UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

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

Trabajo de graduación presentado por Julio Enrique Shin Jo para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

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

Trabajo de graduación presentado por Julio Enrique Shin Jo para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2022

Vo.Bo.:

(f) _____

Ing. Luis Nájera

Tribunal Examinador:

(f) _____

Ing. Luis Nájera

(f) _____

MSc. Carlos Esquit

(f) _____

Ing. Jonathan de los Santos

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

Prefacio

La realización de este trabajo no puedo haber sido posible sin el apoyo que tuve de muchas personas. En primer lugar, quisiera agradecer a mi familia por el apoyo incondicional que me han dado a lo largo de mi vida y por la oportunidad de darme una educación de calidad. Agradezco al MSc Carlos Esquit por sentar las bases de la nanoelectrónica en la Universidad de Valle de Guatemala. Agradezco también a mi asesor, el Ing. Luis Nájera por guiarme y aconsejarme a lo largo de este proyecto. Le doy las gracias al Ing. Jonathan de los Santos por su apoyo con el entorno de *Synopsys*. Finalmente, quiero agradecer al grupo de trabajo del proyecto del *chip* por haber logrado nuestros objetivos y haber obtenido resultados positivos.

Índice

Lista de figuras

Lista de cuadros

Resumen

El presente trabajo trata principalmente acerca del proceso de síntesis física y la validación de reglas eléctricas o *electric rule check* (**ERC**) que se deben de realizar para el diseño del *layout* de un circuito integrado (**IC**). Se busca poder mejorar el proceso iniciado por las iteraciones pasadas de estas etapas del desarrollo de un **IC**. Esto abarca desde el posicionamiento e interconexión de los componentes hasta la corrección de errores que surjan durante las verificaciones a realizar. La refinación de estas etapas permitirá un flujo de diseño más fluido para futuros proyectos en donde sea necesario el diseño de un circuito personalizado.

Para poder realizar las pruebas necesarias para este trabajo, se utilizarán herramientas proveídas por la empresa *Synopsys*. La herramienta de *IC Compiler II* permitió la síntesis de un *layout* en silicio con los componentes necesarios para el funcionamiento del circuito, mientras que *IC Validator* se utilizó para poder verificar que los resultados que se obtuvieron fueran fabricables. Se realizaron distintas pruebas con circuitos como un NOT, un *full adder*, un contador de 4 bits y el circuito diseñado por el grupo de trabajo de la Universidad del Valle de Guatemala (**UVG**).

Los resultados de este trabajo fueron satisfactorios. Se lograron cumplir todos los objetivos planteados. Se obtuvieron los *layouts* de los diferentes circuitos traduciendo los comandos de trabajos anteriores a la *IC Compiler II*. A pesar de realizar circuitos cada vez más complejos, la síntesis física fue realizada sin problemas. Las pruebas de **ERC** fueron exitosas, obteniendo los mismos resultados que se dieron en la iteración pasada de esa parte del proceso. Finalmente, fue posible crear un *script*: para agilizar la etapa de la síntesis física y apoyar en la automatización del proceso.

Abstract

The following work is about the process needed for the physical synthesis and electric rule check (**ERC**) that the layout of any integrated circuit (**IC**) has to go through. The main objective in this iteration of these steps in the development of an **IC** is to improve the process initiated in past works. This means the scope for these stages of the design flow will be the placement and routing of components inside the **IC** and the various fixes that have to be done after checking the design. The betterment of these phases in the design flow of an IC will make the whole process easier and more efficient future works and projects that need a personalized circuit.

All the tools and programs used in this work were provided by the company *Synopsys*. IC Compiler II is the tool of choice for synthesizing a layout in silicon with all the components needed for the circuit, while IC Validator was used for the various checks a layout has to successfully complete to be able to be fabricated. The improved process was tested using circuits like a NOT, a full adder, a 4 bit counter and the chip designed by the Universidad del Valle de Guatemala (**UVG**) students involved in the project.

The results achieved during this work were successful. All the proposed objectives were fulfilled. It was possible to create the layouts for the circuits after migrating the commands used in previous works to IC Compiler II. The physical synthesis of the circuits were made without problems even though the circuits were more complex each time. The *ERC* process was also successful, obtaining the same results as the last iteration of this check. It was also possible to create a script to speed up this process and assist in the automation of the whole flow.

CAPÍTULO 1

Introducción

En la actualidad, todos los productos tecnológicos tienen dentro de ellos más de algún *chip*: que realiza una función designada. Esta revolución tecnológica inició cuando se creó el primer transistor en el año 1947. Esto llevó a la miniaturización de productos electrónicos ya que se fueron descartando el uso de tubos de vacío. En 1958, Jack Killby creó el primer circuito integrado para un *flip-flop*. Esto dio inicio a la creación de **ICs** que realizan distintas funciones.

Con el paso de los años, los circuitos han ido evolucionando y complicándose cada vez más. Existen diseños que contienen miles de millones de transistores dentro de ellos. Los transistores se vuelven cada vez más pequeños y más eficientes, lo que da la posibilidad de crear circuitos mucho más complejos que los que se hacían apenas unos años atrás. Se puede comparar el primer **IC**, un simple *flip-flop* con procesadores como un i7 de Intel para poder ver cambio drástico que se ha tenido la industria tecnológica.

Es por esto que es necesario establecer un flujo para poder crear circuitos a escala nanométrica. Tener claro los pasos a realizar, permite agilizar el proceso de creación de un **IC**. Sin embargo, es un proceso que requiere bastante conocimiento y herramientas específicas para poderse implementar. Dado que esta no es un ámbito muy trabajado en Centroamérica, la Universidad del Valle de Guatemala vio esto como una oportunidad para poder seguir impulsando la ciencia y la tecnología del país y de la región.

CAPÍTULO 2

Antecedentes

Uno de los aspectos que diferencia a los estudiantes de ingeniería electrónica de la Universidad de Valle de Guatemala es el conocimiento acerca del campo de la nanoeléctronica. Este cambio surgió cuando el Ing. Carlos Esquit comenzó a ejercer como director del departamento de Ingeniería Electrónica, Mecatrónica y Biomédica. Se realizaron distintas modificaciones al mapa curricular para que los estudiantes pudieran tener una primera aproximación a la electrónica de escala nanométrica.

El primer cambio que se dio fue en el año 2013, cuando se impartió por primera vez el curso de *Introducción al diseño de sistemas VLSI*. Un año después, se obtuvo un acuerdo con la empresa *Synopsys* para poder utilizar sus herramientas como parte del *University program*. Como resultado, se realizó el primer trabajo de graduación relacionado a **VLSI**. El trabajo puede ser consultado en [1].

Como parte de una reforma curricular, se añadieron los cursos de *Nanoelectrónica 1* y *2* en el año 2015. Este cambio permitió que los estudiantes se familiarizaran más con la teoría y las herramientas relacionadas a **VLSI**. Esto impulsó la iniciativa de crear el primer nanochip de Centroamérica. Al llegar a un acuerdo con *Interuniversity Microelectronics Centre* (**IMEC**) para la manufactura del chip, se iniciaron los primeros trabajos relacionados a la creación del primer circuito **IC** del país. Este acuerdo permitió el acceso a las librerías de *Taiwan Semiconductor Manufacturing Company* (**TSMC**) como un recurso valioso para el desarrollo del proyecto. En el trabajo [2] se especifica un flujo de diseño, mientras que el trabajo [3] lleva a cabo una etapa específica en la elaboración del chip. De especial interés, son los trabajos mostrados en [4], [5] y [6] ya que estos fueron iteraciones pasadas de las etapas de síntesis y verificaciones físicas.

