. Celda IND3D0 - NAND con inversor en entrada A1

Extensión de IND2D0 con tres entradas, donde solo A1 es invertida.

7.5.1. Módulo Verilog

Cuadro 20. Celda IND3D0

```
module IND3D0 (A1, B1, B2, ZN);
    input A1, B1, B2;
    output ZN;
        assign ZN = ~(B1 & B2 & ~A1);
endmodule
```

Nota. El cuadro muestra el código de la celda IND3D0 en Verilog.

7.5.2. Tabla de verdad

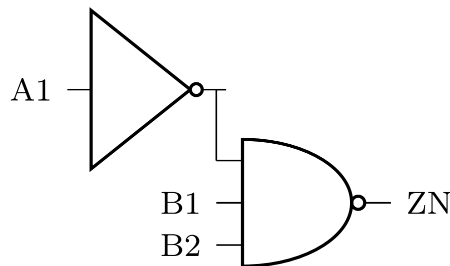
Cuadro 21. Tabla de verdad de la celda IND3D0

A1	**B1**	**B2**	**ZN**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Nota. El cuadro muestra el comportamiento de la celda IND3D0.

7.5.3. Esquemático

Figura 5. Compuertas lógicas que conforman la celda IND3D0



Nota. La imagen muestra el esquemático de la celda IND3D0, que incluye un inversor y una compuerta NAND de tres entradas.

7.6. Celda IND4D0 - NAND con inversor en entrada A1

Versión de 4 entradas donde A1 es invertida antes de la operación NAND.

7.6.1. Módulo Verilog

Cuadro 22. Celda IND4D0

```
module IND4D0 (A1, B1, B2, B3, ZN);
    input A1, B1, B2, B3;
    output ZN;
        assign ZN = ~(B1 & B2 & ~A1 & B3);
endmodule
```

Nota. El cuadro muestra el código de la celda IND4D0 en Verilog.

7.6.2. Tabla de verdad

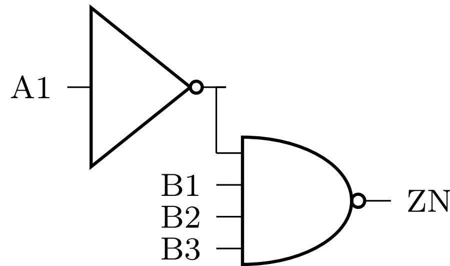
Cuadro 23. Tabla de verdad de la celda IND4D0

A1	**B1**	**B2**	**B3**	**ZN**
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Nota. El cuadro muestra el comportamiento de la celda IND4D0.

7.6.3. Esquemático

Figura 6. Compuertas lógicas que conforman la celda IND4D0



Nota. La imagen muestra el esquemático de la celda IND4D0, que incluye un inversor y una compuerta NAND de cuatro entradas.

7.7. Celda INR2D0 - NOR con inversor en entrada A1

Compuerta NOR de 2 entradas con la entrada A1 invertida.

7.7.1. Módulo Verilog

Cuadro 24. Celda INR2D0

```
module INR2D0 (A1, B1, ZN);
    input A1, B1;
    output ZN;
        assign ZN = ~(B1 | ~A1);
endmodule
```

Nota. El cuadro muestra el código de la celda INR2D0 en Verilog.

7.7.2. Tabla de verdad

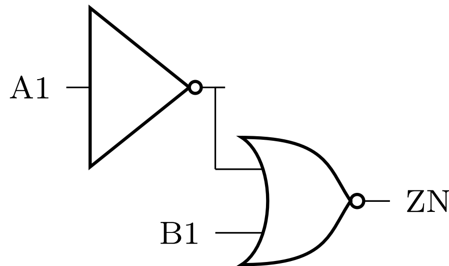
Cuadro 25. Tabla de verdad de la celda INR2D0

A1	B1	ZN
0	0	1
0	1	0
1	0	0
1	1	0

Nota. El cuadro muestra el comportamiento de la celda INR2D0.

7.7.3. Esquemático

Figura 7. Compuertas lógicas que conforman la celda INR2D0



Nota. La imagen muestra el esquemático de la celda INR2D0, que incluye un inversor y una compuerta NOR de dos entradas.

7.8. Celda INR3D0 - NOR con inversor en entrada A1

Versión de 3 entradas de INR2D0.

7.8.1. Módulo Verilog

Cuadro 26. Celda INR3D0

```
module INR3D0 (A1, B1, B2, ZN);
    input A1, B1, B2;
    output ZN;
        assign ZN = ~(B1 | B2 | ~A1);
endmodule
```

Nota. El cuadro muestra el código de la celda INR3D0 en Verilog.

7.8.2. Tabla de verdad

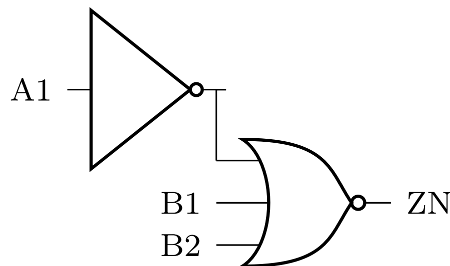
Cuadro 27. Tabla de verdad de la celda INR3D0

A1	**B1**	**B2**	**ZN**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda INR3D0.

7.8.3. Esquemático

Figura 8. Compuertas lógicas que conforman la celda INR3D0



Nota. La imagen muestra el esquemático de la celda INR3D0, que incluye un inversor y una compuerta NOR de tres entradas.

7.9. Celda INR4D0 - NOR con inversor en entrada A1

Compuerta NOR de 4 entradas con A1 invertida.

7.9.1. Módulo Verilog

Cuadro 28. Celda INR4D0

```
module INR4D0 (A1, B1, B2, B3, ZN);
    input A1, B1, B2, B3;
    output ZN;
        assign ZN = ~(B1 | B2 | ~A1 | B3);
endmodule
```

Nota. El cuadro muestra el código de la celda INR4D0 en Verilog.

7.9.2. Tabla de verdad

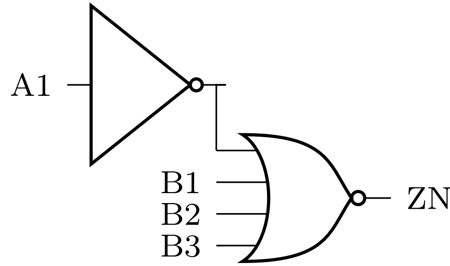
Cuadro 29. Tabla de verdad de la celda INR4D0

A1	B1	B2	B3	ZN
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda INR4D0.

7.9.3. Esquemático

Figura 9. Compuertas lógicas que conforman la celda INR4D0



Nota. La imagen muestra el esquemático de la celda INR4D0, que incluye un inversor y una compuerta NOR de cuatro entradas.

7.10. Celda INVD0 - Inversor

Compuerta inversora básica.

7.10.1. Módulo Verilog

Cuadro 30. Celda INVD0

```
module INVD0 (I, ZN);
    input I;
    output ZN;
        assign ZN = ~I;
endmodule
```

Nota. El cuadro muestra el código de la celda INVD0 en Verilog.

7.10.2. Tabla de verdad

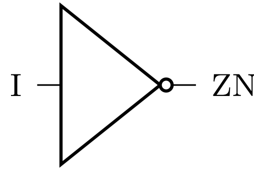
Cuadro 31. Tabla de verdad de la celda INVD0

I	**ZN**
0	1
1	0

Nota. El cuadro muestra el comportamiento de la celda INVD0.

7.10.3. Esquemático

Figura 10. Compuertas lógicas que conforman la celda INVD0



Nota. La imagen muestra el esquemático de la celda INVD0, que consiste en un inversor simple.

7.11. Celda INVD1 - Inversor

Compuerta inversora básica.

7.11.1. Módulo Verilog

Cuadro 32. Celda INVD1

```
module INVD1 (I, ZN);
    input I;
    output ZN;
        assign ZN = ~I;
endmodule
```

Nota. El cuadro muestra el código de la celda INVD1 en Verilog.

7.11.2. Tabla de verdad

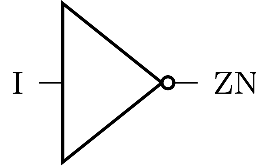
Cuadro 33. Tabla de verdad de la celda INVD1

I	ZN
0	1
1	0

Nota. El cuadro muestra el comportamiento de la celda INVD1.

7.11.3. Esquemático

Figura 11. Compuertas lógicas que conforman la celda INVD1



Nota. La imagen muestra el esquemático de la celda INVD1, que consiste en un inversor simple.

7.12. Celda IOA21D0 - AND-OR-INVERT con inversores en entradas A1 y A2

Implementa una función AND-OR-INVERT donde las entradas A1 y A2 son negadas antes de la operación OR, y el resultado se combina con B mediante NAND.

7.12.1. Módulo Verilog

Cuadro 34. Celda IOA21D0

```
module IOA21D0 (A1, A2, B, ZN);
    input A1, A2, B;
    output ZN;
        assign ZN = ~((~A1 | ~A2) & B);
endmodule
```

Nota. El cuadro muestra el código de la celda IOA21D0 en Verilog.

7.12.2. Tabla de verdad

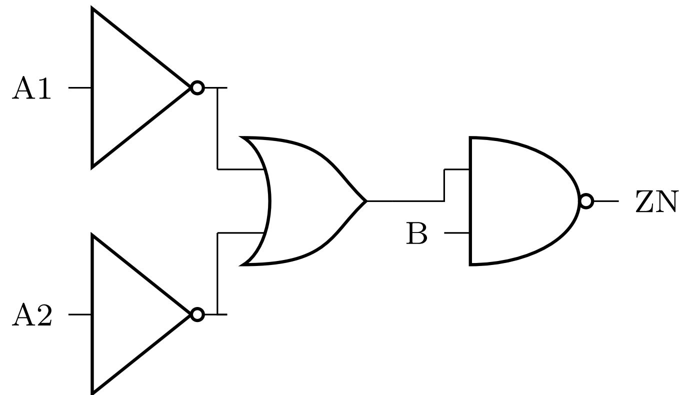
Cuadro 35. Tabla de verdad de la celda IOA21D0

A1	A2	B	ZN
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Nota. El cuadro muestra el comportamiento de la celda IOA21D0.

7.12.3. Esquemático

Figura 12. Compuertas lógicas que conforman la celda IOA21D0



Nota. La imagen muestra el esquemático de la celda IOA21D0, que incluye dos inversores, una compuerta OR y una compuerta NAND.

7.13. Celda MAOI22D0 - *Majority* AND-OR-INVERT

Celda que implementa una función "*majority*" (devuelve 1 cuando la mayoría de sus entradas son 1, y 0 cuando la mayoría son 0) con operaciones AND, y NOR.

7.13.1. Módulo Verilog

Cuadro 36. Celda MAOI22D0

```
module MOAI22D0 (A1, A2, B1, B2, ZN);
    input A1, A2, B1, B2;
    output ZN;
        assign ZN = ~((A1 | A2) & ~(B1 & B2));
endmodule
```

Nota. El cuadro muestra el código de la celda MAOI22D0 en Verilog.

7.13.2. Tabla de verdad

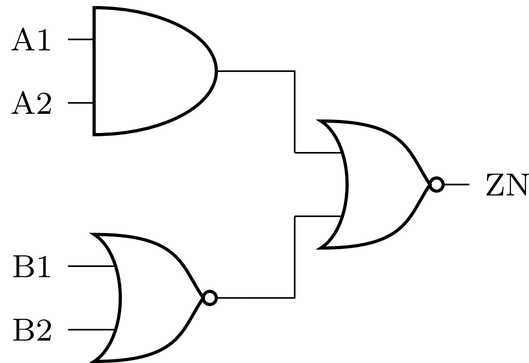
Cuadro 37. Tabla de verdad de la celda MAOI22D0

A1	**A2**	**B1**	**B2**	**ZN**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda MAOI22D0.

7.13.3. Esquemático

Figura 13. Compuertas lógicas que conforman la celda MAOI22D0



Nota. La imagen muestra el esquemático de la celda MAOI22D0, que implementa una función majority con compuertas AND y NOR.

7.14. Celda MOAI22D0 - *Majority* OR-AND-INVERT

Similar a MAOI22D0, pero con configuración OR-AND-INVERT.

7.14.1. Módulo Verilog

Cuadro 38. Celda MOAI22D0

```
module ND3D0 (A1, A2, A3, ZN);
    input A1, A2, A3;
    output ZN;
        assign ZN = ~(A1 & A2 & A3);
endmodule
```

Nota. El cuadro muestra el código de la celda MOAI22D0 en Verilog.

7.14.2. Tabla de verdad

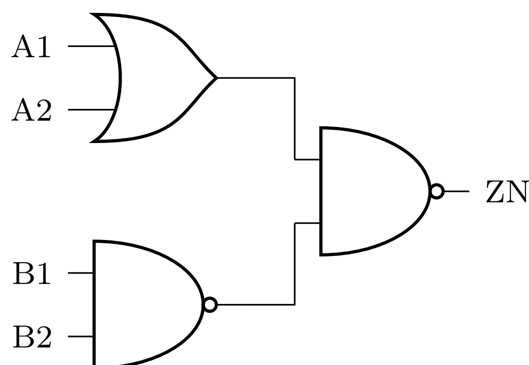
Cuadro 39. Tabla de verdad de la celda MOAI22D0

A1	A2	B1	B2	ZN
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda MOAI22D0.

7.14.3. Esquemático

Figura 14. Compuertas lógicas que conforman la celda MOAI22D0



Nota. La imagen muestra el esquemático de la celda MOAI22D0, que implementa una función majority con compuertas OR y NAND.

7.15. Celda ND3D0 - NAND de 3 entradas

Compuerta NAND estándar de 3 entradas.

7.15.1. Módulo Verilog

Cuadro 40. Celda ND3D0

```
module ND3D0 (A1, A2, A3, ZN);
    input A1, A2, A3;
    output ZN;
        assign ZN = ~(A1 & A2 & A3);
endmodule
```

Nota. El cuadro muestra el código de la celda ND3D0 en Verilog.

7.15.2. Tabla de verdad

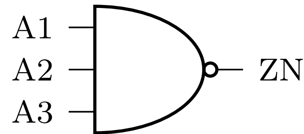
Cuadro 41. Tabla de verdad de la celda ND3D0

A1	A2	A3	ZN
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda ND3D0.

7.15.3. Esquemático

Figura 15. Compuertas lógicas que conforman la celda ND3D0



Nota. La imagen muestra el esquemático de la celda ND3D0, que consiste en una compuerta NAND de tres entradas.

7.16. Celda ND4D0 - NAND de 4 entradas

Compuerta NAND estándar de 4 entradas.

7.16.1. Módulo Verilog

Cuadro 42. Celda ND4D0

```
module ND4D0 (A1, A2, A3, A4, ZN);
    input A1, A2, A3, A4;
    output ZN;
        assign ZN = ~(A1 & A2 & A3 & A4);
endmodule
```

Nota. El cuadro muestra el código de la celda ND4D0 en Verilog.

7.16.2. Tabla de verdad

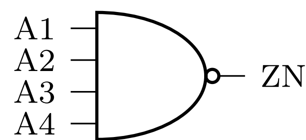
Cuadro 43. Tabla de verdad de la celda ND4D0

A1	A2	A3	A4	ZN
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda ND4D0.

7.16.3. Esquemático

Figura 16. Compuertas lógicas que conforman la celda ND4D0



Nota. La imagen muestra el esquemático de la celda ND4D0, que consiste en una compuerta NAND de cuatro entradas.

7.17. Celda NR2D0 - NOR de 2 entradas

Compuerta NOR estándar de 2 entradas.

7.17.1. Módulo Verilog

Cuadro 44. Celda NR2D0

```
module NR2D0 (A1, A2, ZN);
    input A1, A2;
    output ZN;
    assign ZN = ~(A1 | A2);
endmodule
```

Nota. El cuadro muestra el código de la celda NR2D0 en Verilog.

7.17.2. Tabla de verdad

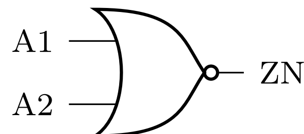
Cuadro 45. Tabla de verdad de la celda NR2D0

A1	**A2**	**ZN**
0	0	1
0	1	0
1	0	0
1	1	0

Nota. El cuadro muestra el comportamiento de la celda NR2D0.

7.17.3. Esquemático

Figura 17. Compuertas lógicas que conforman la celda NR2D0



Nota. La imagen muestra el esquemático de la celda NR2D0, que consiste en una compuerta NOR de dos entradas.

7.18. Celda NR2XD0 - NOR de 2 entradas

Compuerta NOR estándar de 2 entradas.

7.18.1. Módulo Verilog

Cuadro 46. Celda NR2XD0

```
module NR2XD0 (A1, A2, ZN);
    input A1, A2;
    output ZN;
    assign ZN = ~(A1 | A2);
endmodule
```

Nota. El cuadro muestra el código de la celda NR2XD0 en Verilog.

7.18.2. Tabla de verdad

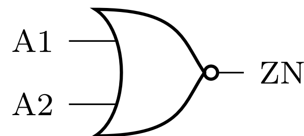
Cuadro 47. Tabla de verdad de la celda NR2XD0

A1	**A2**	**ZN**
0	0	1
0	1	0
1	0	0
1	1	0

Nota. El cuadro muestra el comportamiento de la celda NR2XD0.

7.18.3. Esquemático

Figura 18. Compuertas lógicas que conforman la celda NR2XD0



Nota. La imagen muestra el esquemático de la celda NR2XD0, que consiste en una compuerta NOR de dos entradas.

7.19. Celda NR3D0 - NOR de 3 entradas

Compuerta NOR estándar de 3 entradas.

7.19.1. Módulo Verilog

Cuadro 48. Celda NR3D0

```
module NR3D0 (A1, A2, A3, ZN);
    input A1, A2, A3;
    output ZN;
    assign ZN = ~(A1 | A2 | A3);
endmodule
```

Nota. El cuadro muestra el código de la celda NR3D0 en Verilog.

