Diseño de circuito integrado con tecnología 180 nm usando librerías de diseño de TSMC: Ejecución de extracción de componentes parásitos y automatización del flujo de diseño

Joel Andrés González Herrera

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de circuito integrado con tecnología 180 nm usando librerías de diseño de TSMC: Ejecución de extracción de componentes parásitos y automatización del flujo de diseño

Trabajo de graduación presentado por Joel Andrés González Herrera
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño de circuito integrado con tecnología 180 nm usando librerías de diseño de TSMC: Ejecución de extracción de componentes parásitos y automatización del flujo de diseño

Trabajo de graduación presentado por Joel Andrés González Herrera
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

Vo.Bo.:

(f) _____

Ing. Jonathan de los Santos Chonay

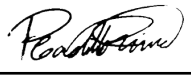
Tribunal Examinador:

(f) _____

Ing. Jonathan de los Santos Chonay

(f) _____

MSc. Carlos Esquit



(f) _____

Ing. Pedro Iván Castillo

Fecha de aprobación: Guatemala, 22 de junio de 2022.

Prefacio

Primeramente, quiero agradecerle a la Universidad del Valle de Guatemala que me ha estado dando una educación estelar durante estos años, siempre poniendo mi desarrollo como primera prioridad y dándome acceso a oportunidades, enseñanzas y seguridad durante mi tiempo de estudio. También se le agradece a la compañía de TSMC, cuyo trabajo, documentación y apoyo han sido inmensos para estas investigaciones y el avance tecnológico de Guatemala. Gracias a ambos, se tiene la oportunidad de realizar un proyecto que marca nuestras vidas.

Quiero agradecerle al ingeniero Jonathan de los Santos, quien apoyó este proyecto como asesor e hizo todo lo posible para cumplir este proyecto demandante con nosotros. Su esfuerzo y enseñanza nos iluminó como estudiantes en esta nueva frontera, permitiéndonos madurar como ingenieros, delvallerianos y como miembros de nuestro país. También agradezco a nuestro director de carrera, MSc. Carlos Esquit, cuyos esfuerzos para avanzar la educación en nuestro país y nuestra región nos han dado constantes oportunidades de aprendizaje.

Finalmente, agradezco a mi familia por su enorme apoyo a lo largo de mi vida. De ellos he aprendido las lecciones más cruciales, y espero con todo mi ser poder hacer bien con lo que ellos me han dado.

Índice

Lista de figuras

Lista de cuadros

Listings

Resumen

El trabajo a continuación reporta sobre la investigación y el desarrollo del trabajo de graduación sobre la verificación y la automatización del flujo de diseño utilizado para el diseño del primer chip nanométrico de la región. Este documento sirve como reporte general de lo que fue desarrollado durante este periodo. Este imparte la metodología, decisiones, y ejecución de la verificación LPE y la automatización. Esta primera fase de verificación es posible por medio del proceso de extracción de componentes parásitos, resumido como LPE por sus siglas en inglés. Tomando referencias de los trabajos anteriores, junto con los recursos proveídos por *Synopsys*, se demuestra la aplicación y validez de la propuesta de un diseño propuesto. El segundo objetivo primario es de establecer y ejecutar por primera vez un método para automatizar dicho flujo de diseño con estas herramientas. El propósito siendo demostrar y probar este método, uno que simplifique el proceso y sea realizable por un solo usuario. El resultado final de este proceso son los archivos válidos y preparados de fabricación. En este trabajo se encuentra un reporte general del método que fue implementado para dichos objetivos.

Abstract

The following work reports on the research and development of the graduation project regarding the verification and automation of a design flow utilized to develop the first nanometric chip of the region. This document serves as a general report over what was developed during this period. It outlines the methodology, the decisions, and execution of the LPE verification and automation. The mentioned first phase of verification is possible through a Layout Parasitic Extraction, summarized as LPE. Taking references from previous works, along with the resources provided by *Synopsys*, the application and validity of the proposed design is shown. The second primary objective is to establish and execute for the first time a method for automating said design flow with these tools. The purpose being to demonstrate and prove the method for simplifying the process as much as possible for a user. The final result of this process are the verified and fabrication-ready files. In this work is a report over the method that was implemented for said objectives.

CAPÍTULO 1

Introducción

La ciencia de chips electrónicos ha existido por un tiempo significativamente más corto al de otras ciencias. Esto se puede atribuir principalmente a su dependencia inicial de otras ciencias. Métodos y aplicaciones de química, ciencia y física fueron requeridos para iniciar la ciencia de la electrónica. Sin embargo, aún con la corta duración de avance que ha tenido esta tecnología, ha llevado a cambios vitales en todos los ámbitos. Desde las calculadoras que se han vuelto parte de las herramientas básicas en toda clase de investigación, a las computadoras más avanzadas que realizan las simulaciones complejas de estos mismos ámbitos. Claramente, toda la electrónica y, específicamente los chips, han llevado a grandes revoluciones en todos los campos de investigación imaginables.

En particular, la nanoelectrónica ha sido un área de evoluciones constantes. La famosa ley de Moore ha ejemplificado la velocidad de avances que se dan en esta ciencia, llevando a soluciones más y más sofisticadas para los problemas a pesar de ya proveer parámetros aceptables. Sin embargo, eso nunca ha sido suficiente. El deseo y el incentivo de mejores soluciones han llevado a este campo de ciencia a tener tantas evoluciones y revoluciones que se podría argumentar que ha tenido el mayor impacto en la vida diaria de la sociedad.

El resultado de este esfuerzo acumulativo de generaciones completas sirve como la parte central de los celulares, televisores, controles, e incluso en la computadora que fue utilizada para escribir este trabajo. Vive en la tecnología más común y más avanzada de estos tiempos.

Este es el contexto del campo al que se desea contribuir con todos los trabajos de investigación acompañando al presente documento. Guatemala como país se ha desarrollado al invertir en sus generaciones más jóvenes, aprendiendo las lecciones de países que están en una etapa avanzada, y adaptándose para tiempos nuevos. El propósito primario de estos esfuerzos siendo educativos, desarrollando las herramientas y métodos que serán dados en salones de esta universidad. Estos métodos serán dados a las generaciones después de la presente.

CAPÍTULO 2

Antecedentes

Empezando en el año 2009, la facultad de ingeniería de la Universidad del Valle de Guatemala inicia esfuerzos para actualizar y expandir aprendizaje de tecnología nanométrica. Con la ocupación del MSc. Carlos Esquit en el puesto de director de la carrera de ingeniería electrónica, se expande los requisitos para dicha carrera. Para 2013, se da por primera vez el curso llamado Introducción al Diseño de Sistemas VLSI.

Durante el año 2019, se recibe la oportunidad de fabricar un circuito integrado en un chip con tecnología nanométrica. Esta oportunidad es gracias a Taiwan Semiconductor Manufacturing Company (de este punto en adelante llamado *TSMC* por sus iniciales). Este trabajo es continuación del trabajo realizado previamente por estudiantes de la Universidad del Valle.

El estudio para la definición del flujo de diseño se puede encontrar en el trabajo hecho por Steven H. Rubio. [13]

El primer paso de este flujo de diseño, específicamente la implementación de circuitos integrados desde lenguaje descriptivo fue visto el trabajo hecho por Luis A. Nájera. [11]

El uso de la herramienta VCS para la simulación fue detallado en el trabajo escrito por Jefferson N. Ruano. [12]

Adicionalmente, se vio la extracción de componentes parásitos en el trabajo escrito, en el mismo proyecto, por Charlie A. Cruz. [5]

Una verificación detallada de las reglas de diseños utilizadas en el proyecto fue hecha por Matthias Sibrian. [17]

Juan R. Girón realizó una verificación del diseño en silicio contrastado con el esquemático para este chip. [7]

También se realizó una corrección al anillo de entradas y salidas del chip y pruebas de antena para el flujo de diseño del chip. Esta fue completada por Marvin G. Flores. [6]

Finalmente, la línea de trabajo hecha con este propósito de utilizar estas herramientas para un flujo de diseño inicia con Jonathan de los Santos, quien durante el 2014 completó un sumador/restador de 32 bits con dichas herramientas. [15]

Como ha sido mencionado, este reporte se enfoca fuertemente en la automatización del flujo de diseño. Este es el primer trabajo con este enfoque, basándose en el esfuerzo hecho por todas las personas arriba y referidas que han desarrollado estos trabajos.

CAPÍTULO 3

Justificación

El trabajo para hacer un chip a estas escalas no es simple. Proporcional a la sofisticación necesaria en los chips, existe un proceso igualmente detallado para encontrar, filtrar o minimizar todo tipo de errores en la implementación, diseño, construcción, fabricación y aplicación de los circuitos finales. Esto incluye también los requisitos del proceso de fabricación, que existen fuera del alcance de este trabajo. Por ejemplo, estos incluyen las condiciones de limpieza necesarios para asegurar el funcionamiento de la solución propuesta.

Como sería evidente, este proceso de fabricación solamente existe después del proceso de diseño. Este se dedica a la implementación de los métodos desarrollados y resulta en un diseño de circuito que cumple todos los requisitos del problema, todas las reglas de diseño, todas las regulaciones requeridas, incluso con sobre-protecciones para asegurar que el proceso completo no sea en vano. Un ejemplo simple de esto es el requisito de distancia entre vías, o en otras palabras, entre conductores. Sin esta clase de regulación, el circuito fácilmente se podría encontrar con un cortocircuito que resulta en una falla completa del circuito.

En este trabajo se prioriza la vista de verificación y automatización del proceso de diseño. Estas dos son partes extremadamente importantes para el desarrollo de las aplicaciones de estas herramientas. La verificación por medio de la extracción de parásitos se presenta como una vista global del comportamiento del circuito. En especial, se establece que capacitancias, resistencias e inductancias parásitas que resultarían por medio del diseño presente de un layout, y que estas no hayan drásticamente alterado o anulado el comportamiento deseado del sistema físico. Estas simulaciones finales podrán validar el diseño como tal, dejando ninguna duda que el layout se comportaría en la realidad como fue planificado.

Segundo, se ve la automatización del proceso general del flujo de diseño. Para esto, se requiere una revisión del sistema completo y de cada fase de diseño, con apoyo y comunicación con los demás grupos de trabajo a fin de establecer el procedimiento de las secciones más específicas del proceso. Estas herramientas, dadas como secciones de cada fase, podrán facilitar el estudio de esta tecnología, estableciendo una ruta más directa con propósitos tanto educativos como de industriales.

CAPÍTULO 4

Objetivos

4.1 Objetivo general

Realizar el proceso de extracción de parásitos en el *Layout* final del chip diseñado y junto con todos los grupos de trabajo con una comunicación constante y efectiva, automatizar el flujo de diseño de tecnología nanométrica mediante scripts, facilitando la elaboración de trabajos futuros.

4.2 Objetivos específicos

- Ejecutar el proceso de extracción de parásitos sobre el *layout* final del chip diseñado, proveído por el último grupo de trabajo de síntesis física.
- Entablar un método para la producción de archivos para simulación en *HSPICE* cuando se esté trabajando con librerías originales. Este método permitirá verificar el funcionamiento del sistema resultante de los circuitos al extraer los componentes parásitos.
- Explorar las herramientas de diseño con Synopsys para la creación de scripts de comandos que sirvan para el avance del proceso, tanto para facilitar casos generales como casos particulares, facilitando *debugging* y sus verificaciones.
- Facilitar el proceso en cada fase de flujo de diseño, de tal manera que se pueda apoyar al resto del grupo en su diseño.
- Automatizar de manera independiente cada fase del flujo de diseño, permitiendo la revisión de errores para generalizar cada caso de diseño.
- Integrar todas las fases del flujo de diseño para desarrollar un *Design Flow* automatizado.

CAPÍTULO 5

Alcance

Debido a la naturaleza de este trabajo y el hecho que este fue hecho en conjunto, este reporte se dedica principalmente a explicar el proceso de automatización y ve la extracción de parásitos del chip propuesto. El funcionamiento detallado de cada proceso interno al diseño de flujo se encuentra en los reportes respectivos de los trabajos que acompañan a este.

En orden de su posición en el flujo de diseño: El proceso de síntesis lógica encuentra en el trabajo de Elmer Torres. La síntesis física con su verificación en DRC y reglas de antena se encuentra en los trabajos de Antonio Altuna, Julio Shin, y José Ayala. La comparación de los diseños de Layout contra el esquemático (LVS) fue visto por José Ruiz. Finalmente, la verificación de síntesis lógica al igual que la verificación final de extracción de parásitos se encuentra en la investigación en proceso por Gerardo Cardoza.

Este trabajo se limitará a explicar la sección de este proyecto que está fuera del alcance de dichos trabajos, por lo que sería recomendable tener estos trabajos a mano para complementar cualquier modificación. Con eso dicho, este reporte puede operar como una introducción o reporte general de todo el proceso de diseño. En otras palabras, en este reporte se encuentra una descripción de todo lo que fue completado durante este proyecto con el propósito de automatizar el trabajo que se realizó anteriormente. Este reporte, complementado por el trabajo especifico hecho por cada uno de los compañeros mencionados, darán una vista detallada de lo que fue trabajado por los estudiantes de ingeniería electrónica involucrados en este proyecto.

Sin embargo, este reporte no estaría completo sin describir lo que hace cada proceso en el flujo de diseño. Por lo que este trabajo se dedicará a dar una descripción completa de lo que se pretende realizar en cada proceso, dando el método de automatización y las opciones utilizadas para cada proceso mencionado. Por lo tanto, en este trabajo, se pretende replicar los procesos hechos por los compañeros, y no se mejoran. Las recomendaciones de cada compañero de trabajo fueron tomadas en cuenta, pero la mejora de los mismos se dejan al siguiente grupo de trabajo. Por medio de todos los reportes hechos, será posible darles un panorama completo del proyecto.

Como parte del flujo a ser completado, también se detalla el proceso de Layout Parasitic Extraction (LPE), junto con los problemas encontrados y sus soluciones propuestas. Esta sección se puede considerar una continuación del trabajo hecho el año anterior por Charlie Cruz. Este trabajo también propone soluciones a los problemas principales encontrados por nuestro compañero Cruz.

Y finalmente, se detallará el proceso de automatización llevado por el equipo. Esto servirá con el motivo principal de facilitar futuras pruebas, habiendo hecho la implementación y con este trabajo conteniendo explicaciones de como se implementó cada sección.

CAPÍTULO 6

Marco teórico

El diseño de un chip nanométrico requiere investigación y dominio de varias herramientas para ejecutar correctamente el flujo de diseño. Y estas herramientas requieren dominio en sus respectivas áreas y detalles para poder explicar sus objetivos y lo que realizan en este proyecto. Existen múltiples temas que se deben investigar para poder analizar esta tecnología. Entre ellos, se incluyen los transistores *MOSFET*, *VLSI*, y fabricación. Es necesario tener el mismo entendido de los temas que forman la base de este proyecto y de los proyectos que lo complementan.

6.1. MOSFETs

La construcción de nanochips y las compuertas lógicas que se han visto es principalmente gracias a los *MOSFETS*. Su construcción es parecida a los transistores *JETS*. Se utiliza el campo generado por un voltaje en la terminal de *gate* que crea un canal conductivo en el cual pueden fluir los electrones de *source* a *drain*. Estos, acumulados y en conjunto, crean una red de señales lógicas que realizan el procesamiento necesario para chips.

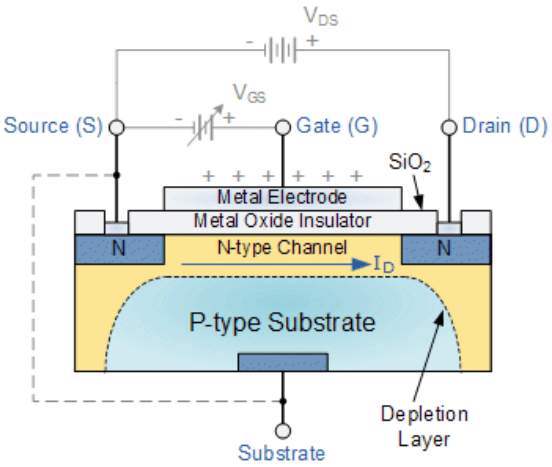


Figura 1: Diagrama de MOSFET [2]

Esta tecnología diminuta es la sección fundamental que permite la lógica necesaria, como el elemento más fundamental, este representa la vista más detallada de la tecnología en estos chips. Esta revolución en tecnología permite el procesamiento a gran escala que se ve en el resto de los chips. Este elemento, por medio de la fabricación que se detalla a continuación, puede ser construido en *waffers* y con una suficiente exactitud, resultan en los dispositivos poderosos y diminutos que se ven hoy en día. [2]

6.2. Very Large-scale Integration (VLSI)

VLSI se define simplemente como la creación de un circuito integrado en un chip, utilizando compuertas CMOS. Primeramente descrito con MOSFETs en *Fairchild Electronics* por Frank Wanlass y Chi-Tang Sah en 1963, estas compuertas lógicas fueron llamadas CMOS. Estas forman la base de los electrónicos de hoy. Esto es debido al costo reducido de producción y el menor consumo de energía dado por esta tecnología.[10]

Esta velocidad del avance tecnológico ha acelerado por décadas. Gordon Moore es famoso y reconocido el día de hoy, no solo por fundar Intel, sino por la observación de *Moore´s Law* que dice que *el número de transistores en un circuito integrado se duplica cada dos años.* La exactitud de dicha predicción considera valiosa. Sin embargo, esta exactitud es secundaria a la observación del aceleramiento que habría en la tecnología mientras fuera avanzando.[4]

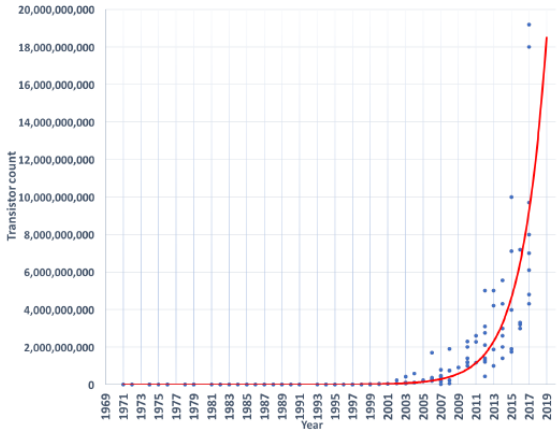


Figura 2: Ley de Moore [4]

Diseño VLSI a apoyado este crecimiento. En la misma área en la que se ingresaban miles de transistores MOS, se ha desarrollado para poder ingresar decenas de miles, millones, decenas de millones. Actualmente, se ha alcanzado 1 000 millones de transistores MOS en un solo circuito integrado. [10].

6.3. Flujo de diseño

El siguiente tema con relevancia es el flujo de diseño. Este flujo se refiere al proceso de diseño e implementación que ha sido estandarizado para esta clase de tecnología. Esta estandarización está hecha para filtrar todo tipo de errores, tanto de aplicación, de lógica, de corto, de falsos contactos, de ruteo, y cualquier otro tipo de error relevante. Este proceso incluso toma en cuenta la posibilidad de error en la fabricación. Esto se debe a que existen varias reglas de diseño que pueden ser consideradas sobre-restringidas. Esto tiene una razón. Estas pretenden garantizar que la solución final operará correctamente dentro el margen de error que puede existir en la fabricación.

Este proceso de diseño se puede dividir en dos partes: Fase *Front-end*, que se refiere a la síntesis lógica y a nivel de aplicación. La segunda parte se le llama *Back-end*, que ve el funcionamiento estructural del sistema. [10]

6.3.1. Front-end Design

Una definición útil de la primera mitad del flujo de diseño puede considerar que esta se encarga del *método* o más bien la solución general del sistema. Este principalmente define el comportamiento del sistema, con el diseño específico y físico perteneciendo a la siguiente sección. Estas incluyen:

1. **Diseño lógico**

   Esta es la sección principal de esta mitad del flujo de diseño y es la más dependiente de asistencia de un diseñador. El propósito de un diseño es de resolver un problema.

Debido a esto, dicho diseño depende principalmente de una descripción del comportamiento del sistema propuesto. El resultado de este proceso va a ser un archivo o una serie de archivos *.v* o Verilog que describan el sistema a ser implementado. Este resultado debe cumplir con las especificaciones del problema y requiere ser verificado apropiadamente.[10]

2. **Síntesis lógica**

   Similar al diseño lógico, esta sección tiene como objetivo tomar el comportamiento de la sección anterior y llevarlos a una descripción estructural. Verilog tiene una facilidad de programación, en la que circuitos se pueden describir de manera que solamente describen comportamiento. Esto tiene su utilidad, pero esto no es utilizado directamente para el proceso de fabricación. Se requiere una definición exacta del sistema o más bien una solución detallada. En eso entra la síntesis lógica, que finaliza la implementación del diseño original. Existen herramientas que facilitan esta transición y tienen como salida un archivo verilog que realmente describen la solución propuesta, con base en una librería importada. Esta que no se limita a comportamiento, sino que detalla a los componentes que serán utilizados en este proceso. [9]

3. **Netlist de compuertas**

   Esta es una sección de vital importancia. La descripción de cualquier circuito Verilog no va a ser suficiente para empezar el proceso de *floor planning*. Para empezar, no se tienen los datos de como es cada compuerta lógica necesaria. Para esto se utiliza una librería. Las herramientas disponibles con Synopsys son capaces de llevar el archivo Verilog a un netlist que conecte el comportamiento con las librerías. En este caso, se utilizarían las librerías TSMC. Estas podrían ser ligadas directamente con layouts respectivos que cumplen los requisitos necesarios en el diseño. [11]

4. **Verificaciones**

   Esta sección, en lugar de ser la parte 4 del flujo de diseño, sería más útil considerarlo como una sección paralela al proceso. Estas verificaciones son un proceso iterativo, donde se realiza constantemente verificaciones de las partes anteriores. Existen tres principales verificaciones a realizar durante esta mitad del flujo.[7]

   La primera de estas es la verificación de funcionalidad, que podría considerarse extensión o prueba del Diseño Lógico. La segunda verificación relevante es la revisión de la síntesis lógica. Finalmente, la última verificación examina las *netlists* como resultados. Estas son partes vitales, operando como correcciones o *feedbacks* del sistema para corregir problemas del diseño desde estos pasos. Para sistemas complejos, es casi garantizado que habrá algún error que deba ser corregido antes de avanzar a los próximos pasos.