CAPÍTULO 3

Justificación

El avance de la tecnología nanométrica ha permitido crear chips cada vez más sofisticados. Estos se encuentran presentes en cualquier dispositivo electrónico que exista, desde una simple calculadora hasta equipos utilizados en plantas industriales. La cantidad de transistores que pueden estar presentes en un *chip* ha crecido rápidamente y se han llegado a tener millones de transistores dentro de un solo empaquetado. Estos diseños cada vez más complejos permiten realizar funciones mucho más complicadas y específicas.

Para poder crear un *chip* de esta magnitud, es necesario tener definido un flujo de diseño funcional. El desarrollo de las distintas etapas de diseño dentro de la universidad significa un gran avance para el país y los estudiantes del departamento de Ingeniería Electrónica, Mecatrónica y Biomédica. El flujo de diseño permitirá la creación de circuitos integrados que cumplan funciones específicas para trabajos y proyectos, además de ser una herramienta educacional que permita conocer de mejor manera el proceso necesario para la producción de un *chip*.

El desarrollo de la etapa de síntesis física y sus diferentes verificaciones permitirá generar un producto que sea fabricable. Estas etapas son unas de las más complicadas ya que se debe de cumplir con una gran cantidad de reglas para que el resultado generado sea adecuado. La síntesis física es de suma importancia ya que es el cuello de botella del flujo de diseño. Además de eso, si no existiera una etapa de síntesis física, un *chip* nunca podría ser una realidad. Solamente se quedaría como una simulación de un circuito que cumple ciertos requisitos. Es por esto que se debe de agilizar este proceso y definir una forma más eficiente de realizarlo.

La etapa de **ERC** junto con las otras verificaciones físicas de *design rule check* (**DRC**) y *antenna rule check* son las que permiten conocer si un diseño en silicio puede ser enviado a un *foundry* para que sea fabricado. Si estas reglas no se cumplen, el diseño propuesto simplemente es rechazado. Específicamente, el **ERC** verifica la conectividad de distintos

elementos físicos para garantizar que se fabriquen de manera correcta. El cumplimiento de este conjunto de validaciones es clave para que el *layout:* propuesto pueda ser creado.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Optimizar la colocación e interconexión de componentes dentro de un circuito integrado como parte de la síntesis física de un circuito integrado así como ejecutar la verificación **ERC** manteniendo la comunicación con los demás equipos de trabajo y miembros del grupo para garantizar la corrección oportuna de los diferentes errores que generan las verificaciones de rigor necesarias para el correcto diseño del circuito integrado.

4.2. Objetivos específicos

- Migrar los comandos utilizados en *IC Compiler* a *IC Compiler II*.
- Generar un archivo `.gds` (*layout*) a partir de un *netlist* proveído por la síntesis lógica.
- Correr la verificaciones física de *electric rule check* utilizando la herramienta de *IC Validator*.
- Interpretar los errores que surjan de la verificación de reglas eléctricas del *layout* generado.
- Corregir errores del *layout* a partir de los resultados de la verificación **ERC**.
- Proveer los archivos e información necesaria para la automatización de la síntesis física y la verificación **ERC**.
- Mantener una comunicación efectiva con los otros grupos de trabajos y miembros del mismo grupo.

CAPÍTULO 5

Alcance

El alcance de este trabajo es poder documentar y detallar el proceso necesario para la síntesis física y verificación de reglas eléctricas, además de mejorar el proceso ya establecido en trabajos anteriores. Para esto, se utilizarán las herramientas *IC Compiler II* y *IC Validator* de la empresa *Synopsys* para la creación de *layouts* y la verificación **ERC**. Para este proceso, se tiene a nuestra disposición las librerías de **TSMC** proveídas por **IMEC**. Con esto es posible ejecutar estas fases con cualquier circuito que haya pasado por la síntesis lógica.

Dado que en iteraciones anteriores se ha ejecutado la síntesis física en *IC Compiler*, se busca poder recopilar los comandos utilizados para generar un *layout* en *IC Compiler II* y explicar qué es lo que hace cada uno. Además de eso, se espera poder dejar un *script* genérico que pueda ser utilizado exitosamente para cualquier circuito que requiera una síntesis física.

Con la verificación de reglas eléctricas se busca poder validar los resultados de la iteración pasada. Debido a que los resultados del **ERC** del trabajo [5] fueron exitosos, se espera poder replicar los mismos resultados con el *layout* resultante de la síntesis física realizada en este trabajo.

CAPÍTULO 6

Marco teórico

6.1. *Very Large Scale Integration*

En el año 1963, Frank Wanlass describió la primera compuerta utilizando transistores pMOSFET: y nMOSFET. Esta manera de crear compuertas se le da nombre de *Complementary Metal Oxide Semiconductor* (**CMOS**). Estos circuitos consumen nanowatts de potencia, por lo que es posible crear **IC** con muchas más compuertas. Dependiendo de la cantidad de compuertas que se utilizan en un **IC**, se pueden clasificar en distintas categorías de implementaciones. Las *Small-Scale Integration* (**SSI**) utilizan alrededor de 10 compuertas, las *Medium-Scale Integration* (**MSI**) tienen como máximo mil compuertas y las *Large-Scale Integration* (**LSI**) contienen hasta diez mil compuertas. Dado al crecimiento en la cantidad de transistores en un solo **IC**, se le llamó *Very Large-Scale Integration* (**VLSI**) a la mayoría de circuitos que se crearon a partir de los años 1980 [7].

6.2. Flujo de diseño

Para el diseño y elaboración de un *chip*, es necesario seguir ciertos pasos para poder agilizar el proceso. Se separan los distintos pasos como parte de un *front end* y *back end*. El *front end* se centra en el funcionamiento lógico del circuito integrado a realizar, mientras que el *back end* trata la estructura física del *chip* [7].