7.19.2. Tabla de verdad

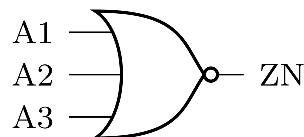
Cuadro 49. Tabla de verdad de la celda NR3D0

A1	A2	A3	ZN
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda NR3D0.

7.19.3. Esquemático

Figura 19. Compuertas lógicas que conforman la celda NR3D0



Nota. La imagen muestra el esquemático de la celda NR3D0, que consiste en una compuerta NOR de tres entradas.

7.20. Celda NR4D0 - NOR de 4 entradas

Compuerta NOR estándar de 4 entradas.

7.20.1. Módulo Verilog

Cuadro 50. Celda NR4D0

```
module NR4D0 (A1, A2, A3, A4, ZN);
    input A1, A2, A3, A4;
    output ZN;
    assign ZN = ~(A1 | A2 | A3 | A4);
endmodule
```

Nota. El cuadro muestra el código de la celda NR4D0 en Verilog.

7.20.2. Tabla de verdad

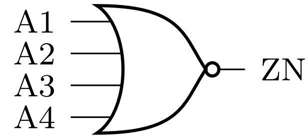
Cuadro 51. Tabla de verdad de la celda NR4D0

A1	A2	A3	A4	ZN
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Nota. El cuadro muestra el comportamiento de la celda NR4D0.

7.20.3. Esquemático

Figura 20. Compuertas lógicas que conforman la celda NR4D0



Nota. La imagen muestra el esquemático de la celda NR4D0, que consiste en una compuerta NOR de cuatro entradas.

CAPÍTULO 8

Flujo de trabajo

El proceso de verificación *Layout Versus Schematic* (LVS) es una de las etapas críticas dentro del flujo de diseño físico de circuitos integrados, ya que asegura que el diseño en silicio (*layout*) sea una representación funcionalmente equivalente a la descrita por el esquemático (*netlist*). Por otro lado, la verificación *Electrical Rule Check* (ERC) garantiza que el diseño del circuito cumpla con las reglas eléctricas establecidas para evitar problemas de funcionamiento y daños en el dispositivo durante su operación.

El flujo que se describe a continuación detalla los pasos anteriores necesarios para llevar a cabo los procesos de verificación, además de cómo preparar, ejecutar e interpretar las verificaciones LVS y ERC utilizando herramientas del ecosistema Synopsys y formatos estándar de la industria.

8.1. Generación del archivo RTL

Primeramente, se debe generar el archivo RTL (*Register Transfer Level*) que describe el comportamiento del circuito, el cual define el comportamiento del circuito en un lenguaje de descripción de *hardware* (HDL), en este caso, Verilog. Este archivo es esencial para la creación del *netlist* que se utilizará en las verificaciones LVS y ERC.

8.1.1. Proceso

1. Se redacta el código Verilog utilizando una estructura ya sea *behavioral* o *structural*.
2. Se definen los módulos, entradas, salidas, cables y las demás instancias que describen el circuito.

3. Este archivo es el punto de partida para la síntesis lógica, generando así un *netlist*.

8.2. Generación del *netlist* a nivel de celdas estádar

Una vez que se tiene el archivo RTL, se transforma el código Verilog en un *netlist* a nivel de celdas estándar, utilizando celdas provenientes de las librerías *Standard Cell Library* y *I/O Cell Library* de TSMC.

El resultado de la transformación, realizada por medio de la síntesis física, es un *netlist* en Verilog estructural que contiene las instancias de las celdas y sus respectivas conexiones.

Se debe tener en cuenta que debido a que el *netlist* generado aún no puede ser utilizado para las pruebas, ya que para esto se requiere un *netlist* que represente las dimensiones físicas de los componentes y geomertría descriptiva del circuito integrado.

8.3. Archivo GDSII

Para llevar a cabo las verificaciones, es necesario contar con un diseño físico del circuito integrado, el cual se representa en un archivo `.gds`. Este contiene el diseño del circuito integrado e información del *layout* del circuito a nivel de máscaras fotolitográficas, que describe capas y geometrías necesarias para la fabricación en silicio.

8.3.1. Proceso

1. Se realiza el procedimiento de *routing* y *placement* del diseño, en este caso mediante un *script* por medio de herramientas de Synopsys.
2. Se genera un archivo `.gds` que contiene la representación gráfica del circuito integrado en formato GDSII, el cual es el estándar de la industria para la representación de diseños físicos.

Este archivo es la fuente desde la cual IC Validator extraerá el netlist físico para realizar la comparación LVS y la revisión ERC.

8.4. Obtención de *headers* y conversión a formato (CDL)

Los *headers* son un archivo Verilog que contiente la declaración mínima de cada módulo utilizado en un circuito, en el que se define solo la interfaz (nombre y puertos) de dichos módulos. Estos sirven para describir la interconexión en el diseño usando celdas estándar sin necesidad de incluir detalles internos de cada módulo. Este archivo se obtiene a partir de las librerías de celdas estándar proporcionadas por el fabricante del proceso de fabricación, que en este caso, son las librerías *Standard Cell Library* y *I/O Cell Library* de TSMC.

La traducción de los *headers* en formato Verilog a formato CDL (*Circuit Description Language*) da paso a que este archivo sea legible por herramientas de verificación física como IC Validator de Synopsys.

8.4.1. Entradas necesarias

- *Headers* de la librería de celdas estándar utilizadas en el diseño (extraídas de la *Standard Cell Library* y *I/O Cell Library*).

Con esta conversión, se genera un archivo en formato `.sp` (CDL), el cual es utilizado posteriormente en el flujo de trabajo.

8.5. Archivo `.icv`

Utilizando el *netlist* estructural en Verilog generado previamente y los *headers* convertidos a formato CDL, se procede a generar un archivo `.icv` que contiene el *netlist* equivalente a un CDL, pero con formato y convenciones adaptadas a IC Validator.

8.5.1. Entradas necesarias

- *Netlist* estructural en Verilog del circuito.
- *Headers* en formato CDL de las librerías de celdas estándar.

8.6. Ejecución de la prueba LVS

Una vez que se cuenta con el *netlist* a nivel de transistor y el archivo `.gds`, se procede a ejecutar la verificación LVS utilizando IC Validator.

8.6.1. Entradas necesarias

- *Layout* del circuito
- Archivo GDSII
- Archivo CDL en formato `.icv` (netlist a nivel de transistor)
- *Runset* de LVS/ERC para IC Validator

8.6.2. Procedimiento

1. Configurar el entorno: se modifica el *runset* en base a las especificaciones del diseño.
2. Ejecución del comando LVS: se comparan jerárquicamente entre sí las celdas del *layout* y del esquemático.
3. Generación de resultados: evaluación *logs*, archivos de resumen, errores, etc.

8.6.3. Interpretación de resultados

Analizar los archivos generados para determinar si el diseño pasa la verificación LVS y corregir errores si es necesario. Si se detectan errores, se debe volver al *layout* o al netlist para ajustar las conexiones, la jerarquía o las definiciones. En el caso de que el diseño pase la verificación LVS, se procede a la siguiente etapa del flujo de diseño, la verificación eléctrica (ERC).

8.7. Ejecución de la prueba ERC

Después de que el diseño haya pasado la verificación LVS, se procede a ejecutar la verificación ERC utilizando IC Validator. Esta prueba utiliza los mismos archivos de entrada que la prueba LVS, pero haciendo pequeñas modificaciones en el *runset* para que este verifique las reglas eléctricas del diseño.

8.7.1. Entradas necesarias

- Archivo GDSII
- Archivo CDL en formato `.icv`
- *Runset* de LVS/ERC para IC Validator

8.7.2. Procedimiento

1. Configurar el entorno: se modifica el *runset* para deshabilitar las funciones de LVS y habilitar las funciones de ERC.
2. Ejecución del comando ERC: se verifica la integridad eléctrica del diseño del circuito.
3. Generación de resultados: evaluación *logs*, archivos de resumen, errores, etc.

8.7.3. Interpretación de resultados

Analizar los archivos generados para determinar si el diseño pasa la verificación ERC y corregir errores si es necesario. Si se detectan errores, es necesario volver al *layout* para ajustar las conexiones, las dimensiones o las definiciones. Media vez el diseño pase la verificación

ERC, este estará listo para ser enviado a TSMC para que realilcen verifiaciones adicionales y posteriormente procedan a la fabricación del circuito integrado.

CAPÍTULO 9

Pasos previos a realizar las pruebas

Al completar la etapa de síntesis lógica del diseño, se obtiene un archivo `.v` que contiene el *netlist* del esquemático del circuito. Sin embargo, este archivo no puede ser utilizado directamente para las pruebas de verificación física, ya que se requiere un *netlist* en formato CDL *(Circuit Description Language)* que represente las conexiones a nivel de transistor del circuito.

9.1. Extracción de *headers* de las librerías de celdas

Para convertir el *netlist* a nivel de celdas estándar a un *netlist* a nivel de transistor, es necesario contar con los *headers* de las librerías de celdas estándar utilizadas en el diseño. Estos *headers* proporcionan la información necesaria sobre las celdas lógicas, incluyendo sus pines y conexiones internas.

Como primer paso, se deben localizar los archivos de las librerías de celdas estándar utilizadas en el diseño del circuito integrado, en este caso, la *Standard Cell Library* (`tcbn65lp.v`) y la *I/O Cell Library* (`tpdn65lpnv2od3.v`), los cuales son proporcionados por IMEC, creados por TSMC.

La ubicación del archivo de la librería *Standard Cell Library* es:

```
TSMC > 65 > CMOS > LP > stclib > 9-track > tcbn65lp-set > tcbn65lp_220a_FE >
tcbn65lp_200a_vlg > TSMCHOME > digital > Front_End > verilog > tcbn65lp_200a
> tcbn65lp.v
```

La ubicación del archivo de la librería *I/O Cell Library* es:

TSMC > 65 > CMOS > LP > stclib > 9-track > tcbn65lp-set > tcbn65lp_220a_FE > tcbn65lp_200a_vlg > TSMCHOME > digital > Front_End > verilog > tpdn65lpnv2od3_140b > tpdn65lpnv2od3.v

Debido a que se le harán modificaciones a los archivos originales, es recomendable crear una copia de los archivos en una ubicación separada para evitar alterar los archivos originales. Para este caso, la copia de la librería *Standard Cell Library* será nombrada como `std_cell_library.v`, y la copia de la librería *I/O Cell Library* será nombrada como `std_io_library.v`.

Luego, se procede a extraer los *headers* de las celdas que se utilizan en el diseño del circuito. Esto se logra conservarndo únicamente la información estructural de los puertos, es decir, las declaraciones `input`, `output` e `inout` de cada celda, y eliminando todo lo demás que no sea necesario para describir los puertos de cada celda. Debido a que dichas librerías en conjunto tienen una longitud de más de 80,000 líneas, se desarrolló un *script* en Python que automatiza este proceso de extracción.

9.1.1. *Script* `extraer_headers.py`

El *script* `extraer_headers.py` toma como entrada un archivo `.v` que contiene la definición de múltiples módulos de Verilog correspondientes a las celdas estándar, y genera un nuevo archivo `.v` que contiene únicamente las declaraciones de los puertos de cada celda, eliminando cualquier lógica interna o directivas del preprocesador de Verilog.

Cuadro 52. *Script* `extraer_headers.py` para extraer los *headers* de las librerías de celdas estándar

```
import re
import os

# Solicita el nombre del archivo
filename = input("Ingresa el archivo .v al cual se le desean
                 extraer los headers: ").strip()

# Lee el contenido
with open(filename, "r") as f:
    text = f.read()

# Se eliminan completamente los bloques "primitive"
text = re.sub(r"primitive\b.*?endprimitive", "", text,
              flags=re.DOTALL)

# Se eliminan las lineas fuera de modulos que inicien con "`"
cleaned_lines = []
inside_module = False
for line in text.splitlines():
    # Detectar inicio o fin de modulo
    if re.match(r"^\s*module\b", line):
        inside_module = True
    elif re.match(r"^\s*endmodule\b", line):
        inside_module = False

    # Si esta fuera de modulo y empieza con "`", se omite
    if not inside_module and re.match(r"^\s*`", line):
        continue

    cleaned_lines.append(line)

text = "\n".join(cleaned_lines)

# Dentro de cada modulo, se conserva solo "input/output/inout"
pattern = re.compile(r"(module\s+\w+\s*\(.*?\);\s*)(.*?)
                     (endmodule)", re.DOTALL)

def clean_module(match):
    header = match.group(1)
    body = match.group(2)
    footer = match.group(3)

...
```

```
...

    # Solo conserva input/output/inout
    lines = []
    for line in body.splitlines():
        if re.match(r"^\s*(input|output|inout)\b", line):
            lines.append(line)
    return header + "\n".join(lines) + "\n" + footer

cleaned_text = re.sub(pattern, clean_module, text)

# Guardar con nuevo nombre
base, ext = os.path.splitext(filename)
new_filename = f"{base}_headers{ext}"

with open(new_filename, "w") as f:
    f.write(cleaned_text.strip() + "\n")

print(f"Los headers de '{filename}' se encuentran en
        '{new_filename}'.")
```

Nota. El cuadro contiene el código del *script* `extraer_headers.py`. en Python.

El *script* realiza tres operaciones principales:

Eliminación de bloques `primitive`

Estos bloques describen primitivas lógicas a nivel transistor o comportamental (como *buffers* o *flip-flops* básicos), pero no forman parte de la jerarquía principal de celdas que definen las interfaces que nos interesan.

Eliminación directivas del preprocesador de Verilog

Estas directivas son útiles para herramientas de simulación o síntesis, pero no aportan información funcional o estructural sobre las entradas/salidas de los módulos, por lo que se descartan.

En este caso, se eliminan las siguientes directivas:

- ‘timescale 1ns / 1ps
- ‘celldefine
- ‘endcelldefine

Depuración del contenido de las celdas

Dentro de cada una de las celdas de la librería se conserva sólamente las líneas que declaran puertos (`input`, `output` e `inout`), eliminando toda lógica interna, especificaciones de retardo, bloques `specify`, entre otros. De esta forma, cada celda queda reducida a una plantilla mínima que describe solo su interfaz.

9.1.2. Obtención de los *headers*

Al ejecutar el *script* `extraer_headers.py` para los archivos `std_cell_library.v` y `std_io_library.v`, se generan dos nuevos archivos: `std_cell_library_headers.v` y `std_io_library_headers.v`, respectivamente. Estos archivos contienen únicamente las declaraciones de los puertos de cada celda, y serán utilizados en el siguiente paso para generar el *netlist* en formato CDL.

Celda de ejemplo

A continuación, se muestra un ejemplo de cómo era una celda originalmente, y cómo luce la celda después de aplicar el *script* de extracción de *headers*.

Cuadro 53. Celda IIND4D0 original

```
`celldefine
module IIND4D0 (A1, A2, B1, B2, ZN);
    input A1, A2, B1, B2;
    output ZN;
    not             (A1N, A1);
    not         (A2N, A2);
    nand                (ZN, A1N, A2N, B1, B2);

  specify
    (A1 => ZN) = (0, 0);
    (A2 => ZN) = (0, 0);
    (B1 => ZN) = (0, 0);
    (B2 => ZN) = (0, 0);
  endspecify
endmodule
`endcelldefine
```

Nota. El cuadro contiene el código original de la celda IIND4D0 en Verilog.

Cuadro 54. Celda IIND4D0 después de aplicar el *script* de extracción de *headers*

```
module IIND4D0 (A1, A2, B1, B2, ZN);
    input A1, A2, B1, B2;
    output ZN;
endmodule
```

Nota. El cuadro contiene el código del *header* de la celda IIND4D0 en Verilog.

9.2. Concatenación de los *headers*

Los archivos `std_cell_library_headers.v` y `std_io_library_headers.v` generados en el paso anterior deben ser concatenados en un solo archivo, ya que la herramienta de conversión utilizada para generar el *netlist* en formato CDL requiere que todos los *headers* estén en un único archivo.

Para concatenar los archivos, se utiliza el siguiente comando en una terminal abierta en la carpeta donde se encuentran los archivos generados anteriormente:

```
cat std_cell_library_headers.v std_io_library_headers.v > headers.v
```

Este comando genera otro archivo en formato Verilog que tiene como nombre `headers.v`.

9.3. Archivo SPICE (CDL)

Con el archivo `headers.v` que contiene los *headers* de las librerías de celdas estándar, se procede a traducir el *netlist* a nivel de compuerta en Verilog al formato CDL utilizando la herramienta `NetTran` de Synopsys.

Para llevar a cabo la traducción, se utiliza el siguiente comando en una terminal abierta en la carpeta donde se encuentra el archivo `headers.v`:

```
icv_nettran -verilog headers.v -outName headers.sp -outType SPICE
```

Donde cada parámetro indica lo siguiente:

- `icv_nettran`: corresponde al ejecutable principal de NetTran como herramienta proveniente de IC Validator.
- `-verliog`: indica las *netlist* de entrada en formato Verilog.
- `-outName`: define el nombre del archivo de salida, que en este caso es `headers.sp`.
- `-outType`: especifica el formato de la *netlist* de salida deseado, en este caso, SPICE (CDL).

Una vez ejecutado el comando, se genera el archivo `headers.sp` que contiene el *netlist* en formato CDL.

Celda de ejemplo

A continuación, se muestra un ejemplo de cómo era una celda en Verilog, y cómo la herramienta NetTran lo traduce al formato SPICE (CDL):

Cuadro 55. Celda IIND4D0 en Verilog

```
module IIND4D0 (A1, A2, B1, B2, ZN);
    input A1, A2, B1, B2;
    output ZN;
endmodule
```

Nota. El cuadro contiene el código de la celda IIND4D0 antes de la traducción.