6.3.2. Back-end Design

Para la segunda mitad de este flujo de diseño, se ve el paso después del método de solución. Es decir, una solución detallada del circuito. En la sección anterior solamente se obtiene una idea general del resultado final. En cambio, en esta mitad, el resultado es el diseño completo de un layout que cumple los requisitos del problema, en otras palabras, el resultado es

un layout preparado para fabricación. Varios pasos de esta sección son simplemente pruebas o *checks* que sirven para asegurar que el layout pueda funcionar correctamente después de ser implementado.

1. **Síntesis física**

   A partir del esquemático original, se debe producir el layout, que sirve como una descripción de como serán colocados cada uno de los elementos necesarios para el circuito. En el caso de nanoelectrónica, este es un chip considerablemente pequeño, y por lo tanto, la fabricación debe ser precisa. Sin un trabajo preciso, sería imposible implementar esta clase de tecnología sin que un error fatal invalide la solución y por lo tanto, invalide el trabajo invertido en el mismo.

2. **Routing**

   Ruteo se refiere al proceso de crear los caminos conductivos necesarios para que se conecten los elementos como fueron propuestos en el esquemático. Parte de este proceso puede ser automatizado, utilizando la herramienta de auto-ruteo. Sin embargo, esta puede fallar en casos más complejos. Por lo que siempre se depende de un usuario que se asegure que se cumpla todas las reglas, y todos los caminos para hacerlo semejante a la solución propuesta en el esquemático. Las siguientes son pruebas utilizadas para encontrar problemas que no cumple el layout con el objetivo de arreglar las discrepancias y los detalles que podrían llevar a un error fatal del diseño.

3. **Design Rule Check**

   Como su nombre indica, DRC se refiere a la parte del proceso de diseño que cumple las reglas de diseño. Como fue mencionado, estas reglas son regulaciones necesarias, en ocasiones son sobre-protecciones que no siempre son vitales. Sin embargo, estas llegan a ser importantes en casos particulares y se debe diseñar tomando encuenta posibles errores en alimentación, voltajes, fabricación, capacitancias o cualquier otro cortocircuito que debe ser eliminado para que se cumpla lo que se busca.

4. **Layout versus Schematic**

   También llamado LVS, el propósito de este prueba es una comparación. El objetivo es encontrar toda diferencia existente entre una solución propuesta en el esquemático y diseño resultante. Esta comparación y su validez es vital para las siguientes secciones. Aunque un diseño de layout haya sido validado por la prueba DRC, eso no garantiza que sí es representativo del esquemático propuesto. Es precisamente en esta sección donde se valida la exactitud de *layouts* para representar la solución propuesta en la primera sección.

5. **Layout Parasitic Extraction**

   Como una de las últimas verificaciones, existe LPE. Una vez ya se valida el diseño con LVS y DRC, demostrando que se está hablando del mismo circuito funcional en ambos casos, se debe correr un proceso que transfiere el circuito de *layout* a un formato en el que puede ser simulado. Para esto, se utiliza *StarRC* para la generación y *HSPICE* para la simulación. Desde este punto, se puede simular un circuito exacto que corresponde al diseño final del plan de piso. Es por medio de esto que se podría hacer una verificación tanto visual, especialmente de comportamiento y función. Sin

esta sección, se toma el riesgo que existan capacitancias parásitas o algún elemento agregado que atrase o incluso invalide el comportamiento pedido del diseño.

CAPÍTULO 7

Metodología

Debido a la dicha complejidad de diseño que será inevitable en esta clase de tecnología, es importante desarrollar un librería clara de estrategias, recursos y herramientas que puedan ser utilizadas durante este proceso. Se ha descrito a detalle el flujo de diseño general utilizado para esta clase de aplicación. Es lógico asumir que existen compañías como *Synopsys* que han desarrollado herramientas para seguir este proceso de manera efectiva. Estas son descritas a continuación.

7.1. Herramientas de diseño

Desde cero, sería difícil diseñar un chip de una calidad satisfactoria. Esta limitación no se da solamente por las dificultades de fabricación, sino por los archivos y programas necesarios para poder ejecutar estas operaciones complejas.

Para lograr esto se basará en los servicios y programas proveídos por *Synopsys*. Dicha compañía se especializa en automatización de diseño de chips electrónicos con una gran número de aplicaciones. Especialmente desde el año 2014 a la actualidad, ha expandido sus servicios. Esta base es importante para avanzar en las áreas relevantes. Estos servicios y productos son la base de este proyecto. [25]

Los distintos servicios o herramientas tienen objetivos distintos. Algunos tienen operaciones similares. A continuación se encuentra una lista de los programas utilizados durante este trabajo:

7.1.1. Custom Compiler

CC es descrito como 'el corazón de la plataforma Synopsys de Diseño'. Este servicio es una herramienta de diseño completo de circuitos integrados. Por medio de diversas interfaces gráficas, este producto permite una implementación manual de cada parte del proceso necesario. Tanto como herramienta educativa y como implementación profesional, tiene una gran utilidad en su simplicidad para facilitar las soluciones más complejas que se desean implementar. Este visualiza y ejecuta cada parte del proceso necesario para un diseño exitoso. [21]

Desde el inicio de este proyecto, y específicamente desde la integración de nanotecnología al currículo de la Universidad del Valle, *Custom Compiler* es la primera herramienta que nuevos estudiantes llegan a utilizar. Este funciona como un *hub* o una centralización de los servicios a continuación. Por medio de este, que hace las llamadas apropiadas a los programas individuales, permite que un usuario familiarizado con sus funciones pueda ejecutar cada sección del flujo de diseño de manera manual.

7.1.2. Design Vision

Síntesis lógica, como fue mencionado anteriormente, debe realizar una transición de una programación de *verilog* a un programa que describe este circuito original en base de los componentes de librería que puedan ser utilizados en la síntesis física. En esta primera fase, es necesario tener una herramienta que pueda analizar una propuesta codificada en *Verilog* y encontrar elementos equivalentes en la librería de uso que pueda ser implementado en un *Layout*. Esta es una parte esencial de la automatización, debido a que el procedimiento sería ineficiente si fuera necesario buscar manualmente cada una de estas equivalencias. Otra opción, que sería incluso menos eficiente, necesitaría la construcción de cada elemento, cada *gate*, y cada *layout* de manera manual cada vez. Debido a que esto claramente no sería realizable para la mayoría de casos, Se presenta la necesidad de una herramienta que procese la síntesis lógica para informar a las siguientes secciones del plan de diseño con los elementos necesarios. Esta es la función de *Design Vision*.[8]

Durante este proyecto, gracias al trabajo de Elmer Torres, se pudo obtener un método directamente de *DV* que permite realizar la síntesis lógica desde una interfaz gráfica. [26] Se pretende generalizar este método para poderlo completar solamente con comandos, desde una terminal.

Con el motivo de mejor explicar el procedimiento, en la siguiente página se encuentra el Cuadro 1. Este es un desglose de los archivos principales del proceso. Este no incluye archivos de referencias generales como librerías en este proceso. Tampoco incluye archivos de registro o *logs* que se abren en el transcurso del programa.

Por supuesto, debido a la necesidad de agregarle puertos de entrada y salidas, esto quiere decir que este proceso debe correrse dos veces. Una vez para realizar la transición del circuito principal, y la segunda para tomar la salida de esta primera corrida, y agregarle los puertos de entradas y salidas.

Cuadro 1: Archivos de *Design Vision* para ejecución de síntesis lógica

Nombre y extensión	**Uso**	**Descripción**
Verilog (.v)	Entrada	Archivo describiendo el circuito de entrada para síntesis lógica.
DVLogSyn (.script)	Script	Script describiendo instrucciones a correr incluyendo la importación de entradas.
Verilog final (.v)	Salida	Archivo describiendo el circuito de entrada ya puesto en términos de la librería solicitada.

7.1.3. IC Compiler

ICC I y II son conocidos como soluciones de *place-and-route*. Se han presentado como la mejor solución y herramienta con calidad de resultados. Facilitan el plan de diseño, exploración de diseño inicial, junto con optimización y velocidad de ruteo. Este nivel de análisis es necesario para la complejidad de circuitos que son requeridos de esta aplicación. [23]

Esta herramienta fue utilizada por el equipo de síntesis física de Altuna, Shin, y Ayala. Por las siglas de sus nombres, este equipo usualmente firma sus trabajos con **ASA**. Según su trabajo, se avanzó y optimizó el trabajo de síntesis física. [3] [16] [1]

A continuación, se listan archivos relevantes en la ejecución de este proceso, esto no incluye archivos de registro o *logs* que se abren en el transcurso del programa:

Cuadro 2: Archivos de *ICC2* para ejecución de síntesis física

Nombre y extensión	**Uso**	**Descripción**
Verilog final (.v)	Entrada	Archivo describiendo el circuito de entrada ya puesto en términos de la librería solicitada.
Libreria_Syn (n/a)	fólder	Folder conteniendo información del proceso y compuertas.
ICC2PS (.tcl)	Script	Script describiendo instrucciones a correr para síntesis física.
Nombre_IO (.gds)	Salida	Extracción de archivo describiendo circuito sintetizado.

7.1.4. IC Validator

Esta herramienta tiene como función principal validar diseños. Esto sería imposible por medio de un método manual debido a las reglas detalladas que son requeridas en DRC y LVS. Para esto, se requiere un programa o servicio que centralice estas verificaciones y provea apoyo en ubicar errores, proponer soluciones y al mismo tiempo mantener una eficiencia y

capacidad de ser escalado apropiadamente con la complejidad del problema a resolver. Este es el rol de *ICV* en el flujo de diseño. [24]

ICV, por supuesto, es de gran utilidad para las tres grandes pruebas necesarias durante el proceso de *Backend* de diseño. Estas pruebas incluyen: *DRC*, Verificación de reglas de Antena y *LVS*. [3] [14]

A continuación, se listan archivos relevantes en la ejecución de estas verificaciones, esto no incluye archivos de registro o *logs* que se crean en el transcurso del programa:

Cuadro 3: Archivos de *ICV* para ejecución de procesos de verificación

Nombre y extensión	**Uso**	**Descripción**
ICVLM18_LM16... (.4a)	Configuración	Archivo con reglas DRC y Antena para la configuración del usuario. Propiedad de TSMC.
LVS_RC... (.4a)	Configuración	Archivo con opciones del proceso LVS, incluyendo la generación de archivos. Propiedad de TSMC.
(.LAYOUT_ERRORS)	Salida	Archivo describiendo las fallas de reglas que fueron encontrados para LVS.
(.RESULTS)	Salida	Archivo describiendo las fallas de reglas que fueron encontrados para DRC.
STARCXT (.mapping y .runset_rep)	Salida	Archivos de salida de LVS, utilizados en extracción de parásitos.

7.1.5. StarRC

Esta herramienta se enfoca principalmente en la extracción de parásitos. Como fue mencionado, dicha prueba es un proceso que transfiere el layout como fue diseñado y lo implementa como un archivo describiendo el circuito para ser simulado. Esta verificación es importante porque se necesitan considerar las capacitancias parásitas, irregularidades, entre otras imperfecciones que surgen en el layout. *StarRC* es capaz de ejecutar este paso, procesando el diseño actual a una versión que pueda ser simulada con parámetros idénticos a los elementos reales. En dicha sección se verán imperfecciones y será posible considerar cambios relevantes para el diseño. [20]

El proceso ejecutado por medio de este programa es extracción de parásitos de layout, o *LPE* por sus siglas en inglés. Este era originalmente ejecutado por medio de *CC*. [5] Después de dificultades abriendo su interfaz gráfica, se pasó a realizarlo por medio de comandos y un archivo *.cmd*. Este proceso es detallado en su sección. A continuación, se listan los archivos utilizados por este sistema:

Cuadro 4: Archivos de *StarRC* para extracción de componentes parásitos

Nombre y extensión	**Uso**	**Descripción**
STARCXT (.mapping y .runset_rep)	Entrada	Archivos de salida de LVS, utilizados en extracción de parásitos.
LPE_TSMC_script (.cmd)	Script	Archivo detallando opciones de extracción incluyendo direcciones a Archivos de entrada.
CIRCUITO_cc (.rep)	Salida	Archivo de reporte de Coupling
CIRCUITO (.sp)	Salida	Archivo descriptivo de circuito

7.1.6. HSPICE

Como herramienta adicional que es particularmente útil para la sección de verificación, existe la herramienta de *HSPICE* que también le pertenece a *Synopsys*. Este es un simulador de circuitos. Este es una de las herramientas principales para estos diseños, ya que por medio de este simulador, es posible realizar numerosas pruebas de manera precisa de todo tipo de circuito válido. Una de estas es la simulación en estado transitorio, que permitirá observar el comportamiento de un sistema una vez recibe instrucciones específicas. [5]

7.1.7. Waveview

Como complemento a *HSPICE*, existe *WV*. Esta es una herramienta de visualización que puede observar los resultados de *HSPICE*. Sin esta clase de herramienta, la verificación de comportamiento dependería principalmente de análisis numérico de los resultados dados de una simulación. Por medio de *WV* es posible realizar una verificación visual del comportamiento. [19]

7.2. Proceso de automatización

El método general propuesto para automatizar el proceso de diseño consistirá en numerosos scripts *.sh* en *Linux*, específicamente *CentOS*. Estos se dedican a llamar a los programas debidos, abriéndolos e ingresando instrucciones en el formato de documentos llamados *runsets*. El inicio de este proceso debe iniciar con archivos *Verilog*, ingresados por el usuario, o importados de algún ejemplo. Los *runsets* adecuados, especialmente en sus secciones donde se apuntan a los archivos de entrada y salida podrán ser modificados de manera manual o utilizando una herramienta de sustitución para la adaptación a un circuito nuevo. Este programa deberá crear, copiar, y/o modificar los scripts utilizados para el proceso. Desde este punto, el objetivo es llegar a obtener los archivos finales de fabricación. Esto es posible debido a la función de comandos que llaman a las herramientas apropiadas. El detalle de cada herramienta, como funciona y los detalles están más allá del alcance de este proyecto.

Sin embargo, es importante estudiar sus limitaciones y como podrá adaptarse el sistema para completarlo.

Por supuesto, no se puede asumir que el proceso es exitoso todo el tiempo. Para adaptarse a esto, se deberá agregar procesos lógicos o verificaciones visuales para los pasos que lo requieran. Estos podrán hacer las pruebas distintas del proyecto, llegando a examinar cada uno de los elementos y fases del diseño. Como mínimo, el usuario debe llegar a conocer cual es el error o los errores como resultados. Esto permitirá corregir cualquier elemento anterior.

La complicación principal que debe considerarse es si resulta un error en una etapa avanzada. Por ejemplo, sea DRC, LVS, u otra prueba avanzada en el flujo. Es probable que un elemento de este layout deba ser corregido. Este puede ser un aspecto que no sea modificable desde el diseño original. La solución propuesta es simple, guiar al usuario a abrir la herramienta y corregir el error de manera manual si es que no se puede desarrollar una solución de manera automática. Esto indica la necesidad de poder continuar el sistema de automatización desde un punto medio del flujo de diseño. Si no fuera así, el mismo diseño lógico siempre terminará en el mismo error. Se tendría que iniciar desde el inicio, o continuar el proceso de manera manual desde ese punto.

Un aspecto importante a considerar es que una automatización de este diseño sería altamente dependiente de la complejidad que se le demanda. Con esto se refiere a la posibilidad de un circuito con una gran cantidad de *gates*, nodos, entre otros. Un programa teórico que realice todas las secciones del flujo de diseño tendría que tener elementos muy específicos a correr. Sería difícil generalizar casos complejos. Sin embargo, se pueden generalizar casos más simples, también con el propósito de poder educar sobre el flujo de diseño.

CAPÍTULO 8

Requerimientos y uso de automatización

La mayoría de los procesos individuales de este flujo de diseño fueron explorados y optimizados por los compañeros de trabajo de este último año. Este equipo entregó sus reportes específicos en enero 2022. Existe una gran ventaja en la construcción de este reporte durante la primera mitad del año 2022. Esta es que se podía utilizar estos reportes y trabajos de graduación como referencias para filtrar errores. Por medio de este proceso, se decidió replicar el proceso de cada uno de estas secciones, esta vez desde una sola computadora con un solo usuario.

Por supuesto, se desea simplificar lo más posible este proceso para un usuario final. El sistema implementado en su final requiere solamente los conocimientos mínimos de *Linux*, solamente el conocimiento de como abrir una terminal y ejecutar un *script* con *sh* o preferiblemente *bash*. Para referencias de personas que no conocen esto, esta se detalla a continuación. Desde *linux* se puede abrir el archivo que contiene estos scripts. Con un clic derecho, se puede elegir la opción de "*Open In Terminal*". Esto abre un cuadro adicional que se prepara para recibir comandos. Se puede utilizar las siguientes líneas para ejecutar uno de estos scripts:

Listing 8.1: Ejecución de scripts .sh desde el fólder que los contiene

```
bash auto_logicsyn.sh        #Sintesis Logica

bash auto_physicsyn.sh       #Sintesis Fisica

bash auto_antenna.sh         #Verificacion de antena

bash auto_LVS.sh             #Verificacion Layout versus Schematic

bash auto_LPE.sh             #Extraccion de componentes parasitos
```

Si estas se ubican en distintas ubicaciones al archivo donde se abrió la terminal, es probable que se requieran utilizar direcciones relativas. Con esto se refiere a añadirles símbolos para hacer referencia al lugar de los archivos respectivos. Una terminal no puede encontrar un archivo si no se le dice donde está. Entonces, para futura referencia, se deja un ejemplo de como se modifican estos comandos para ejecutar estos sistemas si se abre la terminal desde el archivo principal del chip que se desea diseñar.

Listing 8.2: Ejecución de scripts .sh desde un fólder superior

```
bash ./LogicalSynthesis/auto_logicsyn.sh        #Sintesis Logica

bash ./PhysicalSynthesis/auto_physicsyn.sh      #Sintesis Fisica

bash ./ANTENNA/auto_antenna.sh          #Verificacion de antena

bash ./LVS/auto_LVS.sh     #Verificacion Layout versus Schematic

bash ./LPE/auto_LPE.sh     #Extraccion de componentes parasitos
```

Un problema común que se ha dado en estos trabajos, o más bien una recomendación que los autores han dejado es "ser ordenado". Esto se debe al hecho que la gran mayoría de los programas utilizados en la metodología generan una gran cantidad de archivos. Por ejemplo, *StarRC* genera documentos de registros solamente con *intentar* abrirlo. El programa ni siquiera necesita tener éxito en abrirse para generar archivos. Por lo tanto, se necesita tener un buen sistema de organización.

Para esto, se presenta la necesidad de dividir los archivos en tres categorías:

Cuadro 5: Organización de archivos para automatización

Folder	**Uso**	**Descripción**
PROCESO	Generales y salidas	Archivos incluyendo script .sh principal, y salidas a próximo proceso.
PROCESO/files	Configuración y temporales	Configuración de procesos y archivos temporales
PROCESO/logs	Registros y resultados	Archivos secundarios de proceso para debugging y verificar proceso. Este puede estar vacío para correr.

Esta clasificación de archivos ha sido de gran utilidad. Se optimizaron estos mismos archivos scripts y los archivos de configuración para utilizar rutas relativas, es decir, utilizando un símbolo "."para representar la ubicación actual, y ".."para referirse a la ubicación superior.

Para actualizar el sistema para un nuevo chip, es recomendado copiar uno de los últimos

directorios de chip diseñado, y modificar los *runsets* ubicados en cada proceso bajo las carpetas llamada $files$ para tener los nuevos nombres. Se recomienda el uso del siguiente comando:

Listing 8.3: Reemplazo de nombres para archivos en mismo fólder

```
sed -i 's/NOMBRE_ANTERIOR/NOMBRE_NUEVO/g' *
```

En teoría, sería posible crear un script que hace esto automáticamente. No es posible garantizar su éxito debido a nombres cortos como RAM, ALU o NOT que reemplazan secciones de comandos importantes.

Una vez hecho eso y después de agregar al *runset* de LVS las instancias de *blackboxes* de los módulos según detallado por Jose Ruiz [14], será importante correr el sistema, ingresándole un nuevo archivo de *verilog* para ver los errores. Sería ineficiente asumir que el circuito complete todas las pruebas desde la primera vez. Es mucho más probable que se requiera modificaciones en configuraciones o de diseño.

CAPÍTULO 9

Desarrollo de automatización

En este capítulo, se pretende demostrar el método general desarrollado para automatizar cada uno de los procesos involucrados en cada sección del flujo del diseño. Al inicio de cada sección, se mostrará la forma del comando que llama a la herramienta apropiada para ejecutar un archivo script. Este archivo script es distinto para cada proceso, donde se encuentra la información detallada de cada corrida. Cualquier cambio al proceso o a los comandos que deben ser corridos en cada proceso deberán ser modificados dentro del mismo script. Esta automatización hace por suposición que ya se tiene un archivo *Verilog* que se quiere convertir en un layout apropiado.

Esta sección del reporte deberá ser particularmente útil para *debugging* o filtrar errores de corrida, al igual que para hacer correcciones en el proceso mientras se aprenda más del flujo desarrollado.

Es importante notar que para facilidad de lectura, se asume que dentro de los símbolos <> se reemplaza por el directorio apuntando al archivo, seguido por una barra diagonal (/), y finalizando con el nombre respectivo de los archivos solicitados. Algunos comandos piden más información que solamente la ubicación y nombre del archivo solicitado.

9.1. Síntesis lógica

La herramienta para el uso de la síntesis lógica es *Design Compiler*. A este se le llama y se le dice que ejecute un script con el comando:

Listing 9.1: Automatización Design Compiler

```
dc_shell -f <comandos_sintesis_logica.script>
```

El uso es relativamente sencillo, asumiendo que los comandos tienen toda la información necesaria para realizar la síntesis lógica. Detalles de lo que contienen estos scripts se encuentran en el trabajo correspondiente por Elmer Torres.