Primero se realizan los procesos de *front end*. Para proceder, se necesitan las especificaciones del *chip*, es decir, el funcionamiento que este va a presentar. Los diseñadores transforman estos requerimientos a una descripción de su funcionamiento en un lenguaje descriptor de *hardware* (**HDL**:). Este diseño es sometido a pruebas para verificar el correcto funcionamiento del comportamiento descrito. Luego de tener el funcionamiento correcto, se

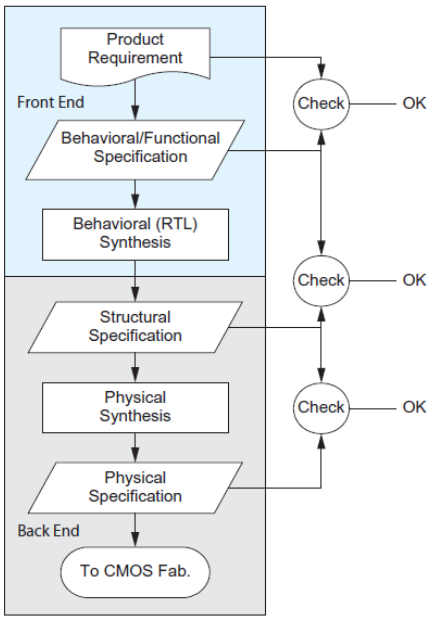

Figura 1: Flujo de diseño generalizado mostrado en [7]

traslada el diseño a un circuito a nivel de compuertas y registros para optimizar velocidad, área, potencia, entre otros aspectos del IC. Para confirmar que este nuevo diseño tiene la misma funcionalidad, se corren nuevamente las pruebas para verificar que realizan las mismas instrucciones. Si se tiene el mismo comportamiento, se prosigue a realizar pruebas de *timing* y de potencia [7].

Luego de finalizar el *front end*, se continúa con el *back end*. Teniendo el diseño con compuertas, se pueden utilizar celda:s ya existentes que contienen el circuito a nivel transistor para poder implementar la función deseada. Estas se utilizan como parte del primer paso que consiste en la colocación de las celdas dentro del *die:*. Al tener las celdas en el lugar deseado, se prosigue con el ruteo de las compuertas para obtener el *layout*. En este paso se interconectan las distintas compuertas según la descripción que se haya recibido. Se sigue con la extracción de parásitos para obtener capacitancias y resistencias que se encuentren dentro del *chip*. Este permite tener una modelo más realista del circuito que se está diseñando. Finalmente, se realizan nuevamente los análisis de *timing* y potencia, adicional al de ruido, para comprobar que aún se estén cumpliendo con los requisitos de diseño [7].

6.2.1. Síntesis física

Esta parte del proceso de diseño de un **IC** consiste en convertir un diseño lógico en uno físico. Esta etapa toma el resultado de las etapas de *front-end*, conocido como un *netlist:*, para obtener un *layout* en silicio en el formato **GDS:**. El *netlist* es un archivo que contiene los componentes necesarios y las conexiones que hay entre ellos. La síntesis física produce un diseño que debería de poder ser enviado para manufactura. Debido a la escala y la cantidad de componentes, gran parte de este proceso es automatizado [7]. Existen tres etapas dentro de la generación de un layout para un **IC**:

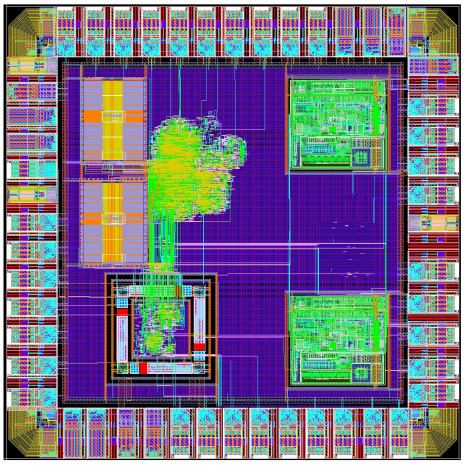

Figura 2: Ejemplo del *layout* de un **IC**

- ***Placement***: Esta parte consiste en la colocación de las celdas estándar dentro del *die*. Este tipo de celdas tienen una altura fija y un ancho variable por lo que es posible colocarlas en varias filas al momento de crear el layout. Dado que los diseños pueden llegar a tener miles de celdas, es necesario utilizar algún algoritmo para asistir en la forma de distribuir los componentes. Se intenta optimizar el largo de las rutas para poder seguir cumpliendo con los requisitos de timing.
- ***Floorplanning***: Dentro del proceso de *placement*, algunas veces puede ser necesario realizar *floorplanning*. Esta etapa se refiere a la designación de áreas dentro del *layout* para la colocación de módulos especiales.
- ***Routing***: Luego de colocar las celdas, se prosigue con el ruteo de las conexiones de los componentes. Este se divide en ruteo global y ruteo detallado. El ruteo global se encarga de generar distintos canales contiguos por los cuales pueden pasar lo metales para generar las interconexiones. Luego de eso, el ruteo detallado genera las conexiones específicas de las celdas. El ruteo automatizado asigna costos a los metales que se utilizan para optimizar las rutas generadas.

6.2.2. *Design Rule Check*

Al tener un layout para el circuito a implementar, es necesario que este pase ciertas condiciones para que pueda ser fabricado. Este proceso se llama *design rule check* (**DRC**). Este es un proceso en el que el layout generado se compara contra un conjunto de restricciones (*runset*) que se deben de cumplir para que un diseño pueda ser manufacturado [8]. Estas reglas son específicas dependiendo del fabricante con el que se esté trabajando. Es decir, el **DRC** puede ser exitoso para un diseño siguiendo las reglas de **TSMC** pero puede fallar para el conjunto de restricciones de Samsung.

Algunas de las reglas que se deben de cumplir para que el **DRC** sea exitoso son [9]:

- **Ancho mínimo:** Se establece un tamaño mínimo para los elementos de una capa del diseño para evitar que estos se generen de manera incorrecta durante la fabricación.

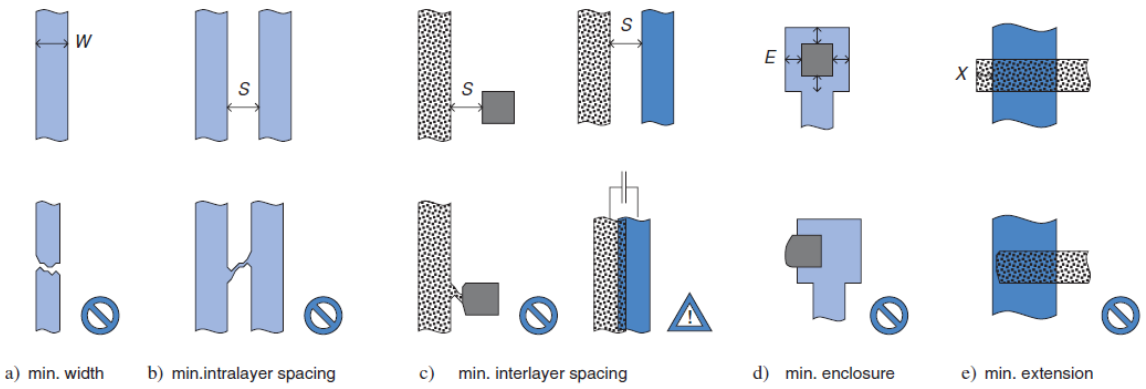

Figura 3: Reglas de **DRC** y sus efectos dentro del **IC**

Dependiendo del tamaño, puede que las conexiones se interrumpan por efectos indeseados durante el proceso de grabado. Este ancho mínimo también puede depender de la densidad de elementos en la misma capa en su cercanía.

- **Espaciado mínimo intracapa:** Esta regla se refiere a la separación que debe de existir entre elementos de una misma capa. En algunos casos el tamaño depende de la existencia de una conexión eléctrica entre los *tracks*. Esta regla evita cortos indeseados entre elementos cercanos dentro de una misma capa.