Cuadro 56. Celda IIND4D0 en formato SPICE (CDL)

```
.SUBCKT IIND4D0 A1 A2 B1 B2 VDD VSS ZN
.ENDS
```

Nota. El cuadro contiene el código de la celda IIND4D0 después de la traducción.

9.4. Archivo ICV

Finalmente, se debe generar el archivo `.icv` que contiene el *netlist* estructural en Verilog en un formato legible por IC Validator, pero antes de realizar la conversión, es necesario realizar algunas modificaciones al archivo Verilog. Este *netlist*, generado en la etapa de síntesis física, además de contener toda las celdas que conforman la lógica del circuito, contiene celdas de componentes puramente físicos, los cuales no son reconocidos por IC Validator. Por lo tanto, se deben eliminar todas las instancias de los siguientes componentes del archivo Verilog:

- `PCORNER`: definen y cierran el perímetro del área digital, manteniendo continuidad estructural.
- `FILL`: rellenan espacios para mantener continuidad eléctrica de potencia y cumplir reglas de espaciamiento.
- `PFILLER`: rellenan huecos en filas de celdas para mantener continuidad de pozos y difusiones.

Para facilitar este proceso, se desarrolló un *script* en Python que automatiza la eliminación de estas instancias del archivo Verilog.

9.4.1. *Script* `limpiar_netlist.py`

Al ejecutar el *script* `limpiar_netlist.py` se toma como entrada el *netlist* estructural en Verilog, y genera un nuevo archivo `.v` que no contiene las instancias de los componentes no reconocidos por IC Validator.

Cuadro 57. *Script* `limpiar_netlist.py` para eliminar instancias no reconocidas por IC Validator

```
import re
import os

input_file = input("Ingresa el nombre del archivo .v
                   a limpiar: ").strip()

if not os.path.isfile(input_file):
    print(f"ERROR: El archivo '{input_file}' no existe.")
    exit(1)

# Archivo de salida
output_file = input_file.replace(".v", "_limpio.v")

with open(input_file, "r") as f:
    lines = f.readlines()

cleaned = []
skip_count = 0

for line in lines:
    s = line.strip()

    # Eliminar instancias fisicas sin significado electrico
    if re.match(r'^\s*PFILLER\w*\s', s):
        skip_count += 1
        continue
    if re.match(r'^\s*PCORNER\w*\s', s):
        skip_count += 1
        continue
    if re.match(r'^\s*FILL\w*\s', s):
        skip_count += 1
        continue

    cleaned.append(line)

with open(output_file, "w") as f:
    f.writelines(cleaned)

print("\n Proceso completado.")
print(f"Se eliminaron {skip_count} instancias
        PFILLER/PCORNER/FILL.")
print(f"Archivo limpio generado: {output_file}\n")
```

Nota. El cuadro contiene el código del *script* `limpiar_netlist.py` en Python.

El *script* realiza una única operación principal en varios pasos:

1. Solicita al usuario el nombre del archivo Verilog que se desea limpiar.
2. Lee el contenido del archivo línea por línea.
3. Para cada línea, verifica si corresponde a una instancia de `PFILLER`, `PCORNER` o `FILL` utilizando expresiones regulares.
4. Si la línea corresponde a alguna de estas instancias, se omite y se incrementa un contador de instancias eliminadas.
5. Si la línea no corresponde a estas instancias, se agrega a una lista de líneas limpias.
6. Finalmente, escribe las líneas limpias en un nuevo archivo Verilog con el sufijo `_limpio.v`.

9.4.2. Conversión a `.icv`

Para realizar la traducción, se utiliza nuevamente la herramienta `NetTran` de Synopsys con el siguiente comando en una terminal abierta en la carpeta donde se encuentra el *netlist* estructural en Verilog sin las instancias de componentes no reconocidos por IC Validator, utilizando el archivo `headers.sp` como referencia para el mapeo de los modelos de circuitos:

```
icv_nettran -verilog netlist.v -sp headers.sp -outName netlist.icv

-outType ICV
```

Una vez ejecutado el comando, se genera el *netlist* en formato ICV.

Módulo de ejemplo

A continuación, se muestra un ejemplo de cómo era un módulo en Verilog, y cómo la herramienta NetTran lo traduce al formato ICV:

Cuadro 58. Celda IIND4D0 en Verilog estructural

```
IIND4D0 U1010 ( .A1 ( n491 ) ,
                .A2 ( n429 ) ,
                .B1 ( n496 ) ,
                .B2 ( n575 ) ,
                .ZN ( n523 ) ,
                .VDD ( VDD ) ,
                .VSS ( VSS ) ) ;
```

Nota. El cuadro contiene el código de una instancia de la celda IIND4D0 en Verilog estructural.

Cuadro 59. Celda IIND4D0 en formato SPICE

```
.SUBCKT IIND4D0 A1 A2 B1 B2 VDD VSS ZN
.ENDS
```

Nota. El cuadro contiene el código de la celda IIND4D0 en formato SPICE.

Cuadro 60. Celda IIND4D0 en formato ICV

```
{cell IIND4D0
{port A1 A2 B1 B2 ZN VDD VSS}
}
```

Nota. El cuadro contiene el código de la celda IIND4D0 en formato ICV.

CAPÍTULO 10

Prueba *Layout vs. Schematic* (LVS)

El proceso de verificación Layout Versus Schematic (LVS) es una de las etapas críticas dentro del flujo de diseño físico de circuitos integrados, ya que asegura que el diseño en silicio (*layout*) sea una representación funcionalmente equivalente a la descrita por el esquemático (*netlist*).

10.1. Selección del *runset*

El *runset* es un conjunto de configuraciones y parámetros que se utilizan para ejecutar la verificación LVS. En este caso, se utilizará un *runset* brindado por IMEC, y creado TSMC, el cual contiene las configuraciones necesarias para realizar la verificación LVS acorde a las especificaciones de la tecnología de 65 nm.

El *runset* escogido para esta verificación se encuentra en la siguiente ubicación:

```
TSMC > 65 > CMOS > util > T-N65-CL-LS-001-J1_1_6A >
DFM_LVS_RC_ICV_N65RF_1P9M_ALRDL_v16a_all >
DFM_LVS_RC_ICV_N65RF_1P9M_ALRDL_v16a > MAIN_DECK > nonUTM >
DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m
```

10.2. Configuración LVS

Para realizar la comparación entre el *layout* y el esquemático, se empleará la herramienta IC Validator de Synopsys, la cual es ampliamente utilizada en la industria para llevar a cabo verificaciones físicas de circuitos integrados. Como se mencionó en el marco teórico, TSMC

no nos brinda acceso al *back-end* de las celdas que se utilizarán en el circuito, por lo que se ejecutará un *Black Box* LVS.

A diferencia de una prueba de LVS común, un *Black Box* LVS permite la verificación parcial de un diseño cuando las celdas utilizadas aún están en desarrollo o no se tiene acceso al contenido dentro de ellas, garantizando que las interconexiones entre las celdas sean correctas. En lugar de verificar que las conexiones y circuitería interna de cada una de las celdas sea íntegra y sin discrepancias, IC Validator tratará a dichas celdas como cajas negras, comparando los puertos del esquemático y del *layout* por nombre e informará si se encuentran o no errores en la interconexión de las celdas.

10.2.1. Configuración del *runset*

Debido a que el *runset* creado por TSMC no se encuentran definidas las celdas de las librerías *Standard Cell Library* y *I/O Cell Library*, es necesario definirlas manualmente en la sección del *runset* llamada *Environment Setup*, como se muestra en el siguiente cuadro:

Cuadro 61. *Environment Setup* del *runset* sin ninguna modificación

```
//////////////////////////
// ENVIRONMENT SETUP //
//////////////////////////

library(
        library_name = "lvs_top.gds",
        cell = "lvs_top",
        format = GDSII
);

SCHEMATIC_TOPCELL : string = "lvs_top";
sch_db = schematic(
        schematic_file = {{"lvs_top.cdl", SPICE}},
        schematic_library_file = {{"unit.cdl", SPICE}},
        expand_multiple_devices = true
);
```

Nota. El cuadro contiene el código original del *Environment Setup* del *runset*.

En esta sección, se deben establecer los argumentos de varios parámetros, empezando por los de la función `library()`:

- `library_name`: define la ruta y el nombre del archivo (en este caso) de formato GDSII, por lo que se coloca la ruta del archivo `.gds`: Generado en la síntesis física del circuito. Contiene las formas geométricas por capa necesarias para la fabricación, y es el archivo que se utiliza para fabricar el circuito.

- `cell`: indica la celda que se va a comprobar, la cual debe estar en la biblioteca previamente especificada.
- `format`: especifica el formato del archivo de entrada. Para este caso, debido que el archivo de entrada es `.gds`, se establece que el formato es GDSII.

Los parámetros de `schematic()`, que en este caso es una función que se guarda en una variable llamada `sch_db`, también deben ser decretados:

- `schematic_file`: enlista los archivos de esquemáticos, incluyendo la ruta opcional, por lo que se coloca la ruta del archivo que contiene la *netlist* del chip generado el capítulo anterior, y se indica que está en formato ICV.
- `schematic_library_file`: enlista los archivos de esquemáticos, incluyendo la ruta opcional, y los ajustes de formato que no se modifican con las opciones de línea de comandos -s y -sf. En este escenario, se coloca la ruta del archivo `unit.cdl`, el cual es dado por IMEC y creado por TSMC, y se indica que está en formato SPICE.
- `expand_multiple_devices`: indica si los dispositivos se incluyen varias veces en el *netlist* del chip. Para este caso, se coloca como `true`.
- `spice_settings`: especifica la configuración relacionada con las traducciones SPICE. Originalmente, este argumento no se encontraba en la función, pero se agrega para indicar que durante la traducción SPICE, los símbolos *slash* no se deben traducir como espacios.

El archivo `unit.cdl` puede ser encontrado en la siguiente ubicación:

```
TSMC > 65 > CMOS > util > T-N65-CL-LS-001-J1_1_6A >
DFM_LVS_RC_ICV_N65RF_1P9M_ALRDL_v16a_all >
DFM_LVS_RC_ICV_N65RF_1P9M_ALRDL_v16a > unit.cdl
```

También se modifica el valor de la variable `SCHEMATIC_TOPCELL`, la cual es de tipo `string` y establece el nombre de la *top cell* del esquemático.

Para este escenario, el *Environment Setup* luce de la siguiente manera después de las modificaciones:

Cuadro 62. *Environment Setup* del *runset* modificado

```
///////////////////////
// ENVIRONMENT SETUP //
///////////////////////

library(
        library_name = "/home/nanoelectronica/Desktop
                        /Folder_de_Trabajo/65/ICV
                        /GRAN_JAGUAR/LVS/IO_nanochip.gds",
        cell = "IO_nanochip",
        format = GDSII
);

SCHEMATIC_TOPCELL : string = "IO_nanochip";
sch_db = schematic(
        schematic_file = {{"/home/nanoelectronica/Desktop
                              /Folder_de_Trabajo/65/ICV
                              /GRAN_JAGUAR
                              /IO_nanochip.icv", ICV}},
        schematic_library_file = {{"/home/nanoelectronica
                                      /Desktop
                                      /Folder_de_Trabajo/65
                                      /ICV/GRAN_JAGUAR/LVS
                                      /unit.cdl", SPICE}},
        expand_multiple_devices = true
);
```

Nota. El cuadro contiene el código modificado del *Environment Setup* del *runset*.

Al final del *runset*, en la sección llamada *Compare Settings*, se define cómo se comparan las bases de datos del *layout* y del esquemático, qué criterios de equivalencia se usan, y cómo se genera el *netlist* de salida en formato SPICE.

En esta sección, se debe hacer una modificación al parámetro `user_functions_file` de la función `compare()`, más específicamente en la ruta del archivo `icv_compare.rh`.

Cuadro 63. Función `compare()` sin ninguna modificación

```
compare(
#endif
        state = cmp_matrix,
        schematic = sch_db,
        layout = lay_db,
        user_functions_file = "DFM/icv_compare.rh",
        case_sensitive = {},
        push_down_pins = true,
        #ifdef RC_DECK
        memory_array_compare = false,
        #else
        memory_array_compare = true,
        #endif
        schematic_top_cell = SCHEMATIC_TOPCELL,
        action_on_error = EXPLODE,
        delete_layout_cells = {"pnwdio"},
        print_messages = {
                all_merged_device_list = true,
                filtered_device_list = true,
                pre_merge_stats = true
        }
);
```

Nota. El cuadro contiene el código original de la función `compare()` del *runset*.

Este archivo es proporcionado por IMEC y creado por TSMC, y es un *script* auxiliar escrito en el lenguaje de reglas de IC Validator (*ICV Rule Hierarchy*), que tiene como propósito principal definir funciones personalizadas que complementan el proceso de comparación LVS, especialmente para el manejo de dispositivos MOSFET combinados o repetidos dentro del *layout*.

El archivo puede ser encontrado en la siguiente ubicación:

`TSMC 65 > CMOS > util > T-N65-CL-LS-001-J1_1_6A > DFM_LVS_RC_ICV_N65RF_1P9M_ALRDL_v16a_all > DFM_LVS_RC_ICV_N65RF_1P9M_ALRDL_v16a > MAIN_DECK > nonUTM > DFM > icv_compare.rh`

El contenido de este archivo se discutirá más a fondo más adelante, pero por ahora se debe modificar la ruta del archivo para que apunte a la ubicación correcta del archivo `icv_compare.rh`.

Cuadro 64. Función `compare()` modificada

```
compare(
#endif
        state = cmp_matrix,
        schematic = sch_db,
        layout = lay_db,
        user_functions_file = "/home/nanoelectronica/Desktop
                              /Folder_de_Trabajo/65/ICV
                              /GRAN_JAGUAR/LVS
                              /icv_compare.rh",
        case_sensitive = {},
        push_down_pins = true,
        #ifdef RC_DECK
        memory_array_compare = false,
        #else
        memory_array_compare = true,
        #endif
        schematic_top_cell = SCHEMATIC_TOPCELL,
        action_on_error = EXPLODE,
        delete_layout_cells = {"pnwdio"},
        print_messages = {
                all_merged_device_list = true,
                filtered_device_list = true,
                pre_merge_stats = true
        }
);
```

Nota. El cuadro contiene el código modificado de la función `compare()` del *runset*.

10.3. Configuración del flujo de las cajas negras

Con la función `lvs_black_box_options()`, se configura el entorno y se especifican las celdas que se tratarán como cajas negras y cómo deben coincidir sus puertos. Al insertar la función en el *runset* de LVS, se configura el entorno de manera que IC Validator realice la prueba LVS considerando las celdas como cajas negras y no celdas a las que se les deberían verificar sus interior.

En este caso, se utilizará la función `lvs_black_box_options()` de la siguiente manera:

Cuadro 65. Configuración de LVS para cajas negras

```
lvs_black_box_options(
    equiv_cells = {{schematic_cell = "Celda_1",
                    layout_cell = "Celda_1"},
                   {schematic_cell = "Celda_2",
                    layout_cell = "Celda_2"},...}
    remove_schematic_ports = {"Puerto_1",
                              "Puerto_2",...},
);
```

Nota. El cuadro contiene un ejemplo del código de la función `lvs_black_box_options()`.

Para este caso, se deben definir los parámetros de la función `lvs_black_box_options()` de la siguiente manera:

- `equiv_cells`: se establecen las equivalencias para cada celda, colocando el nombre que tiene la celda en el esquemático en `schematic_cell`, y el nombre que tiene la celda en el *layout* en `layout_cell`.
- `remove_schematic_ports`: se enlistan los puertos que se desean eliminar del esquemático, ya que no están presentes en el *layout*. Para este caso, se eliminan todos los puertos de cada celda, ya que al trabajar con celdas incompletas, el *layout* carece de puertos al comparalos con los del esquemático.

Esta función se debe utilizar para cada una de las celdas que se van a utilizar en el diseño. En el caso de "El Gran Jaguar", se utilizan un total de 62 celdas distintas, por lo que se debe definir la versión equivalente de cada una de ellas para poder ejecutar el *Black Box* LVS de manera exitosa.

Creación de las cajas negras

Para agilizar la creación de las cajas negras, se desarrolló un *script* en Python que genera automáticamente las definiciones de las cajas negras para cada una de las celdas utilizadas en el diseño. Cabe recalcar que para el funcionamiento del programa, es necesario contar con dos archivos. El primero de ellos, es el *netlist* en estructural en Verilog sin las instancias de los componentes no reconocidos por IC Validator, generado por el *script* `limpiar_netlist.py` previamente.

El segundo archivo, es un archivo con terminación `_lvs.log`, generado por IC Validator después de ejecutar LVS, por lo que se sugiere avanzar momentáneamente a la sección 10.5 para generar dicho archivo y poder hacer uso del *script*.

Después de ejecutar LVS, se creará una carpeta llamada `run_details`, en la cual se encuentra el archivo con terminación `_lvs.log`. Este archivo contiene un resumen detallado de los resultados de la verificación LVS, incluyendo información sobre las celdas utilizadas

en el diseño, y, en este caso, celdas que deberían de haber sido tratadas como cajas negras, pero no lo fueron. Esta distinción es instanciada por el error:

```
ERROR:  Found schematic empty cell XXXX not defined as device.
```

Por medio de este error es que el *script* extrae automáticamente los nombres de las celdas que deben ser definidas como cajas negras.