Sin embargo, es importante notar la importancia de la creación de un archivo .script para esta sección. Esta contiene todos los comandos necesarios para la síntesis que ordinariamente se ingresarían de manera manual a la consola. Se recomienda utilizar el comando # < *Comentariodeinstrucciones* > para diferenciar cada una de las secciones mientras se está corriendo, verificar visualmente el estado actual del proceso, y de esa manera facilitar el análisis de errores. Este genera partes redundantes de títulos que no sirven más función que facilitar la lectura. A continuación, se describe el proceso general que debe pasar cada script para realizar la síntesis lógica:

Listing 9.2: Síntesis lógica: Script de automatización (.script)

```
#------IMPORTANDO LIBRERIAS RELEVANTES:-----------
    lappend search_path <libs_path>
    set link_library <files.db>
    set target_library <file.db>
#------IMPORTANDO ARCHIVO VERILOG:-----------
    read_file -format verilog {<file.v>}
#------REVISION DE DISENIO y COMPILAR:-----------
    check_design
    reset_design
    set_max_area 0
    compile -map_effort high -area_effort high
#------REPORTES:-----------
    report_area
    report_design
    report_power
#------OUTPUT:-----------
    write -format ddc -h -o <out.ddc>
    write -hierarchy -format verilog -output <out.v>
    write_sdc <out.sdc>
exit
```

Como se podría observar al correr, los títulos comentados también van a aparecer en la terminal. El propósito de estos es legibilidad, sin causar mayor pérdida de procesamiento. Estos comandos se deben colocar en un archivo *.script* y luego guardar en una carpeta en donde pueda acceder la información que quiere importar cuando sea corrido desde una terminal. La salida de este proceso son los archivos *.ddc*, *.v*, y *.sdc* que fueron especificados en el último capítulo, tanto para el módulo principal como para los IO. Estos sirven como entradas en la síntesis física. Estos pueden ser verificados por medio de Verdi, sin embargo, esto está afuera del alcance de este trabajo. Esto se puede encontrar en el trabajo siendo realizado por Gerardo Cardoza.

9.2. Síntesis física

Esta sección viene justamente después de la síntesis lógica, con el siguiente diagrama mostrando los roles que tiene cada programa en el flujo. El rol tomado por Design Compiler es transferir un archivo de *HDL* (como *verilog* en este caso) a un diseño descrito por la librería de elección. Este proceso hace lo posible para optimizar tiempos, largos, área, potencia, entre otros. Un análisis puede ser necesario para validar estas propuestas ya que fueron hechas por programa. A continuación, se agrega dicho diagrama para describir la relación entre estos:

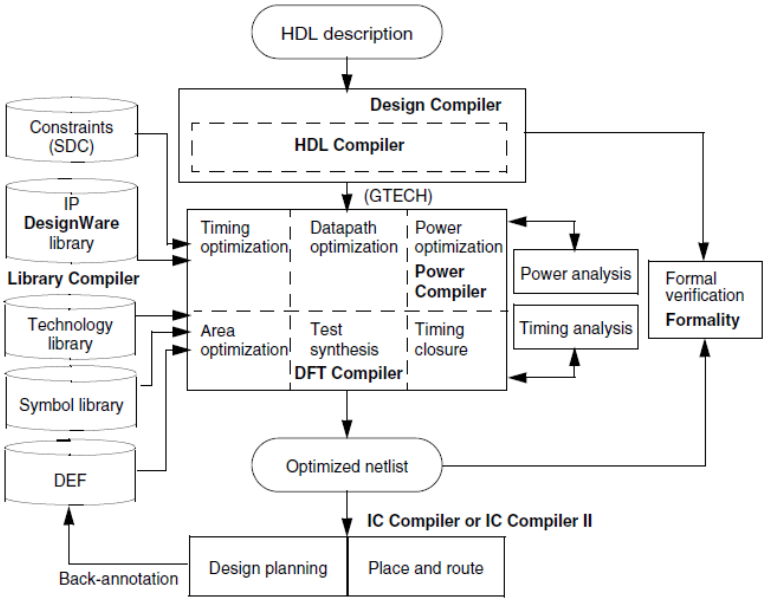


Figura 3: DC and ICC Flow [22]

Durante el periodo del proyecto, se realizó una transición importante en las herramientas que estaban siendo utilizadas. Esta transición fue hecha para utilizar *IC Compiler 2* en lugar original de *IC Compiler*. Esto fue hecho debido a las recomendaciones hechas durante el año pasado, donde la versión anterior provocó varias complicaciones en el diseño. [7] A medida que se avanzó en el proyecto, esto demostró ser una sabia decisión.

El comando utilizado en la terminal para ejecutar un script de diseño.

Listing 9.3: Automatización IC Compiler II

```
icc2_shell -file <comandos_sintesis_fisica.tcl>
```

Dentro de *IC Compiler 2* es que se hace la conversión del archivo de tipo verilog creado por *Design Compiler* para preparar archivos de chip que puede ser fabricado. Es importante notar que varios parámetros están fijos en este ejemplo, es probable que sea necesario modificar widths, pitch, o tamaños respectivos dependiendo de la aplicación:

Listing 9.4: Síntesis física: Primera parte de command file (.tcl)

```
1  #-----------------<CREACION_LIBRERIA>-----------------
2  create_lib <CIRCUITO_SYN.ndm> -technology /usr/synopsys/TSMC/180
       Complete/TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/
       TSMCHOME/digital/Back_End/milkyway/tcb018gbwp7t_270a/techfiles/
       tsmc018_6lm.tf \
3   -ref_libs <TSMCWorkspace.ndm>
4
5  #--------<IMPORTACION_VERILOG_SINTETIZADO>---------------
6  read_verilog <CIRCUITOio.v>
7
8  read_sdc -echo <LogicalSynthesis/CIRCUITOio.sdc>
9
10 #---------------<IMPORTACION_TLU+_Y_MAP>-----------------
11 read_parasitic_tech -tlup /usr/synopsys/TSMC/180Complete/TSMC/180/
       CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/
       Back_End/milkyway/tcb018gbwp7t_270a/techfiles/tluplus/
       t018lo_1p6m_typical.tluplus -layermap /usr/synopsys/TSMC/180
       Complete/TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/
       TSMCHOME/digital/Back_End/milkyway/tcb018gbwp7t_270a/techfiles/
       tluplus/star.map_6M
12
13 #-------------<LIMPIEZA_PG>-----------------------
14 remove_pg_strategies -all
15
16 #---------------<REGLAS_ANTENA>-----------------------
17 #source -echo -verbose "/usr/synopsys/TSMC/180Complete/TSMC/180/
       CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/
       Back_End/milkyway/tcb018gbwp7t_270a/clf/
       MODIFIED_antennaRule_018_6lm.tcl"
18
19 #-----------------<CREACION_CORNERS>-----------------
20 create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER
21
22 #-----------------<VDD_Y_VSS_PADS>-----------------
23 create_cell {PVDD} PVDD1CDG
24 create_cell {PVSS} PVSS1CDG
25
26 #---------------<VDD_Y_VSS_NETS>-----------------------
27 resolve_pg_nets
28 create_net -power VDD
29 create_net -ground VSS
30 connect_pg_net -net VDD [get_pins -physical_context *VDD]
31 connect_pg_net -net VSS [get_pins -physical_context *VSS]
32 connect_pg_net -automatic
33 report_cells -power
34
```

```
35 #-------------<FLOORPLAN-INICIO>-------------------
36 initialize_floorplan -site_def unit -use_site_row -keep_all -
      side_length {285 285} -core_offset {125}
37
38 #---------------<ANILLO-IO>-------------------------
39 create_io_ring -name anillo_IO -corner_height 115
40
41 #-----------<IO-PADS>---------------
42 #Coloca los pines de entradas y salidas (Pads) en un lugar
      arbitrario de no ser especificado en el floorplan
43 add_to_io_guide [get_io_guides anillo_IO.left] PVDD*
44 add_to_io_guide [get_io_guides anillo_IO.right] PVSS*
45 place_io
46
47 #----------<CREACION-ANILLO-PG>-----------------
48 create_pg_ring_pattern ring_pattern -horizontal_layer METAL2    -
      horizontal_width {2} -horizontal_spacing {2}    -vertical_layer
       METAL3 -vertical_width {2} -vertical_spacing {2}
49 set_pg_strategy core_ring    -pattern {{name: ring_pattern}    {
      nets: {VDD VSS}} {offset: {1 1}}} -core
50 compile_pg -strategies core_ring
51
52 #----------<IO-CONEXIONES>-------------------------
53 create_pg_macro_conn_pattern hm_pattern -pin_conn_type
      scattered_pin -layers {METAL2 METAL3} -nets {VDD VSS} -
      pin_layers {METAL2}
54 set_app_options -name plan.pgroute.treat_pad_as_macro -value true
55 set_pg_strategy macro_conn -macros [get_cells {PVDD* PVSS*}] -
      pattern {{name: hm_pattern} {nets: {VDD VSS}}}
56 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}}}
57 compile_pg -strategies macro_conn -via_rule macro_conn_via_rule -
      tag test
58
59 #-----------<VDD-Y-VSS-MESH>----------------
60 connect_pg_net -automatic
61 create_pg_mesh_pattern mesh_pattern -layers { {{vertical_layer:
     METAL3} {width: 4.2} {pitch: 42} {spacing: interleaving}} }
62 set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {409.480
      414.240}} -pattern {{pattern: mesh_pattern}{nets: {VDD VSS}}}
     -blockage {macros: all}
63 create_pg_std_cell_conn_pattern std_cell_pattern
64 set_pg_strategy std_cell_strategy -core -pattern {{pattern:
     std_cell_pattern}{nets: {VDD VSS}}}
65 compile_pg
```

Los comentarios agregados, demostrados por símbolos de numeral (#), ayudan a titular o describir el propósito de cada sección de los comandos. Esta es una costumbre importante ya que por medio de estas definiciones se puede dar una idea de lo que ejecuta cada bloque. En este punto, ya se tiene cargado la gran mayoría de los archivos, permitiendo realizar un análisis, agregando las conexiones restantes hacia afuera del chip, y finalizar las rutas.

Listing 9.5: Síntesis física: Segunda parte de command file (.tcl)

```
1  #-------------<MERGE DE MESH Y ANILLO PG>-------------------------
2  merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2
       METAL3}
3
4  #-------------<PLACEMENT CREACION>--------------------------
5  set_app_options -name place.coarse.fix_hard_macros -value false
6  set_app_options -name plan.place.auto_create_blockages -value auto
7  create_placement -floorplan -timing_driven -congestion -effort
       high -congestion_effort high
8  legalize_placement
9
10 #-----<SINTETIZAR RELOJES>-(Solo chips secuenciales)-------
11 check_clock_trees -clocks clk
12 check_design -checks pre_clock_tree_stage
13 synthesize_clock_trees -clocks clk -postroute -routed_clock_stage
       detail_with_signal_routes
14 clock_opt -list_only
15 check_design -checks cts_qor
16
17 #----------<RUTEADO>-------------------------------
18 check_routability -check_pg_blocked_ports true
19 check_design -checks pre_route_stage
20 route_auto
21
22 #-----------------<CORE FILLER Y ANILLO IO>-----------------
23 create_io_filler_cells -io_guides [get_io_guides {anillo_IO.top
       anillo_IO.right anillo_IO.left anillo_IO.bottom}] \
24 -reference_cells {PFILLER1 PFILLER5 PFILLER05 PFILLER0005
       PFILLER10 PFILLER20}
25 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}]
26 connect_pg_net -automatic
27 remove_stdcell_fillers_with_violation
28 check_legality
```

Después de correr la parte anterior, se pueden realizar las pruebas debidas. Sin embargo,

es importante notar que después de estas pruebas, es necesario generar un archivo *.gds* para las pruebas después. El siguiente comando es capaz de realizar este objetivo:

Listing 9.6: Síntesis física: Creación de .gds

```
write_gds -library <libreria.ndm> -design <TOP_CELL> -view design
    -hierarchy all -lib_cell_view frame <CIRCUITO.gds>
```

Sin embargo, antes de generar este gds, es preferible correrlo después de la prueba de DRC. Este se corre en el mismo script *.tcl*. Se explican como secciones separadas para no sobre complicar la explicación del mismo:

9.3. Design Rule Check

Repitiendo la función de este proceso, el propósito de *DRC* es la verificación de un layout para que este cumpla cada una de las reglas de fabricación, colocando de primera prioridad que el diseño analizado no resulte en problemas inesperados de fabricación o fallas de diseño por error humano. Un ejemplo de las reglas más fundamentales es la separación de nodos con designaciones distintas (es decir, que señales distintas se mantengan separadas), y que esa separación sea lo suficiente para que el efecto que tengan entre sí sea nulo o insignificante.

En este caso, TSMC tiene su reglamento extenso, descrito en detalle y digitalizado en el archivo $ICVM18\_LM16\_LM152\_6M.215A\_pre041518$ que sirve para informarle al programa de *ICV* cuales son las reglas utilizadas. Debido al acuerdo de confidencialidad con *TSMC*, no es apropiado describir cuales son estas en este trabajo. Lo relevante es mencionar que estas son las reglas que fueron utilizadas en este proyecto.

Actualmente, es importante mostrar la forma del comando que *Custom Compiler* utiliza para producir todos los archivos para continuar el proceso. En este ejemplo, se ha generalizado para poder describir lo que se agrega en cada parte del proceso:

Listing 9.7: DRC: icv comando para openaccess

```
exec-oa22.60.021.icv icv -f openaccess -i <nombre_libreria> -c <
    nombre_de_bloque> -oa_view <nombre_de_vista> -oa_lib_defs <
    archivo_libreria.defs> -oa_layer_map <archivo_layer_mapping.map
    > -rc <runset_configuracion> -oa_dm6 -vue <runset> <log_name.
    log> 2>&1
```

Como se podría asumir, *openaccess* se refiere a las librerías preparadas por Synopsys para realizar sus diseños. Estas no son las librerías que fueron utilizadas por este proyecto. El formato anterior simplemente es un guía. Una opción para correr esta prueba es utilizar siguiente comando:

Listing 9.8: DRC: icv comando para correr script

```
icv -i <CIRCUITO_IO.gds> -c <nombre-de-celda> ICV vue <DRC_RUNSET>
```

El método de antena utiliza este mismo comando. Sin embargo, para facilitar el uso de síntesis física al igual de realizar las correcciones debidas, se integra su corrida desde que se está creando las rutas.

El método utilizado al final se hace en medio de síntesis física, desde *IC Compiler 2*, justo después de haber completado las rutas. A continuación, se muestra el final del documento *.tcl* que fue corrido al inicio de la Síntesis física:

Listing 9.9: DRC: Script de configuración y corrida

```
#-----------------<DRC_RUNSET_FILE>-----------------------
set_app_options -list {signoff.check_design.run_dir {../DRC/}}
set_app_options -list {signoff.check_drc.run_dir {../DRC/}}
set_app_options -list {signoff.check_design.runset {../files/
    ICVLM18_DRC.215a_pre041518}}
set_app_options -list {signoff.check_drc.runset {../files/
    ICVLM18_DRC.215a_pre041518}}


#---------------<DRC_Y_GUARDAR>---------------
save_block <CHIP_SYN.ndm>:<CHIP_IO>
signoff_fix_drc

save_block <CHIP_SYN.ndm>:<CHIP_IO>
signoff_check_drc

check_lvs
save_block <CHIP_SYN.ndm>:<CHIP_IO>
```

Es relevante notar que en el bloque de script para DRC, también se está realizando una prueba de LVS. Esta funciona como una prueba propia dentro de la herramienta de *ICC 2*. Esta prueba opera con criterios distintos a los necesarios por TSMC. Debido a esto, todavía resulta necesario correr estas pruebas por medio de *ICV* directamente.

La sección de reglas de antena es similar a esta. Simplemente definiendo un script de configuración del sistema y corriendo el sistema. Este no requiere mayor explicación, simplemente optimizar el archivo de configuración para correr las reglas de antena. Esta se corre con un solo comando de *icv* similar al encontrado con *DRC* y se le ingresa los comandos apropiados.

9.4. Layout versus Schematic

La validación de *LVS* se refiere a una comparación entre el diseño propuesto de layout contra el esquemático que soluciona el problema que se desea resolver. Es útil mostrar el uso de *Custom Compiler* para correr LVS. Cada opción está asociada a distintas configuraciones dentro de la herramienta, incluyendo la interfaz gráfica, las librerías utilizadas, el archivo

de *mapping*, configuración del *runset*, entre otros. Es importante notar que este no es el utilizado por la automatización desarrollada, simplemente se agrega como referencia:

Listing 9.10: LVS: icv comando para openaccess

```
exec-oa22.60.021.icv icv -f openaccess -i <nombre_libreria> -
    oa_view <nombre_de_vista> -oa_lib_defs <archivo_libreria.defs>
     -oa_layer_map <archivo_mapping.map> -rc <runset_configuracion>
     -s <esquematico.sp> -sf SPICE -stc <root_cell> -sf SPICE -
    verbose  -oa_dm6 -vue <runset> <log_name.log> 2>&1
```

Debido a que en la automatización, no se tienen definidas secciones de librerías o vistas, es necesario encontrar otro método de LVS. La relación entre *IC Validator* y *StarRC* es evidente en la manera que dependen del otro. Esto es porque no es permitido hacer una extracción de parásitos sin antes haber aprobado un LVS. A continuación, se muestra un diagrama describiendo el proceso de ICV para realizar LVS y luego LPE:

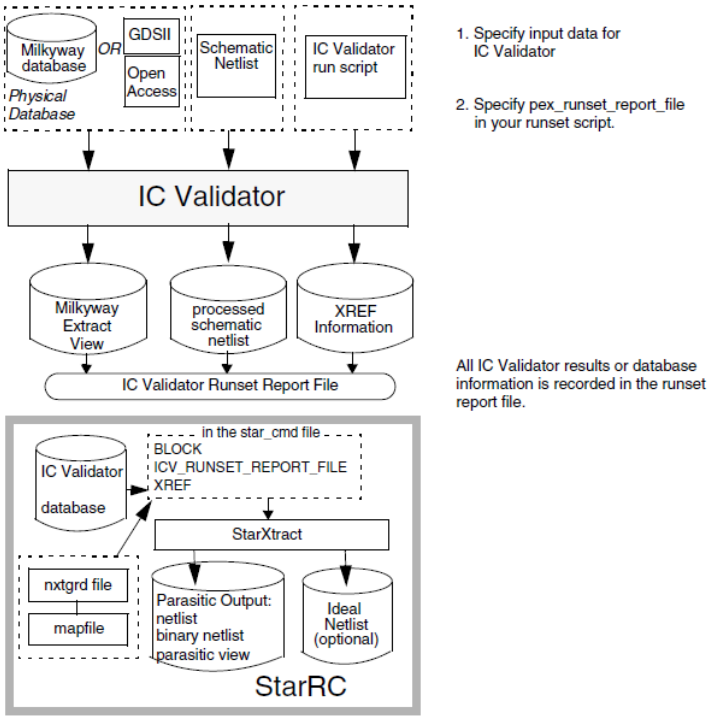


Figura 4: ICV and StarRC Flow [18]

Este proceso es descrito con más detalle en el trabajo de José Ruiz. [14]

El primer paso para realizar LVS va a involucrar la traducción de *Netlist*, primero de archivo *Verilog* a *ICV*, y luego de *Verilog* a *SPICE*. Esto se puede realizar con los siguientes dos comandos:

Listing 9.11: LVS primer paso: Traducción de Netlist

```
icv_nettran -verilog <documento.v> -outType ICV -outName <
    documento.icv>
icv_nettran -verilog <documento.v> -outType SPICE -outName <
    documento.sp>
```

El segundo paso involucra copiar los dichos *netlists* de $StandardCellLibrary$ y $I/OCellLibrary$. Estos se definen de la siguiente manera como fue descrito por el trabajo hecho en esta sección:

Listing 9.12: LVS segundo paso: Definición de headers

```
Standard Cell Library > CORE.v
I/O Cell Library > IO.v
```

El tercer paso involucra concatenar estos dos en un archivo *headers.v*. Y esto se puede realizar con el comando *cat*:

Listing 9.13: LVS tercer paso: Concatenación de headers

```
cat CORE.v IO.v > headers.v
```

El cuarto paso involucra traducir este archivo a *CDL* o *SPICE*:

Listing 9.14: LVS cuarto paso: Traducción a CDL desde headers

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

Dicho archivo viene incompleto, y por lo tanto, se le debe agregar las siguientes lineas a su inicio:

Listing 9.15: LVS 4.5 Paso: agregado a headers.sp

```
.GLOBAL VDD VSS VDDPST VSSPST
*.EQUATION
*.SCALE METER
*.MEGA
.PARAM
.INCLUDE <source.added>
```

Cabe mencionar que el archivo de *source.added* es dado por TSMC, y en general, se puede encontrar en el servidor en la siguiente carpeta

$$/usr/synopsys - old/TSMC/SCRIPTS\_NUEVOS/20191128 - 124344$$

Pasando al quinto paso, se requiere la creación del archivo *.icv*, que contiene la netlist de este dispositivo:

Listing 9.16: LVS quinto paso: Creación de netlist.icv

```
icv_nettran -verilog <out_not_io.v> -sp <headers.sp> -outType ICV
    -outName <CDL_netlist.icv>
```

En este punto existía un sexto paso que involucraba la creación de un archivo *.gds* para el que era necesario importar la librería de síntesis lógica a *CC* y exportarlo manualmente. Sin embargo, en el proceso de exploración de las herramientas, se encontró un método para generar el mismo desde la síntesis lógica. Por lo tanto, siempre que este sea completado, se omite el sexto paso de *LVS*.

Anteriormente, lo que se había hecho es generar este archivo con *CC*, importando la información debido a *Custom Compiler* y solicitar la generación del archivo desde ahí. Esta solución temporal solucionaba debidamente la generación.

Finalmente, el séptimo paso, luego de realizar una revisión completa del *runset* y de los archivos, es la prueba LVS. Este se puede realizar con un solo comando, utilizando todos los archivos generados durante estos pasos:

Listing 9.17: LVS séptimo paso: Prueba de LVS

```
icv -i <ARCHIVO_GDS> -c <NOMBRE_TOP_CELL> -s <CDL_NETLIST> -sf ICV
    -vue <LVS_RUNSET>
```

Por supuesto, este paso involucra revisar los resultados. Con esta información, y los archivos generados por la herramienta, incluido los reportes involucrados, es posible correr la extracción de parásitos.

9.5. Layout Parasitic Extraction

El trabajo hecho originalmente por Charlie Cruz era principalmente de ejecución del método. En esa etapa del proyecto, no fue posible utilizar apropiadamente *StarRC*, siendo difícil abrir su interfaz gráfica. Por lo tanto, Cruz se dedicó a establecer un método para ejecutar *LPE* por medio de *CC*.[5] Por lo tanto, durante este tiempo, se saltó inmediatamente a automatizar este proceso, ya que el programa independiente operaba correctamente. Esto se sabía porque *CC* hace la llamada a *StarRC*. Fue originalmente basado en este proceso en el que se ubicó el método de automatización que ya hizo posible durante este periodo independizarse de *Custom Compiler*.