- **Espaciado mínimo intercapa:** Similar a la regla anterior, esta establece un separación mínima entre elementos de capas distintas. Esto se realiza para poder evitar efectos indeseados dentro del **IC** como capacitancias parásitas o corrientes de fuga.

- **Cercado mínimo:** Esta restricción es válida para estructuras que se encuentran en varias capas dentro del **IC**. Un ejemplo de estas son las vías o contactos. Se define un tamaño mínimo que debe de encerrar al elemento para poder asegurar un buen contacto entre las capas que se están comunicando sin importar pequeñas imperfecciones durante el proceso de manufactura. Otro elemento en donde se utiliza esta regla es en los *wells* embebidos dentro de las zonas de difusión.

- **Extensión mínima:** Esta regla se aplica a elementos que se traslapan en distintas capas. En estos casos es necesario que los *tracks* se superpongan una cierta distancia para evitar problemas durante el grabado en donde se afecta a las conexiones entre las capas y su conductividad.

- **Ancho máximo:** De manera similar al ancho mínimo, también existe un tamaño máximo. Estas reglas generalmente se aplican a contactos y vías. El tamaño máximo se establece para poder generar elementos de mejor calidad.

- **Reglas de densidad:** Al momento de realizar el *layout* es posible que no se llene el área designada. Las reglas de densidad verifican que, dentro de la región, se esté ocupando cierta porción por elementos del **IC**. Se tienen cotas superiores e inferiores, en donde no puede ni estar muy lleno ni muy vacío. Si un *layout* no cumple con la cota inferior de densidad, es posible solucionarlo con elementos *dummy* que no tienen una función en el diseño del **IC**.

6.2.3. *Electrical Rule Check*

El **ERC** es una manera de comprobar la robustez de algún diseño en forma de esquemático o de *layout* [10]. Este utiliza reglas que no son revisadas en el proceso de **DRC** o en el *layout versus schematic* (**LVS**) [11]. Este puede revisar el diseño para encontrar dispositivos flotantes, pines flotantes, *wells* flotantes, entre otras reglas más. Debido a que el conjunto de reglas comprobado por el **ERC** depende del proyecto que se está implementando, se utilizarán las mismas reglas que en la iteración anterior de esta etapa del diseño del *chip* presentadas en [5]. Estas reglas fueron:

- ***Soft connect check*:** Revisa el diseño para encontrar si existe alguna conexión entre algún material de alta resistividad (*n-wells* o difusiones) que pueda causar efectos indeseados en el funcionamiento del *chip*.
- ***Path check*:** Revisa si existe algún *net* que no esté conectado a tierra, alimentación o a algún *net* etiquetado.
- ***PTAP/NTAP connectivity check*:** Revisa las conexiones de los *taps* de los MOSFETs.
- ***MOSFET power and ground check*:** Verifica que el *drain* o el *source* de los transistores esté conectado a alimentación o a tierra.
- ***Gate directly connected to power or ground*:** Esta regla comprueba que no existan *gates* de transistores de pMOSFET conectados a alimentación o *gates* de nMOSFET conectados a tierra ya que esto puede dañar a los transistores.
- ***Floating gate*:** Revisa si existe contacto con el polisilicio de los *gates* de los transistores.
- ***Floating well*:** Verifica que el sustrato tipo p de los transistores esté conectado a tierra o que el *n-well* esté conectado a alimentación.

6.2.4. *Antenna Rule Check*

Durante el proceso de manufactura de los **IC**, es posible que los metales que se encuentran conectados a los *gates* de los transistores acumulen suficiente carga como para generar una descarga hacia la capa de óxido del *gate*. Este fenómeno se conoce como el *antenna effect*. La descarga daña al transistor por lo que incrementa la corriente de fuga, cambia el voltaje de umbral y disminuye la vida útil del transistor [7].

El *antenna rule check* se encarga de encontrar conexiones entre metales y *gates*. Las reglas que utiliza para verificar el diseño se basan en una proporción entre el área del metal que se está conectando y el área del *gate* del transistor. Mientras más área se tenga en el *gate*, se permite una mayor cantidad de carga acumulada antes de que ocurra una descarga. La razón entre las áreas depende del grosor de la capa de óxido [7].

Existen dos maneras de resolver estos problemas. La primera es interrumpiendo el *track*: con una conexión a otra capa de metal. Esto disminuye el área del metal conectado, por

lo que la carga acumulada ya no causa daños en los transistores. La segunda manera de corregirlo, es utilizando diodos. Esto se colocan conectando el metal problemático a tierra para poder eliminar la carga acumulada. Estas soluciones se pueden ver en la Fig. 4.

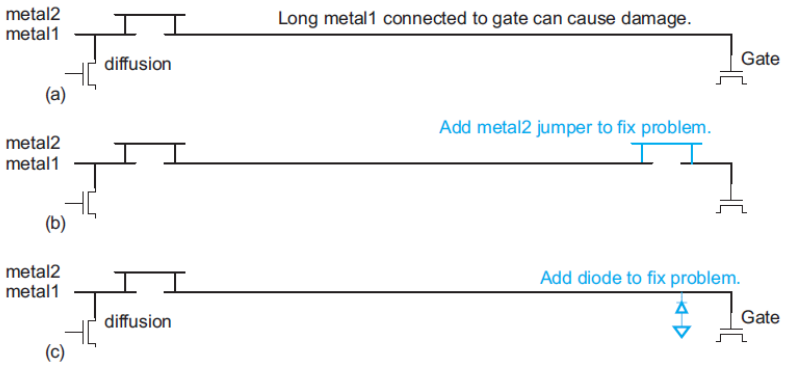

Figura 4: Soluciones para problemas con *antenna rule check*

6.3. Software *Synopsys*

Para poder realizar las diferentes etapas del flujo de diseño, es necesario contar con software especializado. Algunas de las empresas que desarrollan estas herramientas son *Cadence* y *Synopsys*. Los programas a los que se tienen acceso en la **UVG** son aquellos proveídos por *Synopsys*. Estos programas se encargan de etapas específicas en el diseño de un *chip*. Algunas herramientas realizan simulaciones de los circuitos armados, otras realizan validaciones del *layout* creado, etc.

6.3.1. *IC Compiler II*

IC Compiler II (**ICC 2**) una herramienta de *place-and-route*:. Es también conocido como un software de *netlist-to-GDS II* debido a los archivos que recibe como entrada y el formato del archivo generado como salida. Debido a sus algoritmos patentados para el proceso de planificación y optimización, **ICC 2** provee una mejora en la productividad en la síntesis física de los *chip*s a nanoescala [12].

6.3.2. *IC Validator*

IC Validator (**ICV**) es una plataforma de verificación física que permite realizar diferentes comprobaciones al diseño de un **IC**. Esta herramienta permite la verificación de *layouts* generados por **ICC 2** debido a que son parte de la mismo entorno de Synopsys. **ICV** es capaz de realizar verificaciones de **LVS**, **DRC**, **ERC** y antenna rule check [13].

