UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación de interfaz gráfica de la automatización de las fases de diseño de un circuito integrado y uso avanzado de *IC Compiler II.*

Trabajo de graduación presentado por Diego Rodrigo Equité Pirir para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación de interfaz gráfica de la automatización de las fases de diseño de un circuito integrado y uso avanzado de *IC Compiler II.*

Trabajo de graduación presentado por Diego Rodrigo Equité Pirir para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

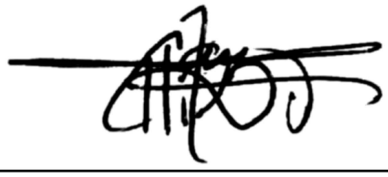
2022

Vo.Bo.:

(f) _____

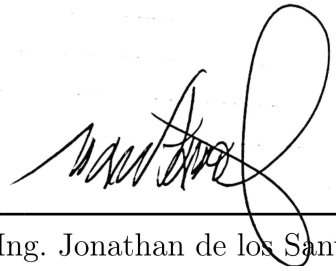
MSc. Carlos Esquit

Tribunal Examinador:



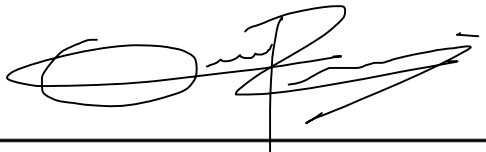
(f) _____

MSc. Carlos Esquit



(f) _____

Ing. Jonathan de los Santos



(f) _____

Ing. Ricardo Girón

Fecha de aprobación: Guatemala, 8 de diciembre de 2022.

Prefacio

Este trabajo está dedicado y en agradecimiento a Dios, mi familia, amigos y la Fundación Juan Bautista Gutiérrez por acompañarme en este camino al cumplir uno de mis sueños de graduarme de ingeniero electrónico.

Índice

Lista de figuras

Lista de cuadros

Resumen

Este trabajo se basa en optimizar el proceso de automatización para la síntesis física además de optimizar el *script* ya existente para la generación del HDL. Esto se logra mejorando el proceso iniciado por las iteraciones pasadas de estas etapas del desarrollo de un IC.

Dado que ya se han realizado iteraciones en años pasados, se realizó una réplica de los trabajos anteriores con el fin de analizar de los resultados y utilizarlos como base para optimizar la síntesis física. Para realizar este trabajo se utilizó un *script* basado en bash, el cual utiliza la herramienta *IC Compiler II* para realizar la síntesis física, este proceso se probó con circuitos como: *not*, ALU y el chip de 180nm.

Para optimizar el *script* ya existente para la generación del HDL se realizó una actualización a *python 3* y luego se optimizó, este HDL es la base para la realización de la síntesis lógica. El proceso para realizar el chip esta dividido por lo que se realizó un interfaz gráfica que una todas las fases, esta son síntesis lógica, síntesis física y verificaciones. Estas fases se integraron en la automatización del flujo, para esta integración se verificó que los archivos generados de la fase previa funcionen para la fase para siguiente.

En la fase de síntesis física se encontraron errores, los cuales se solucionaron, entre estos errores se destacan: reglas de diseño y densidad de metales, para las reglas de diseño se añadió las reglas faltantes con comandos más específicos además de corregir algunas reglas. Para corregir los errores de densidad de metales, se exploraron varios métodos, de los cuales destaca el uso del comando *signoff* para el relleno de metal. En este método se realizó la conversión de un *runset* de calibre a *runset* válido para las herramientas de calibre, este proceso de relleno de metal se hizo de forma iterativa hasta corregir los errores.

Abstract

This work is based on optimizing the automation process for the physical synthesis as well as optimizing the existing script for the generation of the HDL.This is achieved by improving the process started by past iterations of these stages of IC development.

Since iterations have already been carried out in past years, a replica of the previous works was carried out in order to analyze the results and use them as a basis to optimize the physical synthesis.To carry out this work, a bash-based script was used, which uses the IC Compiler II tool to perform the physical synthesis, this process was tested with circuits such as: not, ALU and the 180nm chip.

To optimize the existing script for the generation of the HDL, an update to python 3 was made and then it was optimized, this HDL is the base for the realization of the logical synthesis.The process to make the chip is divided so a graphical interface was made that unites all the phases, these are logical synthesis, physical synthesis and verifications. These phases were integrated in the flow automation, for this integration it was verified that the files generated from the previous phase work for the next phase.

In the physical synthesis phase errors were found, which were solved, among these errors are: design rules and metal density, for the design rules the missing rules were added with more specific commands in addition to correcting some rules. To correct the metal density errors, several methods were explored, the most important of which is to use the signoff command for metal filling, in this method the conversion of a gauge runset to a valid runset for the gauge tools was performed, this metal filling process was done iteratively until the errors were fixed.

CAPÍTULO 1

Introducción

En la actualidad, la nanoelectrónica ha tenido relevancia y avances significativos, dado estos avances es necesario automatizar los procesos de diseño que conocemos, sobre todo en un circuito integrado, dado que este es la pieza fundamental para muchos de los productos tecnológicos.

En la Universidad del Valle de Guatemala en los últimos años se ha incursionado en el proceso de diseño de un chip. Para diseñar un chip ya se estableció un flujo de diseño. El proceso de automatización está dividido en distintas fases: síntesis lógica, síntesis física y verificaciones.

Para validar el proceso de automatización es necesario realizar varias iteraciones con el fin de encontrar la mayor cantidad de errores y corregirlos. En los capítulos siguientes se optimiza y automatiza este flujo específicamente en la fase de síntesis física, utilizando la herramienta *IC Compiler II* , además que se implementa una interfaz gráfica con el fin de expandir el diseño de chips en otras ramas de la ciencia.

CAPÍTULO 2

Antecedentes

Los primeros avances realizados en el campo de la nanoelectrónica en la Universidad del Valle de Guatemala se dieron en el año 2013 con el curso introductorio VLSI, un año después se creó una alianza con la empresa Synopsys donde se obtuvieron las herramientas necesarias para avanzar en el campo de la nanoelectrónica, esto se comprueba en el primer trabajo de graduación relacionado con VLSI [1], este trabajo ha servido como base para los trabajos posteriores hasta el día de hoy.

En los últimos años se añadió los cursos de nanoelectrónica 1 y 2, estos cursos son la base para continuar con la investigación en el campo de VLSI, se logró un acuerdo con *Inter-university Microelectronics Centre (IMEC)* para la manufactura del chip, con este acuerdo se desarrollaron los proyectos [2] y [3].

Durante los años 2020 al 2022 se han dado avances significativos al chip, en el año 2020, Luis Abadía [4], Marvin Flores [5] y Matthias Sibrian [6] obtuvieron resultados con simulaciones funcionales utilizando las librerías de TSMC en los programas *IC Validator* y *IC Compiler I*. En el año 2021, Elmer Torres[7] realizó y verificó la etapa de síntesis lógica en el flujo de diseño, en los trabajos de Antonio Altuna[8], José Ayala [9] y Julio Shin[10] se desarrolló la ejecución de la síntesis física, verificaciones de Antena y corrección de errores obtenidos. Durante estos trabajos hubo un cambio de *IC Compiler I* a *IC Compiler II*, este cambio produjo atrasos a los avances dado que se dieron cambios significativos en los programas, por lo que se encontraron errores los cuales no se pudieron solucionar, estos errores son debido a la densidad en la capa de los metales.

En el año 2022 se replicó todos los avances de los años anteriores, estos son: la generación del verilog, síntesis lógica, síntesis física, comprobaciones físicas; DRC, ERC, Antena, LVS y extracción de parásitos. Luego de realizar la síntesis lógica se procedió a realizar la síntesis física como se muestra en la Figura 6, se obtuvo como resultado el nanochip que se muestra en la Figura 7 y el *core* que se muestra en la Figura 8, luego se procedió con las verificaciones donde se obtuvo los resultados de LVS que se muestran en la Figura 1, para DRC se obtuvo los mismos errores de densidad de metal que se muestran en la Figura 3, además de errores de *black boxes*, los errores de densidad aún no se solucionan. Los resultados de Antena se

muestran en la Figura 2, para realizar la verificación de extracción de parásitos se necesita de un *deck* generado en HSPICE que se muestra en la Figura 5, con este *deck* se realizó la extracción de parásitos que se muestra en la Figura 4.

Figura 1: Resultado verificación LVS.

```
                                RESULTS: CLEAN

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

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


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


                                    IC Validator

            Version S-2021.06-SP3-2 for linux64 - Jan 06, 2022 cl#7200131

                        Copyright (c) 1996 - 2022 Synopsys, Inc.
       This software and the associated documentation are proprietary to Synopsys,
     Inc. This software may only be used in accordance with the terms and conditions
     of a written license agreement with Synopsys, Inc. All other use, reproduction,
       or distribution of this software is strictly prohibited.  Licensed Products
         communicate with Synopsys servers for the purpose of providing software
        updates, detecting software piracy and verifying that customers are using
        Licensed Products in conformity with the applicable License Key for such
      Licensed Products. Synopsys will use information gathered in connection with
        this process to deliver software updates and pursue software pirates and
                                      infringers.

     Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
                Inclusivity and Diversity" (Refer to article 000036315 at
                              https://solvnetplus.synopsys.com)

Called as: icv -i chip.gds -c chip_IO -sf ICV -vue ICVLM18_LM16_LM152_6M.ANT.215a_pre041518

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

Input file name:         chip.gds
Top cell name:           chip_IO
Layout format:           GDSII
Runset file:             ICVLM18_LM16_LM152_6M.ANT.215a_pre041518
Working directory:       /home/nanoelectronica/Desktop/Trabajos/Antena
Run details directory:   /home/nanoelectronica/Desktop/Trabajos/Antena/run_details
Layout error file:       chip_IO.LAYOUT_ERRORS
User name:               nanoelectronica
Time started:            2022/02/18 04:05:19PM
Time ended:              2022/02/18 04:06:06PM
```

Figura 2: Resultado verificación antena.

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

Rule and DRC Error Summary

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

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

Rule                            Violations Found
M1.R.1                          v = 1
M2.R.1                          v = 1
M3.R.1                          v = 1
M4.R.1                          v = 1
M5.R.1                          v = 1
M6.R.1                          v = 1
```

Figura 3: Errores de densidad DRC.

```
                                StarRC (TM)

              Version S-2021.06-SP4 for linux64 - Nov 18, 2021

                    Copyright (c) 1999 - 2021 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
   or distribution of this software is strictly prohibited.  Licensed Products
     communicate with Synopsys servers for the purpose of providing software
    updates, detecting software piracy and verifying that customers are using
    Licensed Products in conformity with the applicable License Key for such
  Licensed Products. Synopsys will use information gathered in connection with
    this process to deliver software updates and pursue software pirates and
                                  infringers.

 Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
            Inclusivity and Diversity" (Refer to article 000036315 at
                        https://solvnetplus.synopsys.com)

ExecName: /usr/synopsys/starrc/S-2021.06-SP4/linux64_starrc/bin/StarXtract
Version: S-2021.06-SP4
Built on: Nov 18 2021 18:35:04
Start Time: Fri Mar 11 01:50:08 2022

Host ............ uvgiemtbmcit11805
```

Figura 4: Verificación extracción de parásitos.

```
*|NET ln_N_10 0.0450721PF
*|I (ln_N_10:F1 chip/U212 ln_N_4 B 0 238.9775 216.8400)
*|I (ln_N_10:F2 chip/U203 ln_N_4 B 0 244.5750 248.2000)
*|I (ln_N_10:F3 chip/U220 ln_N_7 B 0 246.5200 244.6200)
*|I (ln_N_10:F4 chip/U393 ln_N_5 B 0 281.5050 295.2400)
*|I (ln_N_10:F5 chip/U344 ln_N_4 B 0 258.0150 212.9200)
*|I (ln_N_10:F6 chip/U164 ln_N_4 B 0 285.1325 200.8875)
*|I (ln_N_10:F7 chip/U40 ln_N_6 B 0 248.4650 240.3600)
*|I (ln_N_10:F8 chip/U492 ln_N_5 B 0 248.4800 286.7250)
*|I (ln_N_10:F9 chip/U565 ln_N_7 B 0 252.9600 291.8800)
Cg2_1 ln_N_10:F1 0 1.73697e-16
Cg2_2 ln_N_10:F2 0 3.45924e-16
Cg2_3 ln_N_10:F3 0 1.24615e-16
Cg2_4 ln_N_10:F4 0 7.63994e-16
Cg2_5 ln_N_10:F5 0 2.50804e-16
Cg2_6 ln_N_10:F8 0 3.61245e-16
Cg2_7 ln_N_10:F9 0 2.56219e-16
Cg2_8 ln_N_10:10 0 1.86555e-15
Cg2_9 ln_N_10:11 0 1.03303e-15
Cg2_10 ln_N_10:12 0 3.95561e-15
Cg2_11 ln_N_10:13 0 1.53536e-15
Cg2_12 ln_N_10:14 0 2.26442e-15
Cg2_13 ln_N_10:15 0 7.2509e-16
Cg2_14 ln_N_10:16 0 4.0117e-16
Cg2_15 ln_N_10:18 0 1.71717e-16
Cg2_16 ln_N_10:19 0 4.98499e-17
Cg2_17 ln_N_10:20 0 2.958e-15
R2_1 ln_N_10:F1 ln_N_10:20 28.2297
R2_2 ln_N_10:F2 ln_N_10:12 13.658
R2_3 ln_N_10:F3 ln_N_10:13 12.8613
R2_4 ln_N_10:F4 ln_N_10:14 46.8784
R2_5 ln_N_10:F5 ln_N_10:15 19.746
R2_6 ln_N_10:F6 ln_N_10:11 0.015638
R2_7 ln_N_10:F7 ln_N_10:10 6.4
R2_8 ln_N_10:F8 ln_N_10:16 20.058
R2_9 ln_N_10:F9 ln_N_10:18 0.001
R2_10 ln_N_10:F9 ln_N_10:14 14.0863
R2_11 ln_N_10:10 ln_N_10:20 25.3035
R2_12 ln_N_10:10 ln_N_10:13 8.09027
R2_13 ln_N_10:11 ln_N_10:19 0.00227354
R2_14 ln_N_10:11 ln_N_10:15 42.452
R2_15 ln_N_10:12 ln_N_10:13 14.828
R2_16 ln_N_10:12 ln_N_10:16 26.3012
R2_17 ln_N_10:14 ln_N_10:16 22.788
R2_18 ln_N_10:15 ln_N_10:20 10.7123
R2_19 ln_N_10:17 ln_N_10:18 0.001