Una vez se tienen estos dos archivos, se procede a ejecutar el siguiente *script* en una terminal abierta en la carpeta donde se encuentran los archivos mencionados anteriormente:

Cuadro 66. *Script* `crear_blackboxes.py` para generar las definiciones de las cajas negras

```
import re

def find_empty_cells(log_file):
    empty_cells = set()
    pattern = r"ERROR:\s*Found schematic empty cell\s+(\S+)\s+not
                defined as device\."
    with open(log_file, 'r') as f:
        for line in f:
            match = re.search(pattern, line)
            if match:
                empty_cells.add(match.group(1))
    return empty_cells


def parse_verilog_ports(verilog_file, cells):
    cell_ports = {cell: set() for cell in cells}

    with open(verilog_file, 'r') as f:
        content = f.read()

    # Normaliza el texto para evitar depender de saltos de linea
    content = re.sub(r'\s+', ' ', content)

    for cell in cells:
        # Captura instancias del tipo: CellName ( .PORT(signal) );
        pattern = rf"{cell}\s+\S+\s*\((.*?)\)\s*;"
        matches = re.findall(pattern, content, flags=re.DOTALL)

        for match in matches:
            # Extrae nombres de puertos del tipo .PORT_NAME(
            ports = re.findall(r"\.(\w+)\s*\(", match)
            cell_ports[cell].update(ports)

    return cell_ports

...
```

```
...

def generate_black_boxes(cell_ports,output_file="blackboxes.txt"):
    with open(output_file, 'w') as f:
        for cell, ports in cell_ports.items():
            ports_list = '", "'.join(sorted(ports))
            block = (
f'lvs_black_box_options(\n'
f'    equiv_cells = {{ {{schematic_cell = "{cell}",
                       layout_cell = "{cell}"}} }},\n'
f'    remove_schematic_ports = {{"{ports_list}"}}\n'
f');\n\n'
            )
            f.write(block)

def main():
    log_file = input("Ingresa el nombre del archivo .log: ")
    verilog_file = input("Ingresa el nombre del archivo .v: ")

    print("\nBuscando celdas vacias reportadas en el log...")
    cells = find_empty_cells(log_file)

    if not cells:
        print("No se encontraron celdas vacias.")
        return

    print("Celdas detectadas:", ", ".join(cells))

    print("Extrayendo puertos desde el archivo .v...")
    cell_ports = parse_verilog_ports(verilog_file, cells)

    print("Generando archivo blackboxes.txt...")
    generate_black_boxes(cell_ports)

    print("\n Proceso completado. Revisa el archivo
          blackboxes.txt")

if __name__ == "__main__":
    main()
```

Nota. El cuadro contiene el código del *script* `crear_blackboxes.py` en Python.

El *script* realiza las siguientes operaciones principales:

1. Solicita al usuario los nombres de los archivos `.log` y `.v`.
2. Lee el archivo `.log` para identificar las celdas vacías que deben ser tratadas como cajas

negras.

3. Lee el archivo `.v` para extraer los nombres de los puertos asociados a cada una de las celdas identificadas.

4. Genera un archivo de texto llamado `blackboxes.txt` que contiene las definiciones de las cajas negras en el formato requerido por IC Validator.

Después de ejecutar el *script*, se puede proceder a copiar todas las definiciones de las cajas negras en el archivo `blackboxes.txt`, para después pegarlas al final del *Environment Setup* del *runset*. A continuación, se muestran las 62 definiciones de las cajas negras generadas para el caso de "El Gran Jaguar":

Cuadro 67. Definición de las cajas negras en el *runset* utilizadas para "El Gran Jaguar"

```
//////////////////////
///// BLACKBOXES /////
//////////////////////

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OAI21D0",
                     layout_cell = "OAI21D0"} },
    remove_schematic_ports = {"A1", "A2", "B", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "CKND1",
                     layout_cell = "CKND1"} },
    remove_schematic_ports = {"I", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "CKXOR2D0",
                     layout_cell = "CKXOR2D0"} },
    remove_schematic_ports = {"A1", "A2", "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OA21D0",
                     layout_cell = "OA21D0"} },
    remove_schematic_ports = {"A1", "A2", "B", "VDD", "VSS", "Z"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OA31D0",
                     layout_cell = "OA31D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AOI211D0",
                     layout_cell = "AOI211D0"} },
    remove_schematic_ports = {"A1", "A2", "B", "C",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OA32D0",
                     layout_cell = "OA32D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B1", "B2",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "IND2D0",
                     layout_cell = "IND2D0"} },
    remove_schematic_ports = {"A1", "B1", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "TIEL",
                     layout_cell = "TIEL"} },
    remove_schematic_ports = {"VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "PDDW0204SCDG",
                     layout_cell = "PDDW0204SCDG"} },
    remove_schematic_ports = {"C", "DS", "I", "IE",
                              "OEN", "PAD", "PE"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "INR3D0",
                     layout_cell = "INR3D0"} },
    remove_schematic_ports = {"A1", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OR2D1",
                     layout_cell = "OR2D1"} },
    remove_schematic_ports = {"A1", "A2", "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OAI32D0",
                     layout_cell = "OAI32D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "DFCNQD1",
                     layout_cell = "DFCNQD1"} },
    remove_schematic_ports = {"CDN", "CP", "D", "Q", "VDD", "VSS"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OR2D0",
                     layout_cell = "OR2D0"} },
    remove_schematic_ports = {"A1", "A2", "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AO31D0",
                     layout_cell = "AO31D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B",
                              "VDD", "VSS", "Z"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AO22D0",
                     layout_cell = "AO22D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OAI221D0",
                     layout_cell = "OAI221D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2", "C",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "INR2D0",
                     layout_cell = "INR2D0"} },
    remove_schematic_ports = {"A1", "B1", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "INVD1",
                     layout_cell = "INVD1"} },
    remove_schematic_ports = {"I", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "IINR4D0",
                     layout_cell = "IINR4D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AOI221D0",
                     layout_cell = "AOI221D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2", "C",
                              "VDD", "VSS", "ZN"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "CKND2",
                     layout_cell = "CKND2"} },
    remove_schematic_ports = {"I", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "CKND2D0",
                     layout_cell = "CKND2D0"} },
    remove_schematic_ports = {"A1", "A2", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AOI222D0",
                     layout_cell = "AOI222D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2", "C1", "C2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AN3D0",
                     layout_cell = "AN3D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "MAOI22D0",
                     layout_cell = "MAOI22D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AOI22D0",
                     layout_cell = "AOI22D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "NR4D0",
                     layout_cell = "NR4D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "A4",
                              "B1", "B2", "B3",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "PVSS1CDG",
                     layout_cell = "PVSS1CDG"} },
    remove_schematic_ports = {"VSS"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AOI21D0",
                     layout_cell = "AOI21D0"} },
    remove_schematic_ports = {"A1", "A2", "B", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OAI31D0",
                     layout_cell = "OAI31D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "CKND2D1",
                     layout_cell = "CKND2D1"} },
    remove_schematic_ports = {"A1", "A2", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "INR4D0",
                     layout_cell = "INR4D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2", "B3",
                              "VDD", "VSS", "ZN"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AOI32D0",
                     layout_cell = "AOI32D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "ND4D0",
                     layout_cell = "ND4D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "A4",
                              "B1", "B2", "B3",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "INVD0",
                     layout_cell = "INVD0"} },
    remove_schematic_ports = {"I", "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "CKAN2D0",
                     layout_cell = "CKAN2D0"} },
    remove_schematic_ports = {"A1", "A2", "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "IND3D0",
                     layout_cell = "IND3D0"} },
    remove_schematic_ports = {"A1", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OA211D0",
                     layout_cell = "OA211D0"} },
    remove_schematic_ports = {"A1", "A2", "B", "C",
                              "VDD", "VSS", "Z"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "DFQD1",
                     layout_cell = "DFQD1"} },
    remove_schematic_ports = {"CP", "D", "Q", "VDD", "VSS"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OAI22D0",
                     layout_cell = "OAI22D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "TIEH",
                     layout_cell = "TIEH"} },
    remove_schematic_ports = {"VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "MOAI22D0",
                     layout_cell = "MOAI22D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AO21D0",
                     layout_cell = "AO21D0"} },
    remove_schematic_ports = {"A1", "A2", "B",
                              "VDD", "VSS", "Z", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "IAO21D0",
                     layout_cell = "IAO21D0"} },
    remove_schematic_ports = {"A1", "A2", "B", "VDD", "VSS", "ZN"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "PVDD1CDG",
                     layout_cell = "PVDD1CDG"} },
    remove_schematic_ports = {"VDD"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OA221D0",
                     layout_cell = "OA221D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2", "C",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "IND4D0",
                     layout_cell = "IND4D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2", "B3",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OAI211D0",
                     layout_cell = "OAI211D0"} },
    remove_schematic_ports = {"A1", "A2", "B", "C",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AO221D0",
                     layout_cell = "AO221D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2", "C",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "NR3D0",
                     layout_cell = "NR3D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "IIND4D0",
                     layout_cell = "IIND4D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "ND3D0",
                     layout_cell = "ND3D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B1", "B2",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "NR2D0",
                     layout_cell = "NR2D0"} },
    remove_schematic_ports = {"A1", "A2", "B1",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OA22D0",
                     layout_cell = "OA22D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AOI31D0",
                     layout_cell = "AOI31D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "B",
                              "VDD", "VSS", "ZN"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OR3D0",
                     layout_cell = "OR3D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "VDD", "VSS", "Z"}
);

...
```

```
...

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AO211D0",
                     layout_cell = "AO211D0"} },
    remove_schematic_ports = {"A1", "A2", "B", "C",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OR4D0",
                     layout_cell = "OR4D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "A4",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "AN4D0",
                     layout_cell = "AN4D0"} },
    remove_schematic_ports = {"A1", "A2", "A3", "A4",
                              "VDD", "VSS", "Z"}
);

lvs_black_box_options(
    equiv_cells = { {schematic_cell = "OAI222D0",
                     layout_cell = "OAI222D0"} },
    remove_schematic_ports = {"A1", "A2", "B1", "B2",
                              "C1", "C2", "VDD", "VSS", "ZN"}
);

...
```

Nota. El cuadro contiene la definición completa de las cajas negras utilizadas en el *runset* para “El Gran Jaguar”, utilizando la función `lvs_black_box_options()`.

10.4. Bloque de comparación

En el flujo interno de LVS, una vez que se ha realizado la extracción eléctrica del *layout* y se ha generado la base de datos correspondiente (`lay_db`), el siguiente paso consiste en comparar dicha base con el *netlist* del esquemático (`sch_db`). Esta comparación se ejecuta en la sección *Compare Settings* del *runset*, específicamente mediante la función `compare()`, la cual analiza la equivalencia topológica, los nombres de redes (*nets*), y las propiedades geométricas y algunas eléctricas de los componentes del circuito integrado.

A continuación, se muestra el código completo de la sección *Compare Settings* del *runset*:

Cuadro 68. *Compare Settings* del *runset*

```
/* COMPARE SETTINGS */
match(
        state = cmp_matrix,
        detect_permutable_ports = true,
        match_by_net_name = true,
        match_condition = {
                property_mismatch = ERROR,
                missing_required_property = ERROR,
                top_layout_ports_without_name = ERROR,
                empty_cell_not_defined_as_device = ERROR,
                top_ports_matched_with_different_name = ERROR
        }
);

#ifdef CROSS_REFERENCE
xref_db = compare(
#else
compare(
#endif
        state = cmp_matrix,
        schematic = sch_db,
        layout = lay_db,
        user_functions_file = "DFM/icv_compare.rh",
        case_sensitive = {},
        push_down_pins = true,
        #ifdef RC_DECK
        memory_array_compare = false,
        #else
        memory_array_compare = true,
        #endif
        schematic_top_cell = SCHEMATIC_TOPCELL,
        action_on_error = EXPLODE,
        delete_layout_cells = {"pnwdio"},
        print_messages = {
                all_merged_device_list = true,
                filtered_device_list = true,
                pre_merge_stats = true
        }
);

...
```

```
...

/* SPICE NETLISTING */
spice_fh = spice_netlist_file(file = get_top_cell() + ".sp");
#ifdef CROSS_REFERENCE
write_xref_spice(
        xref_db = xref_db,
#else
write_spice(
#endif
        device_db = device_db,
        output_file = spice_fh,
        model_name_format = SPICE,
        precision = 6
);
```

Nota. El cuadro contiene el código de la sección *Compare Settings* del *runset*.

Dentro de esta sección, se pueden identificar tres bloques principales que definen el flujo de comparación LVS:

10.4.1. Bloque `match()`

El primer bloque define las reglas y condiciones bajo las cuales ICV considerará dos elementos (componentes o nodos) como equivalentes entre *layout* y esquemático.

Cuadro 69. Función `match()` del *runset*

```
match(
        state = cmp_matrix,
        detect_permutable_ports = true,
        match_by_net_name = true,
        match_condition = {
                property_mismatch = ERROR,
                missing_required_property = ERROR,
                top_layout_ports_without_name = ERROR,
                empty_cell_not_defined_as_device = ERROR,
                top_ports_matched_with_different_name = ERROR
        }
);
```

Nota. El cuadro contiene el código de la función `match()` utilizada en el *runset*.

Parámetro `state`

Define la matriz de comparación donde se almacenarán los resultados intermedios del proceso de emparejamiento. En este caso, la matriz (`cmp_matrix`) servirá como fuente para el análisis posterior del bloque `compare()`.

Parámetro `detect_permutable_ports`

Permite que ICV reconozca automáticamente puertos permutables, es decir, terminales que pueden intercambiarse sin afectar la funcionalidad del componente. Esto es especialmente útil en:

- Transistores MOS, donde las terminales de drenaje y fuente pueden ser equivalentes.
- Celdas pasivas como resistencias o capacitores simétricos.

Activar esta opción evita falsos errores LVS por diferencias de orientación o rotación de componentes en el *layout*.

Parámetro `match_by_net_name`

Habilita la comparación basada en nombres de red (*nets*). Esto implica que ICV utilizará los nombres de los nodos eléctricos como referencia principal para asociar nodos entre el esquemático y el *layout*. Si los nombres coinciden, ek chequeo puede resolver correspondencias más rápido y con menor ambigüedad.

Parámetro `match_condition`

Define las condiciones de error que deben considerarse críticas durante la comparación. Cada elemento dentro de este bloque especifica una regla de validación, por lo que si se incumple, se registra un error LVS.

- `property_mismatch = ERROR`: reporta error si las propiedades de un componente difieren entre *layout* y esquemático.
- `missing_required_property = ERROR`: indica si falta una propiedad esencial en algún componente del *layout*.
- `top_layout_ports_without_name = ERROR`: da error si existen puertos en la *top cell* del *layout* sin nombre definido. Esto impide la correcta correspondencia con el esquemático.
- `empty_cell_not_defined_as_device = ERROR`: genera un error si se encuentra una celda vacía que no está declarada explícitamente como un dispositivo.

- `top_ports_matched_with_different_name = ERROR` denota si un puerto del *layout* se conecta a un nombre distinto al del esquemático, lo cual garantiza consistencia en la nomenclatura de las conexiones.

Este bloque, en conjunto, asegura que el proceso de comparación se base tanto en estructura topológica como en consistencia de propiedades y aspectos eléctricos relevantes.

10.4.2. Bloque `compare()`

El siguiente bloque es el núcleo del flujo LVS. Aquí se realiza la comparación propiamente dicha entre las bases de datos del *layout* (`lay_db`) y del esquemático (`sch_db`).

Cuadro 70. Función *compare()* del *runset*

```
#ifdef CROSS_REFERENCE
xref_db = compare(
#else
compare(
#endif
        state = cmp_matrix,
        schematic = sch_db,
        layout = lay_db,
        user_functions_file = "DFM/icv_compare.rh",
        case_sensitive = {},
        push_down_pins = true,
        #ifdef RC_DECK
        memory_array_compare = false,
        #else
        memory_array_compare = true,
        #endif
        schematic_top_cell = SCHEMATIC_TOPCELL,
        action_on_error = EXPLODE,
        delete_layout_cells = {"pnwdio"},
        print_messages = {
                all_merged_device_list = true,
                filtered_device_list = true,
                pre_merge_stats = true
        }
);
```

Nota. El cuadro contiene el código de la función `compare()` utilizada en el *runset*.

Parámetro `xref_db`

Cuando la opción `CROSS_REFERENCE` está habilitada, los resultados de la comparación se almacenan en una base de datos de referencia cruzada (`xref_db`). Esto permite que IC Validator genere archivos que documenten las correspondencias exactas entre nodos y componentes del *layout* y del esquemático.

Si `CROSS_REFERENCE` está deshabilitada, la función `compare()` se ejecuta normalmente, pero sin guardar este mapa detallado.

Parámetros `schematic` y `layout`

Estas variables indican las dos bases de datos que serán comparadas:

- `schematic`: especifica el identificador de la base de datos de la *netlist* del esquemático.
- `layout`: especifica el identificador de la base de datos de la *netlist* del *layout*.