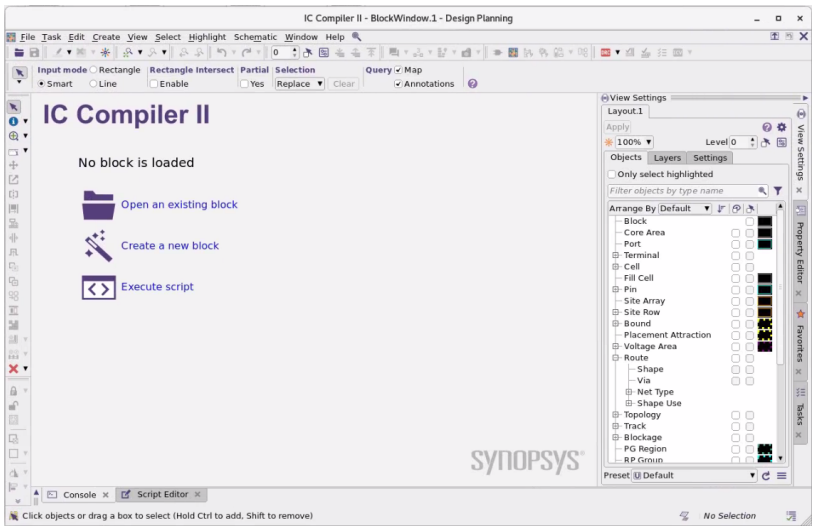

Figura 5: Interfaz gráfica de **ICC 2**

6.4. Iteraciones anteriores

El presente trabajo es una nueva iteración de las etapas de la síntesis lógica y la verificación **ERC** de un *layout*. Es por esto que es necesario poder conocer más acerca de los trabajos que preceden al este para poder plantear mejoras al proceso anterior.

6.4.1. Trabajo de Luis Abadía

El trabajo [6] consiste en la elaboración de la síntesis física para el flujo de diseño de un **IC**. En este se propone una metodología que propone un procedimiento para realizar la colocación y la interconexión de componentes separado en cuatro etapas:

- Preparación del software **ICC**
- Configuración y creación del *floorplan*
- Configuración y creación del *placement*
- Configuración y creación del *routing*

Para iniciar la herramienta **ICC** es necesario correr `icc_shell -gui -shared_license` para iniciar la interfaz gráfica.

1. **Preparación del software ICC**
   La etapa de preparación de software consiste en la ejecución de distintos comandos para preparar el entorno de **ICC**. Para esto es necesario indicar la ubicación de los archivos .db e indicarle las librerías a utilizar. La ruta indicada no necesariamente va a ser la misma ya que dependen de donde se encuentren las librerías dentro del sistema de archivos del sistema operativo. En el caso de las `link_library`, es necesarios

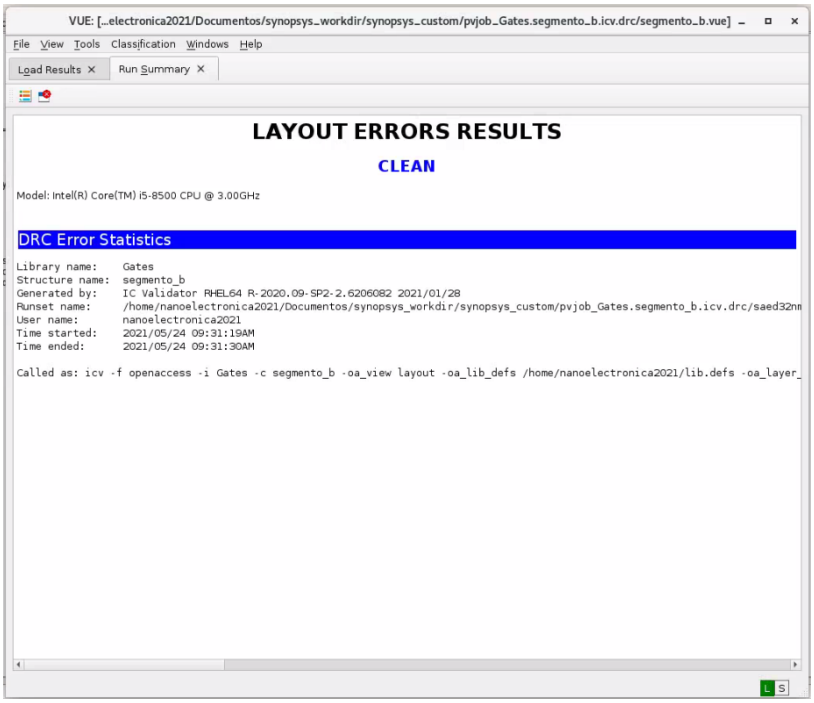

Figura 6: Ventana con los resultados de **ICV**

agregar `ttc.db`, `twc.db` y `tbc.db` para indicar los casos típicos, peor caso y mejor caso respectivamente. La `target_library` indica la librería específica con la que se va a trabajar. Los comandos a ejecutar en esta etapa son los siguientes:

```
lappend search_path /home/administrador/Escritorio/FA_TSMC_DRC/Libs
/tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM
set link_library " * tcb018gbwp7ttc.db tcb018gbwp7twc.db
tcb018gbwp7tbc.db tpd018nvtc.db"
set target_library "tcb018gbwp7ttc.db"
```

Luego de agregar estas librerías, es necesario añadir los archivos `.tluplus` y `.map`. Nuevamente, las rutas no serán las mismas. Esto se hace con el siguiente comando:

```
set_tlu_plus_files -max_tluplus /home/administrador/Escritorio/
FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE/tluplus/t018lo_1p6m_typical.
tluplus -min_tluplus /home/administrador/Escritorio/FA_TSMC_DRC/Libs
/tcb018gbwp7t_290a_FE/tluplus/t018lo_1p6m_typical.tluplus -
tech2itf_map  /home/administrador/Escritorio/FA_TSMC_DRC/Libs/
tcb018gbwp7t_290a_FE/tluplus/star.map_6M
```

Luego de esto, es necesario configurar y crear la librería *MilkyWay*. Las primeras dos rutas son ubicaciones de los archivos necesarios, mientras que la última es la ubicación en donde se crea la librería *MilkyWay*.

```
create_mw_lib  -technology /home/administrador/Escritorio/
FA_TSMC_DRC/Libs/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf  -
mw_reference_library {/home/administrador/Escritorio/FA_TSMC_DRC/
Libs/tcb018gbwp7t_290a_FE/tcb018gbwp7t /home/administrador/
```

```
Escritorio/FA_TSMC_DRC/Libs/iolib/tpd018nv} -bus_naming_style {[%d
]} -open  /home/administrador/Escritorio/LUIS_ABADIA/Milky_Aba
```

Finalmente, se indican los archivos `.v` y `.sdc` generados por la síntesis lógica.

```
read_verilog -dirty_netlist -allow_black_box -verbose {/home/
administrador/Escritorio/FA_TSMC_DRC/Out/FA_syn_p_m.v}
read_sdc -echo -syntax_only -version Latest "/home/administrador/
Escritorio/FA_TSMC_DRC/Out/FA_sdc_p.sdc"
```

Con esto, se termina la etapa de preparación del entorno de **ICC** y se pasa a configuración y creación del *floorplan*.