*|NET ln_N_100 0.0104824PF
*|I (ln_N_100:F1 chip/U25 ln_N_3 B 0 256.3450 205.0800)
*|I (ln_N_100:F2 chip/U2/U1 ln_N_4 B 0 235.3200 224.6800)
```

Figura 5: Archivo generado de HSPICE.

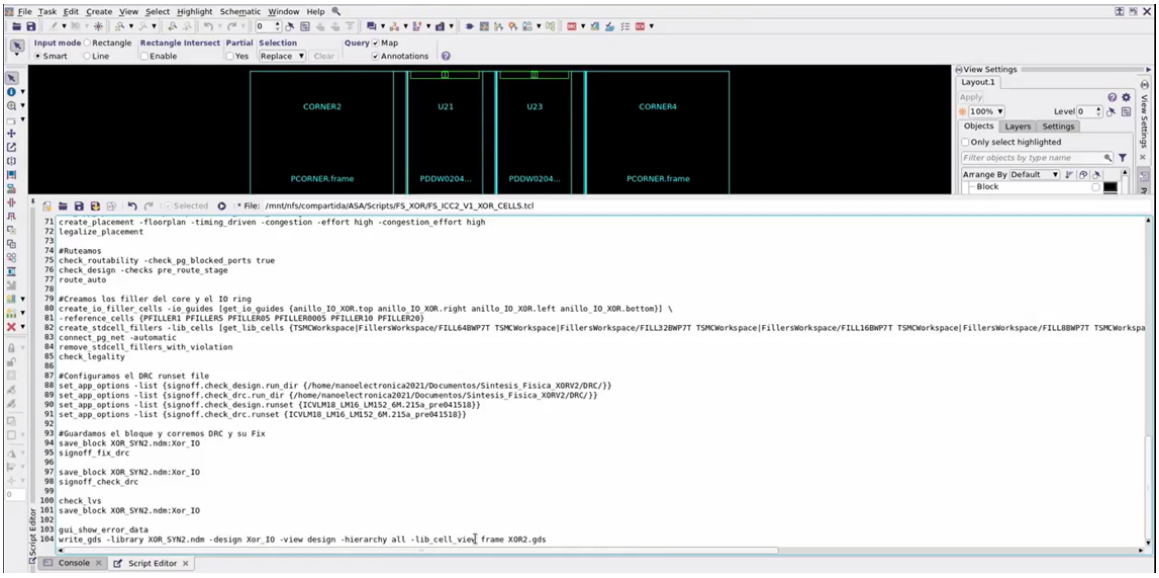


Figura 6: Ejecución síntesis física compuerta XOR.

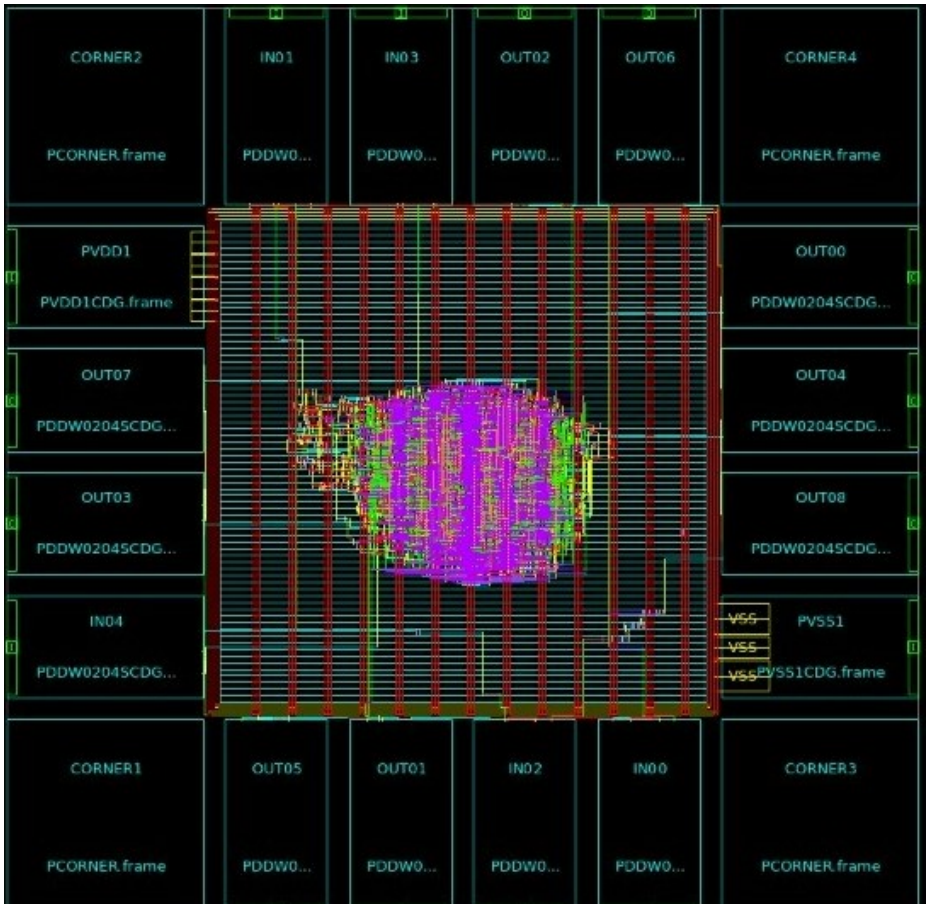


Figura 7: Síntesis física nanochip

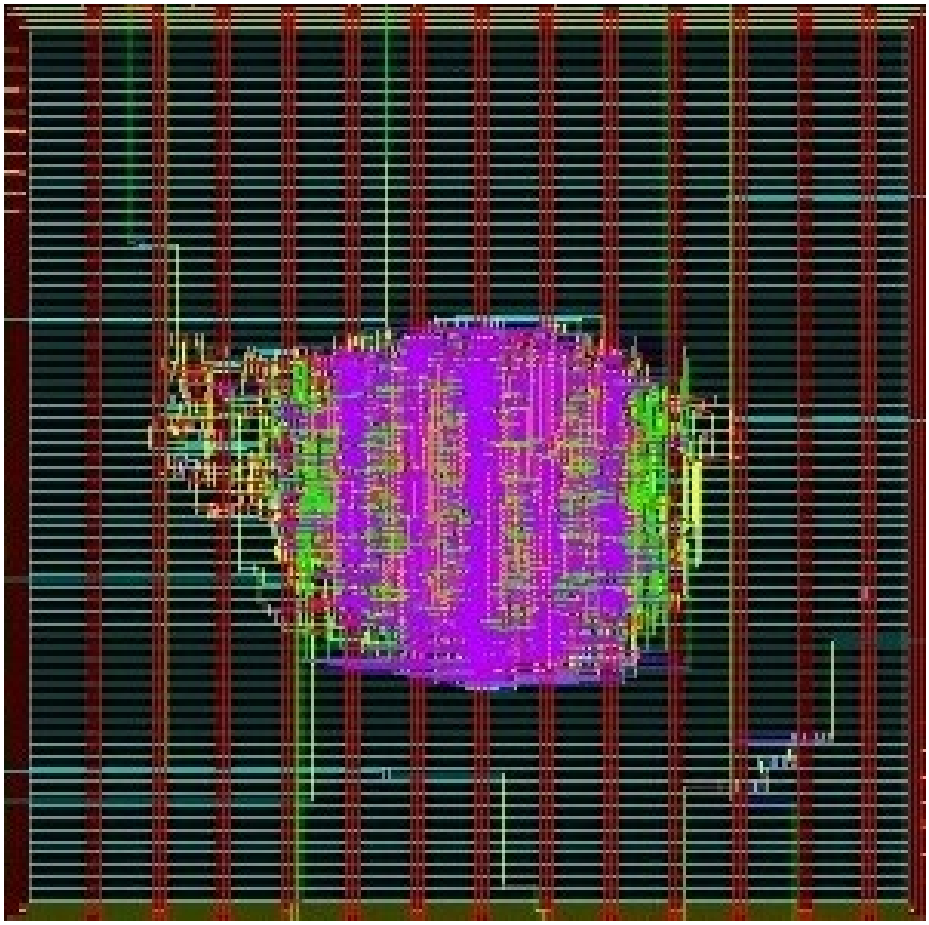


Figura 8: Core nanochip

CAPÍTULO 3

Justificación

La tecnología ha tenido un avance importante durante los últimos años, sobre todo en el campo de la nanoelectrónica donde cada vez los circuitos son más complejos. El desarrollo de un chip es el proceso más complicado que realiza el ser humano por lo que es necesario que se tengan distintos conocimientos para poder desarrollarlo. Uno de los grandes avances en los últimos años es tener un flujo de diseño funcional para desarrollar chips, existen varios flujos de diseño dependiendo la complejidad que se busca en el chip, en este trabajo se busca optimizar el flujo de diseño desarrollando una mejor manera de crear el HDL y automatizar el flujo de síntesis física.

Se busca automatizar este proceso dado que así se reducen costos, aumenta la productividad, el rendimiento, además con esto personas de otras ciencias podrán utilizar las herramientas para crear sus propios chips según sus necesidades sin conocer tanto el proceso de creación, esto abre las puertas para desarrollar más tecnología en Guatemala, impulsando así el desarrollo de la sociedad guatemalteca. Con la optimización del archivo se establece una base sólida para la creación del chip, impulsando a los estudiantes de la Universidad del Valle de Guatemala a seguir con el proceso de la creación del chip, es tanto una herramienta para fabricar como para material didáctico.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar los *scripts* necesarios utilizando *python* y *bash* para optimizar la automatización del flujo de diseño del chip de 180 nm así como desarrollar una interfaz gráfica, además aprender de modo avanzado *IC Compiler II*.

4.2. Objetivos específicos

- Optimizar un *script* que genere un verilog para cualquier texto y actualizar el *script* de *python* 2 a *python* 3.
- Optimizar el *script* existente para la automatización del proceso de síntesis física.
- Generar los archivos necesarios para realizar las verificaciones físicas como: DRC, ERC, LVS, Antena y extracción de parásitos, estos archivos son las extensiones; gds, v, nmd, runset y rundir.
- Identificar y corregir los errores que se den en el proceso de automatización.
- Aprender a utilizar de modo avanzado *IC Compiler II* para encontrar nuevas opciones que puedan apoyar el diseño del nanochip.
- Integración de las diferentes fases de automatización.
- Validar el orden de entradas y salidas de los *decks* desarrollados que se utilizan para la implementación de *Hspice*.

CAPÍTULO 5

Alcance

El alcance de este trabajo se basa principalmente en optimizar y automatizar el flujo de diseño del chip de 180 nm específicamente la fase síntesis física, así como desarrollar una interfaz gráfica que implemente las otras fases del flujo de diseño. La interfaz gráfica está limitada a ser una interfaz básica y funcional.

Para realizar esto se utilizó la herramienta *IC Compiler II*, la documentación y los *runset* existentes para la tecnología 180 nm. Este trabajo será una guía para otras tecnologías que se requieran implementar y automatizar, dado que entre tecnología cambia la documentación y *runset* existentes. Para corregir algunos errores no existen *runsets* compatibles con las herramientas *synopsys* por lo que se tradujo los *runsets* de calibre para ser compatibles con las herramientas de *synopsys* y de forma iterativa se solucionó los errores existentes.

CAPÍTULO 6

Marco teórico

6.1. Bash

Bash es el intérprete de lenguaje de comandos, para el sistema operativo gnu. El nombre es un acrónimo de Bourne-Again SHell. El shell Bourne es el shell tradicional de Unix escrito originalmente por Stephen Bourne. Todos los comandos integrados de Bourne Shell están disponibles en Bash. Las reglas para la evaluación y cotización se toman de la especificación posix para el shell Unix estándar. Sintaxis shell: El shell lee la entrada, procede a través de una secuencia de operaciones. Si la entrada indica el comienzo de un comentario, el shell ignora el símbolo de comentario (#) y el resto de esa línea. [11]

6.2. Python

Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo, es un lenguaje interpretado, esto significa que ejecuta las aplicaciones directamente por el ordenador utilizando un programa denominado interpretador. Es un lenguaje de código abierto, fácil de utilizar y con una gran similitud al lenguaje humano. Python facilita trabajar con inteligencia artificial, *big data* , *machine learning* y *data science*. Los lenguajes interpretados suelen tener un ciclo de desarrollo/depuración más corto que los compilados, aunque sus programas generalmente también se ejecutan más lentamente.[12][13]

- Variable de clase: Una variable definida en una clase y destinada a ser modificada solo a nivel de clase.
- Variable de contexto: Una variable que puede tener diferentes valores dependiendo de su contexto. Esto es similar al almacenamiento local de subprocesos en el que cada subproceso de ejecución puede tener un valor diferente para una variable

- Expresión: Es una acumulación de elementos de expresión como literales, nombres, acceso a atributos, operadores o llamadas a funciones que devuelven un valor.

6.3. Flujo de diseño

Para diseñar un chip existen distintos flujos de diseño los cuales se basan en distintas características para poder llevar a cabo el chip, uno se estos diseños se muestran en la Figura 9. Lo primero que se hace es definir las especificaciones del chip, luego se crea una descripción del comportamiento para analizar el diseño, se crea el HDL, dado que este es base para todos los procesos. Con la simulación, la síntesis lógica y síntesis física se hacen las verificaciones que cumplan con las reglas de diseño y las especificaciones definidas.

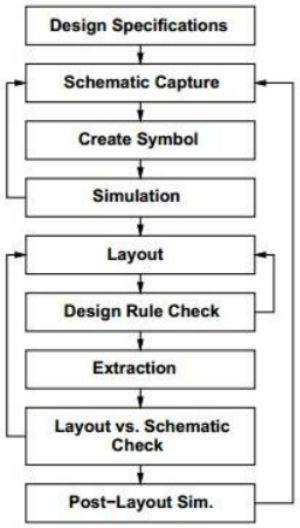


Figura 9: Flujo de diseño para fabricar y diseñar un chip.

6.4. *IC Compiler II*

Es un herramienta de *place and route*, incluye innovación para la planificación de diseño plano y jerárquico, exploración de diseño temprano, ubicación y optimización consciente de la congestión, *clock tree synthesis*, convergencia de *routing* de nodo avanzado, cumplimiento de fabricación y *signoff closure*. [14]

6.5. Síntesis física

Esta parte del flujo de diseño convierte cada subcircuito del diseño lógico a uno físico, en este se muestran las difusiones, pines, el silicio usado. Se obtiene de un *netlist*, para generar un *layout* en formato GDS. Este diseño ya esta listo para manufacturar pero se debe tomar en cuenta que se cumpla con las características del fabricante.

6.6. *Proceso síntesis física*

Para realizar la síntesis física *Synopsys* nos provee la herramienta *IC Compiler II* esta contiene diferentes comandos para cada una de las fases:

1. *Design Initialization*: Para inicializar el diseño, debe realizar las siguientes tareas:[15]
   - Importar la netlist de diseño.
   - Importar la biblioteca.
   - Importar las restricciones de diseño.
   - Configurar las restricciones de tiempo.
   - Crear *floorplan.*
   - Crear *power grid*
2. *Placement and optimization*:Durante la colocación y la optimización, la herramienta encuentra una ubicación física adecuada para cada celda en el bloque de acuerdo con las restricciones de tiempo, congestión y multivoltaje.[15]
3. *Clock tree synthesis and optimization*: Durante la síntesis y optimización del árbol de relojes, la herramienta almacena en búfer los árboles de relojes para equilibrar el sesgo entre los relojes en el diseño colocado de acuerdo con las restricciones de tiempo, congestión y multivoltaje.[15]
4. *Routing*: Durante el enrutamiento, la herramienta conecta los pines de las celdas en el diseño de acuerdo con las reglas de diseño de enrutamiento y las restricciones de tiempo y voltaje múltiple.[15]
5. *Post-route optimization*: Durante la optimización posterior a la ruta, la herramienta optimiza el diseño enrutado de acuerdo con las reglas de diseño y las restricciones de tiempo, congestión y multivoltaje.[15]
6. *Chip finishing*: Durante el acabado de chips, el usuario inserta capacitores de desacoplamiento, celdas de relleno y relleno de metal. El usuario también realiza análisis de electromigración.[15]
7. *Write data*:Durante la escritura de datos, el usuario guarda el diseño final después de completar el lugar y la ruta. El usuario debe guardar los siguientes datos:[15]
   - *Layout data (GDSII or OASIS)*
   - *Floorplanning data (DEF)*
   - *Placed and routed gate-level netlist (Verilog)*
   - *Design database (NDM)*
   - *Library data (NDM)*
   - *Log files*
   - *Report files*

6.7. *Tool Command Language (TCL)*

TCL es una herramienta de secuencias de comandos ampliamente utilizada que se desarrolló para controlar y ampliar aplicaciones, fue creado por John K. Ousterhout en la Universidad de California, Berkeley, se distribuye como software de código abierto. Muchos *shells* de comandos de *Synopsys* utilizan TCL como una herramienta de secuencias de comandos para automatizar los procesos de diseño. TCL proporciona las construcciones de programación necesarias (variables, bucles, procedimientos) para crear *scripts* con comandos de *Synopsys*.[16]

Linux	TCL
ls	glob
rm	file delete
rm -rf	file delete -force
mv	file rename
date	date
sleep	after

Cuadro 1: Comandos TCL equivalentes a comandos Linux

6.8. *Synopsys Design Constraints (SDC)*

El formato *Synopsys Design Constraints (SDC)* se utiliza para especificar la intención del diseño, incluidas las restricciones de tiempo, energía y área para un diseño. SDC se basa en el *Tool Command Language (TCL)*. Las herramientas Synopsys Design Compiler, IC Compiler, IC Compiler II y PrimeTime utilizan la descripción SDC para sintetizar y analizar un diseño. Además, estas herramientas pueden generar descripciones de SDC y leer descripciones de SDC de herramientas de terceros.

6.8.1. *Specifying Design Objects*

La mayoría de los comandos de restricción requieren un objeto de diseño como argumento de comando. SDC admite la especificación de objetos tanto implícita como explícita. Si especifica un nombre simple para un objeto, las herramientas Synopsys determinan el tipo de objeto buscándolo mediante una lista de objetos priorizados. El orden de prioridad varía según el comando y está documentado en la página del manual de la herramienta de cada comando. Esto se denomina especificación implícita de objetos. Los comandos de acceso utilizados para la especificación explícita de objetos son: [17]

Objeto	Comando	Descripción
design	*current_ design*	bloque contenedor
clock	*get_ clocks*	reloj del diseño
	all_ clocks	todos los relojes del diseño
port	*get_ ports*	Un punto de entrada o punto de salida del diseño.
	all_ inputs	todos los puntos de entrada del diseño
	all_ outputs	todos los puntos de salida del diseño
cell	*get_ cells*	instancia del diseño o *library cell*
pin	*get_ pins*	instancia del diseño o *library cell pin*
net	*get_ nets*	Una conexión entre los pines de la celda y los puertos de diseño.
library	*get_ libs*	Un contenedor para celdas de biblioteca.
lib_ cell	*get_ libs_ cells*	elemento lógico
lib_ pin	*get_ libs_ pins*	Un punto de entrada o punto de salida de una *lib_ cell*
registers	*all_ registers*	secuencia lógica

Cuadro 2: Objetos de diseño

6.9. *Design Planning*