El comando utilizado para realizar una extracción de parásitos, con toda la información completa es:

Listing 9.18: Automatizacion StarRC

```
StarXtract <comandos_LPE.cmd> > <log_name.log> 2>&1
```

Como fue explicado anteriormente, el proceso LPE es principalmente realizar un análisis de componentes del layout. Este utiliza la información del layout para generar un *netlist*, o más bien un documento descriptivo para usarse en *HSPICE*. Este documento describe a detalle cada una de las capacitancias y resistencias parásitas. Estas se refieren a las interacciones accidentales que surgen no por agregar elementos, sino por naturaleza de la física en el diseño del layout.

Información de este proceso se puede encontrar en el trabajo hecho por Charlie A. Cruz [5]. Con eso aclarado, es importante aclarar varios detalles generales. La herramienta de *Synopsys* optimizada para el proceso de *LPE* es *StarRC*. Este tiene como entrada información general del proceso de LVS y el layout, produciendo un documento con la capacidad de describir el comportamiento verdadero de un chip con dicho layout de la manera más exacta posible.

Este es un paso crucial para el diseño de cualquier chip. Un diseño de layout podría pasar correctamente la prueba de DRC y LVS, representando un seguimiento correcto de las reglas de diseño y además una similitud exacta al esquemático diseñado. Sin embargo, el comportamiento del circuito puede ser modificado por las capacitancias y resistencias agregadas por imperfecciones del material o interacciones entre vías y componentes. En el mejor caso, estas podrían simplemente alentar las respuestas del circuito con capacitores que deben ser cargados para cambios de voltaje y resistencias que modificar su información. Este efecto podría ser considerable, incluso requiriendo un nuevo diseño para cumplir las necesidades del diseño. Esto muestra la importancia de esta verificación.

Como fue encontrado por Cruz, el comando para llamar a *StarRC* es *StarXTract*. Un comando que hasta el momento parece ser idéntico, que realiza un comando similar solamente utilizando *openaccess*: *StarXTract_oa* es utilizado dentro de *Custom Compiler*. Fue por medio del método utilizado en este que se encontró el método para automatizar esta sección.

A continuación, se muestra un archivo *.cmd* que se puede utilizar para realizar una extracción, este se basa en el archivo *.cmd* utilizado por Charlie Cruz en su trabajo [5]. Este, a su vez, es una versión modificada del archivo de parámetros utilizado por *TSMC* como modelo de opciones de extracción, dadas para referencia de sus clientes. Este es un ejemplo del archivo más simple que puede utilizarse en *StarRC*:

Listing 9.19: LPE script ejemplo para automatización (.cmd)

```
BLOCK                : <nombre_bloque>
MILKYWAY_DATABASE    : <XTOUT_library>
TCAD_GRD_FILE        : <grd_file.nxtgrd>
MAPPING_FILE         : <xt.mapping>

*****************OPCIONES*DE*SALIDA:**********************

NETLIST_FORMAT       : SPF
NETLIST_FILE         : <salida.spf>
STAR_DIRECTORY       : ./star
COUPLE_TO_GROUND     : YES
NETLIST_PASSIVE_PARAMS: YES
```

```
COUPLE_TO_GROUND: NO
COUPLING_REPORT_FILE: <cc.rep>

*****************OPCIONES*ADICIONALES:**********************

EXTRACTION: RC
REDUCTION: YES
CASE_SENSITIVE: NO
HIERARCHICAL_SEPARATOR: /
DENSITY_BASED_THICKNESS: YES
EXTRACT_VIA_CAPS: NO
REMOVE_FLOATING_NETS: YES
REMOVE_DANGLING_NETS: YES
POWER_NETS: VDD VSS
SKIP_CELLS: ! *
TRANSLATE_RETAIN_BULK_LAYERS: YES
NETLIST_FORMAT: NETNAME

XREF: YES
XREF_USE_LAYOUT_DEVICE_NAME: YES
CELL_TYPE: LAYOUT

SKIP_PCELLS : cfmom* cfmom_mx* cfmom_rf* crtmom* crtmom_rf*
ind_std* ind_std_40k* ind_sym* ind_sym_40k* ind_sym_ct*
ind_sym_ct_40k* j v a r * l c e sd1_r f * l c e sd2_r f *
    lowcpad_rf *
mimcap_rf* mimcap_rf_2p0* mos_var* mos_var33* moscap_rf*
moscap_rf33* moscap_rf33_nw* moscap_rf_nw* ndio_hia_rf *
ndio_sbd_mac* pdio_hia_rf * rfnmos2v * rfnmos2v_6t * rfnmos3v *
rfnmos3v_6t * rfpmos2v * rfpmos2v_5t * rfpmos2v_nw* rfpmos2v_nw_5t
* rfpmos3v * rfpmos3v_5t * rfpmos3v_nw* rfpmos3v_nw_5t*
rphpoly_r f * rphr ipo ly_r f * rplpo ly_r f * sbd_rf * sbd_rf_nw*
spiral_std_m2u_a_33k* spiral_std_m2u_x_33k* spiral_std_mu_a_33k
* spiral_std_mu_x_20k* spiral_std_mu_x_33k* spiral_std_mu_x_40k
* spiral_sym_ct_m2u_u_a_33k* spiral_sym_ct_m2u_u_x_33k*
spiral_sym_ct_mu_x_20k* spiral_sym_ct_mu_x_33k*
spiral_sym_ct_mu_x_40k* spiral_sym_ct_mu_x_a_33k*
spiral_sym_m2u_u_33k* spiral_sym_mu_x_20k* spiral_sym_mu_x_33k*
spiral_sym_mu_x_40k*
```

Como se puede notar, el script para el uso de *StarXTract* consiste en referencias a los archivos necesarios de entrada y salida, junto con un listado de las opciones deseadas para la configuración de la extracción. El orden de estos comandos no es relevante, todas estas son

Una sección relevante de esta operación es el archivo *nxtgrd*. Este describe la relación

entre el diseño y el proceso de manufactura. Este puede ser obtenido a partir de los archivos proveídos de TSMC. Más información de este proceso se puede encontrar en el trabajo de Charlie Cruz [5], sin embargo, con el propósito de facilitar la repetición de este trabajo, se deben buscar los archivos *.itf* dentro de la siguiente carpeta:

$$/usr/Synopsys/TSMC/180/CMOS/G/IO3.3V/util$$

Aquí se pueden encontrar varias carpetas comprimidas que contienen los parámetros para los mejores, típicos o peores casos de extracción para estas librerías. En estas se pueden encontrar tanto los *.itf* como *.nxtgrd*. Sin embargo, es importante notar que se puede obtener uno del otro con el comando:

Listing 9.20: Extracción de archivos .nxtgrd

```
grdgenxo <itf_file>
```

Este proceso toma varios minutos, por lo que paciencia es sugerida. Con este archivo y un script de comandos para StarRC, es posible utilizar el comando de *StarXtract* con el formato mostrado al inicio de esta sección, para generar un archivo *.spf* (o *.sp* si las opciones se modifican) que podría ser simulado.

La complicación actual es el hecho que en esta parte del proceso, no se ha generado correctamente las salidas y los pines para poder debidamente simular el circuito. Esto se debe a que actualmente se tiene un paquete educativo de *TSMC*, una vez se pueda realizar este proceso con una librería original, este no será un problema.

Con eso completado, se logra confirmar que con estos métodos, es posible realizar la extracción de parásitos, y generar un archivo que correctamente describe el comportamiento real de un chip con el layout diseñado en la síntesis física.

CAPÍTULO 10

Extracción de ALU

Esta sección se completó durante el año 2021, todavía trabajando en paralelo con los miembros del equipo del mismo año. Este se agrega como el resultado final del trabajo de ese año, desarrollando un método con el circuito más complejo al que se tenía acceso en ese momento. En el capítulo seguido de este, se detalla la replicación del proceso para la última versión de *El Gran Jaguar* con el código de verilog más actual preparado por Elmer Torres.

Por medio de las herramientas mostradas en la sección anterior, es posible hacer el diseño de un chip utilizando la librería asociada de TSMC. Gracias al procedimiento detallado por los estudiantes que avanzaron este proyecto durante el año pasado, fue posible realizar varias pruebas. A continuación, se muestra el archivo en el que se trabajó la automatización:

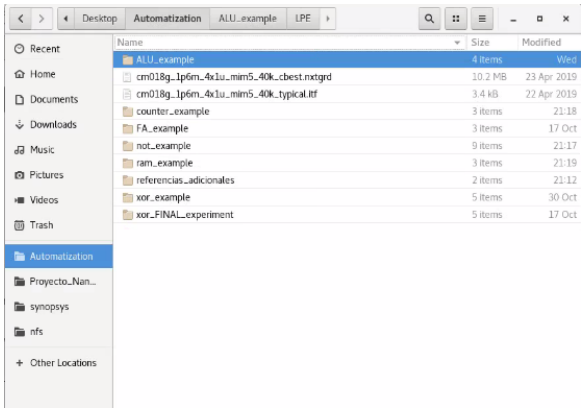


Figura 5: Fólder de automatización

Estos fueron los circuitos diseñados por el equipo de trabajo, llegando hasta la extracción de parásitos. En orden de menor a mayor complejidad: NOT, XOR, $COUNTER$,

FullAdder, *RAM* y *ALU*. El *COUNTER* ha presentó problemas, con el *HNFILE* siendo incapaz de ser analizado. Se recomienda estudiar esta situacicón para evitar problemas en el futuro.

Por el momento, se puede demostrar la extracción de ALU. Utilizando el mismo procedimiento de antes, se crea una carpeta con la siguiente organización:

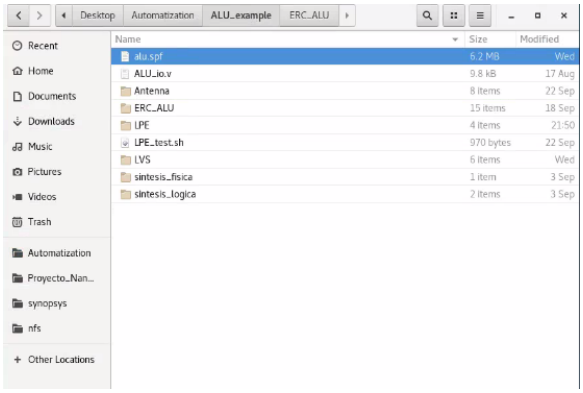


Figura 6: Fólder de ALU

Es importante tomar en cuenta que el script o archivo de comandos de StarRC debe tener bien controlado los directorios internos, específicamente utilizando direcciones relativas (es decir, utilizar "./"para la carpeta actual y "../"para la carpeta de arriba). La preferencia, es realizar un script *.sh* que primero ingrese a *./LPE/logs/* y luego corra *StarXtract* obtener el script un nivel arriba. Esto facilitará el uso de los archivos generados, tanto para modificación o consultas a estos.

Utilizando las direcciones relativas, es posible realizar una actualización rápida de los archivos, al igual que crear un sistema organizado en el que el proceso actual pueda consultar los archivos de salida del proceso anterior. Con buen control, es posible realizar un sistema donde las salidas de cada proceso sean simples de encontrar.

Este proceso no va a ser instantáneo. Esto se debe a la complejidad del layout y lo que se debe analizar. A continuación, se muestra una imagen del layout que se estará extrayendo:

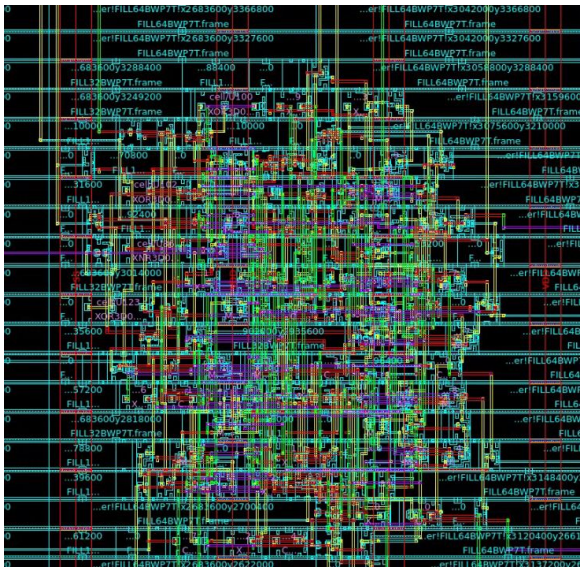


Figura 7: Zoom de layout CHIP

Una vez más, este proceso ha fallado en generar los puertos debidos. Esto se debe a que en los procesos anteriores, no se han definido completamente los puertos. En LVS, se ha encontrado una sección en la que ha se remueven los puertos del esquemático. Esto se presume ser una solución temporal. Según el trabajo de Charlie Cruz, esta se debe a que el paquete recibido no es comercial, y por lo tanto, tiene capas ocultas que serían esenciales para las pruebas finales.

Por ahora, se demuestra el proceso que podrá ser utilizado. El resultado de este proceso va a ser semejante a lo siguiente:

Listing 10.1: LPE Result: ALU (parcial)

```
*
*|DSPF 1.3
*|DESIGN ALU_IO
*|DATE "Wed Nov 10 11:42:50 2021"
*|VENDOR "Synopsys"
*|PROGRAM "StarRC"
*|VERSION "R-2020.09-SP3"
*|DIVIDER |
*|DELIMITER :
**FORMAT SPF
*

** COMMENTS

** OPERATING_TEMPERATURE 25
** DENSITY_OUTSIDE_BLOCK 0
** GLOBAL_TEMPERATURE 25
**
```

```
** TCAD_GRD_FILE ../../../../cm018g_1p6m_4x1u_mim5_40k_cbest.
   nxtgrd
**   TCAD_TIME_STAMP Tue Apr 23 22:51:06 2019
**   TCADGRD_VERSION 62

.SUBCKT ALU_IO

*|GROUND_NET 0

*LAYER_MAP

*0 SUBSTRATE
*1 poly     ITF=poly
*2 poly_rf    ITF=poly
*3 metal1    ITF=metal1
*4 metal2    ITF=metal2
*5 metal3    ITF=metal3
*6 metal4    ITF=metal4
.
.
.
```

El archivo *.spf* final ocupa 6.1 MB de espacio, y toma 86,273 líneas en describir el circuito. Debido a esto, no sería productivo ingresarlo en su totalidad en este trabajo.

Este documento describe correctamente un esquemático verdadero del layout. En teoría, deberá ser posible manualmente agregar los puertos al sub-circuito. Esto no sería mayor problema para circuitos como NOT y XOR. Sin embargo, mientras un circuito va incrementando en complejidad, se vuelve difícil a imposible realizar estas modificaciones. De hecho, esto fue intentado por medio de un programa *.py* para Charlie Cruz. Lastimosamente, este no tuvo éxito [5]. Y esta posibilidad está afuera del alcance de este trabajo que va a priorizar la replicación de resultados del año 2021 en una implementación automatizada.

CAPÍTULO 11

El Gran Jaguar

Como se mencionó anteriormente, este trabajo está compuesto por dos partes. El primero se refiere al desarrollo del método de extracción de parásitos. Segundo, es la automatización del flujo de diseño. Parte de lo que se dejó pendiente durante el 2021 fue el desarrollo del diseño para la última versión del Gran Jaguar. Este fue obtenido del trabajo originalmente utilizado Elmer Torres. [26]

La propuesta de este circuito es un circuito que exporte un texto definido por el usuario. En este caso, se quiere trabajar con dos textos:

"*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 graduados de Ingenieria en Electronica entre 2019 y 2021*"

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

Su desarrollo original, incluída la automatización de su código por medio de *Python*, se encuentra en el trabajo hecho por Elmer Torres. [26]

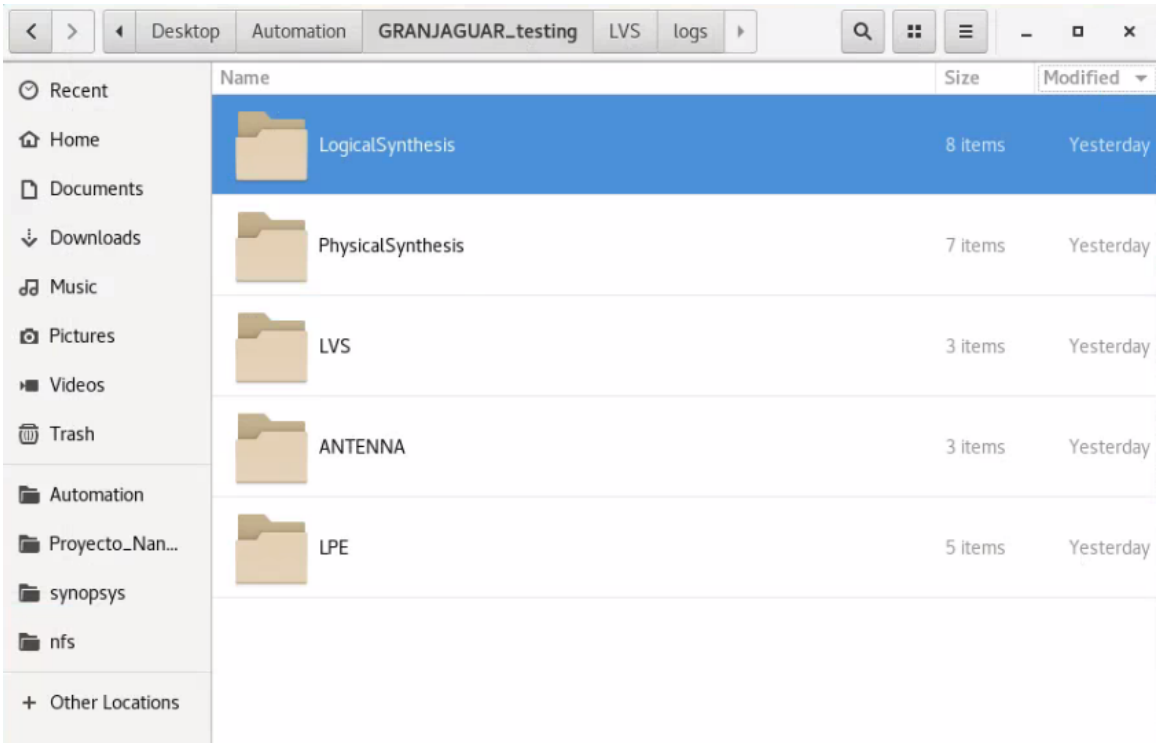


Figura 8: GJ Organización de procesos

11.1. Síntesis lógica del Gran Jaguar

A continuación se muestra la entrada a síntesis lógica, definida como un archivo *Verilog* de *HDL*. Este es el archivo que se le debe pasar una vez por *Design Vision*, seguido de añadirle la información de *IO*, y finalmente pasado otra vez por *Design Vision* para obtener una solución lógica que cumpla los requisitos para iniciar la síntesis física:

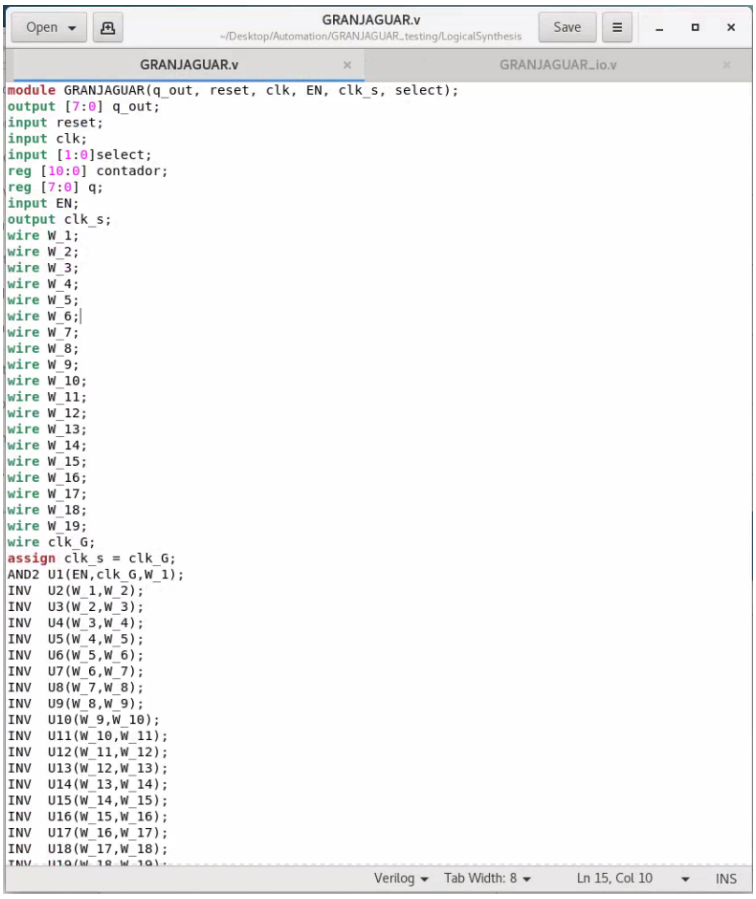


Figura 9: GJ Entrada a síntesis lógica

Se puede correr el proceso simplemente abriendo una terminal en el fólder que contiene a *auto_logicsyn.sh* y correrlo con el comando *bash* o *sh*. Es importante notar que en el archivo *.log* se puede observar los reportes de poder, y área. A continuación se muestra una imagen presentando la síntesis lógica en proceso:

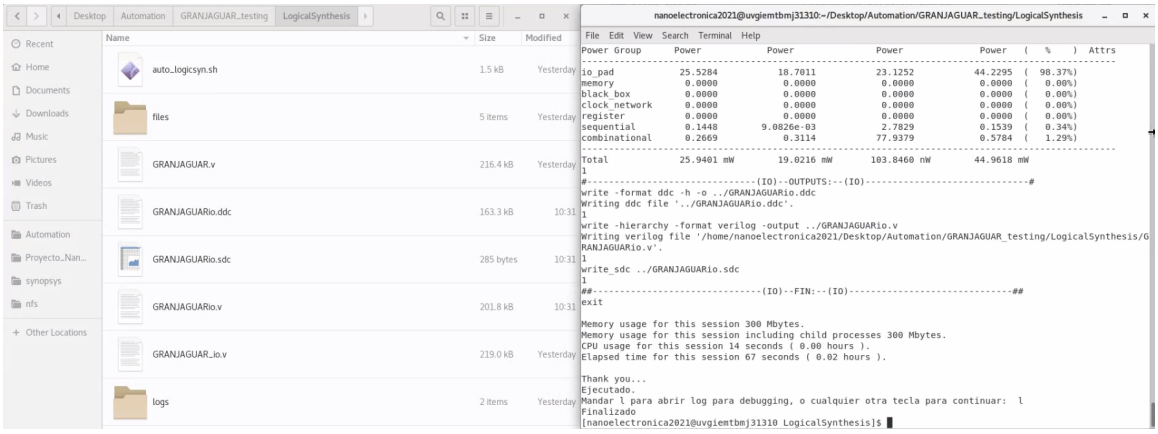


Figura 10: Síntesis lógica en proceso

Finalmente se presenta la salida más relevante de este proceso. Una vez ya finalizado, se crea otro archivo *verilog*. Se puede ver que los nombres de las celdas que este ya utiliza la librería proveída por *TSMC* para describir el circuito propuesto:

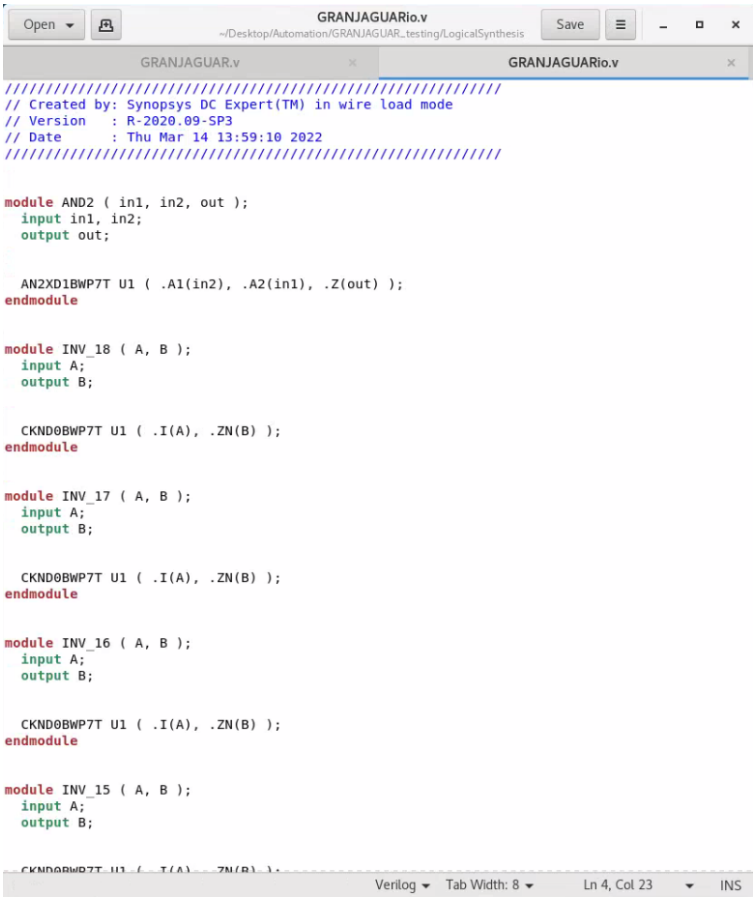


Figura 11: GJ Salida de síntesis lógica

11.2. Síntesis física del Gran Jaguar

La siguiente sección utiliza un método o protocolo parecido, abriendo una terminal en la carpeta de síntesis física y se inicia el proceso con *bash auto_physicsyn.sh*. Se podrá ver en este programa que se sigue el protocolo propuesto por Antonio Altuna, creando una librería de trabajo. Se borra la librería anterior y se inicia una nueva librería debido al hecho que no se puede garantizar si ya existe. [1]

Este es el proceso más largo, especialmente si es la primera vez que se corre. A continuación se muestra este proceso en camino:

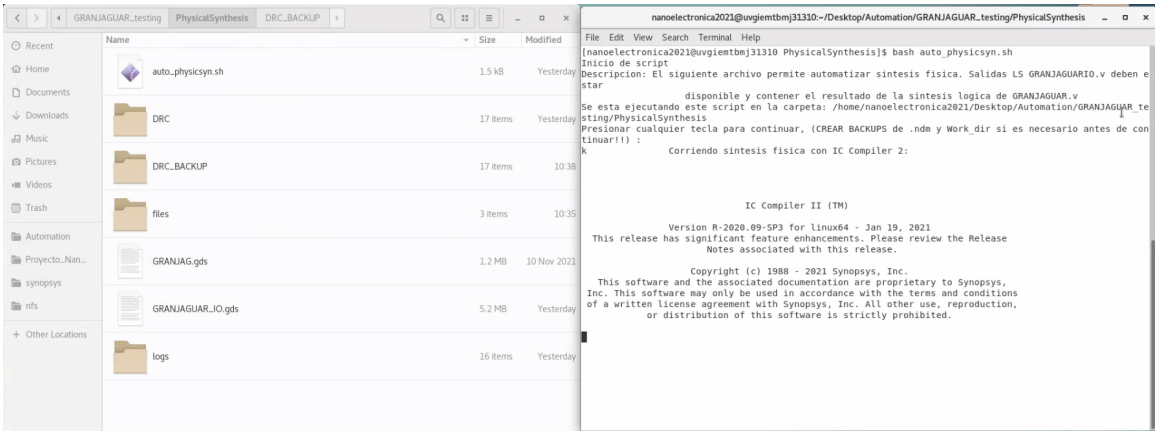


Figura 12: GJ Síntesis física en proceso

Por supuesto, realizar este proceso sin la terminal significa que no sería posible ver como se ejecuta cada paso. Con el motivo de obtener esto, se abrió *ICC2* para tomar estas imágenes, y corriendo cada paso, se tomaron las siguientes imágenes.

Este mismo proceso puede ser realizado en la automatización, sin la necesidad de una interfaz gráfica:

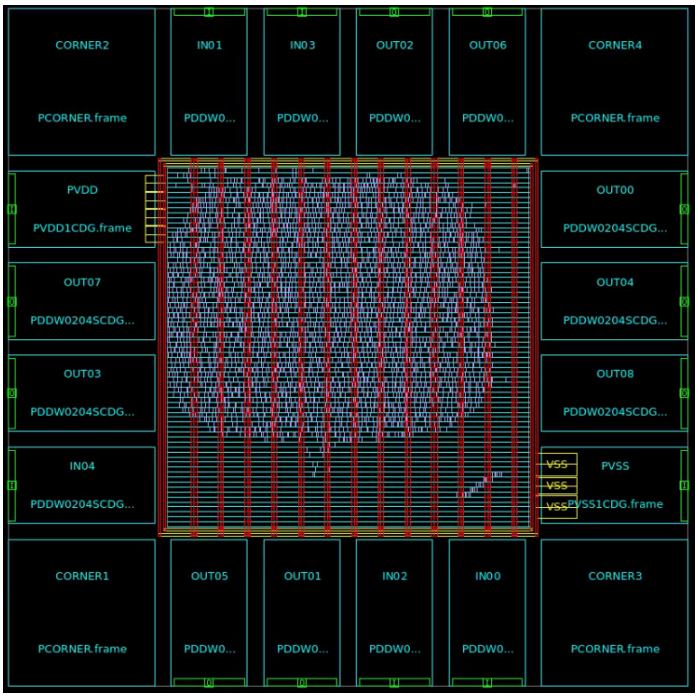


Figura 13: GJ Síntesis física - Placement

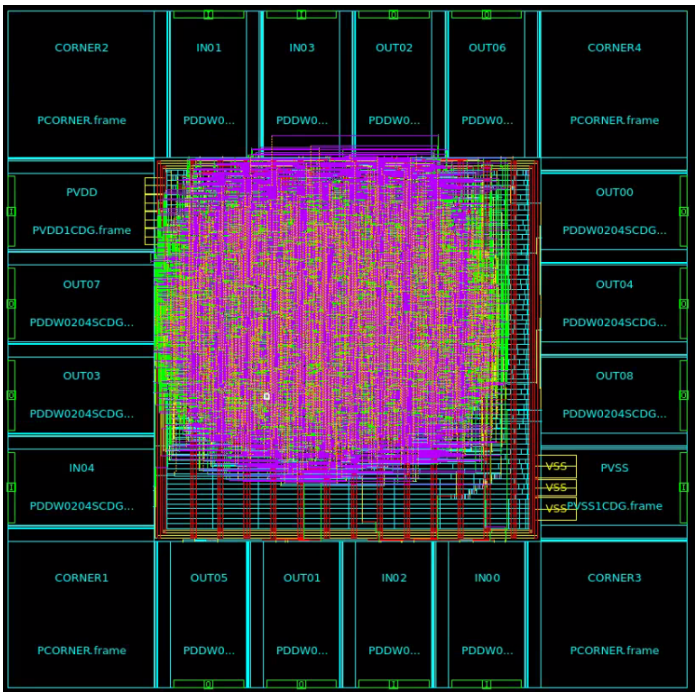


Figura 14: GJ Síntesis física - Routing

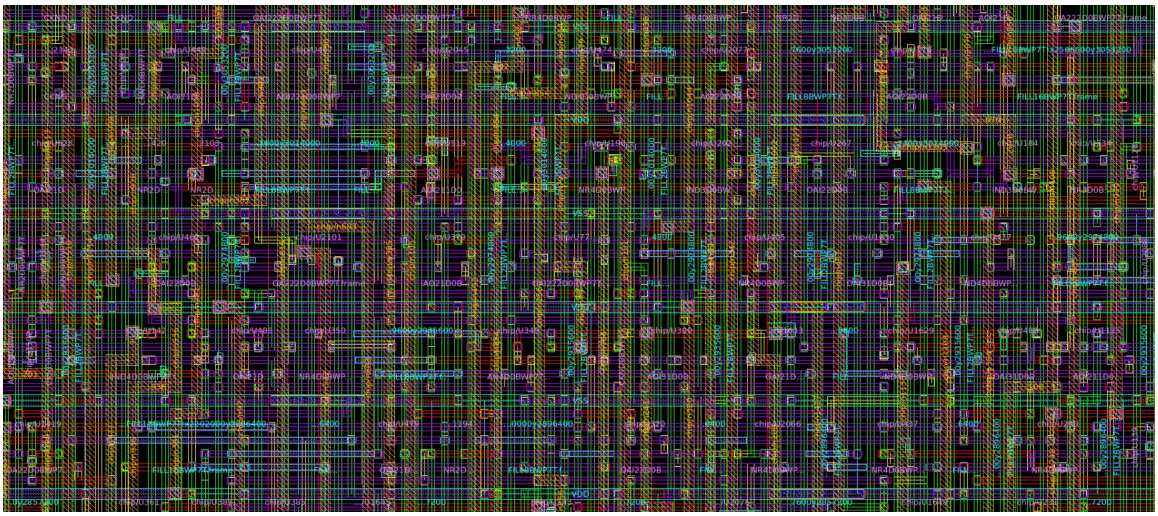


Figura 15: GJ Síntesis física - Routing Detallado

En medio de este proceso, se quiere utilizar *DRC* para hacer constantes correcciones al *Layout*. Para eso, se utiliza el siguiente proceso antes de generar el archivo *.gds* para las últimas etapas.

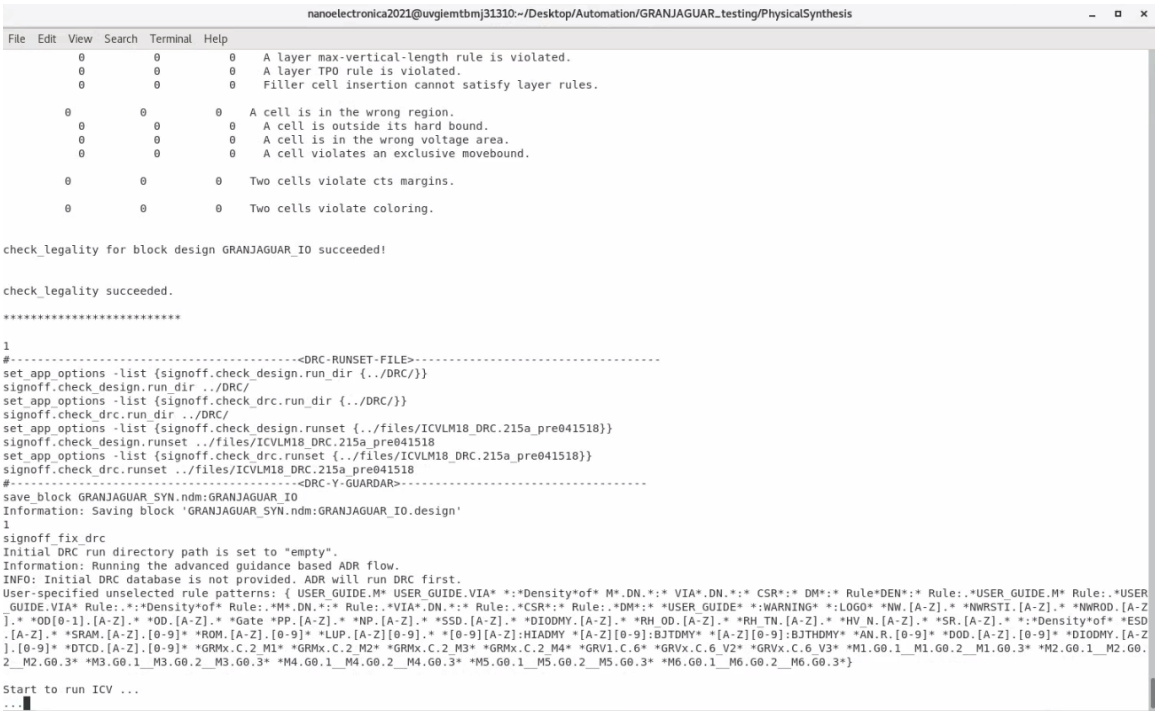


Figura 16: GJ Síntesis física - Finalizado

11.3. DRC del Gran Jaguar

DRC es corrido una vez más para ubicar los errores finales que no fueron resueltos en las iteraciones pasadas. Como fue mencionado anteriormente *DRC* en medio de la síntesis física, utilizado para realizar las correcciones apropiadas. A continuación, se muestra como esto se presenta en la terminal abierta:

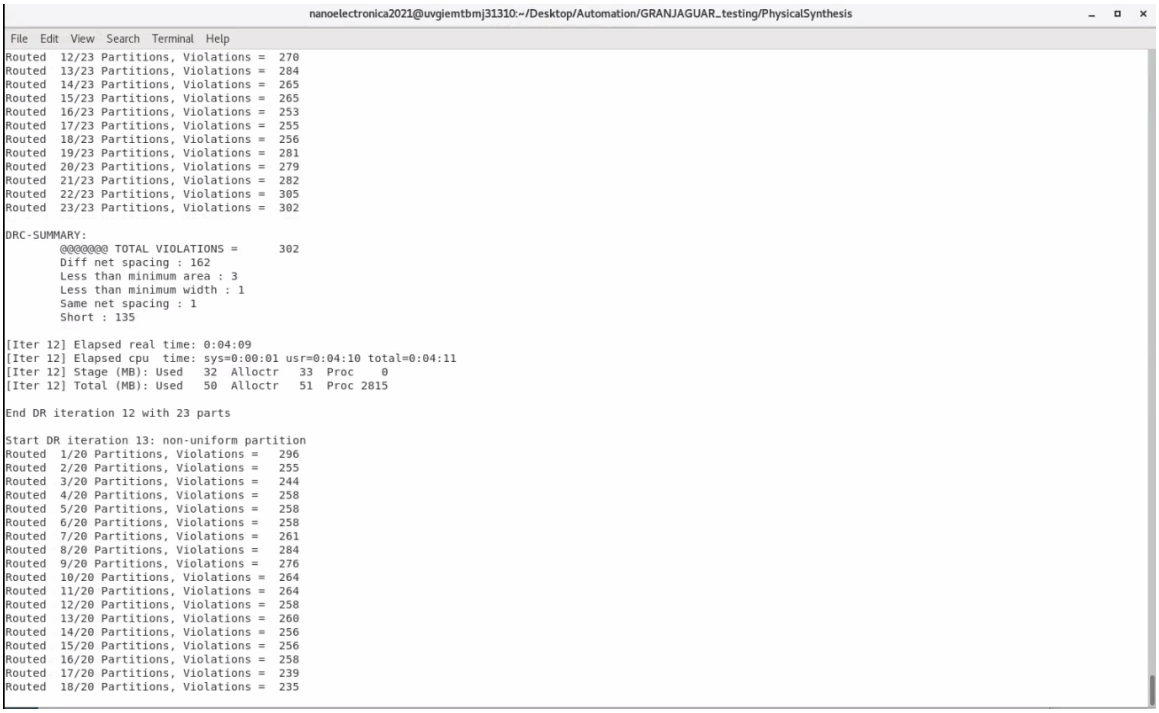


Figura 17: GJ DRC - En proceso

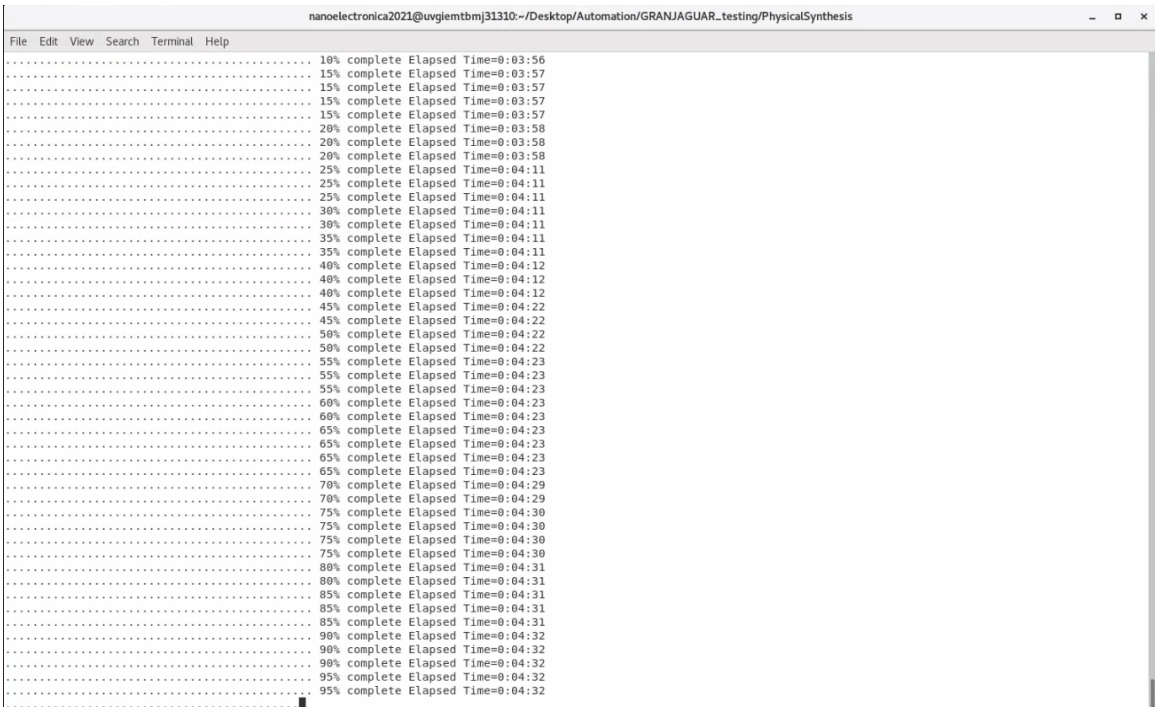


Figura 18: GJ DRC - Procesamiento final

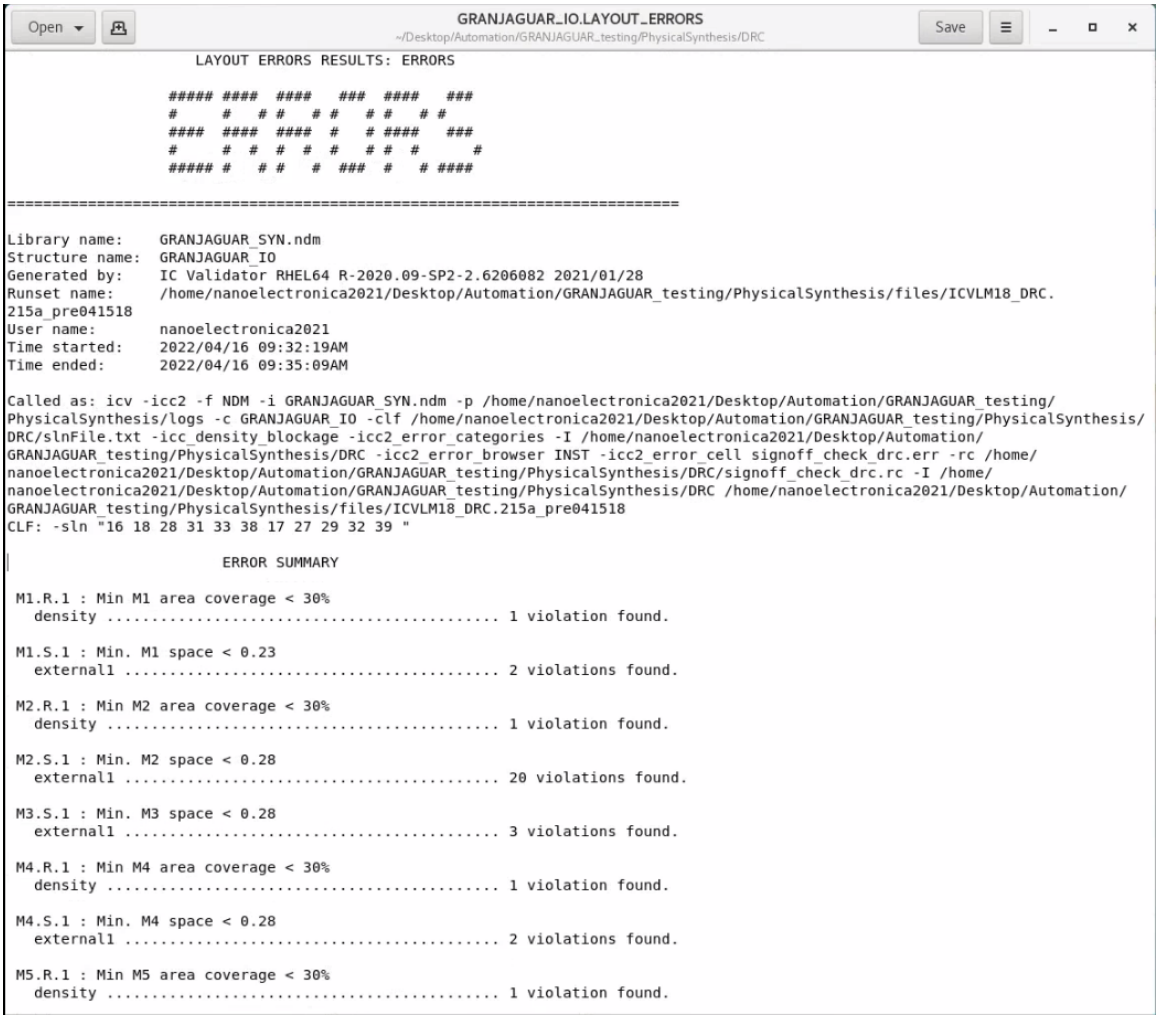


Figura 19: GJ DRC - Errores detectados

Se recuerda que este fue un intento completamente independiente al trabajo del equipo anterior de síntesis lógica utilizando la última del Gran Jaguar preparada por Elmer Torres. [26] Este resultado indica que el Gran Jaguar fue sintetizado, pero todavía con errores relevantes de DRC. Con esto se cumplen los objetivos de este trabajo, y una vez cumplido estas metas, se desea realizar una replicación de los resultados del equipo anterior. Para replicar los resultados, se utiliza el Gran Jaguar versión 2, que fue sintetizado una vez por el equipo anterior. Corriendo este, se obtuvieron los siguientes resultados:

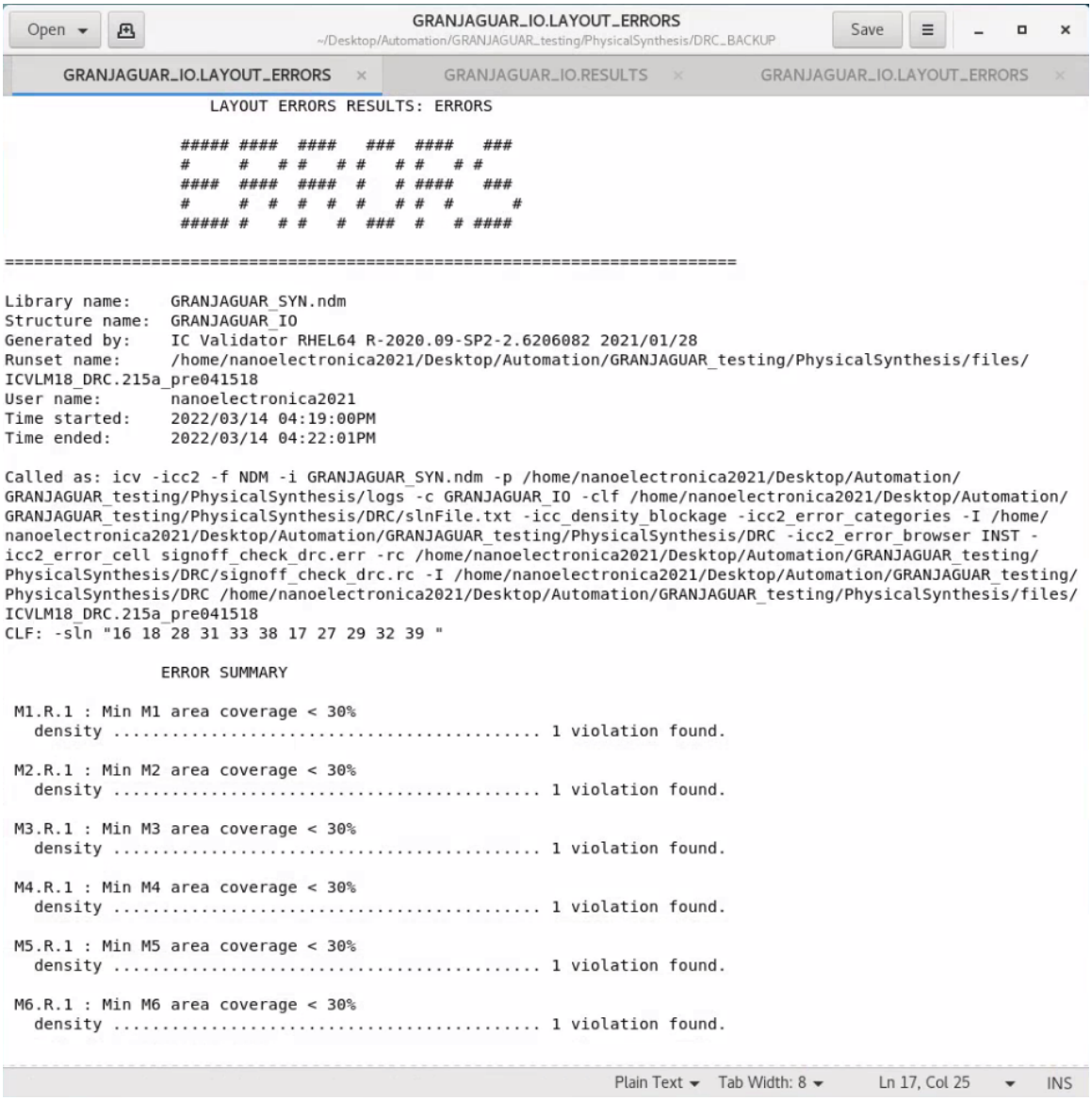


Figura 20: GJ DRC - Errores detectados para versión 2

Se resalta que el proceso de DRC no salió limpio. Es importante aclarar que seis de los errores presentados replican los resultados encontrados por el equipo anterior de síntesis lógica. Estas reglas pendientes son las únicas que no han sido posibles eliminar. [16] Como sus descripciones indican, Mejorar los resultados de *DRC* fue intentado, pero no fue la prioridad de este trabajo. Aun así, estos se presentan como parte de lo que fue logrado por el equipo, y que fue replicado por la automatización.

11.4. Verificación de antena del Gran Jaguar

El proceso de antena es idéntico a LVS, solamente necesitando correr $auto\_ANTENNA.sh$ desde su fólder. Este realiza la verificación rápidamente. A continuación, se agrega el resul-

tado de este proceso:

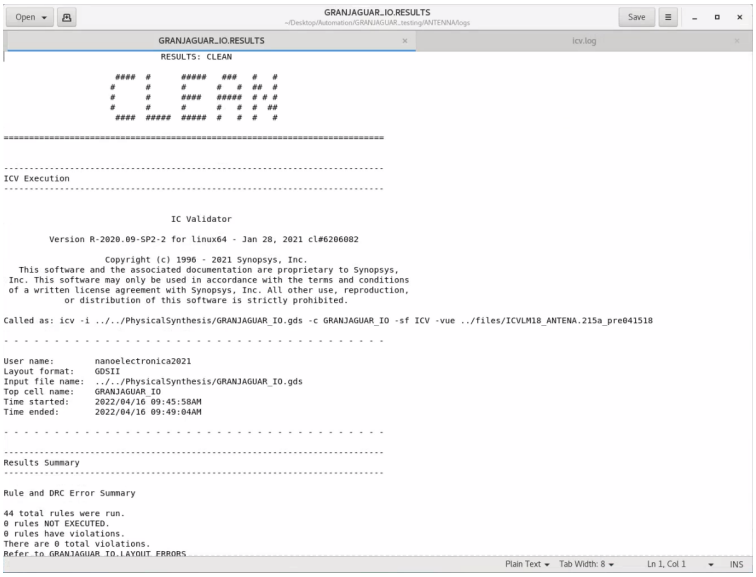


Figura 21: GJ Verificación de antena

Verificación de antena nunca ha presentado un problema. Por medio de esto, se verifica el proceso de diseño realizada por el equipo para esta sección. [3]

11.5. LVS del Gran Jaguar

Como fue explicado anteriormente, *LVS* es un proceso rápido una vez automatizado. Para añadir un nuevo diseño, es necesario agregar al *runset* los registros de nuevas celdas, y agregarlos como *blackboxes*.

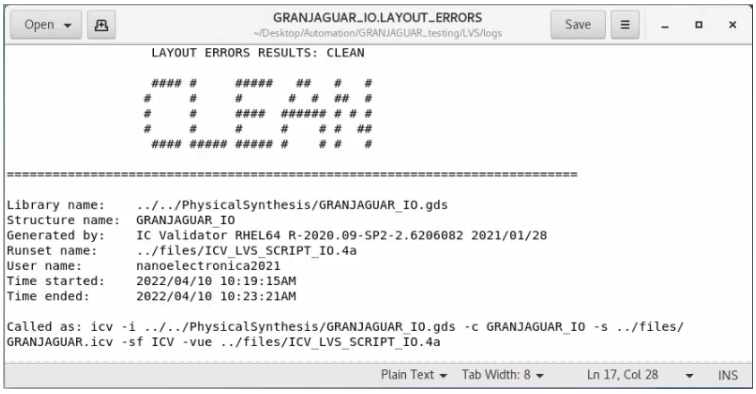


Figura 22: GJ LVS LAYOUT RESULTS

Estos resultados son idénticas a los que fueron presentados por José Ruiz. [14]

11.6. LPE del Gran Jaguar

Finalmente, la última sección realiza la extracción de componentes parásitos. Este previamente fue desarrollado por Charlie Cruz, utilizando *Custom Compiler*. [5] Durante el transcurso de este trabajo, se saltó a la automatización después de presentar problemas en la interfaz gráfica. El resultado es el siguiente proceso:

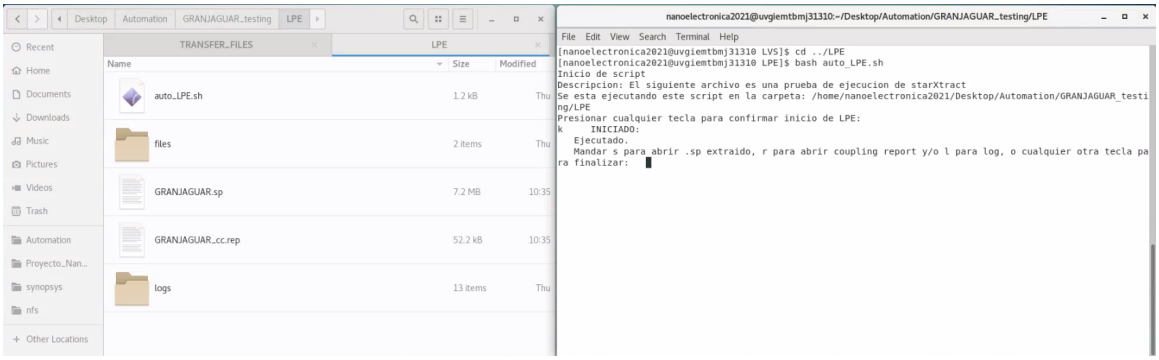


Figura 23: GJ LPE - En proceso

El resultado de este proceso es el siguiente archivo. Es importante notar que este no tiene generado las *nets* de entradas y salidas para facilitar la simulación. Como fue explicado al final del capítulo anterior, esto se debe a que actualmente se utiliza una librería no comercial de *TSMC*, esta tiene toda la información y data utilizada, sin embargo, tiene data o capas ocultas. [5]

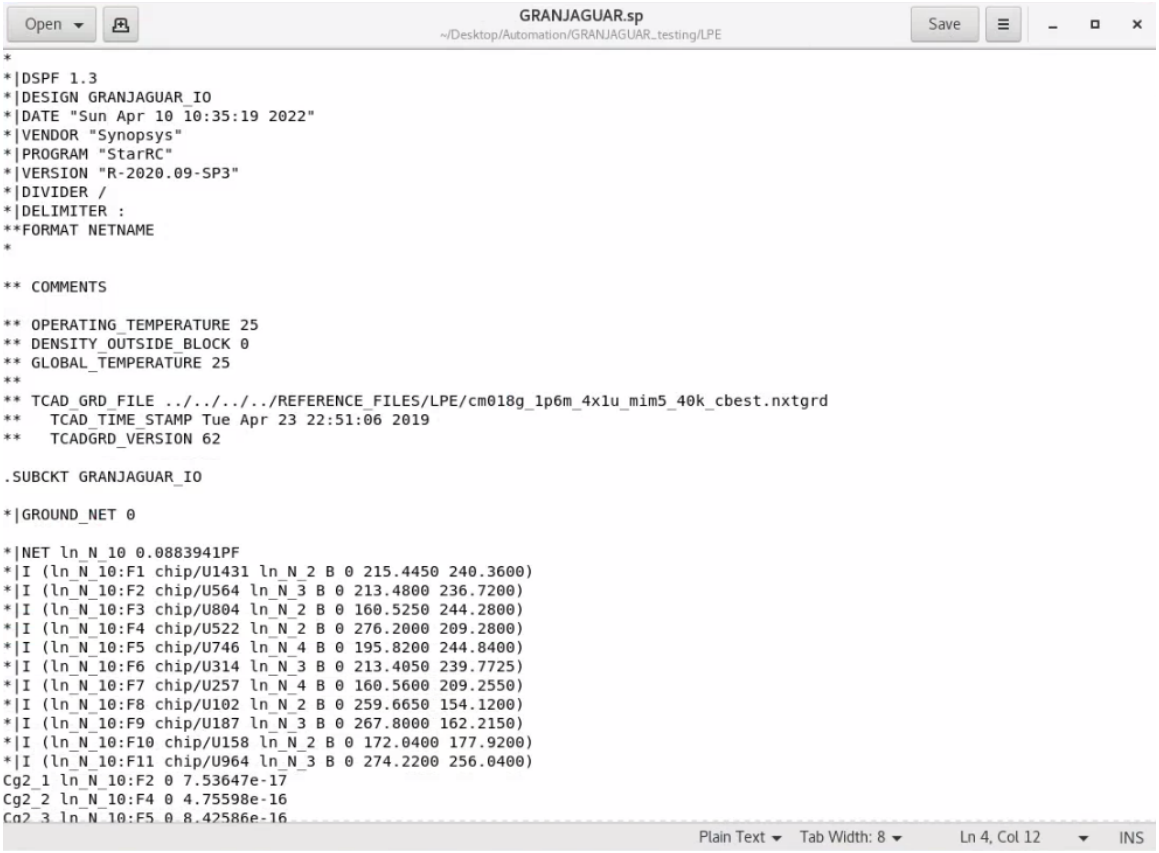


Figura 24: GJ LPE - Finalizado

Con eso, se presenta todo el proceso que fue automatizado para el Gran Jaguar. Este requiere intervención mínima del usuario, permitiendo que un solo usuario con conocimientos básicos de *Linux* utilice estos sistemas desde cualquier posición en la que está definida el directorio de .ªutomation".

CAPÍTULO 12

Conclusiones

1. Se analizó y detalló el proceso de extracción de componentes parásitos para facilitar el uso de una simulación. Esta principalmente se podría utilizar para asegurar que la potencia y el tiempo de respuesta sean aceptables y precisos para la aplicación deseada.

2. Se implementó un método para automatizar los procesos del flujo de diseño, con el propósito de facilitar lo más posible estos procesos para un usuario, potencialmente agilizando futuros diseños y facilitando la capacitación de estudiantes para usar estas herramientas.

3. Utilizando este proceso propuesto, se ejecutaron diversas extracciones de los diseños hechos por el equipo de trabajo. Se demostró el uso efectivo de los archivos proveídos por TSMC para estas aplicaciones.

4. Se exploraron las herramientas ofrecidas por *Synopsys*. Se verificaron y describieron sus limitaciones, requisitos, y ventajas. Se realizaron todas estas de tal manera que no se dependiera de la herramienta de *Custom Compiler*.

5. La falta de generación de puertos en el circuito final se debe a una falta de definición de entradas y salidas en su layout. Este se debe al hecho que las librerías de *TSMC* vienen con capas ocultas, es decir no comerciales, aunque estén completas. La simulación final de este sistema requiere de estas capas ocultas. Una librería alternativa puede ser necesaria para finalizar este paso.

6. Se implementó y desarrolló todo el flujo de diseño para la última versión propuesta para el Gran Jaguar. Esto incluye síntesis lógica, síntesis física, *DRC*, verificación de antena, *LVS* y *LPE*.

7. El sistema automatizado se impartió al próximo equipo del chip, de tal manera que un usuario entrenado en el flujo de diseño y con conocimientos básicos de *Linux* puede ejecutar todo el diseño y su verificación.

CAPÍTULO 13

Recomendaciones

1. Llevar buen registro y documentación de las soluciones temporales cuando se encuentra un problema. Estas pueden tener efectos mayores más adelante en el proceso, y por lo tanto, se debe tener una idea de donde se implementó un cambio que pudo haber desviado el proceso.

2. El análisis del documento de reglas para DRC es una fuente importante de conocimiento. Alteraciones internas pueden ser utilizadas para implementar el reglamento de diversas compañías de manufactura.

3. Conflictos de compatibilidad tienden a surgir cuando no todas las herramientas están actualizadas. Es importante estar al tanto de nuevas versiones, y de cuáles problemas estas solucionan, especialmente dado que estos pueden facilitar alguna parte hecha previamente.

4. Utilizar un servidor común para poder compartir archivos con el resto del equipo. Este sirven como un solo lugar donde todos pueden tomar referencias entre si. Es preferible si incluso se corren las pruebas y los archivos en la misma carpeta, ya que esta podría mejorar comunicación.

5. Como fue pasado al siguiente equipo, se recomienda la creación de scripts o interfaces gráficas que faciliten al usuario el uso de estas herramientas. En la versión actual, se prefirió entrar manualmente a los archivos de configuración para realizar modificaciones.

6. Se recomienda utilizar este documento como un introducción para capacitar a nuevos estudiantes en las herramientas de *Synopsis*. Configuraciones detalladas y definiciones para innovar en los procesos individuales requerirán el uso de los demás trabajos de graduación del equipo.

CAPÍTULO 14

Bibliografía

[1] A. Altuna Hernández, "Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: Ejecución de la síntesis física, Verificación de Reglas de Diseño y Corrección de Errores Obtenidos," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2021, pág. 125.

[2] E. T. |. Aspencore. (2013). "MOSFET and Metal Oxide Semiconductor Tutorial," dirección: `https://www.electronics-tutorials.ws/transistor/tran_6.html`. (accessado: 04.02.2021).

[3] J. A. Ayala Escobar, "Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: Ejecución de la Síntesis Física, Verificaciones de Antena y Corrección de Errores Obtenidos," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2021, pág. 99.

[4] U. of Berkely. (2017). "History and Impact of Computers | Unidad 6," dirección: `https://bjc.edc.org/bjc-r/course/bjc4nyc.html`. (accessado: 04.02.2021).

[5] C. A. Cruz Girón, "Ejecución y utilización de un flujo de diseño para el desarrollo de un chip con tecnología nanométrica: Extracción de componentes parásitos y simulaciones de HSPICE," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2019, pág. 59.

[6] M. G. Flores Espino, "Corrección de anillo de entradas/salidas y pruebas de antenna y ERC para la definición del flujo de diseño del primer chip con tecnología nanométrica desarrollado en Guatemala," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2019, pág. 88.

[7] J. R. Girón Rubio, "Etapa de Verificación física de Diseño en Silicio vs. Esquemático (LVS) en el flujo de diseño para un chip a nanoescala," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2019, pág. 90.

[8] D. D. S. H. Jinsik Yun, "Design Vision - Verilog Logic Synthesis Tool," en *AMCOM Communications*, MICS IAP Members, 2021.

[9] H. B. y. H. M. Kommuru, "ASIC Design Flow Tutorial," en *San Francisco, CA*, San Franscisco State University, 2009.

[10] A. P. Malvino y D. J. Bates, "Electronic Principles," en *Electronic Principles*, McGrawHill Education, 2016, pág. 654.

[11] L. A. Nájera Vásquez, "Implementación de circuitos sintetizados a nivel netlist a partir de un diseño en lenguaje descriptivo de hardware como primer paso en el flujo de diseño de un circuito integrado," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2019, pág. 72.

[12] J. N. Ruano Orellana, "Definición del flujo en la herramienta VCS para la simulación de HDLs en la Fabricación de un Chip con Tecnología Nanométrica CMOS," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2019, pág. 56.

[13] S. H. Rubio Vasquez, "Diseño del Flujo de Diseño para Fabricación de un Chip con Tecnología VLSI CMOS," en *Trabajo de Graduación Ingeniería Electrónica*, UVG, 2019, pág. 51.

[14] J. A. Ruiz Orozco, "Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: ejecución de la fase de verificación física Layout vs Schematic (LVS)," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2021, pág. 68.

[15] J. A. de los Santos, "Diseño de un sumador/restador completo de 32 bits con tecnología CMOS en un proceso de 28 nanómetros usando aplicaciones de diseño de la empresa Synopsys," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2014, pág. 108.

[16] J. E. Shin Jo, "Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: ejecución de la síntesis física, validación de reglas eléctricas y corrección de errores obtenidos," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2021, pág. 85.

[17] M. Sibrian Illescas, "Verificación de reglas de diseño (DRC) para el desarrollo de un flujo funcional de un circuito integrado con tecnología nanométrica," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2019, pág. 79.

[18] Synopsys, "StarRC User Guide and Command References," en *Software User Guide*, Synopsys, Inc., 2011, pág. 934.

[19] ——, "Custom WaveView," en *Software Security Datasheets*, Synopsys, Inc., 2013, págs. 1-7.

[20] ——, "StarRC Parasitic Extraction," en *Software Security Datasheets*, Synopsys, Inc., 2015, págs. 1-7.

[21] ——, "Custom Compiler DataSheet," en *Software Security Datasheets*, Synopsys, Inc., 2018, págs. 1-5.

[22] ——, "Design Compiler User Guide and Command References," en *Software User Guide*, Synopsys, Inc., 2019, pág. 792.

[23] ——, "IC Compiler II," en *Software Security Datasheets*, Synopsys, Inc., 2019, págs. 1-5.

[24] ——, "IC Validator Physical Verification DataSheet," en *Software Security Datasheets*, Synopsys, Inc., 2019, págs. 1-8.

[25] ——, (2021). "Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions," dirección: `https://www.synopsys.com/`. (accessed: 07.05.2021).