2. **Configuración y creación del *floorplan***
El primer paso es la creación de dos variables lógicas para VDD y VSS.

```
set_app_var mw_logic1_net "VDD"
set_app_var mw_logic0_net "VSS"
```

Luego de esto, se generan los *power nets* y los *power pins*.

```
derive_pg_connection -power_net VDD -power_pin VDD -ground_net VSS
-ground_pin VSS
derive_pg_connection -power_net VDD -ground_net VSS -tie
```

En este punto, se puede generar un reporte general de las conexiones realizadas. Esto puede ser útil para revisar lo que se ha hecho y se puede correr en cualquier momento de la síntesis física. El comando es `report_cell_physical -connections`.

Luego de las conexiones lógicas, se prosigue a la creación de celdas para los *pads* y *corners:*. Es necesario conocer el nombre de las celdas a utilizar al momento de correr estos comandos. Esto se hace de la siguiente manera:

```
create_cell {c1 c2 c3 c4} PCORNER
create_cell {VDD} PVDD1CDG
create_cell {VSS} PVSS1CDG
```

Luego de crear estas celdas, se les deben de asignar ciertas restricciones. Estas indican el nombre que se le da a la celda al igual que la posición que va a tener dentro del *layout*. El número que lleva el comando hace referencia a una distinta posición. Estas pueden ser 0 (defecto), 1 (izquierdo), 2 (superior), 3 (derecho) y 4 (inferior). También se indica el orden en el que se van a colocar.

```
set_pad_physical_constraints -pad_name "c1" -side 1
set_pad_physical_constraints -pad_name "c2" -side 2
set_pad_physical_constraints -pad_name "c3" -side 3
set_pad_physical_constraints -pad_name "c4" -side 4
set_pad_physical_constraints -pad_name "VDD" -side 3 -order 2
set_pad_physical_constraints -pad_name "VSS" -side 4 -order 2
```

Luego de tener esto, se puede generar el *floorplan*. Este comando genera un resultado que puede ser observado en la interfaz gráfica de **ICC**. El comando a ejecutar permite modificar las distancias que van a tener los *pads* de entradas y salidas del *core:* del layout así como el mismo tamaño del *core*.

```
create_floorplan -control_type width_and_height -core_utilization
0.7 -core_width 5 -core_height 50 -no_double_back -top_io2core 10 -
bottom_io2core 10 -left_io2core 5 -right_io2core 5
```

El comando `check_physical_design` se puede ejecutar para poder realizar una verificación de los requisitos o de las configuraciones realizadas.

Luego de generar el *floorplan*, este puede ser ajustado para que el programa decida tamaños y otras especificaciones que no se han indicado. Esto lo hace encontrando los valores que mejor se acoplen al diseño realizado. Esto se hace con el comando `adjust_fp_floorplan`. Este es un paso opcional.

Una vez finalizado el *floorplan*, se puede comenzar con la configuración y la ejecución del *placement*.

3. **Configuración y creación del *placement***

   Para iniciar, se debe de configurar la forma en la que se llevará la colocación de elementos. Algunas de las configuraciones que se pueden realizar son la separación entre celdas, el *fanout* y la calidad del *placement*. El primer comando que se corre permite la configuración de más aspectos del proceso de colocación, el segundo posiciona los elementos y el tercero se encarga de optimizar las ubicaciones si es posible.

   ```
   set_fp_placement_strategy -adjust_shapes on
   create_fp_placement -effort High -max_fanout 800 -optimize_pins -
   congestion_driven -timing_driven -consider_scan
   place_opt -effort high
   ```

   Al estar satisfechos con las posiciones de las celdas dentro del *layout*, se pasa a la legalización del *placement*. Esto significa aceptar las posiciones a las que llegaron durante esta etapa de la síntesis física. Esto se hace corriendo `legalize_placement`. Una vez se haya ejecutado, no se podrá modificar el *floorplan* ni el *placement*. Con esto se da por terminada la etapa de *placement*.

4. **Configuración y creación del *routing***

   Para iniciar con la interconexión de las celdas, es necesario generar un anillo de VDD y VSS, al igual que *power straps*. Estas son necesarias para que la alimentación y la tierra del circuito sea accesible para todas las celdas dentro del *core*. Esto se realiza con los siguientes comandos:

   ```
   create_rectangular_rings -nets {VDD VSS} -around core
   create_power_straps -direction vertical -start_at 155 -
   num_placement_strap 4 -increment_x_or_y 50 -nets {VDD} -layer METAL2
    -width 2 -do_not_route_over_macros
   create_power_straps -direction vertical -start_at 150 -
   num_placement_strap 4 -increment_x_or_y 50 -nets {VSS} -layer METAL3
    -width 2 -do_not_route_over_macros
   ```

   Luego de los anillos y los *straps*, se especifica el tipo de ruteo con el que se trabajará. Se ejecuta el comando `set_route_mode_options -zroute false` para esto.

   Con estos elementos creados, se debe de verificar que sí sea posible la interconexión de elementos cumpliendo con varias restricciones. Se debe de tener conexiones de VDD

y VSS (se debe de realizar nuevamente con los primeros 4 comandos del paso 2), no deben de haber señales de reloj y se tienen las *Technology Files* asociadas (realizado al inicio de la síntesis). Para verificar que no existan señales de reloj, se puede ejecutar `report_clock_tree`. Finalmente, existen tres requisitos más de congestión, tiempo y capacitancia que pueden ser verificados por medio de `check_routeability`.

Después de haber cumplido con los requisitos anteriores, se puede configurar el *routing*. El primer comando mostrado ayuda a configurar conexiones en el equipo y permite revisar errores o problemas de *timing*. El segundo comando ayuda con la verificación DRC del *layout*.

```
set_route_opt_strategy
set_preroute_drc_strategy
```

Se prosigue con la ejecución de la etapa de *routing*. Este se separa en el ruteo de los nodos VSS, VDD y, finalmente, las celdas del *layout*. También se realiza una optimización en caso de que sea necesario. El último comando realiza una última verificación a lo realizado durante la síntesis física.

```
preroute_standard_cells -mode net -connect both -nets {VDD}
preroute_standard_cells -mode net -connect both -nets {VSS}
preroute_instances
route_opt
verify_route
```

Luego de tener interconectadas todas las celdas, es necesario poder generar *fillers*. Esto es debido a que no pueden haber espacios vacíos dentro del layout generado. Es necesario generar estos elementos para los *pads* y el *core* del *layout*. Esto se hace por medio de los siguiente comandos:

```
insert_pad_filler -cel "PFILLER1 PFILLER5 PFILLER05 PFILLER0005
PFILLER10 PFILLER20" -overlap_cell "PFILLER0005"
insert_stdcell_filler -cell_without_metal "FILL1BWP7T FILL2BWP7T
FILL4BWP7T FILL8BWP7T FILL16BWP7T FILL32BWP7T FILL64BWP7T" -
connect_to_power "VDD" -connect_to_ground "VSS"
```

Finalmente, se puede obtener el archivo `.gds`, sin embargo, esta parte no funcionó de la manera esperada. Algunos de los comandos utilizados para esto fueron:

```
set_write_stream_options -output_pin {text geometry} -keep_data_type
write_stream -lib_name Milky_Aba -format gds "NOT.gds"
```

Este trabajo permitió realizar de manera exitosa el *floorplan*, *placement* y *routing* de la síntesis física a partir del *script* inicial que se tenía. Además de eso, se implementaron *pads*, *corners* y *power straps* que no estaban en la iteración anterior de esta parte del trabajo.