La planificación del diseño es una parte integral del proceso de diseño de RTL a GDSII. Durante la planificación del diseño, evalúa la viabilidad de diferentes estrategias de implementación al principio del flujo de diseño. Para diseños grandes, la planificación del diseño lo ayuda a dividir y conquistar el proceso de implementación al dividir el diseño en partes más pequeñas y manejables para un procesamiento más eficiente.[18]

6.10. *Floorplanning*

Es el proceso de dividir bloques lógicos en bloques físicos, dimensionar y colocar los bloques, realizar una ubicación a nivel de plano de macros y celdas estándar, y crear un plan de energía. El objetivo de la planificación es aumentar la eficiencia de los pasos de diseño físico posteriores para permitir un diseño sólido y optimizado. Para generar estimaciones de presupuesto precisas para los bloques físicos, genere estimaciones de tiempo para guiar la asignación de presupuesto de tiempo entre los bloques y las celdas de nivel superior en el diseño.[18]

6.11. *Hierarchical Design Planning Flow*

El flujo de planificación de diseño jerárquico proporciona un enfoque eficiente para gestionar diseños grandes. Al dividir el diseño en múltiples bloques, diferentes equipos de diseño pueden trabajar en diferentes bloques en paralelo, desde RTL hasta la implementación física. Trabajar con bloques más pequeños y usar bloques con múltiples instancias puede reducir el tiempo de ejecución general.[18]

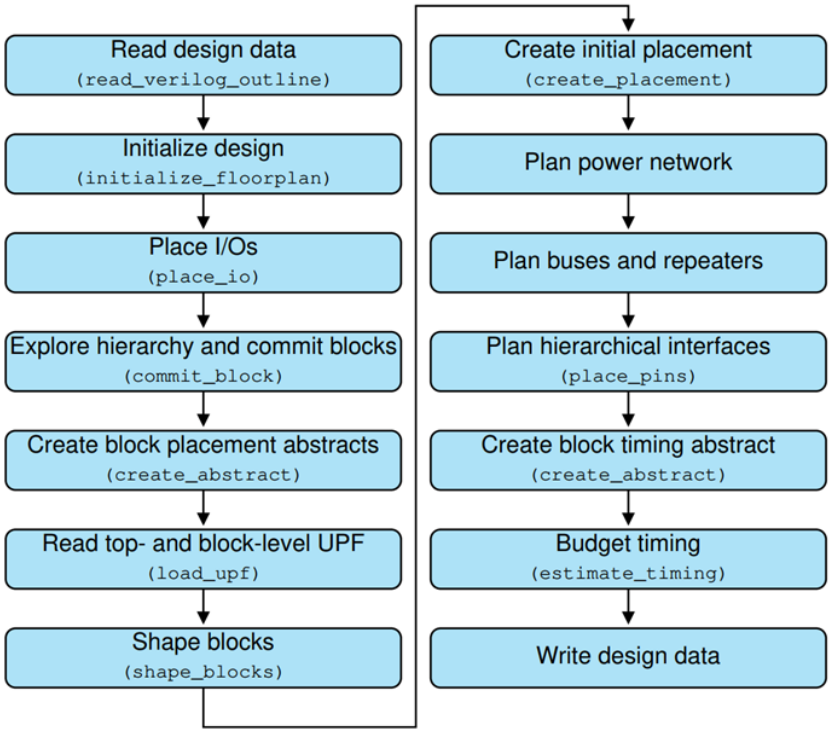


Figura 10: *Hierarchical Design Planning Flow*

6.12. *Creating a Floorplan*

Esta sección describe cómo crear y refinar un plano de planta a partir de una lista de conexiones existente o una descripción del plano de planta. El plano describe el tamaño del núcleo; la forma y ubicación de filas de celdas estándar y canales de enrutamiento; restricciones de colocación de celdas estándar; y la colocación de celdas de E/S periféricas, alimentación y relleno. [18] Para conservar recursos y minimizar el tiempo de respuesta, el lector de netlist crea una vista de esquema de los bloques en los diferentes niveles de la jerarquía de diseño, esto lo hace leyendo el archivo verilog. Para realizar esto se siguen los siguientes comandos:
$icc2\_shell > set\ \ libs\ \ [glob\ \ -directory\ \$\ lib\_dir\ \ *.ndm]$
$icc2\_shell > create\_lib - technology\ \ technology\_file.tf\ \ -ref\_libs\ \$\ libs\ \ lib\backslash design$
$icc2\_shell > read\_verilog\_outline - top\ \ ORCA\ \ ORCA.v$

Después de leer la lista de conexiones de Verilog, use el comando *initialize_floorplan*. Puede crear un plano de planta especificando la relación de longitud entre los bordes del plano de planta, la longitud de los bordes del núcleo, la relación de aspecto o el ancho y la altura exactos.
$icc2\_shell >$ initialize_floorplan -core_offset 100
Para crear un plano de planta con diferentes longitudes de lado, especifique una relación de aspecto con la opción *side_ratio*. Para establecer la relación de aspecto del plano de planta:
$icc2\_shell >$ initialize_floorplan -core_utilization 0.7 -shape R -orientation N -side_ratio

{3.0 1.0} -core_offset {100.0} -flip_first_row true -coincident_boundary true

Para crear pistas de enrutamiento en el plano de planta, use el comando *create_track*. Puede especificar la capa, el número de pistas, la distancia de tono entre pistas, etc.
$icc2\_shell > remove\_tracks - all$
$icc2\_shell > create\_track - layerMETAL3 - space0.5$
$icc2\_shell > report\_tracks$

6.12.1. Creación de reglas adicionales

Para crear reglas de plano de planta para planos de planta que tienen requisitos adicionales, use los comandos que se describen en la siguiente tabla.

Comando	Descripción
set_floorplan_area_rules	Define un área mínima, área máxima, lista de válidos o áreas no válidas, o un rango de áreas válidas o no válidas.
set_floorplan_enclosure_rules	Define una regla de recinto para verificar el espacio legal entre un objeto encerrado y su objeto encerrado.
set_floorplan_halo_rules	Define una regla de halo para verificar el espacio legal entre un objeto cerrado y su objeto envolvente.
set_floorplan_spacing_rules	Define una regla de espaciado para verificar el espacio legal entre objetos.
set_floorplan_width_rules	Define una regla de ancho para verificar el espacio legal entre objetos.
set_floorplan_density_rules	Define una regla para verificar la densidad del plano de planta.

Cuadro 3: Comandos adicionales

6.13. *Design Libraries*

Una biblioteca de diseño almacena toda la información sobre un diseño, incluida la información de tecnología asociada y la información de configuración de la biblioteca de referencia. Además de la información de configuración de la biblioteca de referencia y tecnología, una biblioteca de diseño contiene varias versiones de los diseños almacenados en la biblioteca de diseño. Una versión de un diseño se denomina bloque. El nombre del bloque puede ser igual o diferente al nombre del módulo superior del diseño.[19]

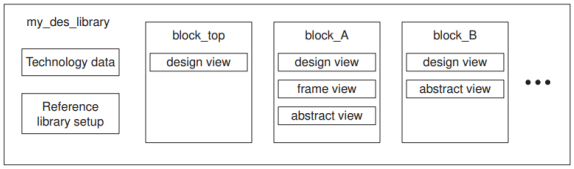


Figura 11: *Design Library Contents*

Comando utilizados para *Design Libraries*[20]:

- *create_lib*: Este comando crea la biblioteca en la memoria y la establece como la biblioteca actual. Por ejemplo: icc2_shell> create_lib ../my_lib_dir/my_libA

- *open_lib*: Este comando abre la biblioteca especificada, convierte esa biblioteca en la biblioteca actual y abre todas sus bibliotecas de referencia asociadas. Por ejemplo: icc2_shell> open_lib my_libA

- *save_lib*: Cuando crea o cambia una biblioteca, los cambios se almacenan solo en la memoria. Para guardar una biblioteca en el disco, use este comando. Por ejemplo: icc2_shell> save_lib my_libA

6.14. *Performing Parasitic Extraction*

El análisis de tiempo preciso depende de la información RC (parásita) precisa para las redes. La herramienta *IC Compiler II* proporciona capacidades de extracción de RC en las etapas previa y posterior al enrutamiento. Además, la herramienta proporciona la capacidad de realizar anotaciones en la información parásita detallada.[21]

Para la estimación de rutas virtuales, la herramienta *IC Compiler II* calcula los coeficientes RC aplicando técnicas de promedio a los coeficientes RC específicos de la capa porque las capas no están asignadas en esta etapa del flujo de diseño.[21]

6.15. *Routing Design Rules*

6.15.1. *No Nonpreferred Direction Routing Rule*

De forma predeterminada, el enrutamiento se puede ejecutar en la dirección preferida o no preferida. Para restringir el enrutamiento en una capa a su dirección preferida, establezca el atributo *nonPreferredRouteMode* en la sección Capa en 1. La sintaxis de esta regla es: [22]
Layer "MetalX" {nonPreferredRouteMode = 1}

6.15.2. *Rectangle-Only Rule*

La regla de solo rectángulo especifica que una capa de metal solo permite formas rectangulares. Cualquier movimiento o forma rectilínea es una violación de las reglas de diseño.La sintaxis de esta regla es: [22]
Layer "MetalX" {hasRectangleOnly = 1}

Figura 12: Regla *Rectangle-Only*

6.15.3. *Dense Wire Minimum Dimensions Rule*

Define la longitud y el ancho mínimo de un rectángulo de metal entre alambres muy juntos. Cuando el espacio entre un rectángulo y cualquiera de los cables cercanos es menor que el valor S, el rectángulo debe tener una longitud de al menos L o un ancho de al menos W.[22]
La sintaxis de esta regla es:
Layer "MetalX"{ denseWireMinWidth = W denseWireMinLength = L denseWireMinSpacing = S }

6.15.4. *Metal Width Rules*

- *Minimum Width Rule*: La regla de ancho mínimo define el ancho mínimo de una forma de metal. La sintaxis de esta regla es:
  Layer "MetalX"{ minWidth = Wmin [nonPreferredWidth = WN] }

- *Default Width Rule*: La regla de ancho predeterminado define el ancho predeterminado de una ruta de señal. La sintaxis de esta regla es:
  Layer "MetalX"{defaultWidth = W [xDefaultWidth = WX] [yDefaultWidth = WY] }

- *Discrete Metal Widths Rule*: La regla de anchos discretos de metal define un conjunto limitado de anchos permitidos para cualquier forma en una capa determinada. Se especifica una lista separada para la dirección x y la dirección y.

- *Signal Routing Maximum Metal Width Rule*:La regla de ancho máximo de metal define el ancho máximo de una ruta de señal en una capa determinada.La sintaxis de esta regla es:
  Layer "MetalX"{signalRouteMaxWidth = Wmax }

6.15.5. *Metal Density Rules*

Las reglas de densidad del metal definen los requisitos de densidad del metal para una capa de metal. Debe especificar estas reglas para cada capa de metal. La sintaxis de esta regla es:
DensityRule { layer = "MetalX" windowSize = winsize minDensity = minpercent maxDensity = maxpercent densityGradient = maxgradient }

6.16. *Routing and Postroute Optimization*

6.16.1. *Routing Application Options*

La herramienta IC Compiler II proporciona opciones de aplicación que afectan a los motores de enrutamiento individuales (enrutamiento global, asignación de pistas y enrutamiento detallado), así como opciones de aplicación que afectan a los tres motores de enrutamiento.

6.16.2. *Routing Clock Nets*

Para enrutar redes de reloj antes de enrutar el resto de las redes en el bloque, use el comando *route_group* con la opción adecuada para seleccionar las redes. Las opciones son: *-all_clock_nets* para todos los relojes y *-nets* para seleccionar relojes especificos. La sintaxis es:
icc2_shell> route_group -all_clock_nets

6.16.3. *Routing Signal Nets*

Antes de enrutar las redes de señal, todas las redes de reloj deben enrutarse sin violaciones. Puede enrutar las redes de señal utilizando uno de los siguientes métodos:

- Para realizar el enrutamiento global, utilice el comando *route_global*.
- Para realizar la asignación de pistas, utilice el comando *route_track*.
- Para realizar un enrutamiento detallado, use el comando *route_detail*.

6.17. *Performing Signoff Design Rule Checking*

Para la verificación de *design rule checking* (DRC) *IC Compiler II* ejecuta la herramienta *IC Validator* dentro de la herramienta para verificar las reglas de diseño de enrutamiento definidas. Para realizar la verificación de las reglas de diseño se ejecuta el comando *signoff_check_drc*. *Zroute* puede usar los resultados de DRC generados por el comando *signoff_check_drc* para corregir automáticamente las violaciones de las reglas de diseño detectadas,

para realizar esta corrección se utiliza el comando *signoff_fix_drc*, esta comando no soluciona todos los errores de DRC para otros errores existe el comando *signoff_create_metal_fill* comando para realizar la inserción de relleno de metal en un bloque. La sintasis para estos comandos es:
icc2_shell> signoff_check_drc
icc2_shell> signoff_fix_drc
icc2_shell> signoff_create_metal_fill -mode replace -select_layers {M1 M3}

CAPÍTULO 7

Optimización generación de archivo verilog

En el trabajo [7], se estableció un *script* de *python* para realizar la generación de un archivo verilog, el cual es la base para realizar el flujo de diseño de un chip. El circuito que se implementa se basa en *if* y *else* para poder mostrar cada carácter de los textos predefinidos.

El *script* se divide en dos, la parte no repetitiva y la parte estática. En la parte estática se incluye la declaración del módulo, los pines de entrada y salida, los cables que se encargan de conectar los módulos y compuertas, el reloj a utilizar, un circuito de 19 compuertas *NOT* que junto a una compuerta *AND* forma un circuito conocido como *Ring oscillator*, además de la declaración del contador y el inicio del texto,

La parte no repetitiva se basa en leer los textos predefinidos, separando cada carácter a mostrar y haciendo su conversión de *ASCII* a binario para poder mostrarlo en los pines de salida del chip, esta parte es la que se compone de sentencias *if* y *else* dependiendo del valor del contador. En la versión realizada en el trabajo [7], la parte de conversión del carácter a binario se realizaba de la siguiente manera:

```
1 binario = bin(int(binascii.hexlify(texto[i+1]) ,16))
2 for x in range(10-len(binario)):
3     archivo.writelines("0")
4 for j in range(len(binario) -2):
5     archivo.writelines(str(binario[j+2]))
6 archivo.writelines(";\n")
```

Esta parte funciona bien para *python* 2 sin embargo al actualizar a *python* 3, la conversión ya no funciona, además de este método no funciona para cualquier texto, ya que manualmente hay que cambiar el valor de la conversión del primer carácter. Para corregir la conversión del primer carácter se sumó uno al valor de cada carácter haciendo que el primer carácter sea nulo. Con esta modificación el texto funciona para *python* 2 y cualquier texto.

Para que esta versión no quede obsoleta se actualizó a *python* 3, el primer cambio que se hizo es cambiar la forma de conversión, con el siguiente comando:

```
1 binario=bin(ord(texto[i+1]))
```

Este comando reduce y optimiza la conversión, además que soluciona el error del primer carácter sin importar que sea nulo o que este sea algún carácter especial. El *script* se ejecutó con diferentes textos para validar que sea funcional, dando como resultado el *script* que se muestra en los anexos.

CAPÍTULO 8

Interfaz gráfica

8.1. Diseño interfaz gráfica

Para realizar la interfaz gráfica se utilizó el paquete Tkinter para *python*, en la mayoría de aplicaciones se requieren de dos módulos de este paquete los cuales son: `tkinter` y `tkinter.ttk`, estos se importan de la siguiente manera:

```
1 from tkinter import *
2 from tkinter import ttk
```

Luego de las importaciones hay que crear una instancia de clase Tk, también crea una ventana raíz, que sirve como ventana principal de la aplicación, para mostrar la ventana se utiliza `.mainloop()`, de la siguiente manera:

```
1 from tkinter import *
2 from tkinter import ttk
3 root = Tk()
4 root.mainloop()
```

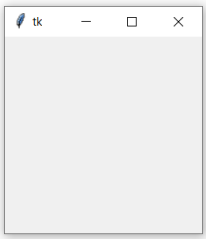


Figura 13: Ventana principal de la aplicación

8.2. Implementación de *widgets*

El paquete Tkinter tiene varios *widgets*, cada *widget* se representa como un objeto de *python*, instanciado a partir de clases como: `ttk.Frame`, `ttk.Labely` y `ttk.Button`, estos se organizan en una jerarquía y tienen opciones de configuración, que modifican su apariencia y comportamiento.

8.2.1. Configuración de *widgets*

Para cada *widget* existen distintas configuraciones, las más utilizadas son:

- *bg*: indica el color de fondo.
- *fg*: indica el color del texto.
- *height*: altura en líneas del componente.
- *width*: anchura en caracteres del componente
- *font*: nos permite especificar la fuente, tamaño y estilo.
- *text*: texto que se muestra en el botón.
- *command*:indica qué función se ejecute cuando se haga *click* en el botón.

Algunos ejemplos de como se implementan los *widgets*:

```
ttk.Frame(root, padding=10)
frm.grid()
ttk.Label(frm, text="Hello World!").grid(column=0, row=0)
ttk.Button(frm, text="Quit", command=root.destroy)
```

8.2.2. Interfaz gráfica flujo de diseño

El siguiente *script* es la base para la interfaz gráfica utilizada para implementar el flujo de diseño.

```
from tkinter import *
from tkinter import ttk
import subprocess
root = Tk()
frm = ttk.Frame(root, padding=10)
frm.grid()
ttk.Label(frm, text="Diseño de un circuito integrado").grid(column=0, row=0)
ttk.Button(frm, text="Síntesis Lógica", command=root.destroy).grid(column=1,
    row=1)
ttk.Button(frm, text="Síntesis Física", command=root.destroy).grid(column=2,
    row=1)
ttk.Label(frm, text="Verificaciones:").grid(column=0, row=2)
ttk.Button(frm, text="DRC", command=root.destroy).grid(column=1, row=3)
ttk.Button(frm, text="LVS", command=root.destroy).grid(column=2, row=3)
```

```
13 ttk.Button(frm, text="ERC", command=root.destroy).grid(column=3, row=3)
14 ttk.Button(frm, text="Antena", command=root.destroy).grid(column=4, row=3)
15 root.mainloop()
```

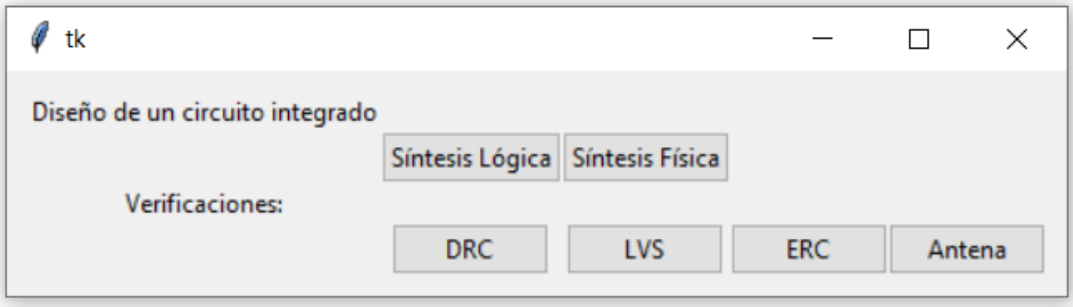


Figura 14: Interfaz gráfica implementada.

CAPÍTULO 9

Automatización síntesis física

En los trabajos [8], [9]y [10] se establecieron los comandos para la realización de la síntesis física utilizando la herramienta *IC Compiler II*, estos se usaron como base para la automatización. Para comprobar que el *script* es funcional se realizaron múltiples iteraciones hasta comprender cada comando.

9.1. Proceso de automatización

Para realizar la automatización se utilizó *bash scripting*, el *script* tiene extensión `.sh`, es necesario que al inicio del archivo se coloque el siguiente comando: `#!/bin/bash`, luego de este comentario se sigue con los otros comentarios. La primera versión del *script* solo ejecuta la herramienta *IC Compiler II*, utilizando el siguiente comando:

```
1 icc2_shell -file plantilla.tcl
```

En este se especifica el archivo con extensión `.tcl` el cual contiene los comandos necesarios para ejecutar la síntesis física, como se observa en los anexos. Al ejecutar el *script*, el primer error que obtenemos hace referencia a la creación de la librería NDM, esto se debe a que ya existe la librería, para evitar esto se añade el siguiente comando, al inicio del archivo plantilla.tcl:

```
1 file delete -force  EL_GRAN_JAGUAR.ndm
```

El siguiente error se debe a que no encuentra los archivos de entrada y los archivos que hacen referencia a la tecnología, esto se debe a que no se especifica el directorio de estos archivos, por lo que se especificó cada directorio, al realizar esta modificación la síntesis se desarrolló de manera adecuada llegando a obtener los mismos resultados de los trabajos anteriores.

Al realizar esta modificación obtenemos un problema al cambiar el directorio de trabajo o de computadora, dado que los directorios no coinciden, por lo que se optó por cambiar las direcciones absolutas a direcciones relativas, creando la siguiente jerarquía en los directorios.

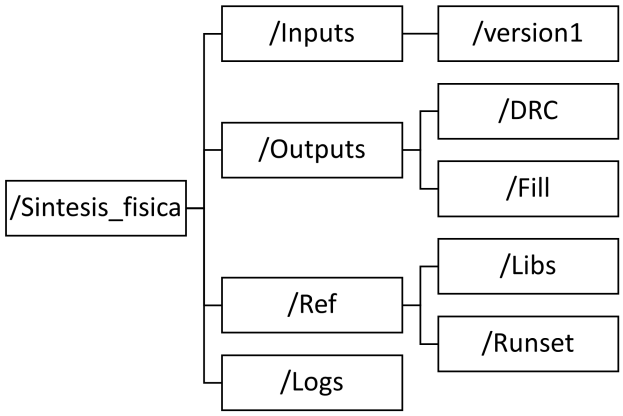


Figura 15: Jerarquía del sistema de archivos implementada.

Esta misma jerarquía funciona para poder implementar las otras fases del flujo de diseño. La siguiente modificación se dio debido a la variación de tamaño entre los diferentes chips, por lo que se le pide al usuario que ingrese: *side legth x*, *side legth y* y *core offset*, esto para que se adapte de mejor manera el tamaño y evitar que el espacio sea o demasiado pequeño o muy grande, como se muestra en los siguientes comandos:

```
echo ingrese el valor de side legth x
read var1
echo ingrese el valor de side legth y
read var2
echo ingrese el valor de core offset
read var3
```

Estos parámetros se buscan en un archivo plantilla que contiene el *script* de la síntesis física, con el siguiente comando:

```
sed "s/initialize_floorplan -site_def unit -use_site_row -keep_all -
    side_length {285 285} -core_offset {125}/initialize_floorplan -site_def
    unit -use_site_row -keep_all -side_length {$var1 $var2} -core_offset {
    $var3}/"<plantilla.tcl >scriptfs.tcl
```

Se sustituyen en el archivo que se ejecuta, esto con el fin de mantener un archivo sin modificaciones para que la búsqueda sea siempre la misma. Cada variable se sustituye en la siguiente línea del archivo a ejecutar:

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

Luego de esto se realiza el comando para ejecutar el archivo:

```
icc2_shell -file scriptfs.tcl
```

Durante la ejecución de la síntesis física se crean muchos archivos los cuales no son útiles para las siguientes fases, pero si para tener información del proceso por lo que se mueven a una carpeta llamada Logs, con los siguientes comandos:

```
mv *.log /logs
mv *.svf /logs
mv *.txt /logs
```

9.2. Ejecución síntesis física

Se utilizó el *script* generado en la sección anterior para realizar la síntesis física, para ejecutar este *script* se utiliza el siguiente comando en cualquier consola:

```
./scriptfs.sh
```

Para que se ejecute es necesario comprobar que las carpetas tengan los archivos necesarios en cada carpeta, para la carpeta de *Inputs* se debe tener los archivos que muestra la Figura 16, para la carpeta *Runset* archivos que muestra la Figura 17, para la carpeta *Libs* archivos que muestra la Figura 18 y para la carpeta *techfiles* archivos que muestra la Figura 19.

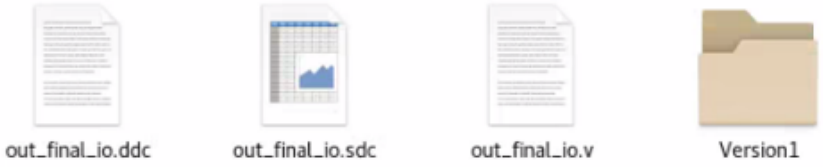


Figura 16: Archivos carpeta *Inputs*

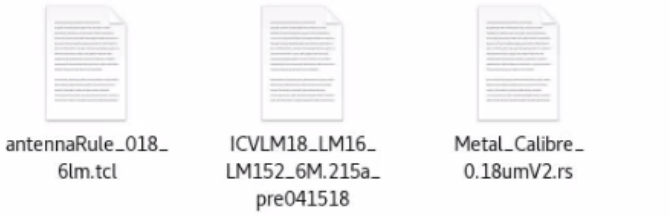


Figura 17: Archivos carpeta *Runset*

Figura 18: Archivos carpeta *Libs*

Figura 19: Archivos carpeta *techfiles*

Los archivos de las carpetas *Libs*, *techfiles* y *Runset* dependen de la tecnología del chip, por lo que puede variar para otra tecnología. Al ejecutar este *script* se ingresa los parámetros requeridos y se ejecuta la síntesis física. Se ingresaron los valores de: `x=285` `y=285` y `core_offset=125` dado como resultado la Figura 20.

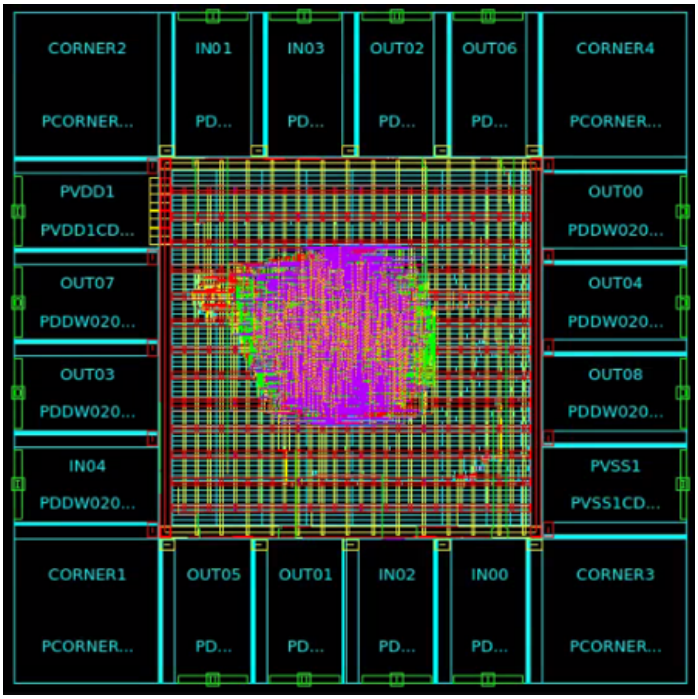


Figura 20: Chip generado con la automatización.

9.3. Integración de las fases

Para obtener un flujo de diseño óptimo es necesario integrar las fases: síntesis lógica, síntesis física y verificaciones, esta integración se divide en varias partes, la primera es identificar los archivos necesarios para cada fase, identificados estos archivos es necesario corroborar que archivos se pueden generar y cuáles nos provee la empresa *TSMC*. Para la fase de síntesis física se requieren un archivo verilog, un archivo sdc, las librerías de la tecnología a utilizar, los *runsets* y los archivos *techfiles*.

Los errores más comunes al integrar las fases se dieron con las carpetas dado que en un principio la fase anterior y posterior utilizaban directorios absolutos por lo que leer los

archivos que se generan era complicado, esto se solucionó implementando una jerarquía similar en las carpetas que se muestra en la Figura 15, esto con el fin de tener un mejor control de los archivos.

Para obtener una mejor integración se creó una carpeta compartida en un servidor, en el cual se creó una carpeta para cada fase, en esa carpeta se colocó todo lo necesario para que se pueda realizar cada fase y luego se actualizó los directorios. Luego de esto se realizaron pruebas donde se encontraron errores por los nombres de archivos y algunas librerías que no se encontraron. Para solucionar el error de las librerías se creó una carpeta que contiene todas las librerías utilizadas en todas las fases. Para solucionar los errores de nombres y no interferir con cada fase, se implementó en cada fase un cambio de nombre en los archivos y se realizó una copia de los archivos para que no se sustituyan si se están realizando al mismo tiempo.

CAPÍTULO 10

Corrección de errores de metal

Al ejecutar la verificación DRC en la síntesis física se encontraron errores de densidad de metal, estos errores se muestran en la Figura 3. Para corregir estos errores se intentaron diferentes técnicas, la primera técnica que se utilizó fue cambiar el *width* en ciertas partes y en cada metal, con esta técnica se corrigen los errores de metal 1, 2 y 3, además que los metales 4, 5 y 6 se aumentó la densidad. Los problemas con esta técnica es que es un proceso largo que puede llevar bastante tiempo, además en cada síntesis que se haga se debe modificar cada *width* y se debe saber que *width* hay que modificar para no generar otros errores de DRC, por lo que para un usuario final es complicado.

La segunda técnica que se utilizó es cambiar el texto del chip aumentando los caracteres que va a mostrar, con esta técnica se mejoró las densidades, pero no se corrigió ningún error. La tercera técnica combina las primeras dos, pero no se obtuvo cambios significativos en las densidades, además que los cambios de *width* anterior no funcionan con el nuevo texto por lo que se tuvo que escoger otros *width*. Para la cuarta técnica se utilizó polígonos de metal en cada uno de los metales, con esta técnica se solucionó los errores de cada uno de los metales, pero se agregaron algunos errores de DRC, esto debido a la ubicación y tamaño de los polígonos, por lo que esta técnica tiene errores similares a la primera técnica.

10.1. Traducción de *Runset*

Las herramientas de *synopsys* contienen comandos específicos para el relleno de metal y corrección de errores de densidad, para ejecutar estos comandos se requiere de un *runset* compatible con las herramientas, dado que los *runset* que nos dieron no son compatibles se realizó una traducción de un *runset* de calibre a uno compatible.

La primera fase de la traducción se hizo de forma manual, utilizando el manual de calibre

y el manual de PXL, esta traducción fue larga y con muchos errores dado que algunas de las funciones implementadas requieren de un mayor conocimiento de calibre y PXL, la traducción que se obtuvo con este método no funcionó, por lo que se descartó.

La siguiente fase se realizó utilizando una herramienta que nos provee *synopsys*, esta herramienta se encuentra en la carpeta de *IC Validator*, se puede encontrar en el siguiente directorio: `$ICV_HOME_DIR/contrib/cal2pxl`.

Para ejecutar esta herramienta se utiliza la terminal de linux, desde el directorio anterior y se ejecuta el siguiente comando: `cal2pxl runset_calibre runset_pxl`, donde `runset_calibre` se coloca el nombre del archivo de calibre que se desea traducir y en `runset_pxl` el nombre del archivo de salida. Otra forma de ejecutar la herramienta es copiar el archivo cal2pxl a alguna carpeta y ejecutar desde esa carpeta el mismo comando. En este caso se optó por copiar el archivo a otra carpeta y se ejecutó dando como resultado la Figura 21.

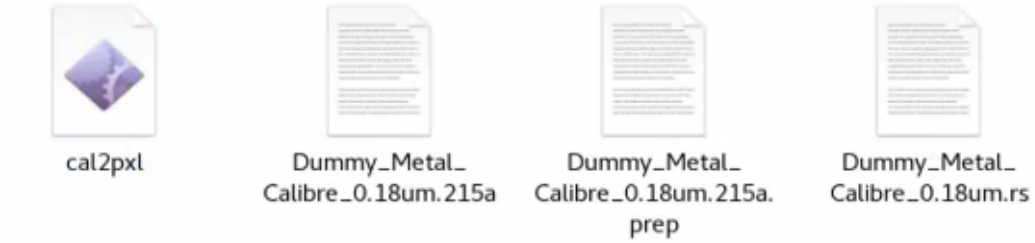


Figura 21: Archivos generados al traducir.

Al realizar la traducción con la herramienta se dio un *warning*, se busco en la documentación y no era un *warning* que afecte el funcionamiento. Para probar el *runset* se ejecuto el comando `signoff_create_metal_fill`, con este comando se obtuvo los errores que se muestran en la Figura 22.