Parámetro `user_functions_file`

Este parámetro tiene como argumento el archivo `icv_compare.rh`, el cual indica cómo debe ser el tratamiento de transistores MOS, gestionando cómo se agrupan, fusionan o interpretan cuando existen configuraciones en paralelo o en serie dentro del *layout*:

Cuadro 71. Archivo `icv_compare.rh`

```
#include <icv_compare.rh>
#include <math.rh>

mos_parallel_merge_ratio : entrypoint function (void)
returning void {
        device = lvs_current_device();
        isMos : boolean = false;
        isParallel : boolean = lvs_is_parallel(device);

        if (lvs_is_nmos(device) || lvs_is_pmos(device)) {
                isMos = true;
        }

        if (isParallel && isMos) {
                multi_wl : double = 0;
                div_wl : double = 0;
                width : double = 0;
                length : double = 0;
                result1 = lvs_sum_of_products(device, "w", "l",
                                                         multi_wl);
                result2 = lvs_sum_of_divisions(device, "w", "l",
                                                           div_wl);

...
```

```
...
                length = sqrt(multi_wl / div_wl);
                width = sqrt(multi_wl * div_wl);
                lvs_save_double_property("w", width);
                lvs_save_double_property("l", length);
        }
}

mos_series_merge_ratio : entrypoint function (void)
returning void {
        device = lvs_current_device();
        isMos : boolean = false;
        isSeries : boolean = lvs_is_series(device);

        if (lvs_is_nmos(device) || lvs_is_pmos(device)) {
                isMos = true;
        }

        if (isSeries && isMos) {
                multi_wl : double = 0;
                div_wl : double = 0;
                width : double = 0;
                length : double = 0;
                result1 = lvs_sum_of_products(device, "w", "l",
                                                       multi_wl);
                result2 = lvs_sum_of_divisions(device, "w", "l",
                                                          div_wl);
                length = sqrt(multi_wl * div_wl);
                width = sqrt(multi_wl / div_wl);
                lvs_save_double_property("w", width);
                lvs_save_double_property("l", length);
        }
}

recalculate_schematic_mos_nf : entrypoint function (void)
returning void {
        device = lvs_current_device();
        isSch : boolean = lvs_is_schematic_device(device);
        noNF : boolean = true;

        nf : double;
        if (lvs_get_double_property(device, "nf", nf)) {
                noNF = false;
        }

...
```

```
...

        if (isSch && noNF) {
                lvs_save_double_property("nf", 1);
        }
}
mf_mos_merge_parallel_func : entrypoint function (void)
returning void {
        device = lvs_current_device();

        is_mos : boolean = false;
        if ( lvs_is_nmos(device) || lvs_is_pmos(device) ) {
            is_mos = true;
        }

        if(is_mos) {
            nf : double=0;
            if( lvs_is_parallel(device)) {
                number = lvs_count_members(device);
                mos_parallel_merge_ratio();
                if(lvs_get_double_property(device,"nf",nf)) {
                    nf = nf / number;
                    lvs_save_double_property("nf",nf);
                } else {
                    lvs_save_double_property("nf",1.0);
                }
            }
        }
}

mf_mos_parallel_merge_exclude_func : entrypoint function (void)
returning void {
        count = lvs_count_candidates();
        for(i = 0 to count-1 step 1) {
            device = lvs_get_candidate(i);
            defined_nf = lvs_get_double_property(device,
                                                  "nf", nf);
            if(dblgt(nf, 1.0)) {
                lvs_exclude_from_merge(device);
            }
        }
}
```

Nota. El cuadro contiene el código del archivo `icv_compare.rh` utilizado en el *runset*.

En un flujo LVS, los transistores pueden aparecer físicamente divididos en múltiples

segmentos (por ejemplo, para reducir la resistencia o mejorar la distribución de corriente), aunque en el esquemático estén representados como un solo dispositivo. Este archivo permite que el IC Validator reconozca y combine correctamente esas estructuras equivalentes, evitando falsos errores de comparación. Para ello, implementa funciones como:

- `mos_parallel_merge_ratio()` y `mos_series_merge_ratio()`: recalculan los valores efectivos de ancho (W) y largo (L) de los componentes resultantes de combinaciones paralelas o en serie, asegurando que el valor eléctrico total coincida con el del esquemático.
- `recalculate_schematic_mos_nf()`: asigna un valor por defecto al parámetro `nf` (*number of fingers*) si no está presente en el esquemático, para evitar errores por propiedades ausentes. (Este parámetro indica si un transistor MOS es un transistor *single-gate* o un transistor *multifinger*).

También se incluyen funciones para excluir ciertos dispositivos del proceso de fusión cuando su número de *fingers* o características físicas no cumplen con los criterios de equivalencia.

Parámetro `push_down_pins`

Permite que el ICV descienda en la jerarquía de celdas para buscar coincidencias en los pines internos. Es útil si hay pines en el *layout* que no están explícitamente definidos en un nivel superior.

Parámetro `memory_array_compare`

La configuración de este parámetro depende de si `RC_DECK` está habilitado. Si está activo, se desactiva la comparación de arreglos de memoria, ya que estos se modelan de forma distinta en el *layout*. Si está inactivo, evita falsos errores LVS en bloques de memoria modelados como celdas *macro*.

Parámetro `schematic_top_cell`

Define cuál es la *top cell* del esquemático que será utilizada como punto de comparación con la celda superior del *layout*, que en este caso, manda a llamar a la variable `SCHEMATIC_TOPCELL` definida en el *Environment Setup*.

Parámetro `action_on_error`

Indica cómo actuar si ocurre un error de correspondencia entre celdas, para este caso se indica que el sistema debe expandir la jerarquía (explosionar) el diseño para identificar la causa raíz dentro de los niveles inferiores.

Parámetro `delete_layout_cells`

Excluye explícitamente ciertas celdas del *layout* (en este caso, diodos parasitarios (pnwdio)) que no deben considerarse en la comparación.

Parámetro `print_messages`

Habilita la creación de reportes de diagnóstico durante la comparación:

- `all_merged_device_list = true`: imprime todos los dispositivos fusionados durante el proceso.
- `filtered_device_list = true`: muestra los dispositivos descartados.
- `pre_merge_stats = true`: resume estadísticas previas a la fusión de estructuras equivalentes.

10.4.3. Bloque SPICE

El bloque final genera la *netlist* SPICE del diseño, basada en los resultados de la extracción y comparación. Dependiendo de si `CROSS_REFERENCE` está habilitado, el archivo contendrá o no información de mapeo entre esquemático y *layout*.

Cuadro 72. Bloque SPICE del *runset*

```
spice_fh = spice_netlist_file(file = get_top_cell() + ".sp");
#ifdef CROSS_REFERENCE
write_xref_spice(
        xref_db = xref_db,
#else
write_spice(
#endif
        device_db = device_db,
        output_file = spice_fh,
        model_name_format = SPICE,
        precision = 6
);
```

Nota. El cuadro contiene el código del bloque SPICE utilizado en el *runset*.

Función `spice_netlist_file()`

Crea el archivo de salida con extensión `.sp`, nombrado según la *top cell*.

Parámetro `write_xref_spice()` / `write_spice()`

- `write_xref_spice()`: genera una netlist SPICE que incluye información de referencia cruzada (`xref_db`), útil para depuración de LVS.
- `write_spice()`: genera una netlist estándar sin correspondencias explícitas.

Parámetro `device_db`

Indica que la fuente de datos para generar la *netlist* proviene de la base de dispositivos extraída en el flujo RC/LVS.

Parámetro `precision`

Define la precisión numérica (en este caso, seis dígitos significativos) para los parámetros de los componentes.

10.5. Ejecución de la prueba LVS

Con el *runset* modificado, se puede proceder a ejecutar la prueba LVS, pero antes de esto se recomienda seguir los siguientes pasos:

10.5.1. Preparación de archivos

- Para correr esta prueba de una manera ordenada, se recomienda crear una carpeta dedicada exclusivamente a la prueba LVS, donde se coloquen únicamente los archivos necesarios.
- En la carpeta creada exclusivamente para la prueba, colocar el archivo `.gds`, el archivo `.icv`, el archivo `unit.cdl`, el archivo `icv_compare.rh` y el *runset* modificado (aunque este archivo no sea una entrada en sí, es el medio por el cual se corre la prueba).
- Verificar que los nombres y las rutas de los archivos en el *runset* hagan referencia a los archivos pertinentes.

10.5.2. Comando de ejecución

Una vez se tienen todos los archivos requeridos para la prueba en la carpeta, se debe abrir una terminal para ejecutar la prueba. Una vez ahí, se debe ejecutar el siguiente comando:

```
icv -i archivo.gds -c top_cell -s netlist.icv -sf ICV -vue runset
```

Donde cada parámetro indica lo siguiente:

- `icv`: corresponde al ejecutable principal de IC Validator.
- `-i`: indica el archivo de *layout* de entrada, en formato GDSII. Este archivo contiene la representación física del diseño.
- `-c`: especifica el nombre de la celda superior (*top cell*) dentro del archivo GDS, a partir de la cual se inicia la extracción y verificación.
- `-s` define el archivo de esquemático o *netlist* de referencia. Este archivo es la fuente contra la cual se comparará el *netlist* extraído del *layout*.
- `-sf`: señala el formato del netlist de referencia indicado con `-s`. En este caso, *ICV* especifica que el archivo está en un formato reconocido directamente por IC Validator.
- `-vue`: hace referencia a un archivo que define las reglas y el ambiente de la prueba, que en este caso, es el *runset*.

La estructura del comando puede variar según las necesidades específicas del diseño y las configuraciones del entorno de trabajo, para este caso, el comando proporcionado fue hecho en base al manual *IC Validator User Guide* [30], y a las necesidades de diseño de "El Gran Jaguar".

10.6. Archivos generados

Después de ejecutar el comando, IC Validator generará varios archivos de salida en la carpeta de archivos de salida. Estos archivos de salida tienen el nombre de la *top cell*, seguido de un sufijo que indica el tipo de archivo:

- `top_cell.LVS_ERRORS`: este archivo almacena el listado detallado de los errores de LVS. Al igual que el archivo `.RESULTS`, este muestra con un texto en grande si la verificación aprueba el diseño o no (*PASS / FAIL*). El archivo contiene información sobre todas las celdas utilizadas, con el objetivo de determinar si el esquemático y el *layout* coinciden en todos los puntos equivalentes previstos por el usuario establecidos en el *runset* utilizados para la comparación. Consecuentemente, sirve como guía para localizar y corregir las discrepancias entre los dos tipos de archivos evaluados, y hace referencia a otros archivos generados que contienen detalles adicionales sobre los errores encontrados.
- `top_cell_lvs.log`: este archivo contiene registros detallados del proceso de verificación LVS. Incluye información sobre cada paso realizado durante la comparación, así como mensajes de error, advertencias y otros eventos relevantes que ocurrieron durante la ejecución de la prueba. Se encuentra dentro de la carpeta `run_details`.
- `sum.cell.cell`: este archivo proporciona un resumen de la verificación LVS para una celda específica dentro del diseño. Contiene estadísticas y resultados relacionados con esa celda, ayudando a identificar problemas o confirmar la correcta implementación de esa parte del diseño. Dentro de la carpeta `compare`, que a su vez está dentro de la

carpeta `run_details`, se encuentra una carpeta por cada una de las celdas utilizadas en el diseño, y dentro de cada una de estas carpetas es que se puede encontrar la variante de este archivo para cada celda.

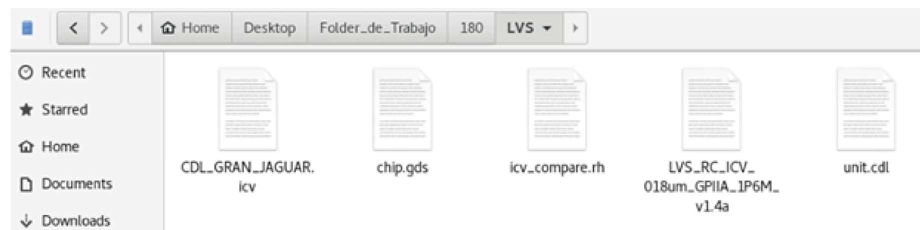
Llegar a conseguir un diseño que pase la prueba LVS es un proceso iterativo, ya que es común encontrar errores que requieren volver al *layout* o al esquemático para hacer correcciones. Para que este proceso sea más eficiente, es importante interpretar correctamente los archivos de salida y entender la naturaleza de los errores reportados, por lo que se recomienda acudir a la documentación oficial de IC Validator para entenderlos a profundidad y determinar el método correcto para corregirlos.

10.7. Réplica de la prueba LVS con tecnología de 180 nm

Para validar el flujo de trabajo descrito en este capítulo, se decidió replicar la prueba LVS utilizando la tecnología de 180 nm, utilizando el diseño de "El Gran Jaguar" desarrollado por estudiantes de la universidad en años anteriores. Para replicar el procedimiento, se siguieron los mismos pasos descritos en este capítulo, utilizando las herramientas de Synopsys y el *runset* correspondiente a esa tecnología.

10.7.1. Preparación de archivos

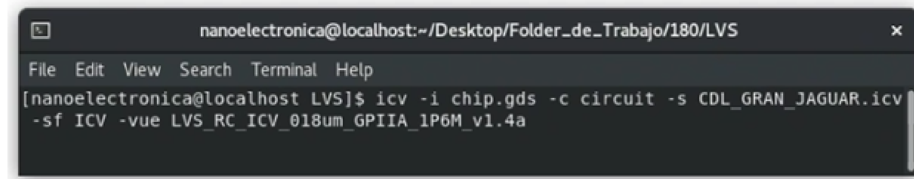
Figura 21. Carpeta dedicada a LVS con los archivos necesarios para la prueba



Nota. La imagen muestra una captura de pantalla de la carpeta preparada para correr la verificación LVS. `Elaboración propia.`

10.7.2. Comando de ejecución

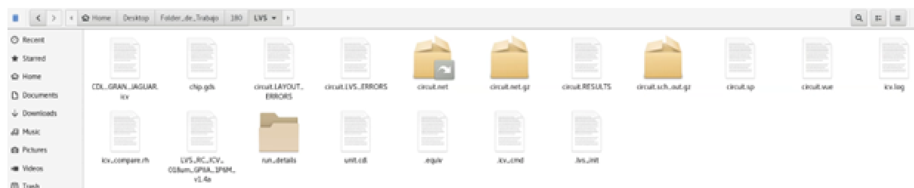
Figura 22. Terminal con el comando introducido para correr la prueba LVS



Nota. La imagen muestra una captura de pantalla de la terminal preparada para correr la verificación LVS. `Elaboración propia.`

10.7.3. Archivos generados

Figura 23. Archivos generados después de correr la prueba LVS



Nota. La imagen muestra una captura de pantalla de los archivos generados después de correr la verificación LVS. `Elaboración propia.`

Figura 24. Archivo `topcell.LVS_ERRORS` de la prueba LVS con tecnología de 180 nm

```
 1 +---------------------------------------------------------------------+
 2 |                    ICV_Compare LVS Comparison Report                |
 3 +---------------------------------------------------------------------+
 4
 5 ICV_Compare (R) Hierarchical Layout Vs. Schematic
 6         RHEL64 W-2024.09-SP4.11451981 2025/02/26
 7 Copyright (C) Synopsys, Inc. All rights reserved.
 8
 9
10 ---------------------------------------------------------------------
11 LVS error file     = circuit.LVS_ERRORS
12 Layout error file  = circuit.LAYOUT_ERRORS
13 Schematic netlist  = /home/nanoelectronica/Desktop/Folder_de_Trabajo/180/LVS/circuit.sch_out.gz
14 Layout netlist     = /home/nanoelectronica/Desktop/Folder_de_Trabajo/180/LVS/circuit.net.gz
15 Runset file        = LVS_RC_ICV_018um_GPIIA_1P6M_v1.4a
16 Working directory  = /home/nanoelectronica/Desktop/Folder_de_Trabajo/180/LVS
17 Compare directory  = run_details/compare
18 Compare start time = 2025-10-23 11:40:52
19
20
21
22 ---------------------------------------------------------------------
23 Final comparison result:PASS
24
25                         #####   ##   ##### #####
26                         #   #  #  #  #     #
27                         ##### ###### ##### #####
28                         #     #    #     #     #
29                         #     #    # ##### #####
30
31
32 TOP equivalence point:
33                 [circuit, circuit]
34
35 Comparison summary
36
37     67 Successful blackbox cells
38      0 Failed blackbox cells
39
40      1 Successful equivalence points
41      0 Failed equivalence points
```

Nota. La imagen muestra una captura de pantalla del archivo `topcell.LVS_ERRORS` generado después de correr la verificación LVS. `Elaboración propia.`

10.8. Resultados de la prueba LVS con tecnología de 65 nm

Se realizó la prueba LVS para un total de 3 circutos integrados diferentes, cada uno incrementando en complejidad en comparación al anterior, hasta llegar a "El Gran Jaguar".