6.4.2. Trabajo de Marvin Flores

El trabajo [5] trata acerca de la creación del anillo de entradas y salidas, la ejecución de **ERC** y el *Antenna Rule Check*. Sin embargo, las secciones de interés son aquellas que tratan específicamente sobre el **ERC**.

Regla	Configuración por defecto	Configuración aplicada
WELL TO PG CHECK	Habilitado	Habilitado
GATE TO PG CHECK	Deshabilitado	Habilitado
PATH CHECK	Deshabilitado	Habilitado
DS TO PG CHECK	Habilitado	Habilitado
FLOATING GATE CHECK	Habilitado	Habilitado
FLOATING WELL CHECK	Habilitado	Habilitado
NW RING	Deshabilitado	Deshabilitado

Cuadro 1: Reglas configuradas para el **ERC** [5]

Los archivos requeridos para poder correr esta verificación son el `.gds`, `.icv` y el *runset* para el **ERC**. El `.gds` es el que se genera por la síntesis física por el software **ICC**. El *runset* utilizado es el que fue proveído por **TSMC**. En este es necesario realizar modificaciones para poder realizar todas las verificaciones necesarias. Estos cambios se muestran en el Cuadro 1. Finalmente, el archivo `.icv` del esquemático es generado mediante el siguiente proceso [14]:

1. **Extracción de *headers* en *netlists* de la *Standard Cell Library* y de la *I/O Cell Library***

   Este paso consiste en eliminar líneas del archivo `.v` de estas librerías para dejar únicamente las definiciones de los módulos con sus *input*, *output* e *inout*.

2. **Concatenación de *headers***

   En este paso, se combinan los archivos generados en el paso anterior en uno solo con el comando:

   ```
   cat headers_standard_cell.v headers_IO_cell.v > headers.v
   ```

   El archivo `headers.v` se puede llamar de cualquier forma y será útil en los siguientes pasos.

3. **Traducción de *headers* a CDL (SPICE)**

   El tercer paso es necesario para poder obtener un archivos **CDL** que contenga las referencias a las librerías utilizadas. En el comando mostrados, se pasa el nombre del archivo generado en el paso anterior y el nombre para el resultado, en este caso `libraries.sp`.

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

4. **Unión de CDL de librerías con *netlist***

   Con este archivo **CDL** y el *netlist* original, ya es posible obtener el archivo `.icv`. esto se hace mediante el siguiente comando:

   ```
   icv_nettran -verilog netlist_sintesis_logica.v -sp libraries.sp -
   outType ICV -outName CDL_netlist.icv
   ```

Una vez se tenga el `.icv`, es posible realizar la verificación junto con el *runset* y el *layout*. Esto se hace por medio de

```
icv -i DireccionArchivo.gds -c NombreCelda -s DireccionArchivo.icv -sf
ICV -vue DireccionRunset
```

Luego de correr este comando, se generan distintos archivos dentro de la carpeta en donde se está trabajando. Los más importantes son el `.RESULT` y `.LAYOUT_ERRORS`. Como lo indican sus extensiones, estos archivos indican el resultado de la verificación **ERC** y los errores que se encontraron durante el proceso.

Los errores que se detectaron en las pruebas fueron el mismo para todos los diseños. Este consistía en tener el sustrato tipo p de un nMOSFET flotando. Esto era un error en el *runset* del **LVS**, que se corrigió al cambiar la línea `psub = psube not nxwell` del archivo por

```
psub = psube and nxwell
```

Además del error, se tenía una advertencia por el tipo de *pin* con el que estaba definido el *Bulk*. Esta advertencia se solucionó siguiendo la sugerencia de que se mostraba en la terminal. Este cambio era modificar el tipo de *pin* que estaba definido como `TERMINAL` a `BULK` en las líneas del *runset* que eran similares a estas:

```
{psub,"BULK",pin_type="BULK"}
```

Como resultado de este trabajo, se obtuvieron resultados satisfactorios para las verificaciones de **ERC** de los distintos circuitos generados.

CAPÍTULO 7

Library Manager

Una de las recomendaciones indicadas en trabajos de graduación [6] y [5] es el cambio en la herramienta de **ICC** ya que esta podría ser descontinuada en un futuro, además de no generar los resultados esperados para el archivo `.gds`. Es por esto que se decidió utilizar la herramienta **ICC 2** como parte de este trabajo para mejorar el flujo que se ha realizado en iteraciones anteriores.

Una de las diferencias entre **ICC** y **ICC 2**, es el tipo de librería con el que trabaja cada herramienta. **ICC** utiliza librerías *Milky Way*, mientras que **ICC 2** utiliza librerías del formato *New Data Model* (**NDM**). Es por esto que se vuelve necesario la creación de librerías utilizando la herramienta *Library Manager* (**LM**).

7.1. Archivos necesarios para la librería NDM

Estas librerías **NDM** contienen la información lógica y física de las celdas que se definen dentro de ellas. **LM** recibe como entradas tres archivos para la creación de la `.ndm`:

- ***Technology File:*** Este es un archivo que contiene la información acerca de las diferentes capas de metal y algunas de las reglas de ruteo para una tecnología en específico. Estos generalmente tiene una extensión `.tf`.
- **Librería Física:** Estas son las librerías que contienen las características físicas de una celda, es decir, las formas que tienen los metales de una celda específica. Estas librerías pueden ser generadas a partir archivos `.lef` o `.gds`.

- **Librería Lógica:** En estos archivos se especifican características como el *timing*, la potencia y el ruido de una celda. Los archivos que son utilizados para este tipo de librerías son de extensión `.db`.

Al tener estos tres archivos, es posible generar una librería de celdas que contiene la información física y lógica [15].

Para el caso de las librerías proporcionadas por **TSMC**, se tienen varios archivos con la misma extensión. En el caso de los `.tf`, estos están definidos para las diferentes cantidades de capas de metal que se van a estar utilizando en el diseño. La ruta en la que se encuentran dentro de la carpeta de **TSMC** es la siguiente:

```
TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/
Back_End/milkyway/tcb018gbwp7t_270a/techfiles/
```

En esta carpeta se tienen los *Technology Files* para utilizar 4, 5 y 6 capas de metal en el diseño que se está realizando.

En el caso de los archivos `.db`, estos muestran los distintos escenarios en los que pueden estar operando como se menciona en el trabajo [6]. Estos pueden ser encontrados en las rutas que se muestran en el Cuadro 2. Las vistas que contienen estos archivos en las librerías de **TSMC** es la de *timing*.

Para los archivos de la librería física, también se muestran diferentes opciones. Los archivos `.lef` que se tienen en las carpetas que se muestran en el Cuadro 2 corresponden a la representación física de las celdas en diferentes cantidades de capas de metal como los `.tf`. En los archivos `.lef` están definidas las vistas *frame* que son similares a las de *layout*, sin embargo, estas solo contienen las posiciones de los pines de las celda y áreas de *routing blockage* en donde no es permitido que pasen metales [15].