```
Dummy_Metal_Calibre_0.18um.rs:4: error: Excessive token sequence "(2021, 6, 3, 0)"
    #ifdef VERSION_LT(2021, 6, 3, 0)
Dummy_Metal_Calibre_0.18um.rs:5: error: #error This PXL runset was generated to run with ICV version 2021.06-SP3 and newer.

    #error This PXL runset was generated to run with ICV version 2021.06-SP3 and newer.
Dummy_Metal_Calibre_0.18um.rs:1350: warning: Encryption pragma "encrypt" found in input runset. Please check if this runset requires encryption.
    #pragma PXL encrypt begin
Dummy_Metal_Calibre_0.18um.rs:1406: warning: Encryption pragma "encrypt" found in input runset. Please check if this runset requires encryption.
    #pragma PXL encrypt end
Dummy_Metal_Calibre_0.18um.rs:2658: warning: Encryption pragma "encrypt" found in input runset. Please check if this runset requires encryption.
    #pragma PXL encrypt begin
Dummy_Metal_Calibre_0.18um.rs:4208: warning: Encryption pragma "encrypt" found in input runset. Please check if this runset requires encryption.
    #pragma PXL encrypt end
```

Figura 22: Errores generados al traducir 1.

Los errores que se dieron no permiten ejecutar el *runset* por lo que no se puede saber si existen más errores de traducción, se solucionaron cambiando la versión que se ejecuta *IC Validator* con esos errores corregidos el *runset* se ejecutó y se obtuvo un error llamado *Unexpected token* combinado con *Unexpected EOF*, con este error hace referencia a que falta cerrar alguna función, para corroborar este se hizo un *find* para encontrar que tantas llaves, corchetes y paréntesis sean los correctos, donde se encontró que faltaba un paréntesis, por lo que se buscó manualmente y se solucionó.

Se ejecutó de nuevo el comando y se encontró los errores que se muestran en la Figura 23. Se analizó cada error obtenido en la Figura 23 en la línea correspondiente y luego se solucionaron. Al solucionar estos errores se ejecutó nuevamente el *runset* obteniendo los errores que se muestran en las figuras 24 y 25. Se solucionó los 45 errores obtenidos y se volvió a ejecutar el *runset*, obteniendo 10 errores que se muestran en la Figura 26. Se solucionaron los errores y se obtuvieron otros 5 errores que se muestran en la Figura 27, estos errores se solucionaron y se obtuvo un *runset* válido para ejecutar con el comando `signoff_create_metal_fill`.

```
Metal_Calibre_0.18um.rs:1788: error: unexpected token: =
Metal_Calibre_0.18um.rs:1794: error: unexpected token: =
Metal_Calibre_0.18um.rs:1952: error: unexpected token: WindowSize1
Metal_Calibre_0.18um.rs:1960: error: unexpected token: emptyWindowSize
failed with 4 errors
fix previous errors to continue
```

Figura 23: Errores generados al traducir 2.

```
In call of 'internal1' (function returning type polygon_layer)...
Metal_Calibre_0.18um.rs:1402: error: 'internal1' (function returning type polygon_layer) has no parameter named 'gBULK'
Metal_Calibre_0.18um.rs:1402: error: missing value for required parameter 'layer1'
Metal_Calibre_0.18um.rs:1402: error: missing value for required parameter 'distance'
/usr/synopsys/icvalidator/S-2021.06-SP3-2/include/drc_public.rh:7288: 'internal1' (function returning type polygon_layer) is declared here
At top level...
Metal_Calibre_0.18um.rs:1464: error: operator '<=' prohibits operands of type constraint of double and type double
Metal_Calibre_0.18um.rs:1464: error: expression of type boolean is illegal as a statement
Metal_Calibre_0.18um.rs:1470: error: operator '<=' prohibits operands of type constraint of double and type double
Metal_Calibre_0.18um.rs:1470: error: expression of type boolean is illegal as a statement
In call of 'external1' (function returning type polygon_layer)...
Metal_Calibre_0.18um.rs:1607: error: 'external1' (function returning type polygon_layer) has no parameter named 'gM1F'
Metal_Calibre_0.18um.rs:1607: error: missing value for required parameter 'layer1'
Metal_Calibre_0.18um.rs:1607: error: missing value for required parameter 'distance'
/usr/synopsys/icvalidator/S-2021.06-SP3-2/include/drc_public.rh:6191: 'external1' (function returning type polygon_layer) is declared here
In call of 'not_interacting' (function returning type polygon_layer)...
/usr/synopsys/icvalidator/S-2021.06-SP3-2/include/drc_public.rh:9058: 'not_interacting' (function returning type polygon_layer) is declared here
At top level...
Metal_Calibre_0.18um.rs:1623: error: operator '<=' prohibits operands of type constraint of double and type double
Metal_Calibre_0.18um.rs:1623: error: expression of type boolean is illegal as a statement
Metal_Calibre_0.18um.rs:1629: error: operator '<=' prohibits operands of type constraint of double and type double
Metal_Calibre_0.18um.rs:1629: error: expression of type boolean is illegal as a statement
In call of 'external1' (function returning type polygon_layer)...
Metal_Calibre_0.18um.rs:1772: error: 'external1' (function returning type polygon_layer) has no parameter named 'gM2F'
Metal_Calibre_0.18um.rs:1772: error: missing value for required parameter 'layer1'
Metal_Calibre_0.18um.rs:1772: error: missing value for required parameter 'distance'
/usr/synopsys/icvalidator/S-2021.06-SP3-2/include/drc_public.rh:6191: 'external1' (function returning type polygon_layer) is declared here
In call of 'not_interacting' (function returning type polygon_layer)...
/usr/synopsys/icvalidator/S-2021.06-SP3-2/include/drc_public.rh:9058: 'not_interacting' (function returning type polygon_layer) is declared here
At top level...
Metal_Calibre_0.18um.rs:1790: error: operator '<' prohibits operands of type constraint of double and type double
Metal_Calibre_0.18um.rs:1790: error: expression of type boolean is illegal as a statement
Metal_Calibre_0.18um.rs:1796: error: operator '<' prohibits operands of type constraint of double and type double
Metal_Calibre_0.18um.rs:1796: error: expression of type boolean is illegal as a statement
In call of 'external1' (function returning type polygon_layer)...
```

Figura 24: Errores generados al traducir 3.

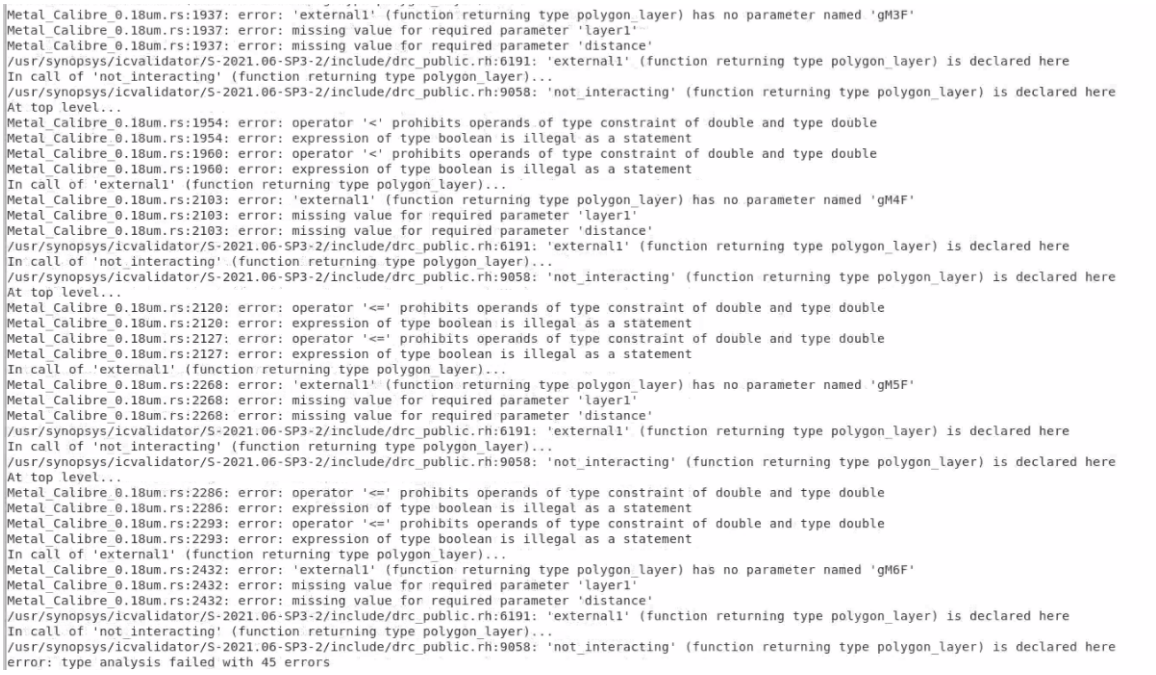


Figura 25: Errores generados al traducir 4.

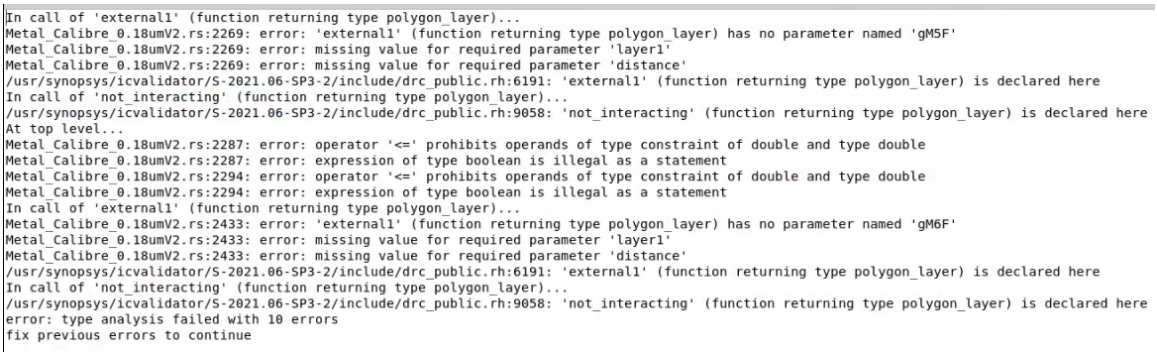


Figura 26: Errores generados al traducir 5.

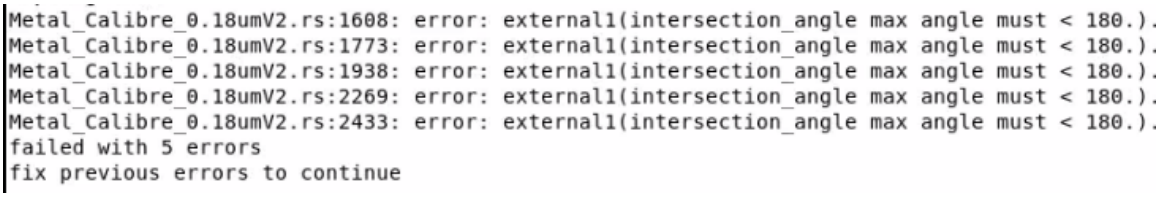


Figura 27: Errores generados al traducir 6.

10.2. Ejecución del *Runset*

Con el *runset* sin errores, se ejecutó la síntesis física con el fin de comprobar y corregir los errores de densidad de metal, de la síntesis física se obtuvo el archivo `chip.gds` este archivo

se utilizó con la herramienta *ic validator* en conjunto con el *runset* para hacer un relleno de metal. Se utilizó el comando: `icv -i chip.gds -c Chip_IO Metal_Calibre_0.18umV2.rs`, al ejecutar el comando se obtuvo el resultado *not clean* que se muestra en la Figura 28 y los errores que se muestran en la Figura 29

Figura 28: Resultado al realizar relleno de metal.

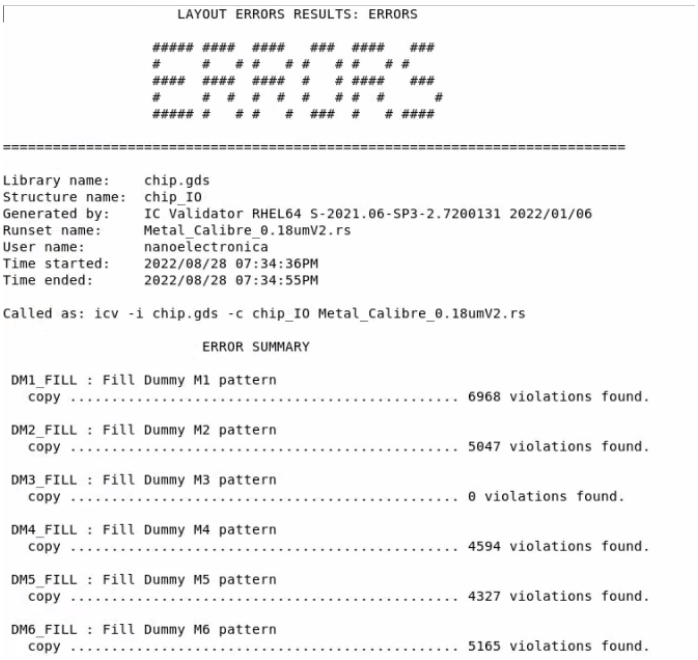


Figura 29: Errores al realizar relleno de metal.

Se realizó modificaciones a la síntesis física y al *runset*, además se añadió el proceso de relleno de metal al *script* existente para poder automatizar la parte del relleno de metal. Con estas modificaciones se realizó de nuevo el relleno de metal dando como resultado los errores que se muestran en la Figura 30, en este se redujeron los errores en cada metal y aumentó la densidad de cada uno.