10.8.1. Compuerta NOT

Figura 25. Archivo `IO_NOT.LVS_ERRORS` de la prueba LVS con tecnología de 65 nm

```
1 +---------------------------------------------------------------------+
2 |                     ICV_Compare LVS Comparison Report               |
3 +---------------------------------------------------------------------+
4
5 ICV_Compare (R) Hierarchical Layout Vs. Schematic
6         RHEL64 W-2024.09-SP4.11451981 2025/02/26
7 Copyright (C) Synopsys, Inc. All rights reserved.
8
9
10 ----------------------------------------------------------------------
11 LVS error file     = IO_not.LVS_ERRORS
12 Layout error file  = IO_not.LAYOUT_ERRORS
13 Schematic netlist  = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/NOT/LVS/IO_not.sch_out.gz
14 Layout netlist     = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/NOT/LVS/IO_not.net.gz
15 Runset file        = DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m
16 Working directory  = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/NOT/LVS
17 Compare directory  = run_details/compare
18 Compare start time = 2025-11-07 20:12:38
19
20
21
22 ----------------------------------------------------------------------
23 Final comparison result:PASS
24
25                         #####   ##   ##### #####
26                         #   #  #  #  #     #
27                         ##### ###### ##### #####
28                         #     #    #     #     #
29                         #     #    # ##### #####
30
31
32 TOP equivalence point:
33                 [IO_not, IO_not]
34
35 Comparison summary
36
37     6 Successful blackbox cells
38     0 Failed blackbox cells
39
40     1 Successful equivalence points
41     0 Failed equivalence points
```

Nota. La imagen muestra una captura de pantalla del archivo `IO_NOT.LVS_ERRORS` generado después de correr la verificación LVS. `Elaboración propia.`

10.8.2. ALU

Figura 26. Archivo `IO_ALU.LVS_ERRORS` de la prueba LVS con tecnología de 65 nm

```
 1 +--------------------------------------------------------------------+
 2 |                    ICV_Compare LVS Comparison Report               |
 3 +--------------------------------------------------------------------+
 4
 5 ICV_Compare (R) Hierarchical Layout Vs. Schematic
 6         RHEL64 W-2024.09-SP4.11451981 2025/02/26
 7 Copyright (C) Synopsys, Inc. All rights reserved.
 8
 9
10 ---------------------------------------------------------------------
11 LVS error file     = IO_ALU_with_ring_osc.LVS_ERRORS
12 Layout error file  = IO_ALU_with_ring_osc.LAYOUT_ERRORS
13 Schematic netlist  = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/ALU/LVS/IO_ALU_with_ring_osc.sch_out.gz
14 Layout netlist     = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/ALU/LVS/IO_ALU_with_ring_osc.net.gz
15 Runset file        = DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m
16 Working directory  = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/ALU/LVS
17 Compare directory  = run_details/compare
18 Compare start time = 2025-11-07 22:37:10
19
20
21
22 ---------------------------------------------------------------------
23 Final comparison result:PASS
24
25                         #####   ##   ##### #####
26                         #   #  #  #  #     #
27                         ##### ###### ##### #####
28                         #     #    #     #     #
29                         #     #    # ##### #####
30
31
32 TOP equivalence point:
33                 [IO_ALU_with_ring_osc, IO_ALU_with_ring_osc]
34
35 Comparison summary
36
37     37 Successful blackbox cells
38      0 Failed blackbox cells
39
40      1 Successful equivalence points
41      0 Failed equivalence points
```

Nota. La imagen muestra una captura de pantalla del archivo `IO_ALU.LVS_ERRORS` generado después de correr la verificación LVS. `Elaboración propia.`

10.8.3. El Gran Jaguar

Figura 27. Archivo `IO_nanochip.LVS_ERRORS` de la prueba LVS con tecnología de 65 nm

```
 1 +-----------------------------------------------------------------------+
 2 |                    ICV_Compare LVS Comparison Report                  |
 3 +-----------------------------------------------------------------------+
 4
 5 ICV_Compare (R) Hierarchical Layout Vs. Schematic
 6         RHEL64 W-2024.09-SP4.11451981 2025/02/26
 7 Copyright (C) Synopsys, Inc. All rights reserved.
 8
 9
10 -------------------------------------------------------------------------
11 LVS error file     = IO_nanochip.LVS_ERRORS
12 Layout error file  = IO_nanochip.LAYOUT_ERRORS
13 Schematic netlist  = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/GRAN_JAGUAR/LVS/IO_nanochip.sch_out.gz
14 Layout netlist     = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/GRAN_JAGUAR/LVS/IO_nanochip.net.gz
15 Runset file        = DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m
16 Working directory  = /home/nanoelectronica/Desktop/Folder_de_Trabajo/65/ICV/GRAN_JAGUAR/LVS
17 Compare directory  = run_details/compare
18 Compare start time = 2025-11-08 02:57:07
19
20
21
22 -------------------------------------------------------------------------
23 Final comparison result:PASS
24
25                        #####   ##   ##### #####
26                        #   #  #  #  #     #
27                        ##### ###### ##### #####
28                        #     #    #     #     #
29                        #     #    # ##### #####
30
31
32 TOP equivalence point:
33                 [IO_nanochip, IO_nanochip]
34
35 Comparison summary
36
37     62 Successful blackbox cells
38      0 Failed blackbox cells
39
40      1 Successful equivalence points
41      0 Failed equivalence points
```

Nota. La imagen muestra una captura de pantalla del archivo `IO_nanochip.LVS_ERRORS` generado después de correr la verificación LVS. `Elaboración propia.`

10.9. Análisis de resultados

Como se puede observar en la sección anterior, los resultados obtenidos para los 3 circuitos diferentes fueron exitosos, por lo que sólo se realizará un análisis más a fondo de los archivos resultantes de la prueba correspondiente a “El Gran Jaguar”.

10.9.1. `IO_nanochip.LVS_ERRORS`

El archivo`IO_nanochip.LVS_ERRORS` detalla que se verificaron 62 celdas en formato de caja negra con 0 fallas, y 1 punto de equivalencia con 0 fallas; además, el resumen *"Post-compare"* a nivel `TOP` muestra 0 desparejos de dispositivos, *nets* o puertos para los tipos de celdas listados. También se observa la configuración de `global_nets` y dominios de potencia/masa coherentes entre esquemático y *layout*, lo que ayuda a estabilizar la comparación en ausencia del *back-end*. El mapeo de celdas y dominios es correcto, ya que no se evidencian problemas de equivalencia de alto nivel, por lo que se recomienda mantener el flujo LVS establecido.

CAPÍTULO 11

Prueba *Electrical Rule Check* (ERC)

El proceso de verificación de reglas eléctricas (ERC) es una etapa crucial en el flujo de diseño de circuitos integrados, ya que asegura la integridad eléctrica del diseño antes de su fabricación. A diferencia de la verificación LVS, que se enfoca en la correspondencia entre el esquemático y el *layout*, la verificación ERC se centra en identificar posibles problemas eléctricos que podrían afectar el rendimiento y la funcionalidad del circuito.

11.1. Selección del *runset*

Al igual que en la prueba LVS, se utilizará un *runset* brindado por IMEC y creado por TSMC, el cual contiene las configuraciones necesarias para realizar la verificación ERC acorde a las especificaciones de la tecnología de 65 nm. En este caso, el *runset* es el mismo que se utiliza para la prueba LVS, pero con pequeñas modificaciones en la sección *Environment Setup*.

11.2. Configuración de la prueba ERC

Para realizar la prueba y consecuentemente la detección de errores de conectividad y consistencia eléctrica en el diseño de un circuito integrado, también se utilizará la herramienta IC Validator de Synopsys. Nuevamente, debido a que TSMC no nos brinda acceso al *back-end* de las celdas que se utilizarán en el circuito, se ejecutará un *Black Box* ERC.

El término "*Black Box* ERC", a diferencia de "*Black Box* LVS", no es un término estándar en la industria, pero se utiliza aquí para describir un enfoque similar al de la prueba LVS, donde el contenido de las celdas no será relevante durante la verificación eléctrica. En

lugar de verificar que las conexiones y circuitería interna de cada una de las celdas sea íntegra y sin discrepancias, IC Validator tratará a dichas celdas como cajas negras, por lo que sólo se verificarán las conexiones eléctricas entre las celdas.

11.2.1. Configuración del *runset*

Para este caso, se deben realizar algunas modificaciones en el *runset* utilizado anteriormente, en la sección *Environment Setup*:

Cuadro 73. Sección del *Environment Setup* sin modificaciones para correr ERC

```
/* EDIT: The following section contains all of the runset
variables for RC extraction tools. */
//#define DFM_RULE
//#define RC_DECK
#define CROSS_REFERENCE
//#define ZERO_NRS_NRD
//#define FILTER_DGS_TIED_MOS

#define WELL_TO_PG_CHECK
//#define GATE_TO_PG_CHECK
//#define PATH_CHECK

#define DS_TO_PG_CHECK
#define FLOATING_WELL_CHECK

//#define NW_RING
```

Nota. El cuadro contiene el código de la sección del *Environment Setup* del *runset* sin modificaciones para correr la prueba ERC.

Para correr la prueba ERC, se deben realizar las siguientes modificaciones:

Cuadro 74. *Environment Setup* del *runset* modificado para correr ERC

```
/* EDIT: The following section contains all of the runset
variables for RC extraction tools. */
//#define DFM_RULE
//#define RC_DECK
//#define CROSS_REFERENCE
//#define ZERO_NRS_NRD
//#define FILTER_DGS_TIED_MOS

#define WELL_TO_PG_CHECK
#define GATE_TO_PG_CHECK
#define PATH_CHECK

#define DS_TO_PG_CHECK
#define FLOATING_WELL_CHECK

//#define NW_RING
```

Nota. El cuadro contiene el código de la sección del *Environment Setup* del *runset* modificado para correr la prueba ERC.

Las opciones que no se encuentran comentadas después de las modificaciones son las que se utilizarán para correr la prueba ERC, y corresponden a cada una de las verificaciones mencionadas anteriormente.

- `#define WELL_TO_PG_CHECK`: activa la verificación para detectar si un *n-well* está conectado incorrectamente a tierra (VSS) o si un *p-substrate* está conectado erróneamente a potencia (VDD). Ambas condiciones representan errores de polarización de los pozos, los cuales pueden conducir a problemas graves como *latch-up* o fugas de corriente.

  El *runset* incluye verificaciones específicas, como `ERC_npvss49` y `ERC_ppvdd49`, que comprueban si cada tipo de *tap* está efectivamente vinculado al potencial correspondiente. Si un *tap* se encuentra desconectado o asociado al nodo incorrecto, el pozo pierde su polarización estable, pudiendo generar fugas de corriente, modulación no deseada del umbral (*body effect*), o incluso conducir a fallas de aislamiento entre transistores adyacentes.

- `#define GATE_TO_PG_CHECK`: revisa si las compuertas de transistores están directamente conectadas a nodos de potencia o tierra. Aunque en algunos casos esta conexión se utiliza intencionalmente (por ejemplo, en transistores usados como *pull-up* o *pull-down*), en la mayoría de los diseños lógicos esto es un error.

  Las reglas del *runset*, tales como `ERC_npvss150` y `ERC_ppvdd150`, detectan este tipo de conexiones, reportando las compuertas que no están asociadas a una red lógica válida.

- `#define PATH_CHECK`: esta directiva activa las 4 verificaciones del tipo *path checks*, que detectan:

    - Nodos conectados únicamente a potencia.
    - Nodos conectados únicamente a tierra.
    - Nodos completamente aislados.
    - Nodos sin conexión a ninguna red etiquetada o identificable.

- `#define DS_TO_PG_CHECK`: activa la búsqueda de transistores cuyos terminales de drenaje (D) y fuente (S) están conectados simultáneamente a potencia y tierra, respectivamente. Esta conexión directa entre VDD y VSS a través del canal del transistor es equivalente a un cortocircuito eléctrico en el dispositivo,

  En el *runset*, esta verificación se implementa mediante reglas del tipo `ERC_mnpg` y `ERC_mppg`, las cuales analizan la conectividad de los dispositivos dentro del *netlist* físico. Cuando se detecta que las terminales de un transistor comparten conexiones hacia ambos nodos de alimentación, se genera un reporte de error, indicando la posible existencia de un corto circuito interno. Este tipo de error puede provocar el consumo excesivo de corriente, el sobrecalentamiento del chip e incluso su destrucción durante su funcionamiento. Por lo tanto, esta verificación representa una de las comprobaciones más críticas dentro del proceso ERC.

- `#define FLOATING_WELL_CHECK`: esta opción revisa si existen pozos o substratos que no estén conectados ni a potencia ni a tierra. Los pozos flotantes pueden cargarse de forma indeterminada, provocando variaciones de umbral, inestabilidad o incluso activación de estructuras parásitas (como el *latch-up*).

  El *runset* define reglas específicas como `ERC_floating_nxwell`, `ERC_floating_psub` o `ERC_floating_hvwell`, que aseguran que todas las regiones de pozo y substrato se encuentren conectadas correctamente. La detección temprana de estos errores es crucial, especialmente en tecnologías de escala nanométrica donde las corrientes de fuga y el acoplamiento entre pozos son más pronunciados.

11.3. Bloque de verificación

Para cada una de las verificaciones eléctricas mencionadas, el *runset* incluye bloques de código específicos que implementan las reglas necesarias para detectar los errores. A continuación, se describen los bloques de código correspondientes a cada verificación, denotando que, debido a que en repetidas ocasiones se utiliza un mismo bloque de código con leves modificaciones para verificar diferentes aspectos de una misma prueba, se utilizará un bloque de código genérico para explicar el funcionamiento de cada verificación con textos entre comillas angulares (`<< >>`) para indicar las partes que varían.

11.3.1. *Well to Power/Ground Check*

Para realizar la verificación de *taps* a potencia/tierra, se utiliza el siguiente bloque de código dentro del *runset*:

Cuadro 75. Bloque de verificación de *Well to P/G Check*

```
#ifdef WELL_TO_PG_CHECK
ERC_XXXXXXX @= { @ "XXXXXXX : <<tap>> are connected to <<node>>";
        <<tap>>_<<node>> = net_texted_with(
                cdb,
                text = <<node>>_NAME,
                output_from_layers = {<<tap>>_not_var},
                texted_at = TOP_CELL,
                colon_text = EQUATE_NETS,
                processing_mode = HIERARCHICAL
        );
        copy(<<tap_cell>>_<<node>>);
}
#endif
```

Nota. El cuadro contiene el código del bloque de verificación de *Well to P/G Check* utilizado en el *runset*.

La función `net_texted_with()` selecciona polígonos de redes (*nets*) que contienen texto con las cadenas especificadas. En este caso, se busca identificar los *taps* (`tap`) que están conectados a la red de potencia (`POWER_NAME`) tierra (`GROUND_NAME`). Luego, se copia el resultado en otra variable (`copy(<<tap_cell>>_<<node>>)`) para marcar las áreas que cumplen la condición, y se reporta un error si un *ntap* o *ptap* está conectado al contacto incorrecto. La función opera de manera jerárquica, revisando todas las celdas dentro de la *top cell*.

11.3.2. *Gate to Power/Ground Check*

En el caso de la verificación de compuertas conectadas a potencia o tierra, el *runset* corre la prueba por medio del siguiente bloque de código:

Cuadro 76. Bloque de verificación de *Gate to P/G Check*

```
#ifdef GATE_TO_PG_CHECK
tndiff_<<node>> = net_texted_with(
        cdb,
        text = <<node>>_NAME,
        output_from_layers = {tndiff},
        texted_at = TOP_CELL
);
[gate]_GND = net_texted_with(
        cdb,
        text = <<node>>_NAME,
        output_from_layers = {<<gate>>},
        texted_at = TOP_CELL
);
XXXXXXXXX = interacting(<<gate>>_<<node>>, tndiff_<<node>>, == 2);

ERC_XXXXXXXX @= { @ "XXXXXXXX : <<gate>> is connected
                    to <<node>>";
        <<gate>>_<<node>> not XXXXXXXX;
}
#endif
```

Nota. El cuadro contiene el código del bloque de verificación de *Gate to P/G Check* utilizado en el *runset*.

La función `net_texted_with()` busca compuertas (gates) conectadas directamente a nodos de potencia (`POWER_NAME`) o tierra (`GROUND_NAME`). La función `interacting()` selecciona los polígonos de una capa que tocan o se superponen a los datos de otra capa. En este caso, se utiliza para determinar si las compuertas (`gate`) está en conectacto a los nodos de potencia (`tndiff_POWER`) o tierra (`tndiff_GROUND`) en más de un punto (`== 2`). En el caso de que se cumpla alguna de las condiciones establecidas, se reporta un error.

11.3.3. *Path Check*

Para llevar a cabo la verificación de rutas eléctricas o conexiones, el *runset* utiliza el siguiente fragmento de código:

Cuadro 77. Bloque de verificación de *Path Check*

```
#ifdef PATH_CHECK
ERC_PATH<<#>> @= { @ "<<label>>: Nodes have no path to ...";
        <<label>> = net_path_check(
                device_db,
                path_to = <<node>>,
                other_nets = {"*"},
                filter_nets = {FLOATING, TOP_PORT},
                output_from_layers = UNSPECIFIED_LIST_
                        POLYGON_LAYER
        );
        copy(<<label>>);
}
#endif
```

Nota. El cuadro contiene el código del bloque de verificación de *Path Check* utilizado en el *runset*.

La función `net_path_check()` selecciona polígonos de redes (*nets*) en la ruta especificada por el argumento `path_to`, obtenida de la base de datos del dispositivo que se ajustan a los criterios especificados. En este caso, la función comprueba si existen nodos con una ruta a una red de potencia (`POWER`), tierra (`GROUND`), o ninguna (`NONE`) a través de componentes como resistores, o la fuente o el drenaje de transistores MOS. Todas las redes se procesan dentro del contexto de la *top cell*.

En este caso, se realizan cuatro verificaciones distintas utilizando esta función:

- `ERC_PATH1`: identifica nodos que sólo tienen una ruta hacia potencia (`POWER`), pero no hacia tierra (`GROUND`).
- `ERC_PATH2`: detecta nodos que sólo tienen una ruta hacia tierra (`GROUND`), pero no hacia potencia (`POWER`).
- `ERC_PATH3`: encuentra nodos que no tienen rutas hacia ni potencia (`POWER`) ni tierra (`GROUND`), es decir, nodos completamente aislados (`NONE`).
- `ERC_PATH4`: localiza nodos que no tienen rutas hacia ninguna *net* etiquetada o identificable, por lo que al igual que en la variantes anterior, son nodos completamente aislados (`NONE`).

11.3.4. *Drain/Source to Power/Ground Check*

Para realizar la verificación de terminales de drenaje/fuente a potencia/tierra, se utiliza el siguiente bloque de código dentro del *runset*:

Cuadro 78. Bloque de verificación de *D/S to P/G Check*

```
#ifdef DS_TO_PG_CHECK
N1tndiff = net_texted_with(
        cdb,
        text = POWER_NAME,
        output_from_layers = {tndiff},
        colon_text = EQUATE_NETS,
        processing_mode = HIERARCHICAL,
        texted_at = TOP_CELL
);
T1tndiff = erc_<<transistor>>_gates outside_touching N1tndiff;

N2tndiff = net_texted_with(
        cdb,
        text = GROUND_NAME,
        output_from_layers = {tndiff},
        colon_text = EQUATE_NETS,
        processing_mode = HIERARCHICAL,
        texted_at = TOP_CELL
);
T2tndiff = erc_<<transistor>>_gates outside_touching N2tndiff;

ERC_XXXX @= { @ "XXXX : erc_<<transistor>>_gates S/D connected to
                both power and ground";
        T1tndiff and T2tndiff;
}
#endif
```

Nota. El cuadro contiene el código del bloque de verificación de *D/S to P/G Check* utilizado en el *runset*.

La función `net_texted_with()` localiza redes (*nets*) de conexión entre las terminales de fuente (*source*) o drenaje (*drain*) de transistores (NMOS o PMOS) y redes de potencia (`POWER_NAME`) o tierra (`GROUND_NAME`) al mismo tiempo. En este caso, se crean dos variables (`N1tndiff` y `N2tndiff`) que representan las redes de difusión tipo n, conectadas a potencia y tierra respectivamente. Luego, se crean otras dos variables (`T1tndiff` y `T2tndiff`) que representan los transistores NMOS o PMOS cuyas terminales están en contacto con nodos de potencia o tierra, respectivamente. Si se cumple la condición de que un transistor tiene sus terminales de drenaje y fuente conectadas a ambos nodos, se reporta un error.