[26] E. O. Torres Garza, "Diseño de un circuito integrado con tecnología de 180 nm usando librerías de diseño de TSMC: Ejecución y simulación para la etapa de síntesis lógica," en *Universidad del Valle de Guatemala*, UVG Facultad de Ingeniería, 2021, pág. 99.

CAPÍTULO 15

Anexos

Existen varios documentos, códigos y scripts que no encajan en la explicación del desarrollo de automatización. El propósito de este reporte es que sirva como documentación para optimizar el uso de estos scripts y replicarlo. Especialmente para facilitar el proceso de *debugging*, se añaden a continuación varias referencias utilizadas, al igual que algunos de los códigos más largos que no están completos en los demás trabajos:

15.1. Flujo de diseño

Las siguientes dos imágenes han sido útiles para explicar el proceso llevado para todas las secciones. Se añaden a continuación para dejar referencia del proceso replicado:

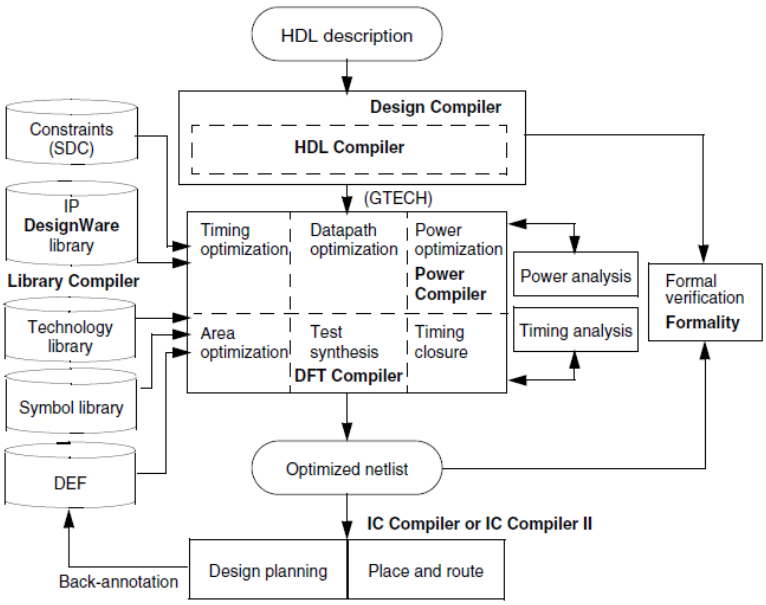


Figura 25: DC and ICC Flow [22]

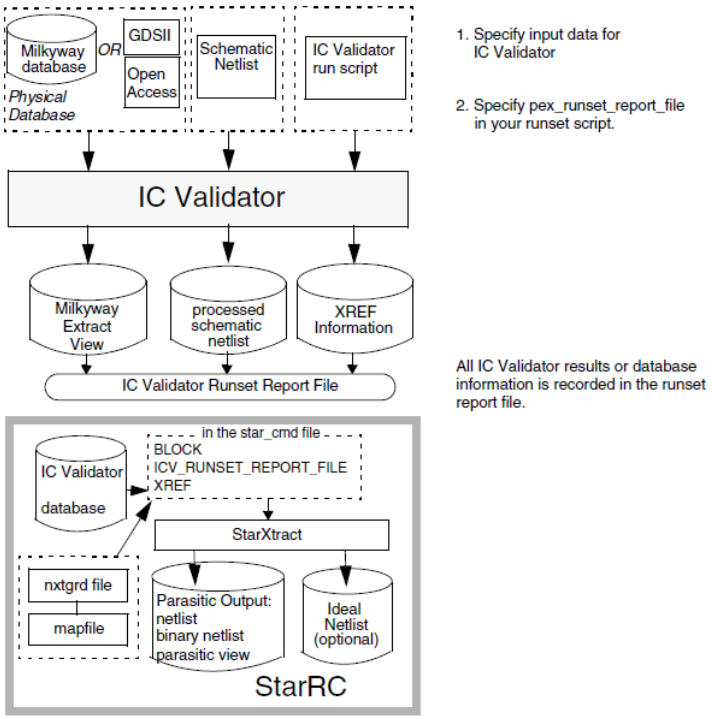


Figura 26: ICV and StarRC Flow [18]

15.2. Resultados

La organización del *fólder* ha resultado como una de las mayores ventajas de este trabajo. Sin esta organización, se haría difícil encontrar archivos específicos en cada sección. Sin embargo, se hace necesario explicar esta jerarquía adecuadamente:

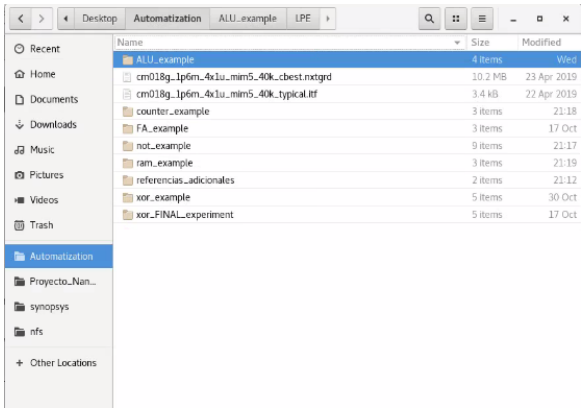


Figura 27: Fólder de automatización

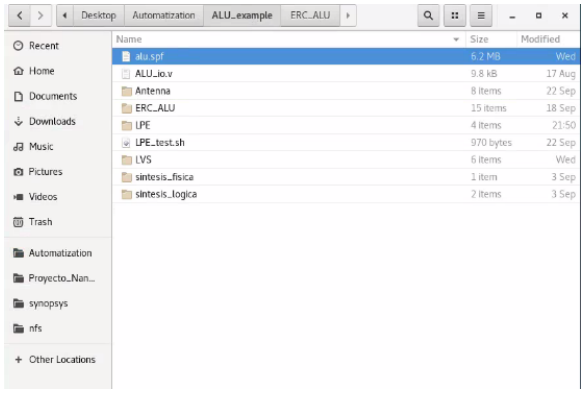


Figura 28: Fólder de ALU

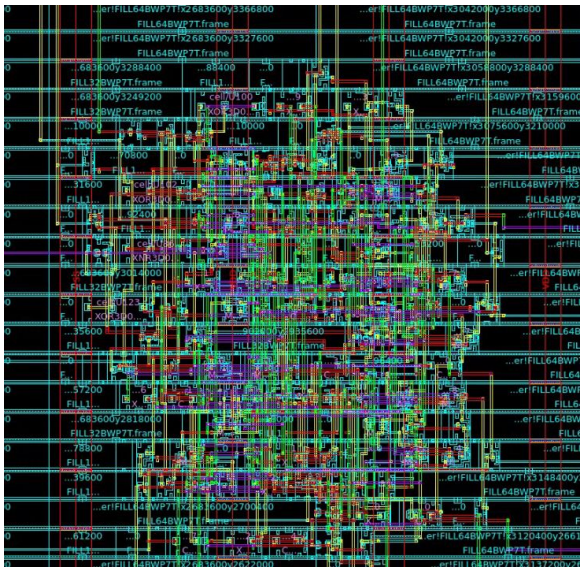


Figura 29: Zoom de layout ALU

Se repiten los resultados para el Gran Jaguar en esta sección:

GRANJAGUARio.v
~/Desktop/Automation/GRANJAGUAR_testing/LogicalSynthesis

GRANJAGUAR.v | GRANJAGUARio.v

```
////////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : R-2020.09-SP3
// Date      : Thu Mar 14 13:59:10 2022
////////////////////////////////////////////////////////////////


module AND2 ( in1, in2, out );
  input in1, in2;
  output out;

  AN2XD1BWP7T U1 ( .A1(in2), .A2(in1), .Z(out) );
endmodule


module INV_18 ( A, B );
  input A;
  output B;

  CKND0BWP7T U1 ( .I(A), .ZN(B) );
endmodule


module INV_17 ( A, B );
  input A;
  output B;

  CKND0BWP7T U1 ( .I(A), .ZN(B) );
endmodule


module INV_16 ( A, B );
  input A;
  output B;

  CKND0BWP7T U1 ( .I(A), .ZN(B) );
endmodule


module INV_15 ( A, B );
  input A;
  output B;
```

Verilog | Tab Width: 8 | Ln 4, Col 23 | INS

Figura 30: GJ salida de síntesis lógica

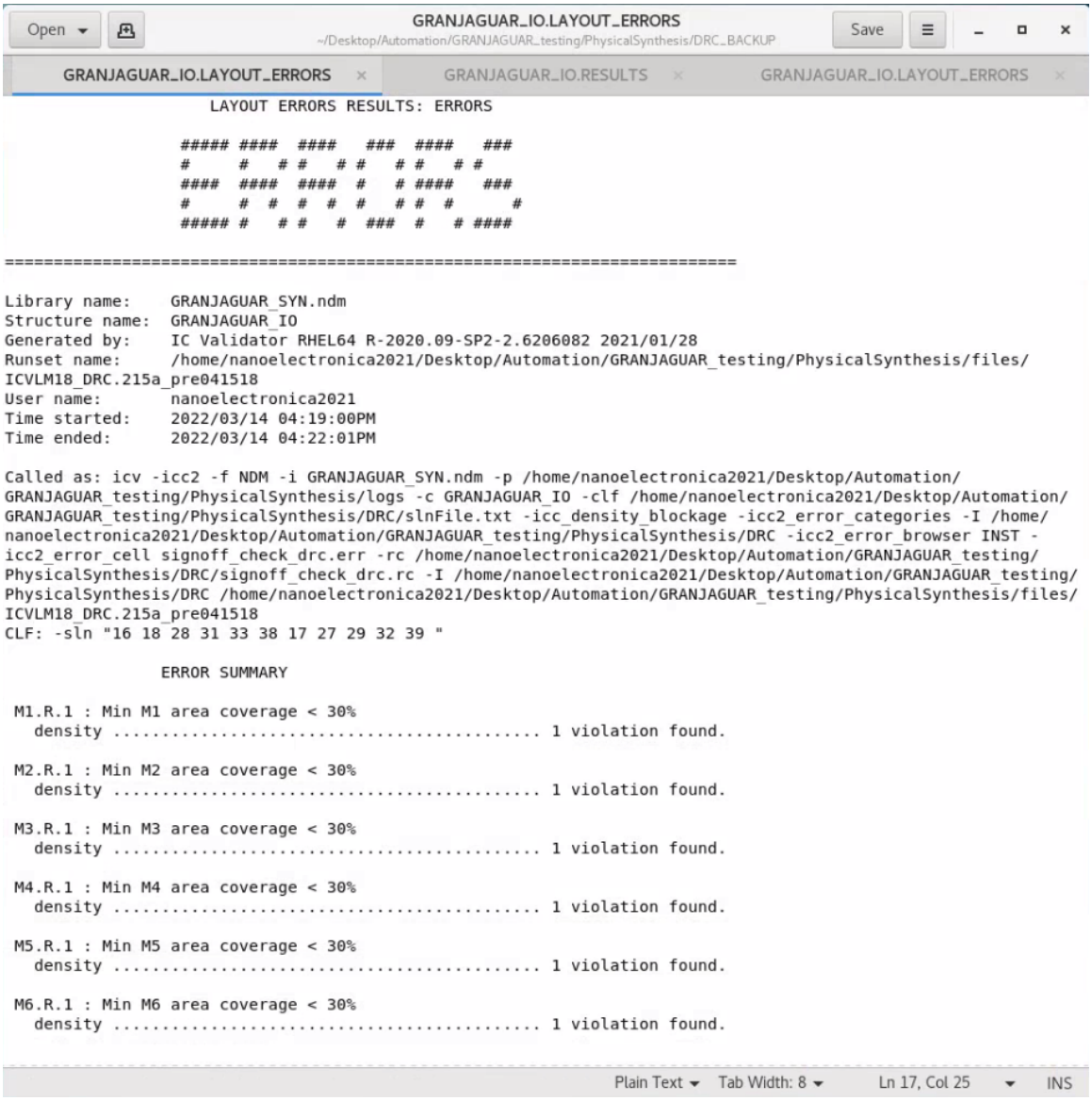


Figura 31: GJ DRC - Errores detectados para versión 2

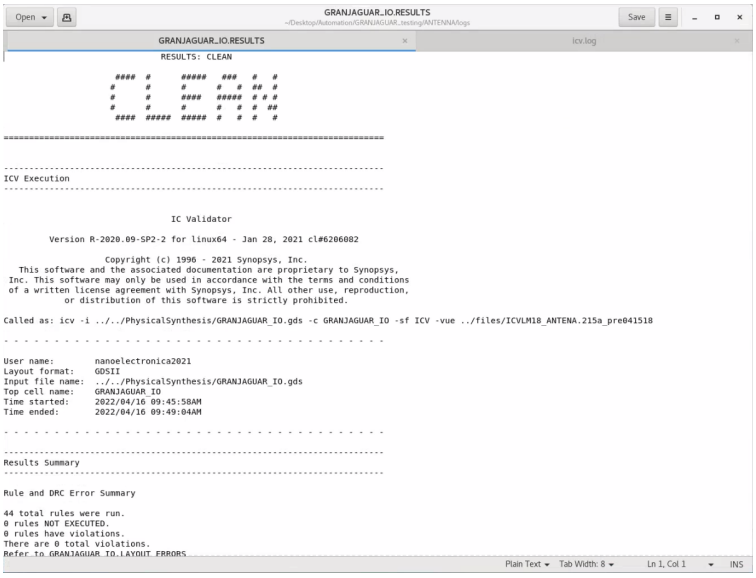


Figura 32: GJ verificación de antena

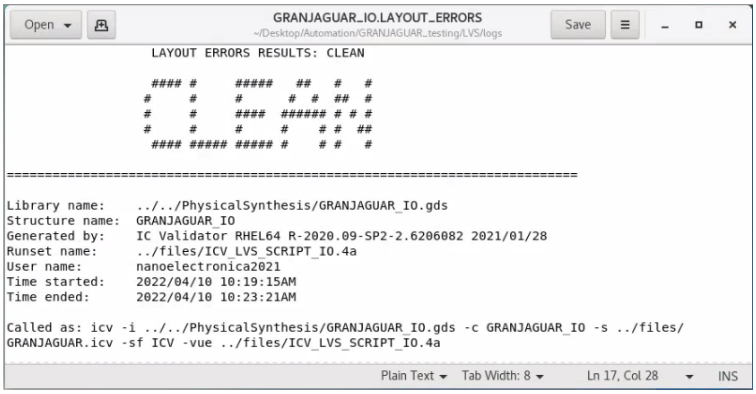


Figura 33: GJ LVS LAYOUT RESULTS

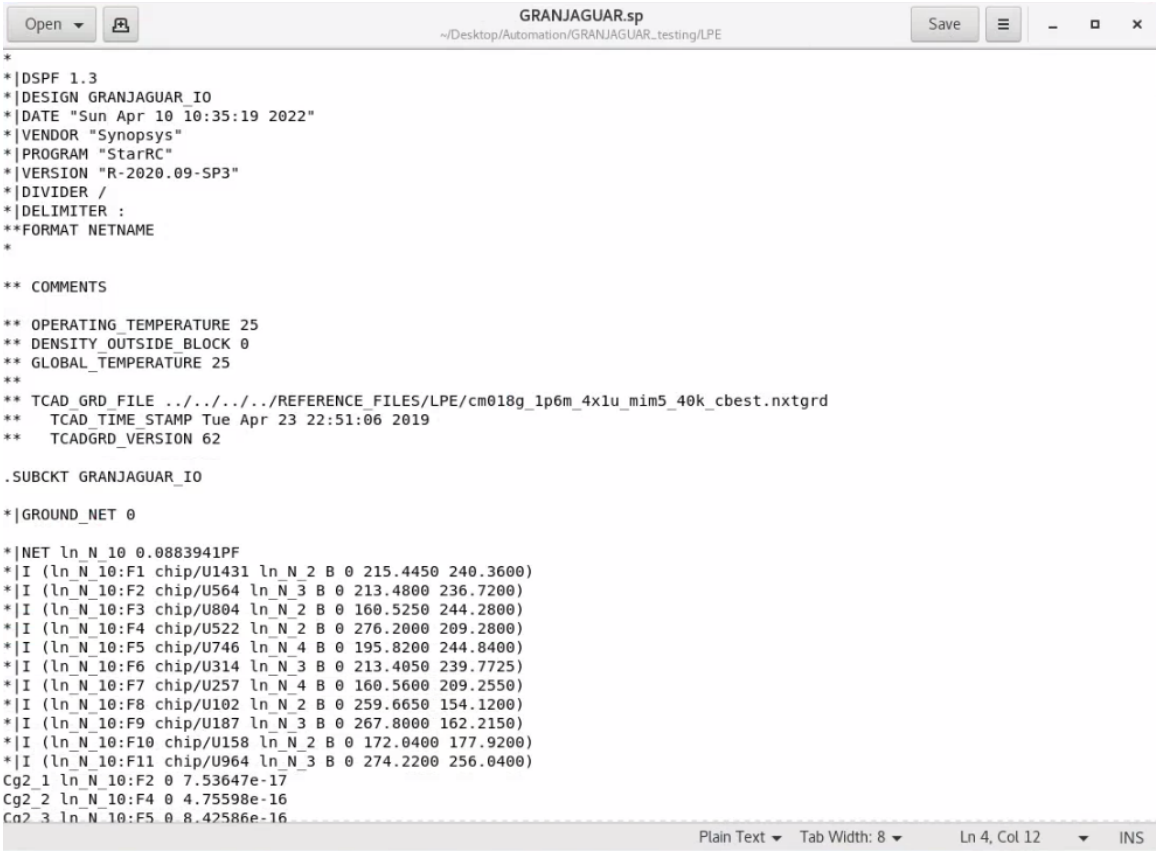


Figura 34: GJ LPE - Finalizado

15.3. Scripts de automatización

Como fue explicado anteriormente, es necesario crear scripts de automatización, y no solamente instruir al usuario cuales comandos se utilizan para cargar los runsets. Esto es debido al hecho que varios pasos del flujo de diseño requieren hacer carpetas, construir archivos, llamar múltiples veces un programa para ejecutar distintas secciones de un proceso. Es debido a esto que se utilizan scripts *.sh* para correr estas funciones. En orden paralelo al flujo de diseño, estos se agregan a continuación:

Listing 15.1: autologicsyn.sh | Script para síntesis lógica

```
1 #!/bin/bash
2 #=============================================================================

3 # Nombre:        auto_logicsyn.sh
4 # Fecha:         20/8/2021
5 # Autor:         J. Gonzalez Herrera
6 # Requisitos:    Asegurar permiso de ejecucion con chmod +x <nombre.sh>
7 #                Preparar lista de comandos para design vision en un .
      script
8 #                Asegurar rutas correctas en comandos de .script
```

```
9  #                Correr este .sh con comando: sh <nombre.sh> o
       equivalente
10 #
       ================================================================================


11
12 echo "Inicio de script"
13 echo "Descripcion: El siguiente archivo permite automatizar sintesis
       logica."
14 echo " Archivo GRANJAGUAR.v debe estar"
15 echo "              disponible y contener el resultado de la sintesis "
16 echo "logica de GRANJAGUAR.v"
17 echo -n "Se esta ejecutando este script en la carpeta: "
18 echo $PWD
19
20 echo "Presionar cualquier tecla para continuar:"
21 while [ true ] ; do
22 read -t 7 -n 1
23 if [ $? = 0 ]; then
24 break ;
25 else
26 echo "Esperando por entrada de usuario..."
27 fi
28 done
29
30
31 echo "          Corriendo sintesis logica con Design Vision:"
32 cd ./logs
33
34 dc_shell -f ../files/DV_LogSyn.script
35
36 echo "Presionar cualquier tecla para continuar con IO:"
37 while [ true ] ; do
38 read -t 10 -n 1
39 if [ $? = 0 ]; then
40 break ;
41 else
42 echo "Esperando por entrada de usuario..."
43 fi
44 done
45
46 dc_shell -f ../files/DV_LogSynio.script
47
48 cd ..
49
50 echo "Ejecutado."
51
52 read -p "Mandar l para abrir log para debugging, o cualquier otra tecla
       para continuar: " str1
53 if [[ "$str1" == *"l"* ]]; then
54 gio open ./logs/command.log
55 #xdg-open ./files/logs/command.log
56 fi
57
```

```
58 echo "Finalizado"
59
60 exit
```

Listing 15.2: autophysicsyn.sh | Script de síntesis física y DRC

```
1  #!/bin/bash
2  #==========================================================================
3  # Nombre:         auto_physicsyn.sh
4  # Fecha:          9/2/2022
5  # Autor:          J. Gonzalez Herrera
6  # Requisitos:     Asegurar permiso de ejecucion con chmod +x <nombre.sh>
7  #                 Preparar lista de comandos para IC Compiler 2 en un .tcl
8  #                 Asegurar rutas correctas en comandos de .tcl
9  #                 Correr este .sh con comando: sh <nombre.sh> o
       equivalente
10 #==========================================================================
11
12 echo "Inicio de script"
13 echo "Descripcion: El siguiente archivo permite automatizar sintesis
       fisica. "
14 echo "Salidas LS GRANJAGUARIO.v deben estar"
15 echo "              disponible y contener el resultado de la sintesis
       logica "
16 echo "de GRANJAGUAR.v"
17 echo -n "Se esta ejecutando este script en la carpeta: "
18 echo $PWD
19
20 echo "Presionar cualquier tecla para continuar, "
21 echo "(CREAR BACKUPS de .ndm y Work_dir si es necesario antes de
       continuar!!) :"
22 while [ true ] ; do
23 read -t 7 -n 1
24 if [ $? = 0 ]; then
25 break ;
26 else
27 echo "Esperando por entrada de usuario..."
28 fi
29 done
30
31
32 echo "           Corriendo sintesis fisica con IC Compiler 2:"
33 cd ./logs
34
35 rm -rf ./*_SYN.ndm
36 rm -rf ./work_dir
37
38 icc2_shell -file ../files/ICC2_PS.tcl
39
40 echo "Ejecutado."
41
42 cd ..
43
```

```
44 read -p "Mandar l para abrir log para debugging, r para DRC results, o
       cualquier otra tecla para continuar on antena:   " str1
45 if [[ "$str1" == *"l"* ]]; then
46 gio open ./logs/icc2_output.txt
47 fi
48 if [[ "$str1" == *"r"* ]]; then
49 gio open ./DRC/GRANJAGUAR_IO.RESULTS
50 fi
51
52
53 echo "Finalizado"
54
55 exit
```