```
Input file name:        EL_GRAN_JAGUAR.ndm
Top cell name:          chip_IO
Layout format:          NDM
Runset file:            /usr/synopsys/icc2/S-2021.06-SP4/etc/pxl/signoff_metal_fill.rs
Working directory:      /mnt/nfs/var/Automation/Integracion/Diego/Sintesis_Fisica/Outputs/Fill/icv_run_1
Run details directory:  /mnt/nfs/var/Automation/Integracion/Diego/Sintesis_Fisica/Outputs/Fill/icv_run_1/run_details
Layout error file:      chip_IO.LAYOUT_ERRORS
User name:              nanoelectronica
Time started:           2022/08/28 08:34:30PM
Time ended:             2022/08/28 08:36:04PM

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

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

Rule and DRC Error Summary

129 total rules were run.
0 rules NOT EXECUTED.
5 rules have violations.
There are 12840 total violations.
Refer to chip_IO.LAYOUT_ERRORS

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

Rule                        Violations Found
DM1_FILL                    v = 4
DM2_FILL                    v = 2396
DM4_FILL                    v = 3153
DM5_FILL                    v = 3926
DM6_FILL                    v = 3361
```

Figura 30: Errores al realizar relleno de metal.

10.3. Otras modificaciones

Con el fin de reducir los errores de DRC y densidad se realizaron pruebas con distintos escenarios modificando el texto y algunas reglas, además de realizar cambios en la forma de ejecución del flujo de diseño, se utilizó de base el flujo que se muestra en la Figura 31, en el primer escenario se mantuvo el texto y se realizó 2 veces el comando `signoff_create_metal_fill`, dando como resultado los errores de la Figura 32 y la síntesis física que se muestra en la Figura 33. En el segundo escenario solo se realizó una vez el comando de relleno de metal y se obtuvo los resultados de las figuras 34 y 35. De los escenarios que se realizaron destacan:

- Escenario donde solo hay 2 errores de densidad y ninguno de reglas, los errores se dan en el metal 5 y 6, con densidades cercanas al 30% como se muestra en la Figura 42.

- Escenario donde no tiene errores de densidad pero tiene errores de reglas, estos errores son demasiados por lo que no se pueden solucionar.

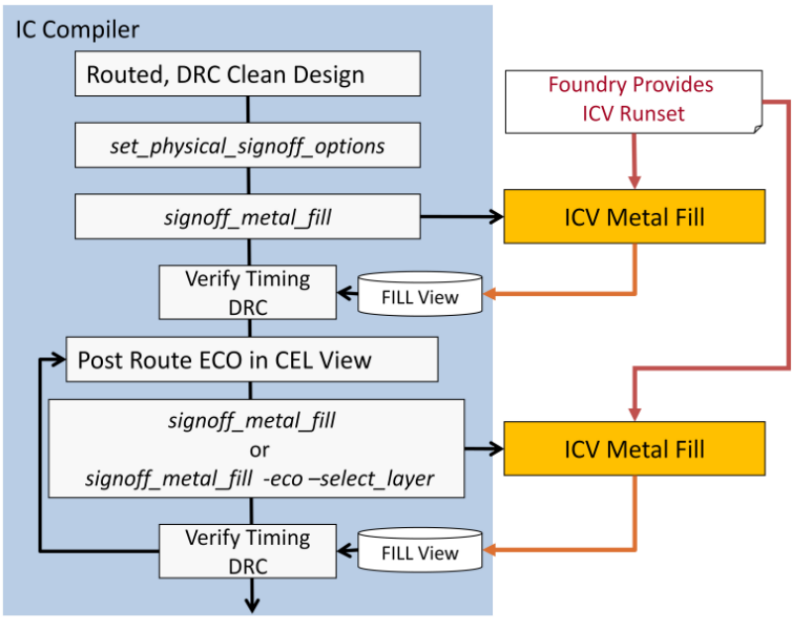


Figura 31: Flujo síntesis física con relleno de metal.

```
--------------------------------------------------------------------------------
Rule:                    M1.S.1 : Min. M1 space < 0.23              166 errors
Rule:                    M1.W.1 : Min. M1 width < 0.23                4 errors
Rule:                    M2.S.1 : Min. M2 space < 0.28             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:                    M2.W.1 : Min. M2 width < 0.28               56 errors
Rule:               M3.A.1 : Min M3 area region < 0.202               7 errors
Rule:                    M3.S.1 : Min. M3 space < 0.28              234 errors
Rule:               M4.A.1 : Min M4 area region < 0.202               4 errors
Rule:                    M4.S.1 : Min. M4 space < 0.28              264 errors
Rule:               M5.A.1 : Min M5 area region < 0.202               1 error
Rule:               M5.R.1 : Min M5 area coverage < 30%               1 error
Rule:                    M5.S.1 : Min. M5 space < 0.28             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:              M6.A.1 : Min. M6 area region < 0.562               82 errors
Rule:               M6.R.1 : Min M6 area coverage < 30%               1 error
Rule:                    M6.S.1 : Min. M6 space < 0.46             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:                 VIA1.S.1 : Min. VIA1 space < 0.26               1 error
Rule:                 VIA2.S.1 : Min. VIA2 space < 0.26               5 errors
Rule:               VIA5.S.1 : Min. VIA5 spacing < 0.35               1 error
Rule:         layout_grid_errors:off_grid_boundary (1)               32 errors
Rule:         layout_grid_errors:off_grid_boundary (2)                4 errors
--------------------------------------------------------------------------------
```

Figura 32: Errores primer escenario.

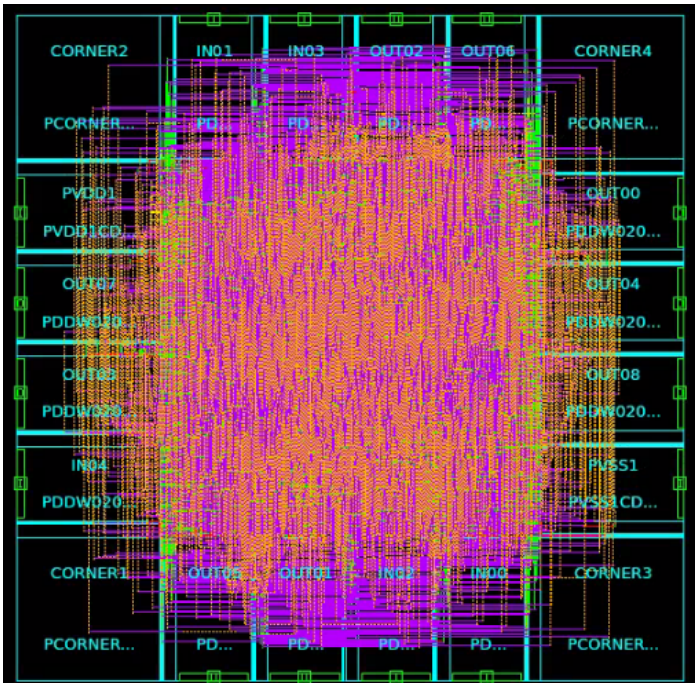


Figura 33: Síntesis física primer escenario.

```
--------------------------------------------------------------------------------
Rule:              M1.R.1 : Min M1 area coverage < 30%                1 error
Rule:                M1.S.1 : Min. M1 space < 0.23                  123 errors
Rule:              M2.R.1 : Min M2 area coverage < 30%                1 error
Rule:                M2.W.1 : Min. M2 width < 0.28                   68 errors
Rule:              M3.A.1 : Min M3 area region < 0.202                9 errors
Rule:                M3.S.1 : Min. M3 space < 0.28                  222 errors
Rule:              M4.A.1 : Min M4 area region < 0.202                3 errors
Rule:              M4.R.1 : Min M4 area coverage < 30%                1 error
Rule:                M4.S.1 : Min. M4 space < 0.28                  160 errors
Rule:              M5.R.1 : Min M5 area coverage < 30%                1 error
Rule:             M6.A.1 : Min. M6 area region < 0.562                1 error
Rule:              M6.R.1 : Min M6 area coverage < 30%                1 error
Rule:                VIA1.S.1 : Min. VIA1 space < 0.26                3 errors
Rule:                VIA2.S.1 : Min. VIA2 space < 0.26                3 errors
Rule:              VIA5.S.1 : Min. VIA5 spacing < 0.35                1 error
--------------------------------------------------------------------------------
```

Figura 34: Errores segundo escenario.

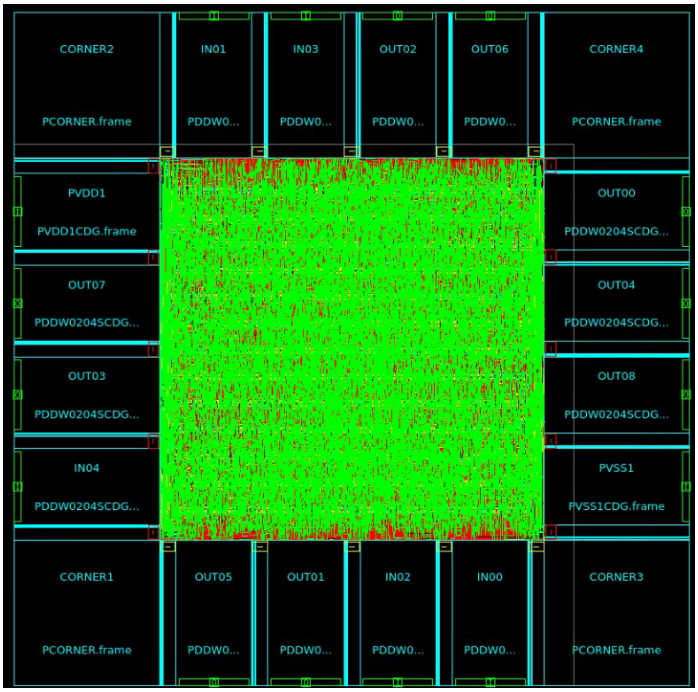


Figura 35: Síntesis física segundo escenario.

```
--------------------------------------------------------------------------------
Rule:              M1.R.1 : Min M1 area coverage < 30%              1 error
Rule:                M1.S.1 : Min. M1 space < 0.23              182 errors
Rule:              M2.R.1 : Min M2 area coverage < 30%              1 error
Rule:                M2.W.1 : Min. M2 width < 0.28               44 errors
Rule:              M3.A.1 : Min M3 area region < 0.202              2 errors
Rule:              M3.R.1 : Min M3 area coverage < 30%              1 error
Rule:                M3.S.1 : Min. M3 space < 0.28              225 errors
Rule:              M4.A.1 : Min M4 area region < 0.202              3 errors
Rule:              M4.R.1 : Min M4 area coverage < 30%              1 error
Rule:                M4.S.1 : Min. M4 space < 0.28              194 errors
Rule:              M5.A.1 : Min M5 area region < 0.202              1 error
Rule:              M5.R.1 : Min M5 area coverage < 30%              1 error
Rule:              M6.R.1 : Min M6 area coverage < 30%              1 error
Rule:                VIA1.S.1 : Min. VIA1 space < 0.26              4 errors
Rule:                VIA2.S.1 : Min. VIA2 space < 0.26              6 errors
Rule:             VIA5.S.1 : Min. VIA5 spacing < 0.35               1 error
--------------------------------------------------------------------------------
```

Figura 36: Errores tercer escenario.

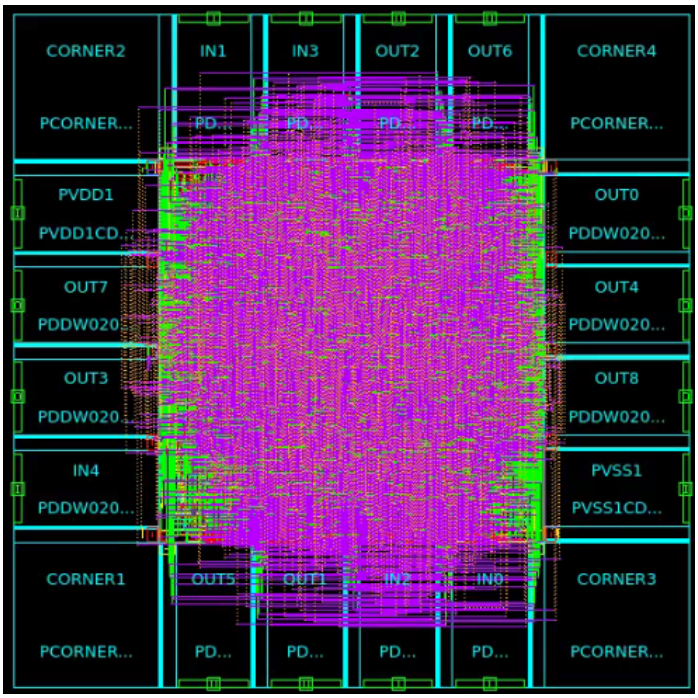


Figura 37: Síntesis física tercer escenario.

```
--------------------------------------------------------------------------------
Rule:                    M1.S.1 : Min. M1 space < 0.23              160 errors
Rule:                    M1.W.1 : Min. M1 width < 0.23                8 errors
Rule:                M2.A.1 : Min M2 area region < 0.202              1 error
Rule:                    M2.S.1 : Min. M2 space < 0.28             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:                    M2.W.1 : Min. M2 width < 0.28               60 errors
Rule:                M3.A.1 : Min M3 area region < 0.202             10 errors
Rule:                    M3.S.1 : Min. M3 space < 0.28              207 errors
Rule:                M4.A.1 : Min M4 area region < 0.202              7 errors
Rule:                    M4.S.1 : Min. M4 space < 0.28              238 errors
Rule:                M5.A.1 : Min M5 area region < 0.202              1 error
Rule:                M5.R.1 : Min M5 area coverage < 30%              1 error
Rule:                    M5.S.1 : Min. M5 space < 0.28             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:               M6.A.1 : Min. M6 area region < 0.562              1 error
Rule:                M6.R.1 : Min M6 area coverage < 30%              1 error
Rule:                    M6.S.1 : Min. M6 space < 0.46             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:                  VIA2.S.1 : Min. VIA2 space < 0.26              7 errors
Rule:                VIA5.S.1 : Min. VIA5 spacing < 0.35              7 errors
--------------------------------------------------------------------------------
```

Figura 38: Errores cuarto escenario.

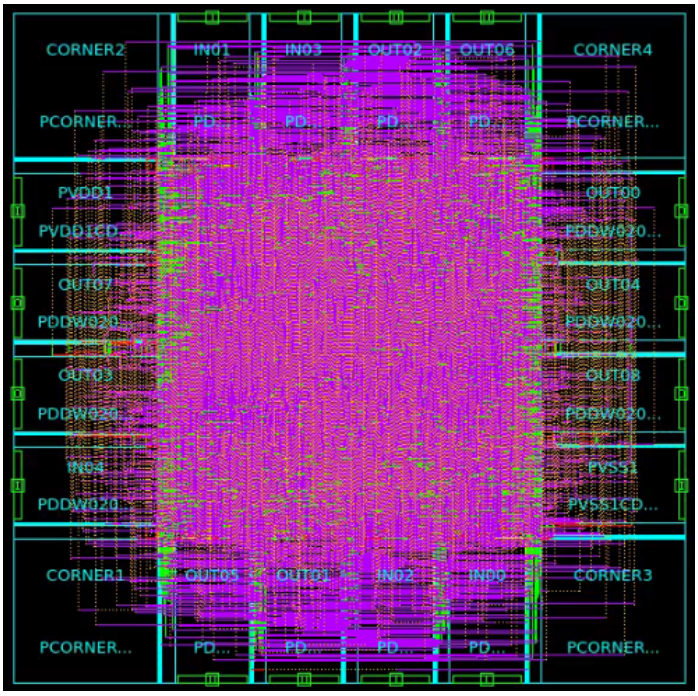


Figura 39: Síntesis física cuarto escenario