11.3.5. *Floating Well Check*

Finalmente, para llevar a cabo la verificación de pozos flotantes, el *runset* utiliza el siguiente fragmento de código:

Cuadro 79. Bloque de verificación de *Floating Well Check*

```
#ifdef FLOATING_WELL_CHECK
<<well/substrate>>_<<suffix>> = net_path_check(
        device_db,
        path_to = <<node>>,
        filter_nets = {},
        output_from_layers = {<<well/substrate>>}
);
<<well/substrate>>_<<suffix>> = net_path_check(
        device_db,
        path_to = POWER_AND_GROUND,
        filter_nets = {},
        output_from_layers = {<<well/substrate>>}
);
<<well/substrate>>_<<suffix>> = <<well/substrate>> not
(<<well/substrate>>_<<suffix>> or <<well/substrate>>_<<suffix>>);

// <<well/substrate>> that are directly connected to <<node>>
<<well/substrate>>_<<suffix>> = net_texted_with(
        cdb,
        text = <<node>>_NAME,
        output_from_layers = {nxwell_float},
        texted_at = TOP_CELL,
        colon_text = EQUATE_NETS,
        processing_mode = HIERARCHICAL
);

ERC_floating_<<well/substrate>> @= { @ "FLOATING.<<well/
                                        substrate>> : Floating
                                        <<well/substrate>> is
                                        not allowed";
        <<well/substrate>>_<<suffix>> or
    <<well/substrate>>_<<suffix>>;
}
#endif
```

Nota. El cuadro contiene el código del bloque de verificación de *Floating Well Check* utilizado en el *runset*.

Para esta verificación, la función `net_path_check()` se utiliza para identificar pozos (`well`) o substratos (`substrate`) que no tienen una ruta hacia potencia (`POWER`) o tierra (`GROUND`), al igual que identificar pozos o substratos que se encuentran conectados a potencia y tierra de manera simultánea (`POWER_AND_GROUND`). Luego, se crean dos variables que representan los pozos o substratos que no están conectados a potencia ni tierra. Posteriormente, se utiliza la función `net_texted_with()` para identificar pozos o substratos que estén directamente conectados a nodos de potencia (`POWER_NAME`) o tierra (`GROUND_NAME`).

Finalmente, si cualquiera de estas condiciones se terminan cumpliendo, se reporta un error.

11.4. Ejecución de la prueba ERC

Con el *runset* modificado, se puede proceder a ejecutar la prueba ERC, siguiendo los mismos pasos que en la prueba LVS:

Preparación de archivos

- Crear una carpeta dedicada exclusivamente a la prueba ERC, donde se coloquen únicamente los archivos necesarios.
- En la carpeta creada exclusivamente para la prueba, colocar el archivo `.gds`, el archivo `.icv`, el archivo `unit.cdl`, el archivo `icv_compare.rh` y el *runset* modificado.
- Verificar que los nombres y las rutas de los archivos en el *runset* hagan referencia a los archivos pertinentes.

11.4.1. Comando de ejecución

Una vez se tienen todos los archivos requeridos para la prueba en la carpeta, se debe abrir una terminal para ejecutar la prueba. Una vez ahí, se debe ejecutar el siguiente comando:

```
icv -i archivo.gds -c top_cell -s cdl_netlist.icv -sf ICV -vue runset
```

El comando es el mismo que se utilizó para correr la prueba LVS, ya que IC Validator determina el tipo de verificación a realizar en función de las opciones definidas en el *runset*.

11.5. Archivos generados

Después de ejecutar el comando, IC Validator generará varios archivos de salida en la carpeta de archivos de salida. Estos archivos de salida tienen el nombre de la *top cell*, seguido de un sufijo que indica el tipo de archivo:

- `top_cell.RESULTS`: este archivo contiene el reporte principal de resultados de la ejecución de LVS (al ser el mismo *runset*, también se corre la prueba LVS), DRC y otras extracciones, entre las cuales ahora se encuentra ERC. Lo más relevante dentro del archivo es el texto en grande que indica si la prueba LVS fue exitosa (*PASS*), si la prueba DRC y otras extracciones fueron exitosas (*CLEAN*); o si se determinó que el *layout* y el esquemático no coinciden (*FAIL*), o si encontraron discrepancias en el diseño *NOT CLEAN*. Luego, resume el estado general de las pruebas, incluyendo estadísticas, número de celdas evaluadas, la configuración utilizada, y otros detalles relevantes para entender el resultado de la verificación.

- `top_cell.LAYOUT_ERRORS`: este archivo contiene los errores de extracción del *layout* detectados durante el análisis. El archivo muestra con un texto en grande si la verificación aprueba el diseño o no (*CLEAN* / *ERRORS*). En este se detallan problemas relacionados a las verificaciones eléctricas mencionadas anteriormente, tales como pozos flotantes, compuertas conectadas incorrectamente, nodos aislados, entre otros.

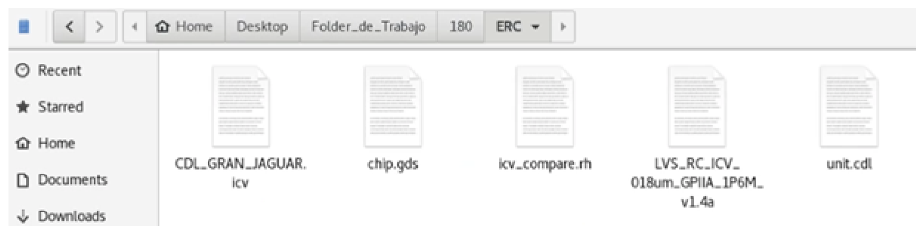
Como con la prueba LVS, llegar a conseguir un diseño que pase la prueba ERC es un proceso iterativo, ya que es común encontrar errores que requieren volver al diseño del circuito para hacer correcciones. Para que este proceso sea más eficiente, es importante interpretar correctamente los archivos de salida y entender la naturaleza de los errores reportados, por lo que se recomienda acudir a la documentación oficial de IC Validator para entenderlos a profundidad y determinar el método correcto para corregirlos.

11.6. Réplica de la prueba ERC con tecnología de 180 nm

Para validar el flujo de trabajo descrito en este capítulo, se decidió replicar la prueba ERC utilizando la tecnología de 180 nm, utilizando el diseño de "El Gran Jaguar" desarrollado por estudiantes de la universidad en años anteriores. Para replicar el procedimiento, se siguieron los mismos pasos descritos en este capítulo, utilizando las herramientas de Synopsys y el *runset* correspondiente a esa tecnología.

11.6.1. Preparación de archivos

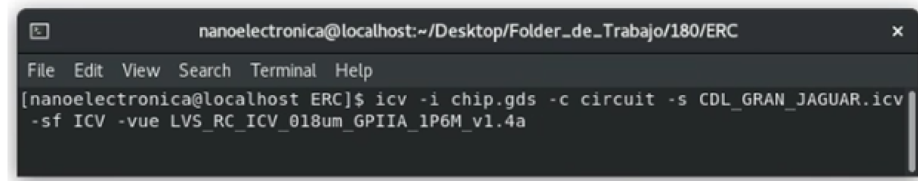
Figura 28. Carpeta dedicada a ERC con los archivos necesarios para la prueba



Nota. La imagen muestra una captura de pantalla de la carpeta preparada para correr la verificación ERC. `Elaboración propia.`

11.6.2. Comando de ejecución

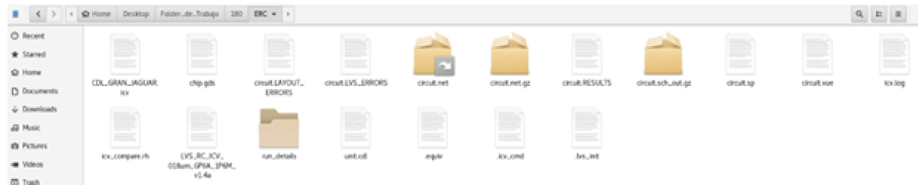
Figura 29. Terminal con el comando introducido para correr la prueba ERC



Nota. La imagen muestra una captura de pantalla de la terminal preparada para correr la verificación ERC. `Elaboración propia.`

11.6.3. Archivos generados

Figura 30. Archivos generados después de correr la prueba ERC



Nota. La imagen muestra una captura de pantalla de los archivos generados después de correr la verificación ERC. `Elaboración propia.`

Figura 31. Archivo `topcell.RESULTS` de la prueba ERC con tecnología de 180 nm

```
1                          LVS Compare Results: PASS
2
3                           ####   ###    ####   ####
4                          #   #  #   #  #      #
5                          ####   #####  #####  #####
6                          #      #   #      #      #
7                          #      #   #  ####   ####
8
9 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
10
11                     DRC and Extraction Results: CLEAN
12
13                      ####  #      #####   ###   #   #
14                     #      #      #      #   #  ##  #
15                     #      #      ####   #####  # # #
16                     #      #      #      #   #  #  ##
17                      ####  #####  #####  #   #  #   #
18
19 ===========================================================================
20
21
22 ---------------------------------------------------------------------------
23 ICV Execution
24 ---------------------------------------------------------------------------
25
```

Nota. La imagen muestra una captura de pantalla del archivo `topcell.RESULTS` generado después de correr la verificación ERC. `Elaboración propia.`

Figura 32. Archivo `topcell.LAYOUT_ERRORS` de la prueba ERC con tecnología de 180 nm

```
1                  LAYOUT ERRORS RESULTS: CLEAN
2
3                   #### #     #####   ##   #   #
4                  #     #     #      #  #  ##  #
5                  #     #     ####  ###### # # #
6                  #     #     #     #    # #  ##
7                   #### ##### ##### #    # #   #
8
9 ======================================================================
10
11 Library name:    chip.gds
12 Structure name:  circuit
13 Generated by:    IC Validator RHEL64 W-2024.09-SP4.11451981 2025/02/26
14 Runset name:     LVS_RC_ICV_018um_GPIIA_1P6M_v1.4a
15 User name:       nanoelectronica
16 Time started:    2025/10/23 11:55:31AM
17 Time ended:      2025/10/23 11:55:33AM
18
19 Called as: icv -i chip.gds -c circuit -s CDL GRAN JAGUAR.icv -sf ICV -vue LVS RC ICV 018um GPIIA 1P6M v1.4a
```

Nota. La imagen muestra una captura de pantalla del archivo `topcell.LAYOUT_ERRORS` generado después de correr la verificación ERC. `Elaboración propia.`

11.7. Resultados de la prueba ERC con tecnología de 65 nm

Se realizó la prueba ERC para un total de 3 circutos integrados diferentes, cada uno incrementando en complejidad en comparación al anterior, hasta llegar a “El Gran Jaguar”.

11.7.1. Compuerta NOT

Figura 33. Archivo `IO_NOT.RESULTS` de la prueba ERC con tecnología de 65 nm

```
                       LVS Compare Results: PASS

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

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

                 DRC and Extraction Results: NOT CLEAN

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

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


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

Nota. La imagen muestra una captura de pantalla del archivo `IO_NOT.RESULTS` generado después de correr la verificación ERC. `Elaboración propia.`

Figura 34. Archivo `IO_NOT.LAYOUT_ERRORS` de la prueba ERC con tecnología de 65 nm

```
                   LAYOUT ERRORS RESULTS: ERRORS

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

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

Library name:    IO_not.gds
Structure name:  IO_not
Generated by:    IC Validator RHEL64 W-2024.09-SP4.11451981 2025/02/26
Runset name:     DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m
User name:       nanoelectronica
Time started:    2025/11/07 08:12:28PM
Time ended:      2025/11/07 08:12:31PM

Called as: icv -i IO_not.gds -c IO_not -s IO_not.icv -sf ICV -vue DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m

                        ERROR SUMMARY

 FLOATING.psub : Floating psub is not allowed
   or ................................................ 9 violations found.




                        ERROR DETAILS



-------------------------------------------------------------------------
FLOATING.psub : Floating psub is not allowed
-------------------------------------------------------------------------
```

Nota. La imagen muestra una captura de pantalla del archivo `IO_NOT.LAYOUT_ERRORS` generado después de correr la verificación ERC. `Elaboración propia.`

11.7.2. ALU

Figura 35. Archivo `IO_ALU.RESULTS` de la prueba ERC con tecnología de 65 nm

```
LVS Compare Results: PASS

DRC and Extraction Results: NOT CLEAN

ICV Execution
```

Nota. La imagen muestra una captura de pantalla del archivo `IO_ALU.RESULTS` generado después de correr la verificación ERC. `Elaboración propia.`

Figura 36. Archivo `IO_ALU.LAYOUT_ERRORS` de la prueba ERC con tecnología de 65 nm

```
LAYOUT ERRORS RESULTS: ERRORS

Library name:    IO_ALU_with_ring_osc.gds
Structure name:  IO_ALU_with_ring_osc
Generated by:    IC Validator RHEL64 W-2024.09-SP4.11451981 2025/02/26
Runset name:     DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m
User name:       nanoelectronica
Time started:    2025/11/18 05:57:42PM
Time ended:      2025/11/18 05:57:45PM

Called as: icv -i IO_ALU_with_ring_osc.gds -c IO_ALU_with_ring_osc -s IO_ALU_with_ring_osc.icv -sf ICV -vue DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m

ERROR SUMMARY

FLOATING.psub : Floating psub is not allowed
  or ................................................ 40 violations found.

ERROR DETAILS

FLOATING.psub : Floating psub is not allowed
```

Nota. La imagen muestra una captura de pantalla del archivo `IO_ALU.LAYOUT_ERRORS` generado después de correr la verificación ERC. `Elaboración propia.`

11.7.3. El Gran Jaguar

Figura 37. Archivo `IO_nanochip.RESULTS` de la prueba ERC con tecnología de 65 nm

```
                     LVS Compare Results: PASS

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

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

                DRC and Extraction Results: NOT CLEAN

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

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


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

```

Nota. La imagen muestra una captura de pantalla del archivo `IO_nanochip.RESULTS` generado después de correr la verificación ERC. `Elaboración propia.`

Figura 38. Archivo `IO_nanochip.LAYOUT_ERRORS` de la prueba ERC con tecnología de 65 nm

```
                      LAYOUT ERRORS RESULTS: ERRORS

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

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

Library name:    IO_nanochip.gds
Structure name:  IO_nanochip
Generated by:    IC Validator RHEL64 W-2024.09-SP4.11451981 2025/02/26
Runset name:     DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m
User name:       nanoelectronica
Time started:    2025/11/08 03:22:16AM
Time ended:      2025/11/08 03:22:19AM

Called as: icv -i IO_nanochip.gds -c IO_nanochip -s IO_nanochip.icv -sf ICV -vue DFM_LVS_RC_ICV_N65_ALRDL_noU_v16a.9m

                        ERROR SUMMARY

 FLOATING.psub : Floating psub is not allowed
   or ................................................. 276 violations found.