Listing 15.3: autoantenna.sh | Script para verificación de antena

```
1  #!/bin/bash
2  #==========================================================================
3  # Nombre:         auto_antenna.sh
4  # Fecha:          19/3/2021
5  # Autor:          J. Gonzalez Herrera
6  # Requisitos:     Asegurar permiso de ejecucion con chmod +x <nombre.sh>
7  #                 Preparar lista de comandos para IC Compiler 2 en un .tcl
8  #                 Asegurar rutas correctas en comandos de .tcl
9  #                 Correr este .sh con comando: sh <nombre.sh> o
       equivalente
10 #==========================================================================
11
12 echo "Inicio de script"
13 echo "Descripcion: El siguiente archivo permite automatizar verificacion
       de "
14 echo "antenna. .gds generado desde Sintesis Fisica debe estar "
15 echo "disponible."
16 echo -n "Se esta ejecutando este script en la carpeta: "
17 echo $PWD
18
19 echo "Presionar cualquier tecla para continuar:"
20 while [ true ] ; do
21 read -t 7 -n 1
22 if [ $? = 0 ]; then
23 break ;
24 else
25 echo "Esperando por entrada de usuario..."
26 fi
27 done
28
29
30 echo "          Ahora corriendo antena con IC Compiler 2:"
31 cd logs
32
33 icv -i ../../PhysicalSynthesis/GRANJAGUAR_IO.gds -c GRANJAGUAR_IO -sf
       ICV -vue ../files/ICVLM18_ANTENA.215a_pre041518
34
35 cd ..
```

```
36
37 echo "Ejecutado."
38
39 read -p "Mandar r para abrir resultados y/o l para log, o cualquier otra
       tecla para continuar con LVS:  " str1
40
41 if [[ "$str1" == *"r"* ]]; then
42 gio open ./logs/GRANJAGUAR_IO.RESULTS
43 fi
44 if [[ "$str1" == *"l"* ]]; then
45 gio open ./logs/icv.log
46 fi
47
48
49 echo "Finalizado"
50
51 exit
```

Listing 15.4: autoLVS.sh | Script para verificación LVS

```
1  #!/bin/bash
2  #=============================================================================
3  # Nombre:         auto_LVS.sh
4  # Fecha:          15/2/2022
5  # Autor:          J. Gonzalez Herrera
6  # Requisitos:     Asegurar permiso de ejecucion con chmod +x <nombre.sh>
7  #                 Preparar lista de comandos para IC Compiler 2 en un .tcl
8  #                 Asegurar rutas correctas en comandos de .tcl
9  #                 Correr este .sh con comando: sh <nombre.sh> o
       equivalente
10 #=============================================================================
11
12 echo "Inicio de script"
13 echo "Descripcion: El siguiente archivo permite automatizar verificacion
       de "
14 echo "  LVS. diversos archivos deben estar disponibles de todos los
       procesos "
15 echo "  anteriores. Pasos detallados encontrados en PASOS_LVS_README.txt
       "
16 echo "  diversos archivos deben estar disponibles de todos los procesos
       "
17 echo "anteriores."
18 echo -n "Se esta ejecutando este script en la carpeta: "
19 echo $PWD
20
21 echo "Presionar cualquier tecla para confirmar inicio de LVS:"
22 while [ true ] ; do
23 read -t 7 -n 1
24 if [ $? = 0 ]; then
25 break ;
26 else
27 echo "Esperando por entrada de usuario...)"
28 fi
29 done
```

```
30
31
32
33
34 cd logs
35 echo "          Traduccion Verilog a Netlist ICV..."
36 icv_nettran -verilog ../../LogicalSynthesis/GRANJAGUARio.v -outType ICV
       -outName ../files/GRANJAGUAR.icv
37 echo "          Traduccion Verilog a Netlist SPICE..."
38 icv_nettran -verilog ../../LogicalSynthesis/GRANJAGUARio.v -outType
       SPICE -outName ../files/GRANJAGUAR.sp
39
40
41 echo "          Concatenacion de headers (CORE.v y IO.v ya generados) en
        files/header.v"
42 cat ../files/CORE.v ../files/IO.v> ../files/headers.v
43 echo "          Conversion a SPICE en files/headers_temp.sp"
44 icv_nettran -verilog ../files/headers.v -outType SPICE -outName ../files
       /headers_temp.sp
45 echo "          Definicion de inicio de headers"
46 cat ../files/TO_ADD.txt ../files/headers_temp.sp > ../files/headers.v
47
48
49 echo "          Ejecucion de LVS"
50 icv -i ../../PhysicalSynthesis/GRANJAGUAR_IO.gds -c GRANJAGUAR_IO -s ../
       files/GRANJAGUAR.icv -sf ICV -vue ../files/ICV_LVS_SCRIPT_IO.4a
51
52 cd ..
53
54
55
56
57 read -p "   Mandar s para abrir summary, r para resultados y/o l para
       log, o cualquier otra tecla para continuar on LPE:        " str1
58
59 if [[ "$str1" == *"s"* ]]; then
60 gio open ./logs/run_details/GRANJAGUAR_IO.sum
61 fi
62 if [[ "$str1" == *"r"* ]]; then
63 gio open ./logs/GRANJAGUAR_IO.RESULTS
64 fi
65 if [[ "$str1" == *"l"* ]]; then
66 gio open ./logs/run_details/ICV_LVS.dp.log
67
68 fi
69
70 echo "Finalizado"
71
72 exit
```

Listing 15.5: autoLPE.sh | Script para extracción LPE

```
1 #!/bin/bash
```

```
2 #=============================================================================

3 # Nombre:        auto_LPE.sh
4 # Fecha:         22/9/2021
5 # Autor:         J. Gonzalez Herrera
6 # Requisitos:    Este script sirve para probar la automatizacion de
      extraccion
7 #                de parasitos para GRANJAGUAR. Esta info sera util para
      el trabajo.
8 #
     ============================================================================

9
10 echo "Inicio de script"
11 echo "Descripcion: El siguiente archivo es una prueba de ejecucion de
      starXtract"
12 echo -n "Se esta ejecutando este script en la carpeta: "
13 echo $PWD
14
15
16 echo "Presionar cualquier tecla para confirmar inicio de LPE:"
17 while [ true ] ; do
18 read -t 7 -n 1
19 if [ $? = 0 ]; then
20 break ;
21 else
22 echo "Esperando por entrada de usuario..."
23 fi
24 done
25
26 echo "     INICIADO:"
27 cd ./logs;
28
29 StarXtract ../files/LPE_TSMC_script.cmd > ./stdout.lpe.log 2>&1
30
31 cd ..
32
33 echo "   Ejecutado."
34
35 read -p "   Mandar s para abrir .sp extraido, r para abrir coupling
      report y/o l para log, o cualquier otra tecla para finalizar:
      " str1
36
37 if [[ "$str1" == *"l"* ]]; then
38 gio open ./logs/stdout.lpe.log
39 fi
40
41 if [[ "$str1" == *"c"* ]]; then
42 gio open ./GRANJAGUAR_cc.rep
43 fi
44
45 if [[ "$str1" == *"s"* ]]; then
46 gio open ./GRANJAGUAR.sp
47 fi
```

```
48
49 exit
```

15.4. Runsets

Naturalmente, los *runsets* son una parte importante de estos procesos. Le instruyen a cada programa los comandos que deben correr en cada sección. Será obvio que en esta sección se utilizan direcciones relativas. De nuevo, el propósito de esta práctica es fácilmente generalizar el proceso para correr estos procesos desde cualquier dirección, con cualquier computadora (siempre que tenga los programas instalados). Existen varios *runsets* incluyendo *DRC*, verificación de antena y *LVS* que contienen 7 MB de data, lo cual los hacen muy largos para añadir en este reporte. Agregado a esto, son propiedad de *TSMC* y solamente fueron modificados para este proyecto. En cambio: síntesis lógica, síntesis física, y LPE contienen *runsets* desarrollados por los integrantes de este equipo. Estos se añaden como referencia de los procesos utilizados en estos reportes:

Listing 15.6: Síntesis lógica: .script para diseño principal

```
1
2 #---------<IMPORTANDO LIBRERIAS RELEVANTES:>---------#
3 lappend search_path /home/nanoelectronica2021/Desktop/Automatization/
    REFERENCE_FILES/libs/tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM
4 set link_library " * tcb018gbwp7ttc.db tcb018gbwp7twc.db tcb018gbwp7tbc.
    db tpd018nvtc.db"
5 set target_library "tcb018gbwp7ttc.db"
6
7 #----------<IMPORTANDO ARCHIVO VERILOG:>----------#
8 read_file -format verilog {../GRANJAGUAR.v}
9
10 #----------<SET DE RESTRICCIONES>----------#
11 reset_design
12 set_max_area 0
13 link
14 uniquify
15 create_clock clk -period 40 -waveform {0 20}
16
17 #----------<REVISION DE DESIGN:>----------#
18 check_design
19
20 reset_design
21 set_max_area 0
22
23 #----------<COMPILACION:>----------#
24 compile -map_effort high -area_effort high
25
26 #----------<REPORTES:>----------#
27 report_area
28 report_design
29 report_power
30
```

```
31 #----------------<OUTPUTS>--------------------#
32 write -format ddc -h -o ../files/GRANJAGUAR.ddc
33 write -hierarchy -format verilog -output ../files/GRANJAGUAR.v
34 write_sdc ../files/GRANJAGUAR.sdc
35
36 ##-----<SINTESIS-LOGICA-PRINCIPAL-FINALIZADA>----INICIANDO-IO-----##
37 exit
```

Listing 15.7: Síntesis lógica: .script para diseño con entradas y salidas

```
1  #------(IO)---IMPORTANDO-LIBRERIAS-RELEVANTES:--(IO)----------#
2  lappend search_path /home/nanoelectronica2021/Desktop/Automation/
       REFERENCE_FILES/libs/tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM
3  set link_library " * tcb018gbwp7ttc.db tcb018gbwp7twc.db tcb018gbwp7tbc.
       db tpd018nvtc.db"
4  set target_library "tcb018gbwp7ttc.db"
5
6
7  #--------(IO)---IMPORTANDO-ARCHIVO-VERILOG:--(IO)---------#
8  read_file -format verilog {../GRANJAGUAR_io.v}
9
10 #-------(IO)---SET-DE-RESTRICCIONES:--(IO)----------------#
11 reset_design
12 set_max_area 0
13 link
14 uniquify
15 create_clock clk -period 40 -waveform {0 20}
16
17 #-------------(IO)---REVISION-DE-DESIGN:--(IO)----------#
18 check_design
19
20 #-------------(IO)---COMPILACION:--(IO)-------------#
21 compile -map_effort high -area_effort high
22
23 #---------(IO)---REPORTES:--(IO)-----------------#
24 report_area
25 report_design
26 report_power
27
28
29 #----------------(IO)---OUTPUTS:--(IO)--------------------#
30 write -format ddc -h -o ../GRANJAGUARio.ddc
31 write -hierarchy -format verilog -output ../GRANJAGUARio.v
32 write_sdc ../GRANJAGUARio.sdc
33
34 ##-------------(IO)---FIN:--(IO)----------------##
35 exit
```

Listing 15.8: Síntesis física: .tcl para diseño

```
1 #--------<CREACION-LIBRERIA>-------------------------
2 create_lib GRANJAGUAR_SYN.ndm -technology /usr/synopsys/TSMC/180Complete
      /TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/
```

```
    digital/Back_End/milkyway/tcb018gbwp7t_270a/techfiles/tsmc018_6lm.tf
     \
 -ref_libs ../../../REFERENCE_FILES/LibreriasNDM/TSMCWorkspace.ndm

#-------<IMPORTACION VERILOG SINTETIZADO>-------------
read_verilog ../../LogicalSynthesis/GRANJAGUARio.v

read_sdc -echo ../../LogicalSynthesis/GRANJAGUARio.sdc

#-------------<IMPORTACION TLU+ Y MAP>-----------------
read_parasitic_tech -tlup /usr/synopsys/TSMC/180Complete/TSMC/180/CMOS/G
    /stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/
    milkyway/tcb018gbwp7t_270a/techfiles/tluplus/t018lo_1p6m_typical.
    tluplus -layermap /usr/synopsys/TSMC/180Complete/TSMC/180/CMOS/G/
    stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/
    milkyway/tcb018gbwp7t_270a/techfiles/tluplus/star.map_6M

#--------------------<LIMPIEZA PG>--------------------
remove_pg_strategies -all

#-------------<REGLAS ANTENA>--------------------
#source -echo -verbose "/usr/synopsys/TSMC/180Complete/TSMC/180/CMOS/G/
    stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/
    milkyway/tcb018gbwp7t_270a/clf/MODIFIED_antennaRule_018_6lm.tcl"

#-----------------<CREACION CORNERS>----------------
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

#-------<VDD Y VSS PADS>-------------
create_cell {PVDD} PVDD1CDG
create_cell {PVSS} PVSS1CDG

#--------<VDD Y VSS NETS>-------------
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 INICIO>-----------------
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {285 285} -core_offset {125}

#-----------------<ANILLO IO>-------------------
create_io_ring -name anillo_IO -corner_height 115

#-------------------<IO PADS>-----------------
#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.left] PVDD*
add_to_io_guide [get_io_guides anillo_IO.right] PVSS*
place_io
```

```
46
47 #-------------<CREACION_ANILLO_PG>---------------
48 create_pg_ring_pattern ring_pattern -horizontal_layer METAL2    -
       horizontal_width {2} -horizontal_spacing {2}    -vertical_layer
       METAL3 -vertical_width {2} -vertical_spacing {2}
49 set_pg_strategy core_ring    -pattern {{name: ring_pattern}    {nets: {
       VDD VSS}} {offset: {1 1}}} -core
50 compile_pg -strategies core_ring
51
52 #-------------<IO_CONEXIONES>---------------
53 create_pg_macro_conn_pattern hm_pattern -pin_conn_type scattered_pin -
       layers {METAL2 METAL3} -nets {VDD VSS} -pin_layers {METAL2}
54 set_app_options -name plan.pgroute.treat_pad_as_macro -value true
55 set_pg_strategy macro_conn -macros [get_cells {PVDD* PVSS*}] -pattern {{
       name: hm_pattern} {nets: {VDD VSS}}}
56 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}}}
57 compile_pg -strategies macro_conn -via_rule macro_conn_via_rule -tag
       test
58
59 #-------------------<VDD_Y_VSS_MESH>-------------------
60 connect_pg_net -automatic
61 create_pg_mesh_pattern mesh_pattern -layers { {{vertical_layer: METAL3}
       {width: 4.2} {pitch: 42} {spacing: interleaving}} }
62 set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {409.480
       414.240}} -pattern {{pattern: mesh_pattern}{nets: {VDD VSS}}} -
       blockage {macros: all}
63 create_pg_std_cell_conn_pattern std_cell_pattern
64 set_pg_strategy std_cell_strategy -core -pattern {{pattern:
       std_cell_pattern}{nets: {VDD VSS}}}
65 compile_pg
66
67 #-------------<MERGE_DE_MESH_Y_ANILLO_PG>---------------
68 merge_pg_mesh -nets {VDD VSS} -types {ring stripe} -layers {METAL2
       METAL3}
69
70 #---------------<PLACEMENT_CREACION>---------------
71 set_app_options -name place.coarse.fix_hard_macros -value false
72 set_app_options -name plan.place.auto_create_blockages -value auto
73 create_placement -floorplan -timing_driven -congestion -effort high -
       congestion_effort high
74 legalize_placement
75
76 #----------<SINTETIZAR_RELOJES>-----------------------
77 check_clock_trees -clocks clk
78 check_design -checks pre_clock_tree_stage
79 synthesize_clock_trees -clocks clk -postroute -routed_clock_stage
       detail_with_signal_routes
80 clock_opt -list_only
81 check_design -checks cts_qor
82
83 #-------------<RUTEADO>-------------------------
84 check_routability -check_pg_blocked_ports true
```

```
85  check_design -checks pre_route_stage
86  route_auto
87
88  #-------------<CORE_FILLER_Y_ANILLO_IO>-----------------------
89  create_io_filler_cells -io_guides [get_io_guides {anillo_IO.top
        anillo_IO.right anillo_IO.left anillo_IO.bottom}] \
90  -reference_cells {PFILLER1 PFILLER5 PFILLER05 PFILLER0005 PFILLER10
        PFILLER20}
91  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}]
92  connect_pg_net -automatic
93  remove_stdcell_fillers_with_violation
94  check_legality
95
96  #-----------<DRC_RUNSET_FILE>-----------------------
97  set_app_options -list {signoff.check_design.run_dir {../DRC/}}
98  set_app_options -list {signoff.check_drc.run_dir {../DRC/}}
99  set_app_options -list {signoff.check_design.runset {../files/ICVLM18_DRC
        .215a_pre041518}}
100 set_app_options -list {signoff.check_drc.runset {../files/ICVLM18_DRC
        .215a_pre041518}}
101
102
103 #-----------<DRC_Y_GUARDAR>-----------------
104 save_block GRANJAGUAR_SYN.ndm:GRANJAGUAR_IO
105 signoff_fix_drc
106
107 save_block GRANJAGUAR_SYN.ndm:GRANJAGUAR_IO
108 signoff_check_drc
109
110 check_lvs
111 save_block GRANJAGUAR_SYN.ndm:GRANJAGUAR_IO
112
113 #----------<VERILOG_CREACION>-----------------
114 write_verilog -include all GRANJAGUAR_SYN.v
115
116 #-------------<.GDS_CREACION>-----------------
117 write_gds -library GRANJAGUAR_SYN.ndm -design GRANJAGUAR_IO -view design
        -hierarchy all -lib_cell_view frame ../GRANJAGUAR_IO.gds
118
119 exit
```

Listing 15.9: LPE: .cmd con comandos utilizados para la extracción

```
1  **-------------------------------------------------------------------**
2  ** Name:         LPE_script.cmd
3  ** Date:         02 nov 2021
4  ** Author:       J. Gonzalez Herrera
5  ** Description: The following is a command file utilized to run StarRC
       to extract
```

```
**                GRANJAGUAR characteristics to be simulated and tested.
**
**                Based on TSMC template made for customer reference.
**------------------------------------INPUT------------------------------------**


BLOCK: GRANJAGUAR_IO
TCAD_GRD_FILE: ../../../REFERENCE_FILES/LPE/
    cm018g_1p6m_4x1u_mim5_40k_cbest.nxtgrd
MAPPING_FILE: ../../LVS/logs/STARRCXT.mapping
ICV_RUNSET_REPORT_FILE: ../../LVS/logs/STARRCXT.runset_rep
*** Subcket Pin's order for Simulation
SPICE_SUBCKT_FILE: ../../LVS/files/GRANJAGUAR.icv


**-----------OUTPUT-----------**

NETLIST_FORMAT: NETNAME
NETLIST_PASSIVE_PARAMS: YES
NETLIST_FILE: ../GRANJAGUAR.sp

COUPLING_REPORT_FILE: ../GRANJAGUAR_cc.rep
**----------------OPTIONS----------------**

CASE_SENSITIVE: NO
HIERARCHICAL_SEPARATOR: /

* MILKYWAY_EXTRACT_VIEW: YES
*** Metal fill extraction
* METAL_FILL_POLYGON_HANDLING: FLOATING
* METAL_FILL_GDS_FILE:
* GDS_LAYER_MAP_FILE:

*** RC Extraction options
* NETLIST_DEVICE_LOCATION_ORIENTATION : COMMENT

COUPLE_TO_GROUND: NO
*** Coupling Caps options
* COUPLING_ABS_THRESHOLD: 3e-15
* COUPLING_REL_THRESHOLD: 0.03
* COUPLING_REPORT_NUMBER: 1000
EXTRACTION: RC
REDUCTION: YES
DENSITY_BASED_THICKNESS: YES
*** For 90nm and below process
*EXTRACT_VIA_CAPS: YES
*** For 0.13um and above process
 EXTRACT_VIA_CAPS: NO
*** DataBase Processing
REMOVE_FLOATING_NETS: YES
REMOVE_DANGLING_NETS: YES
*REMOVE_FLOATING_PORTS: YES
POWER_NETS: VDD VSS
SKIP_CELLS: !*
```

```
59 TRANSLATE_RETAIN_BULK_LAYERS: YES
60
61
62 * NETLIST_COMPRESS_COMMAND: gzip
63
64 * XREF options
65 XREF: YES
66 *XREF_USE_LAYOUT_DEVICE_NAME: YES
67 *CELL_TYPE: LAYOUT
68 *NET_TYPE: LAYOUT
69 *** For shrink flow
70 * MAGNIFICATION_FACTOR : 0.9
71 * MAGNIFY_DEVICE_PARAMS : NO
72 SKIP_PCELLS : cfmom* cfmom_mx* cfmom_rf* crtmom* crtmom_rf* ind_std*
      ind_std_40k* ind_sym* ind_sym_40k* ind_sym_ct* ind_sym_ct_40k* jvar*
       lcesd1_rf* lcesd2_rf* lowcpad_rf* mimcap_rf* mimcap_rf_2p0* mos_var
      * mos_var33* moscap_rf* moscap_rf33* moscap_rf33_nw* moscap_rf_nw*
      ndio_hia_rf* ndio_sbd_mac* pdio_hia_rf* rfnmos2v* rfnmos2v_6t*
      rfnmos3v* rfnmos3v_6t* rfpmos2v* rfpmos2v_5t* rfpmos2v_nw*
      rfpmos2v_nw_5t* rfpmos3v* rfpmos3v_5t* rfpmos3v_nw* rfpmos3v_nw_5t*
      rphpoly_rf* rphripoly_rf* rplpoly_rf* sbd_rf* sbd_rf_nw*
      spiral_std_m2u_a_33k* spiral_std_m2u_x_33k* spiral_std_mu_a_33k*
      spiral_std_mu_x_20k* spiral_std_mu_x_33k* spiral_std_mu_x_40k*
      spiral_sym_ct_m2u_u_a_33k* spiral_sym_ct_m2u_u_x_33k*
      spiral_sym_ct_mu_x_20k* spiral_sym_ct_mu_x_33k*
      spiral_sym_ct_mu_x_40k* spiral_sym_ct_mu_x_a_33k*
      spiral_sym_m2u_u_33k* spiral_sym_mu_x_20k* spiral_sym_mu_x_33k*
      spiral_sym_mu_x_40k*
```