```
--------------------------------------------------------------------------------
Rule:                    M1.S.1 : Min. M1 space < 0.23              122 errors
Rule:                    M2.S.1 : Min. M2 space < 0.28             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:                    M2.W.1 : Min. M2 width < 0.28               82 errors
Rule:                M3.A.1 : Min M3 area region < 0.202              3 errors
Rule:                    M3.S.1 : Min. M3 space < 0.28              163 errors
Rule:                M4.A.1 : Min M4 area region < 0.202              2 errors
Rule:                    M4.S.1 : Min. M4 space < 0.28              141 errors
Rule:                M5.A.1 : Min M5 area region < 0.202              2 errors
Rule:                M5.R.1 : Min M5 area coverage < 30%              1 error
Rule:                    M5.S.1 : Min. M5 space < 0.28             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:                M6.R.1 : Min M6 area coverage < 30%              1 error
Rule:                    M6.S.1 : Min. M6 space < 0.46             1000 errors
  WARNING: Runset error limit exceeded for this rule, some errors omitted.
Rule:                  VIA1.S.1 : Min. VIA1 space < 0.26              5 errors
Rule:                  VIA2.S.1 : Min. VIA2 space < 0.26              1 error
Rule:                VIA2.W.1 : VIA2 must be 0.26 x 0.26              1 error
Rule:                VIA5.S.1 : Min. VIA5 spacing < 0.35              6 errors
--------------------------------------------------------------------------------
```

Figura 40: Errores quinto escenario.

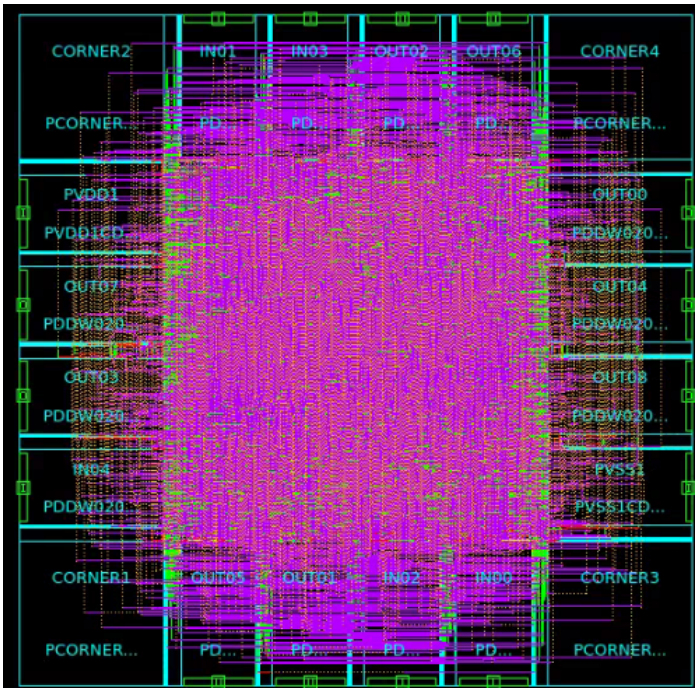


Figura 41: Síntesis física quinto escenario

```
-------------------------------------------------------------------------
Density Report Summary:
-------------------------------------------------------------------------
Window = {{0 0 534.48 532.24}}
--------------------
Layer     Density(%)
--------------------
METAL1    0
METAL2    0
METAL3    0
METAL4    0
METAL5    21.1881
METAL6    22.9467
```

Figura 42: Errores sexto escenario.

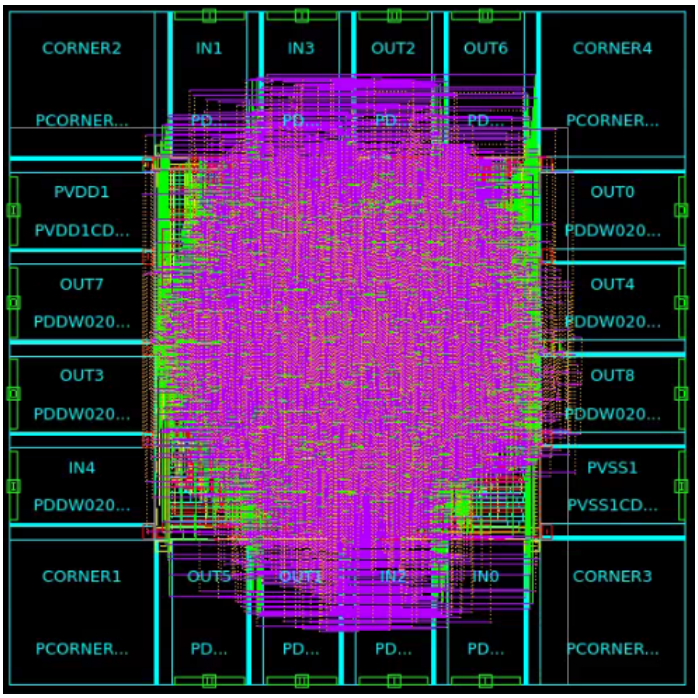


Figura 43: Síntesis física sexto escenario

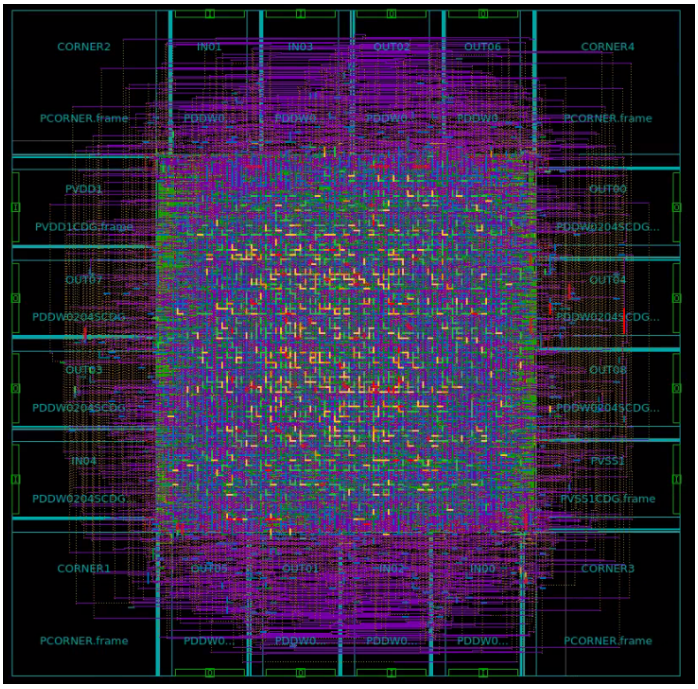


Figura 44: Síntesis física sin errores de densidad

CAPÍTULO 11

Conclusiones

- Se optimizó un *script* que genere un verilog para cualquier texto.
- Se actualizó el *script* de *python* 2 a *python* 3.
- Se optimizó el *script* existente para el proceso de síntesis física.
- Se generaron los archivos necesarios para realizar las verificaciones físicas como: DRC, ERC, LVS, antena y extracción de parásitos.
- Se validó que los archivos generados funcionan en las verificaciones físicas: DRC, ERC, LVS, antena y extracción de parásitos.
- Se identificó y corrigió los errores que se dieron en el proceso de automatización.
- Se integró la fase de síntesis lógica con la fase de síntesis física en la automatización.
- Se integró la fase de síntesis física con la fase de verificaciones en la automatización
- Se validó un *runset* para relleno de metal y con este se redujo a 2 los errores de densidad de metal.

CAPÍTULO 12

Recomendaciones

- Mejorar la comunicación con **IMEC** y **TSMC** para obtener los archivos necesarios como: *runset* de polisilicio y documentación.
- Mantener comunicación con las distintas fases de automatización, además de los grupos encargados de la creación del chip, ya que esto agiliza el proceso y se puede corregir errores logrando un tiempo de respuesta mejor.
- Explorar otro proceso para la creación de la interfaz gráfica, así como añadir más opciones que se pueda ejecutar desde la interfaz.
- Explorar el proceso para automatizar otras tecnologías como 90nm, 32nm y 14nm.
- Mejorar el flujo de síntesis física para obtener mejores resultados.
- Aprender sobre ruteo avanzado y reglas de diseño para mejorar el *script* de síntesis física.
- Realizar la traducción del *runset* `Dummy_OD_PO` utilizando la traducción con el comando cal2pxl.

CAPÍTULO 13

Bibliografía

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

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

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

[4] L. Abadía, *Posicionamiento e interconexión entre componentes de un circuito sintetizado para el flujo de diseño de un circuito a escala nanométrica utilizando la herramienta de IC Compiler*, Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

[5] M. Flores, *Corrección de anillo de entradas/salidas y pruebas de antenna y ERC para la definición del flujo de diseño del primer chip con tecnología nanométrica desarrollado en Guatemala*, Tesis de licenciatura, Universidad del Valle de Guatemala, 2020.

[6] M. Sibrian, *Verificación de reglas de diseño (DRC) para el desarrollo de un flujo funcional de un circuito integrado con tecnología nanométrica*, Tesis de licenciatura, Universidad del Valle de Guatemala, 2020.

[7] E. Torres, *Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: Ejecución y simulación para la etapa de síntesis lógica*, Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

[8] A. Altuna, *Diseño de un Circuito Integrado con Tecnología de 180nm usando Librerías de Diseño de TSMC: Ejecución de la Síntesis Física, Verificación de Reglas de Diseño y Corrección de Errores Obtenidos*, Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

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

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

[11] C. Ramey y B. Fox, *Bash Reference Manual*, 2020.

[12] G. van Rossum, *The Python Language Reference*, 2022.

[13] G. van Rossum, *The Python Library Reference*, 2022.

[14] Synopsys, *IC Compiler II*, 2022. dirección: `https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html`.

[15] Synopsys, *IC Compiler II™ Functional Safety Manual*, 2021.

[16] Synopsys, *Using Tcl With Synopsys Tools*, 2021.

[17] Synopsys, *Using the Synopsys Design Constraints Format Application Note*, 2017.

[18] Synopsys, *IC Compiler™ II Design Planning User Guide*, 2022.

[19] Synopsys, *IC Compiler™ II Data Model User Guide*, 2022.

[20] Synopsys, *IC Compiler™ II Implementation User Guide*, 2022.

[21] Synopsys, *IC Compiler™ II Timing Analysis User Guide*, 2022.

[22] Synopsys, *Synopsys Technology File and Routing Rules Reference Manual*, 2022.

CAPÍTULO 14

Anexos

14.1. *Script* para generar el verilog

```
#version para python 3
import binascii
texto = "Hola, soy El Gran Jaguar, el primer nanochip elaborado en
    Centroamerica por una Universidad. Desarrollado completamente en la
    Universidad del Valle de Guatemala por alumnos graduandos de Ingenieria
    en Electronica entre dos mil diecinueve y dos mil veintidos. El equipo
    encargado de mi creacion se conformo por: Carlos Alberto Esquit
    Hernandez, Jonathan Alberto de los Santos Chonay, Kurt Emmanuel Kellner
    Juarez, Luis Alberto Rivera Estrada, Luis Arturo Najera Vasquez, Steven
    Hiram Rubio Vasquez, Marvin Geovanni Flores Espino, Matthias Sibrian
    Illescas, Julio Enrique Shin Jo, Karol Sophia Cardona Polanco,  Luis
    Estuardo Abadia Lopez, Joel Andres Gonzalez Herrera, Jose Adrian Ayala
    Escobar, Jose Alejandro Ruiz Orozco, Juan Ricardo Giron Rubio, Charlie
    Ayenci Cruz Giron, Elmer Otoniel Torres Garza, Gerardo Cardoza, Antonio
    Altuna Hernandez, Carlos Andres Letona Torres, Helder Arnoldo Ovalle
    Barrios, Jose David Ponce del Cid, Israel Antonio Arevalo Gonzalez,
    Allison Estuardo Aguilar Chocooj, Paola Alejandra Mendizabal Batres,
    Diego Rodrigo Equite Pirir, Estuardo Geovany Mancio Ruballos, Stefan
    Alexander Schwendener Farrington y Luis Ricardo Gomez Velasquez. La idea
     de fabricar un CHIP nacio a inicios del dos mil diecinueve, donde los
    alumnos graduandos consiguieron el apoyo de Euro-practice para financiar
     gran parte de la fabricacion de El Gran Jaguar. Tambien a traves de
    IMEC se tuvo acceso a librerias de fabricacion implementadas por la
    Taiwan Semiconductor Manufacturing Company en un proceso de ciento
    ochenta nanometros. Desde entonces se hicieron importantes avances en
    esta linea de investigacion, obteniendo importantes conclusiones y
    recomendaciones para continuar con el proyecto. En el siguiente periodo,
     correspondiente a dos mil veinte, las investigaciones continuaron segun
     lo previsto para realizar un Design Flow adecuado, estos planes se
    vieron drasticamente afectados con la llegada del virus SARS-CoV-2,
    covid-19. Sin embargo, la universidad del Valle de Guatemala logro sobre
     llevar esta situacion dando a los investigadores las herramientas
    necesarias para continuar avanzando. En dicho periodo se lograron
    avances importantes en las fases de sintesis logica, sintesis fisica y
```

```
    las verificaciones LVS y DRC. No fue hasta en el 2021 donde se tuvieron
    avances cruciales del nanoChip puesto que se continuo con las
    recomendaciones de las cohortes anteriores. Sobre los avances obtenidos
    se puede mencionar la migracion de aplicaciones a versiones mas
    recientes y sofisticadas como IC Compiler II, Design Vision, PrimeSim y
    nuevas funcionalidades de IC Validator. Tambien en 2021 se logro
    realizar de forma exitosa las verificaciones fisicas de Antena, LVS y
    ERC quedando unicamente pendiente la solucion de seis errores de
    densidad provenientes de la verificacion DRC. Actualmente en el 2022,
    los alumnos trabajan en encontrar soluciones a los errores de densidad,
    errores en la generacion del deck resultado de la extraccion de
    parasitos. Al mismo tiempo se realizan pruebas fisicas en un FPGA con el
     fin de validar el resultado de la sintesis logica. Tambien se trabaja
    para automatizar todo el proceso logrado a lo largo de la linea de
    investigacion desde la instalacion y actualizacion de las aplicaciones
    de Synopsys hasta la simulacion final del circuito. El Gran Jaguar es un
     proyecto ambicioso que dara paso a una infinidad de oportunidades en la
     investigacion y desarrollo de nuevas tecnologias y aplicaciones en
    beneficio de la sociedad global y principlamente en la sociedad
    guatemalteca."
texto2 = "Hello, I am El Gran Jaguar, the first nanochip made in Central
    America by a University. Completely developed at the Universidad del
    Valle de Guatemala by undergraduate students of Electronic Engineering
    between 2019 and 2022. The team in charge of my creation was made up of:
     Carlos Alberto Esquit Hernandez, Jonathan Alberto de los Santos Chonay,
     Kurt Emmanuel Kellner Juarez, Luis Alberto Rivera Estrada, Luis Arturo
    Najera Vasquez, Steven Hiram Rubio Vasquez, Marvin Geovanni Flores
    Espino, Matthias Sibrian Illescas, Julio Enrique Shin Jo, Karol Sophia
    Cardona Polanco, Luis Estuardo Abadia Lopez, Joel Andres Gonzalez
    Herrera, Jose Adrian Ayala Escobar, Jose Alejandro Ruiz Orozco, Juan
    Ricardo Giron Rubio, Charlie Ayenci Cruz Giron, Elmer Otoniel Torres
    Garza, Gerardo Cardoza, Antonio Altuna Hernandez, Carlos Andres Letona
    Torres , Helder Arnoldo Ovalle Barrios, Jose David Ponce del Cid, Israel
     Antonio Arevalo Gonzalez, Allison Estuardo Aguilar Chocooj, Paola
    Alejandra Mendizabal Batres, Diego Rodrigo Equite Pirir, Estuardo
    Geovany Mancio Ruballos, Stefan Alexander Schwendener Farrington, and
    Luis Ricardo Gomez Velasquez. The idea of manufacturing a CHIP was born
    at the beginning of 2019, when graduating students obtained the support
    of Euro-practice to finance a large part of the manufacture of El Gran
    Jaguar. Also through IMEC there was access to manufacturing libraries
    implemented by the Taiwan Semiconductor Manufacturing Company in a
    process of one hundred and eighty nanometers. Since then, important
    advances have been made in this line of research, obtaining important
    conclusions and recommendations to continue with the project. In the
    following period, corresponding to two thousand and twenty, the
    investigations continued as planned to carry out and adequate Design
    Flow, these plans were drastically affected with the arrival of the SARS
    -CoV-2 virus, covid-19. However, the University of the Valley of
    Guatemala managed to overcome this situation by giving researchers the
    necessary tools to continue advancing. In this period, important
    advances were made in the phases of logical synthesis, physical
    synthesis and the LVS and DRC verifications. It was not until 2021 that
    crucial advances were made in the nanoChip since the recommendations of
    the precious cohorts were continued. Regarding the progress made, it is
    worth mentioning the migration of applications to more recent and
    sophisticated versions such as IC Compiler II, Design Vision, PrimeSim
    and new functionalities of IC Validator. Also in 2021, the physical
    verifications of Antenna, LVS and ERC were successfully carried out,
    with the only solution remaining for six density errors from the DRC
```