                        ERROR DETAILS


------------------------------------------------------------------------
FLOATING.psub : Floating psub is not allowed
------------------------------------------------------------------------
```

Nota. La imagen muestra una captura de pantalla del archivo `IO_nanochip.LAYOUT_ERRORS` generado después de correr la verificación ERC. `Elaboración propia.`

11.8. Análisis de resultados

Como se puede observar en la sección anterior, los resultados obtenidos para los 3 circuitos diferentes resaltan un mismo tipo de error, por lo que sólo se realizará un análisis más a fondo de los archivos resultantes de la prueba correspondiente a “El Gran Jaguar”.

11.8.1. `IO_nanochip.RESULTS`

El archivo `IO_nanochip.RESULTS` confirma que la comparación LVS pasó (*“LVS Compare Results: PASS”*), mientras que la etapa de DRC/extracción no está limpia (*“DRC and Extraction Results: NOT CLEAN”*), lo cual era un resultado esperado, ya que por cuestiones de tiempo a las personas encargadas de llevar a cabo la prueba DRC no les fue posible corregir todos los errores físicos del *layout* antes de correr la prueba LVS. En conclusión, la conectividad lógica de alto nivel (equivalencias por celda) entre *layout* y esquemático es consistente bajo el enfoque cajas negras, pero el *layout* presenta problemas físicos que afectan la extracción/RC y la confiabilidad. Para un futuro, se recomienda mantener el flujo LVS, y corregir los errores físicos que impiden una extracción limpia; tras corregirlos, ejecutar nuevamente las pruebas DRC y LVS para verificar que el estado global quede limpio sin cambiar el modo de comparación.

11.8.2. `IO_nanochip.LAYOUT_ERRORS`

El archivo `IO_nanochip.LAYOUT_ERRORS` reporta la existencia de errores, cuya principal causa se puede observar en el *“ERROR SUMMARY”*, ya el error `FLOATING.psub : Floating psub is not allowed` se repite un total de 276 veces. Este error indica que existen regiones de *p-substrate* sin contacto a tierra que, aunque no afectan la coincidencia lógica del LVS, sí degradan la extracción y la confiabilidad del circuito (riesgo de latch-up, acoples y variaciones de umbral), por lo que el estado global queda “NOT CLEAN”.

11.9. Trabajo Futuro

Se recomienda establecer un procedimiento sistemático para identificar y corregir de forma temprana cualquier ocurrencia del error `FLOATING.psub`, con el fin de preservar la integridad eléctrica y la confiabilidad a largo plazo del circuito integrado.

En primera instancia, se sugiere utilizar los reportes de ERC para localizar con precisión las regiones del *layout* en las que el substrato tipo P se encuentra sin una ruta hacia tierra, o red de referencia. Una vez identificadas estas áreas, debe verificarse si la causa está asociada a:

- Ausencia o escasez de *p-taps* (contactos de substrato) en el entorno inmediato.
- Distancias excesivas entre *taps* consecutivos, violando la separación máxima definida por la tecnología.

- Falta de conexión eléctrica a la red VSS/GND, ya sea por ausencia de ruteo metálico o por una etiqueta incorrecta.
- Eliminación de celdas de relleno (PFILLER, PCORNER, FILLER) que, en algunas librerías, aportan contactos al substrato. Si este llegara a ser el caso, será necesario encontrar otra manera para tratar estas celdas, de manera que no interfieran durante la prueba LVS.
- Problemas de mapeo de redes en el runset de LVS/ERC, donde la red de tierra física no se reconoce como GND/VSS en las reglas de verificación.

Estas son sólo algunas sugerencias sobre las posibles causas del error, sin asegurar que una de dichas sugerencias sea la causa definitiva. Puede ser que para solucionarlo sea necesario investigar más a fondo su orígen, o cómo se ha solucionado en otros flujos de trabajo similares al que se ha establecido para trabajar en “El Gran Jaguar”.

CAPÍTULO 12

Conclusiones

- Se estableció y ejecutó con éxito un flujo completo de las verificaciones *Layout Versus Schematic* (LVS) y *Electrical Rule Check* (ERC) para el circuito integrado "El Gran Jaguar" en tecnología de 65 nm, utilizando herramientas profesionales de Synopsys.

- La investigación y el análisis de la verificación LVS permitió comprender de manera profunda la relación entre el diseño lógico y su implementación física, evidenciando cómo pequeñas discrepancias en *netlists*, jerarquías o mapeo de puertos pueden afectar directamente la integridad eléctrica del nanochip. La interpretación estructurada de los reportes permitió distinguir errores críticos de diferencias no significativas, fortaleciendo la capacidad de depuración del diseño.

- El estudio y proceso de análisis de la verificación ERC puso en evidencia la importancia de evaluar sistemáticamente la confiabilidad eléctrica de un circuito integrado, más allá de su funcionalidad lógica. Al examinar casos como nodos flotantes, pozos mal conectados o compuertas ligadas a potencia o tierra, fue posible comprender cómo estas condiciones pueden comprometer la estabilidad del chip a nivel físico y operativo.

- La verificación LVS confirmó la equivalencia eléctrica entre el *layout* y el esquemático del nanochip, demostrando que las interconexiones del diseño fueron implementadas correctamente.

- La prueba ERC permitió identificar la existencia de sustratos tipo P flotantes dentro del circuito, un aspecto crítico que debe ser corregidos para garantizar la integridad eléctrica del diseño final, asegurando su correcto funcionamiento y fiabilidad durante la operación.

- La necesidad de aplicar un enfoque de *black box* a las verificaciones permitió reflexionar sobre las estrategias utilizadas en la industria para verificar celdas estándar cuyos detalles internos son propiedad intelectual. Este análisis permitió afianzar la importancia de realizar las verificaciones a nivel de interconexión de puertos, asegurando la integridad del diseño sin comprometer la confidencialidad de las celdas.

- La creación y prueba en FPGA de un subconjunto de celdas permitió verificar funcionalmente la etapa de diseño lógico, reforzando la confiabilidad del diseño final.
- El análisis combinado de LVS y ERC permitió integrar la visión lógica, física y eléctrica del diseño, consolidando una perspectiva completa del ciclo de verificación. Este entendimiento integral es fundamental para futuros proyectos en los que el diseño y la verificación dependerán de la toma de decisiones informadas y basadas en evidencia técnica.

CAPÍTULO 13

Recomendaciones

- Se sugiere documentar y estandarizar el flujo de las verificaciones LVS y ERC desarrollado, para facilitar su adopción en futuros proyectos de diseño de circuitos integrados dentro de la Universidad del Valle de Guatemala.
- Ampliar la documentación técnica y crear material didáctico adicional basado en este flujo para uso en cursos de electrónica y diseño VLSI.
- En futuros trabajos, se podría investigar la optimización o incluso la automatización del proceso de llevar a cabo ambas pruebas de verificación, para reducir los tiempos de ejecución y mejorar la eficiencia del flujo de trabajo ya establecido.
- Se recomienda explorar la integración de otras herramientas EDA que puedan complementar o mejorar las capacidades para llevar a cabo las verificaciones LVS y ERC utilizadas en este trabajo, evaluando su compatibilidad con el flujo actual.
- Se recomienda identificar el origen del error `FLOATING.psub` revisando las zonas del *layout* donde el substrato P carece de conexión a VSS mediante verificar la presencia y densidad adecuada de *p-taps*, la correcta propagación de la *net* de tierra, buscar diferentes maneras de tratar de las celdas de relleno, entre otros aspectos. Una vez detectada la causa, debe corregirse para asegurar la integridad eléctrica del chip y prevenir fallas como variaciones de umbral o posibles eventos de *latch-up*.
- Finalmente, se recomienda fomentar colaboraciones con otras instituciones académicas y la industria local, con el objetivo de fortalecer las capacidades en diseño y verificación de circuitos integrados a nivel nacional.

CAPÍTULO 14

Referencias

[1] J. A. D. los Santos Chonay, «Diseño de un sumador/restador completo de 32 bits con tecnología CMOS en un proceso de 28 nanómetros usando aplicaciones de diseño de la empresa Synopsys,» 2014.

[2] L. A. N. Vásquez, «Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado,» 2019.

[3] S. H. R. Vasquez, «Definición del Flujo de Diseño para Fabricación de un Chip con Tecnología VLSI CMOS,» 2019.

[4] M. S. Illescas, «Verificación de reglas de diseño (DRC) para el desarrollo de un flujo funcional de un circuito integrado con tecnología nanométrica,» 2020.

[5] J. R. G. Rubio, «Etapa de verificación física de Diseño en Silicio vs. Esquemático (LVS) en el flujo de diseño para un chip a nanoescala,» 2020.

[6] M. G. F. Espino, «Corrección de anillo de entradas/salidas y pruebas de antenna y ERC para la definición del flujo de diseño del primer chip con tecnología nanométrica desarrollado en Guatemala,» 2020.

[7] K. S. C. Polanco, «Mejoramiento del proceso de síntesis lógica llevada a cabo para la elaboración de un circuito integrado a escala nanométrica,» 2021.

[8] E. O. T. Garza, «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,» 2022.

[9] C. A. L. Torres, «Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: uso avanzado de StarRC para la generación de un archivo HSPICE con componentes par sitos para su correcta simulación,» 2022.

[10] E. G. M. Ruballos, «Implementación y verificación de la funcionalidad de código obtenido durante la síntesis lógica de arquitectura del nano chip Gran Jaguar utilizando una plataforma de desarrollo con un FPGA Xilinx Virtex-5 Genesys,» 2022.

[11] L. R. G. Velasquez, «Procesamiento de las señales generadas por un circuito integrado con tecnología de 180nm usando librerías de diseño de TSMC montado en un FPGA Digilent Genesys Board demostrando el funcionamiento mediante una aplicación y sintetizador digital,» 2022.

[12] D. S. A. Mota, «Diseño de un circuito integrado con tecnología de 65 nm utilizando librerías de diseño de TSMC: Pruebas de LVS, ERC y extracción de parásitos,» 2024.

[13] A. A. Raj y T. Latha, *VLSI Design*. PHI Learning, oct. de 2008, ISBN: 9788120334311. dirección: https://books.google.com.gt/books?id=CO8zq6_vcr8C.

[14] L. Suresh, *Why is VLSI Used? A Comprehensive Exploration*, ene. de 2025. dirección: https://www.guvi.in/blog/why-is-vlsi-used/.

[15] S. Saurabh, *VLSI Design Flow: RTL to GDS*, 2023. dirección: https://onlinecourses.nptel.ac.in/noc23_ee137/preview#:~:text=This%20course%20covers%20the%20entire%20RTL%20to%20GDS,of%20logic%20synthesis,%20verification,%20physical%20design,%20and%20testing..

[16] S. Inc., *Synopsys Introduces Breakthrough Fusion Technology to Transform the RTL-to-GDSII Flow*, mar. de 2018. dirección: https://news.synopsys.com/2018-03-19-Synopsys-Introduces-Breakthrough-Fusion-Technology-to-Transform-the-RTL-to-GDSII-Flow.

[17] S. Inc., *Synopsys IC Compiler II*. dirección: https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html.

[18] S. Inc., *IC Validator Physical Verification*, 2019. dirección: https://www.synopsys.com/content/dam/synopsys/implementation&signoff/datasheets/ic-validator-ds.pdf.

[19] S. Inc., *07 - How to perform netlist translation & modification using IC Validator Net-Tran utility*. dirección: https://video.synopsys.com/icvalidator/detail/video/5850492853001/07---how-to-perform-netlist-translation-modification-using-ic-validator-nettran-utility.

[20] S. Inc., *What is Layout Versus Schematic Checking?* Dirección: https://www.synopsys.com/glossary/what-is-layout-versus-schematic-checking.html.

[21] S. Inc., *Equivalence File for LVS Run*. dirección: https://www.synopsys.com/implementation-and-signoff/resources/videos/lvs-four.html.

[22] TSMC, *N65/N55 ICV LVS/LPE Deck Usage*.

[23] S. Velivala, *The Evolving Role of Layout-Versus-Schematic (LVS) Checking for Modern SoCs*, abr. de 2024. dirección: https://www.synopsys.com/blogs/chip-design/lvs-checking-process.html.

[24] Z. EDA, *Physical Verification Tools | DRC ERC LVS*, 2011. dirección: https://www.zeni-eda.com/veri.html.

[25] S. Inc., *LVS Black Box Flow in IC Validator*. dirección: https://www.synopsys.com/implementation-and-signoff/resources/videos/lvs-six.html.

[26] S. Inc., *Creating a Black-Box LVS Flow*, 2025. dirección: https://spdocs.synopsys.com/dow_retrieve/qsc-x/dg/icvolh/X-2025.06/icvolh/icvlvsug/layout_versus_schematic_flows/creating_black_box_lvs_flow.html.

[27] TSMC, *ERC Usage*, 2013.

[28] SuccessBridge, *ERC in VLSI: Ensuring Electrical Integrity in Chip Design*, nov. de 2024. dirección: https://www.successbridge.co.in/understanding-erc-in-vlsi/.

[29] S. Inc., *What is Programmable Electrical Rules Checking?* Dirección: https://www.synopsys.com/glossary/what-is-programmable-electrical-rules-checking.html.

[30] S. Inc., *IC Validator User Guide*, jun. de 2025. dirección: https://spdocs.synopsys.com/dow_retrieve/qsc-x/dg/icvolh/X-2025.06/icvolh/pdf/icvug1.pdf.

CAPÍTULO 15

Glosario

Anillo de entradas y salidas: región periférica del chip dedicada a las celdas de entrada y salida (*I/O ring*), encargada de la interfaz eléctrica entre el núcleo lógico del circuito integrado y el mundo externo (pads, encapsulado, placas de prueba, etc.).

Black Box LVS: técnica de verificación LVS que trata ciertas celdas o bloques como "cajas negras", verificando únicamente la correcta interconexión de sus puertos externos sin inspeccionar su implementación interna.

Caja negra: modelado abstracto de una celda o bloque cuyo comportamiento interno se omite en la verificación; solo se consideran sus puertos externos y su conectividad. En un LVS tipo *black box* se verifica únicamente la correcta conexión de estos puertos sin requerir el *back-end* de la celda.

CDL: *Circuit Description Language*. Formato de netlist compatible con herramientas de verificación como IC Validator, usualmente derivado de descripciones SPICE y empleado como referencia para la prueba LVS.

Celdas estándar: bloques de lógica predefinidos (por ejemplo, compuertas NAND, NOR, biestables, etc.) caracterizados y proporcionados por la biblioteca tecnológica, utilizados como ladrillos básicos para implementar el diseño lógico de un circuito integrado.

Deck de reglas: conjunto de reglas tecnológicas proporcionadas por la fundición (por ejemplo, TSMC) que define las condiciones geométricas y eléctricas que el diseño debe cumplir para ser manufacturable.

Digilent Genesys: plataforma FPGA utilizada como entorno de verificación funcional del nanochip, permitiendo validar en hardware el comportamiento de las celdas descritas en Verilog antes de su integración en el diseño final.

DRC: *Design Rule Check*. Verificación que asegura que la geometría del *layout* cumple con las reglas de fabricación establecidas por la fundición, tales como espaciamientos mínimos, anchos de capa y superposiciones permitidas.

EOS: *Electrical OverStress* (Sobreesfuerzo Eléctrico). Condición que ocurre cuando un dispositivo semiconductor se somete a una corriente o voltaje que excede sus límites operativos máximos.

ERC: *Electrical Rule Check*. Verificación eléctrica que comprueba el cumplimiento de reglas relacionadas con la polarización de pozos, la existencia de nodos flotantes, las conexiones hacia potencia y tierra, y condiciones de seguridad eléctrica contra fenómenos como ESD o *latch-up*.

ESD: *Electrostatic Discharge*. Descarga electrostática que puede dañar estructuras internas del circuito integrado. El diseño debe incluir mecanismos de protección para evitar fallas permanentes asociadas a estos eventos.

FinFET: tipo de estructura de transistor MOSFET tridimensional (3D) utilizada en tecnologías de nodo avanzado (como 16 nm o inferiores) para mejorar el control del canal y reducir la fuga de corriente.

FPGA: *Field-Programmable Gate Array*. Dispositivo lógico programable que permite implementar y probar hardware digital de forma reconfigurable. En este trabajo se utiliza para verificar funcionalmente una fracción de las celdas estándar del diseño.

GDSII: *Graphic Data System II*. Formato estándar de archivo que almacena la geometría del *layout* de un circuito integrado, utilizado como salida final del flujo físico y como entrada para la fabricación de las máscaras fotolitográficas.

IC Compiler II: herramienta de implementación física de Synopsys que integra las etapas de *floorplanning*, colocación, ruteo, cierre de temporización y generación del *layout* final, optimizando simultáneamente potencia, rendimiento y área.

IC Validator: herramienta de verificación física de Synopsys que ejecuta DRC, LVS, ERC y otras verificaciones en diseños de gran escala, con capacidad de paralelización y estrecha integración con IC Compiler II dentro del flujo de diseño físico.

IMEC: *Interuniversity Microelectronics Centre*. Centro de investigación en microelectrónica con sede en Bélgica que actúa como organismo intermediario entre universidades y fundiciones de semiconductores, facilitando el acceso a tecnologías avanzadas como las librerías y *decks* de TSMC utilizados en este trabajo.

Latch-up: fenómeno indeseado en circuitos CMOS en el que se establece un camino de baja impedancia entre los rieles de alimentación debido a estructuras parasitarias internas, pudiendo causar corrientes excesivas y la destrucción del chip si no se mitiga adecuadamente.

Layout: representación física o geométrica de un circuito integrado, que define la disposición espacial de los componentes, las interconexiones y las diferentes capas que conforman un chip.

LVS: *Layout Versus Schematic*. Verificación que compara el diseño físico (*layout*) con el esquemático o *netlist* de referencia para asegurar que son eléctricamente equivalentes. Un resultado *LVS clean* indica coincidencia entre ambas representaciones.

Multifinger: transistor MOS ancho que, para optimizar rendimiento y área, se divide en varios transistores paralelos más pequeños que comparten las terminales de fuente y drenaje. Esto equivale a tener múltiples "*fingers*" o dedos de compuerta.

Nanochip: circuito integrado fabricado en una tecnología nanométrica (por ejemplo, 65 nm), en el cual las dimensiones características de los dispositivos se encuentran en el orden de decenas de nanómetros.

Netlist: descripción textual de un circuito que especifica los dispositivos electrónicos (celdas, transistores, resistencias, etc.) y las redes (nodos) que los interconectan. Puede expresarse en lenguajes como SPICE, Verilog o el formato ICV utilizado por herramientas de verificación.

NetTran: herramienta de Synopsys para convertir netlists en formatos estándar (SPICE, Verilog, etc.) al formato interno utilizado por IC Validator, y para fusionar diversos netlists en un único archivo de nivel superior que se emplea en las verificaciones.

RTL-to-GDSII: flujo de diseño digital que abarca desde la descripción a nivel RTL hasta la generación del archivo GDSII listo para fabricación. Incluye etapas de síntesis lógica, implementación física (*floorplanning*, *placement*, *routing*), cierre de temporización y verificaciones físicas como DRC, LVS y ERC.

RTL: *Register Transfer Level.* Nivel de abstracción en el que se describe el comportamiento de un circuito digital en términos de registros, operaciones lógicas y flujo de datos entre ellos. Es el punto de partida del flujo de diseño *RTL-to-GDSII*.

Runset: archivo de configuración que contiene las reglas, parámetros y secuencias de verificación utilizadas por herramientas como IC Validator para ejecutar pruebas DRC, LVS y ERC en una tecnología específica.

Single-gate: transistor MOS estándar con una única vía continua entre las terminales de fuente y drenaje (típicamente modelado con `nf = 1`).

Standard Cell Library: conjunto de celdas lógicas pre-diseñadas y caracterizadas (compuertas, *flip-flops*, etc.) que se utilizan como bloques básicos en el diseño digital *semi-custom*.

Síntesis Física: proceso de implementación física que incluye la colocación (*placement*) de las celdas y el enrutamiento (*routing*) de sus interconexiones dentro del área del chip.

Síntesis Lógica: proceso de transformar una descripción RTL de un diseño en una *netlist* a nivel de compuertas, utilizando celdas de una biblioteca tecnológica específica.

TSMC: *Taiwan Semiconductor Manufacturing Company.* Fundición de semiconductores que proporciona las librerías tecnológicas y los *decks* de reglas de 65 nm empleados en el diseño y verificación del proyecto *El Gran Jaguar*.

VLSI: *Very-Large-Scale Integration.* Metodología de diseño que permite integrar una gran cantidad de transistores y componentes electrónicos en un único chip de silicio, habilitando circuitos integrados de alta complejidad y desempeño en tecnologías nanométricas.