```
    verification. Currently in 2022, students work on finding solutions to
    density errors, errors in the generation of the deck resulting from the
    extraction of parasites. At the same time, physical tests are performed
    on an FPGA in order to validate the result of the logic synthesis. Work
    is also being done to automate the entire process achieved throughout
    the research line, from the installation and updating of Synopsys
    applications to the final simulation of the circuit. El Gran Jaguar is
    an ambitious project that will gice way to endless opportunities in the
    research and development of new technologies and applications for the
    benefit of global society and mainly in Guatemalan society."
#print len(texto)
#print len(texto2)
archivo = open("circuito.v", "w")
archivo.writelines("module chip_SP(q_out, reset, clk, EN, clk_s, select);\n"
    )

archivo.writelines("output [7:0] q_out;\n")
archivo.writelines("input reset;\n")
archivo.writelines("input clk;\n")
archivo.writelines("input [1:0]select;\n")

archivo.writelines("reg [11:0] contador;\n")
archivo.writelines("reg [7:0] q;\n")

archivo.writelines("input EN;\n")
archivo.writelines("output clk_s;\n")

archivo.writelines("wire W_1;\n")
archivo.writelines("wire W_2;\n")
archivo.writelines("wire W_3;\n")
archivo.writelines("wire W_4;\n")
archivo.writelines("wire W_5;\n")
archivo.writelines("wire W_6;\n")
archivo.writelines("wire W_7;\n")
archivo.writelines("wire W_8;\n")
archivo.writelines("wire W_9;\n")
archivo.writelines("wire W_10;\n")
archivo.writelines("wire W_11;\n")
archivo.writelines("wire W_12;\n")
archivo.writelines("wire W_13;\n")
archivo.writelines("wire W_14;\n")
archivo.writelines("wire W_15;\n")
archivo.writelines("wire W_16;\n")
archivo.writelines("wire W_17;\n")
archivo.writelines("wire W_18;\n")
archivo.writelines("wire W_19;\n")
archivo.writelines("wire clk_G;\n")

archivo.writelines("assign clk_s = clk_G;\n")

archivo.writelines("AND_2 U1(EN,EN,W_1);\n")
#archivo.writelines("AND_2 U1(EN,W_1,W_2);\n")
archivo.writelines("INV  U2(W_1,W_2);\n")
archivo.writelines("INV  U3(W_2,W_3);\n")
archivo.writelines("INV  U4(W_3,W_4);\n")
archivo.writelines("INV  U5(W_4,W_5);\n")
archivo.writelines("INV  U6(W_5,W_6);\n")
archivo.writelines("INV  U7(W_6,W_7);\n")
archivo.writelines("INV  U8(W_7,W_8);\n")
```

```
archivo.writelines("INV  U9(W_8,W_9);\n")
archivo.writelines("INV  U10(W_9,W_10);\n")
archivo.writelines("INV  U11(W_10,W_11);\n")
archivo.writelines("INV  U12(W_11,W_12);\n")
archivo.writelines("INV  U13(W_12,W_13);\n")
archivo.writelines("INV  U14(W_13,W_14);\n")
archivo.writelines("INV  U15(W_14,W_15);\n")
archivo.writelines("INV  U16(W_15,W_16);\n")
archivo.writelines("INV  U17(W_16,W_17);\n")
archivo.writelines("INV  U18(W_17,W_18);\n")
archivo.writelines("INV  U19(W_18,W_19);\n")
#archivo.writelines("INV  U20(W_19,clk_G);\n")
archivo.writelines("INV  U20(W_19,clk_G);\n")

archivo.writelines("always @ (posedge reset or posedge clk)\n")
archivo.writelines("if (reset)\n")
archivo.writelines("contador<=12'b000000000000;\n")
archivo.writelines("else if(select==2'b00 || select==2'b11) begin\n")
archivo.writelines("if (contador <"+str(len(texto)-1)+ ")\n")
archivo.writelines("contador <= contador + 1;\n")
archivo.writelines("else\n")
archivo.writelines("contador <= 0;\n")
archivo.writelines("end\n")
archivo.writelines("else if(select==2'b01 || select==2'b10) begin\n")
archivo.writelines("if (contador <"+str(len(texto2)-1)+ ")\n")
archivo.writelines("contador <= contador + 1;\n")
archivo.writelines("else\n")
archivo.writelines("contador <= 0;\n")
archivo.writelines("end\n")


archivo.writelines("always @ (posedge clk)\n")
archivo.writelines("if(select==2'b00 || select==2'b11)\n")
archivo.writelines("begin\n")
archivo.writelines("if(contador == 'd0)\n")
archivo.writelines("begin\n")
#archivo.writelines("q<=8'b00100000;\n")
archivo.writelines("q<=8'b")
binario=bin(ord(texto[0]))
for x in range(10-len(binario)):
  archivo.writelines("0")
for j in range(len(binario)-2):
  archivo.writelines(str(binario[j+2]))
archivo.writelines(";\n")
archivo.writelines("end\n")

for i in range(len(texto)-1):
  archivo.writelines("else if(contador == 'd"+str(i+1)+")\n")
  archivo.writelines("begin\n")
  archivo.writelines("q<=8'b")
  #print(ord(texto[i]))
  #binascii.hexlify(texto[i])ord(character)
  #binario = bin(int(binascii.hexlify(texto[i]),16))
  binario=bin(ord(texto[i+1]))
  #print len(binario)
  #print binario
  for x in range(10-len(binario)):
    archivo.writelines("0")
  for j in range(len(binario)-2):
```

```
112     archivo.writelines(str(binario[j+2]))
113   archivo.writelines(";\n")
114   archivo.writelines("end\n")
115   #print(bin(int(binascii.hexlify(texto[i]),16)))
116 archivo.writelines("end\n")
117
118 archivo.writelines("else if(select==2'b01 || select==2'b10)\n")
119 archivo.writelines("begin\n")
120 archivo.writelines("if(contador == 'd0)\n")
121 archivo.writelines("begin\n")
122 #archivo.writelines("q<=8'b00100000;\n")
123 archivo.writelines("q<=8'b")
124 binario=bin(ord(texto2[0]))
125 for x in range(10-len(binario)):
126   archivo.writelines("0")
127 for j in range(len(binario)-2):
128   archivo.writelines(str(binario[j+2]))
129 archivo.writelines(";\n")
130 archivo.writelines("end\n")
131
132 for i in range(len(texto2)-1):
133   archivo.writelines("else if(contador == 'd"+str(i+1)+")\n")
134   archivo.writelines("begin\n")
135   archivo.writelines("q<=8'b")
136   #binario = bin(int(binascii.hexlify(texto2[i]),16))
137   #print len(binario)
138   binario=bin(ord(texto2[i]))
139   for x in range(10-len(binario)):
140     archivo.writelines("0")
141   for j in range(len(binario)-2):
142     archivo.writelines(str(binario[j+2]))
143   archivo.writelines(";\n")
144   archivo.writelines("end\n")
145   #print(bin(int(binascii.hexlify(texto2[i]),16)))
146 archivo.writelines("end\n")
147 archivo.writelines("assign q_out = q;\n")
148 archivo.writelines("endmodule\n")
149
150 archivo.writelines("module INV(A, B);\n")
151 archivo.writelines("input A;\n")
152 archivo.writelines("output B;\n")
153 archivo.writelines("assign B = ~A;\n")
154 archivo.writelines("endmodule\n")
155
156 archivo.writelines("module AND_2(in1, in2, out);\n")
157 archivo.writelines("input in1, in2;\n")
158 archivo.writelines("output out;\n")
159 archivo.writelines("assign out = in1 & in2;\n")
160 archivo.writelines("endmodule\n")
161
162 archivo.close()
163
164 #for i in range(len(texto)):
165 # print(bin(int(binascii.hexlify(texto[i]),16)))
```

14.2. *Script* síntesis física

```
file delete -force  EL_GRAN_JAGUAR.ndm
set command_log_file "./Logs/command.log"
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
create_lib EL_GRAN_JAGUAR.ndm -technology /home/nanoelectronica/Desktop/
    Folder_de_Trabajo/Util/techfiles/tsmc018_6lm.tf \
 -ref_libs ./Libs/LibreriasNDM/TSMCWorkspace.ndm

#Abrimos el verilog file sintetizado

read_verilog ./Inputs/out_final_io.v

read_sdc -echo ./Inputs/out_final_io.sdc

#Importamos las TLU+ y el map
read_parasitic_tech -tlup ./Libs/tcb018gbwp7t_290a_FE/tluplus/
    t018lo_1p6m_typical.tluplus \
-layermap ./Libs/tcb018gbwp7t_290a_FE/tluplus/star.map_6M

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

#antenna
source -echo -verbose "./antennaRule_018_6lm.tcl"
#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

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

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

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

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

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

#Creacion del anillo de PG
create_pg_ring_pattern ring_pattern -horizontal_layer METAL2
```

```
    -horizontal_width {8} -horizontal_spacing {2}    -vertical_layer METAL3
    -vertical_width {8} -vertical_spacing {2}
set_pg_strategy core_ring    -pattern {{name: ring_pattern}    {nets: {VDD
    VSS}} {offset: {1 1}}} -core
compile_pg -strategies core_ring


#anillo layer 5 y 6
create_pg_ring_pattern ring_pattern -horizontal_layer METAL6
    -horizontal_width {8} -horizontal_spacing {2}    -vertical_layer METAL6
    -vertical_width {8} -vertical_spacing {2}
set_pg_strategy core_ring    -pattern {{name: ring_pattern}    {nets: {VDD
    VSS}} {offset: {2 2}}} -core
compile_pg -strategies core_ring
#creamos conexion IO ejemplo 2 -EL MEJOR EJEMPLO-
create_pg_macro_conn_pattern hm_pattern -pin_conn_type scattered_pin -layers
     {METAL2 METAL3} -nets {VDD VSS} -pin_layers {METAL2}
set_app_options -name plan.pgroute.treat_pad_as_macro -value true
set_pg_strategy macro_conn -macros [get_cells {PVDD* PVSS*}] -pattern {{
    name: hm_pattern} {nets: {VDD VSS}}}
set_pg_strategy_via_rule macro_conn_via_rule -via_rule {{{{strategies:
    macro_conn}}{{existing: all} {layers: METAL3}} {via_master: default}} {{
    intersection: undefined}{via_master: NIL}}}
compile_pg -strategies macro_conn -via_rule macro_conn_via_rule -tag test

#Creamos el mesh del circuito para VDD y VSS
connect_pg_net -automatic
create_pg_mesh_pattern mesh_pattern -layers {{{horizontal_layer: METAL3} {
    width: 6}{pitch: 42} {spacing: interleaving}}}
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {409.480 414.240}}
     -pattern {{pattern: mesh_pattern}{nets: {VDD VSS}}} -blockage {macros:
    all}
create_pg_std_cell_conn_pattern std_cell_pattern
set_pg_strategy std_cell_strategy -core -pattern {{pattern: std_cell_pattern
    }{nets: {VDD VSS}}}
compile_pg

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

#Creamos el placement
set_app_options -name place.coarse.fix_hard_macros -value false
set_app_options -name plan.place.auto_create_blockages -value auto
create_placement -floorplan -timing_driven -congestion -effort high
    -congestion_effort high
legalize_placement

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

#Ruteamos
check_routability -check_pg_blocked_ports true
check_design -checks pre_route_stage
route_auto
```

```
95
96 #Creamos los filler del core y el IO ring
97 create_io_filler_cells -io_guides [get_io_guides {anillo_IO_JAGUAR.top
       anillo_IO_JAGUAR.right anillo_IO_JAGUAR.left anillo_IO_JAGUAR.bottom}]
       -reference_cells {PFILLER1 PFILLER5 PFILLER05 PFILLER0005 PFILLER10
       PFILLER20}
98 create_stdcell_fillers -lib_cells [get_lib_cells {TSMCWorkspace|
       FillersWorkspace/FILL64BWP7T TSMCWorkspace|FillersWorkspace/FILL32BWP7T
       TSMCWorkspace|FillersWorkspace/FILL16BWP7T TSMCWorkspace|
       FillersWorkspace/FILL8BWP7T TSMCWorkspace|FillersWorkspace/FILL4BWP7T
       TSMCWorkspace|FillersWorkspace/FILL2BWP7T TSMCWorkspace|FillersWorkspace
       /FILL1BWP7T}]
99 connect_pg_net -automatic
100 remove_stdcell_fillers_with_violation
101 check_legality
102
103 # Configuramos el DRC runset file
104 set_app_options -list {signoff.check_design.run_dir {./Outputs/DRC/}}
105 set_app_options -list {signoff.check_drc.run_dir {./Outputs/DRC/}}
106 set_app_options -list {signoff.create_metal_fill.run_dir {./Outputs/Fill/}}
107 #set_app_options -list {signoff.create_metal_fill.run_dir {/home/
       nanoelectronica/Desktop/Trabajos/FS/Fill/}}
108 set_app_options -list {signoff.check_design.runset {
       ICVLM18_LM16_LM152_6M.215a_pre041518}}
109 set_app_options -list {signoff.check_drc.runset {
       ICVLM18_LM16_LM152_6M.215a_pre041518}}
110 set_app_options -list {signoff.create_metal_fill.runset {
       saed90nm_1p9m_mfill_rules.rs}}
111 #set_app_options -list {signoff.create_metal_fill.fix_density_errors true}
112 signoff_create_metal_fill
113 #Guardamos el bloque y corremos DRC y su Fix
114 save_block EL_GRAN_JAGUAR.ndm:chip_IO
115 signoff_fix_drc
116 save_block EL_GRAN_JAGUAR.ndm:chip_IO
117 signoff_check_drc
118 check_lvs
119 save_block EL_GRAN_JAGUAR.ndm:chip_IO
120
121 #Creamos el verilog equivalente de la sintesis fisica
122 write_verilog -include all ./Outputs/EL_GRAN_JAGUAR.v
123
124 #gds
125 write_gds -library EL_GRAN_JAGUAR.ndm -design chip_IO -view design
       -hierarchy all -lib_cell_view frame ./Outputs/chip.gds
126 exit
```

14.3. *Script* automatización síntesis física

```
#!/bin/bash
#codigo para automatizar la sintesis fisica
echo ingrese el valor de side legth x
read var1
echo ingrese el valor de side legth y
read var2
echo ingrese el valor de core offset
read var3
#echo initialize_floorplan -site_def unit -use_site_row -keep_all -
    side_length {$var1 $var2} -core_offset {$var3}
sed "s/initialize_floorplan -site_def unit -use_site_row -keep_all -
    side_length {285 285} -core_offset {125}/initialize_floorplan -site_def
    unit -use_site_row -keep_all -side_length {$var1 $var2} -core_offset {
    $var3}/"<plantilla.tcl >scriptfs.tcl
icc2_shell -file scriptfs.tcl
```

```
#!/bin/bash
rm /inputs/version1/out_final_io.ddc /inputs/version1/out_final_io.sdc /
    inputs/version1/out_final_io.v
mv /inputs/out_final_io.ddc /inputs/out_final_io.sdc /inputs/out_final_io.v
    /version1
mv /inputs/outchipio.ddc /inputs/out_final_io.ddc
mv /inputs/outchipio.sdc /inputs/out_final_io.sdc
mv /inputs/outchipio.v /inputs/out_final_io.v
icc2_shell -file plantilla.tcl
mv *.log /logs
mv *.svf /logs
mv *.txt /logs
```