Todos los archivos utilizados para generar las librerías **NDM** pueden ser copiados hacia otro directorio para poder acceder a ellos de manera más fácil.

7.2. *Flows* en *Library Manager*

LM también es capaz de tener diferentes *flows* o flujos que dictan cómo se va a comportar la herramienta con las entradas que se le proporcionen. Para la creación de la librería utilizada para la síntesis física de los circuitos, se utilizaron tres flujos distintos [15]:

- ***Normal Flow:*** Este es el flujo estándar para la herramienta, en donde se le indican los archivos que contienen las características físicas y lógicas de las celdas para generar la librería **NDM** que contienen esta misma información de manera consolidada.
- ***Physical-only Flow:*** En este tipo de flujo, se ingresan como entradas los mismos archivos que en el *Normal Flow* pero, a diferencia de ese flujo, se crea una librería **NDM** que contiene solamente las características físicas de las celdas que no tienen una contraparte lógica en el archivo `.db`. Este proceso se puede observar en la Fig. 7, en donde se obtienen librerías distintas para el *Normal Flow* y el *Physical-only Flow*.

Librería	Archivo	Ruta
Celdas estándar	.db	TSMC/180/CMOS/G/stclib/7-track/ tcb018gbwp7t_290a_FE/TSMCHOME/digital/ Front_End/timing_power_noise/NLDM/ tcb018gbwp7t_270a/
	.lef	TSMC/180/CMOS/G/stclib/7-track/ tcb018gbwp7t_290a_FE/TSMCHOME/digital/ Back_End/lef/tcb018gbwp7t_270a/lef/
Pads	.db	/TSMC/180/CMOS/G/IO3.3V/iolib/LINEAR/ tpd018nv_280a_FE/TSMCHOME/Digital/ Front_End/timing_power_noise/NLDM/
	.lef	/TSMC/180/CMOS/G/IO3.3V/iolib/LINEAR/ tpd018nv_280a_FE/TSMCHOME/Digital/ Back_End/lef/tpd018nv_280a/mt_2/6lm/

Cuadro 2: Rutas a archivos para las librerías

- ***Aggregate Flow:*** Este flujo es utilizado para poder añadir diferentes librerías **NDM** dentro de una sola. Este proceso es útil para poder movilizar la librería y también al momento de especificar las librerías a utilizar en la síntesis física.

Existe más tipos de flujos como el flujo para edición y otros que son más especializados que no fueron utilizados para este trabajo. Dependiendo de la librería que se quería obtener, se aplicaba un flujo distinto. para las celdas de la librería de las celdas estándar fue necesario generar un librería con un *normal flow* y otra con un *physical-only flow*. Esto es debido a que se tienen las celdas que contienen las compuertas y también contiene los *fillers* que únicamente tienen una parte física. Para las librerías de los *pads*, también fue necesario generar dos flujos ya que dentro de ellas estaban definidas los *corners* y *fillers* para el anillo de entradas y salidas. Finalmente, se utilizo el *aggregate flow* para poder tener una sola librería que contuviera toda la información de las cuatro librerías generadas por separado.

7.3. Creación de librerías en *Library Manager*

LM permite la creación de librerías por medio de una interfaz gráfica que puede ser invocada por medio de cualquiera de los siguientes comandos:

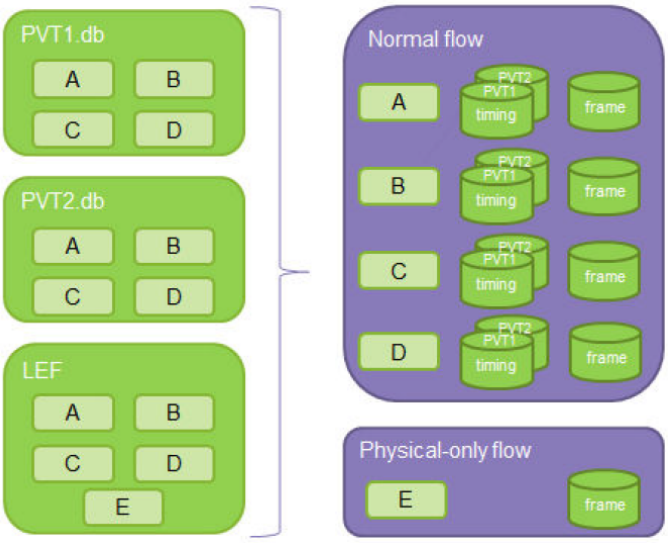

Figura 7: Ejemplo de un *Normal Flow* y un *Physical-only Flow* [15]

```
icc2_lm_shell -gui
lm_shell -gui
```

Al correr este comando, se debería de mostrar una ventana como la que se muestra en la Fig. 8. Esta interfaz permite crear las librerías **NDM** de una manera más interactiva sin la necesidad de conocer los comandos que se deben de ejecutar en la consola de **LM**. Además de crear librerías, también se pueden editar o ver librerías creadas anteriormente.

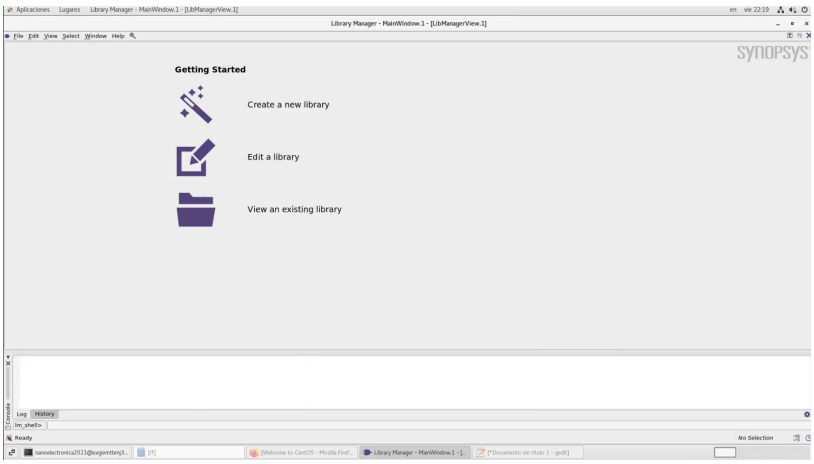

Figura 8: Interfaz gráfica de *Library Manager*

Para iniciar un nuevo flujo para la creación de una librería, se debe de hacer clic en la opción de *Create a new library*. Esto debería de desplegar una nueva ventana que se muestra en la Fig. 9. En esta, se debe de indicar el nombre del *workspace* para la librería, el tipo de flujo a realizar y el archivo `.tf` que se utilizará. Luego de haber seleccionado el *Technology File*, es posible cargar los archivos que contengan la información de las celdas en las pestañas correspondientes. Luego de seleccionar los archivos, se debe dar clic en el botón de *read* para

que cargue los contenidos de los archivos al *workspace* de la librería. Luego de haber cargado los archivos, la interfaz gráfica debería de verse de manera similar a la Fig. 10. De esta manera, se encuentran cargados todos los archivos necesarios para crear la librería.

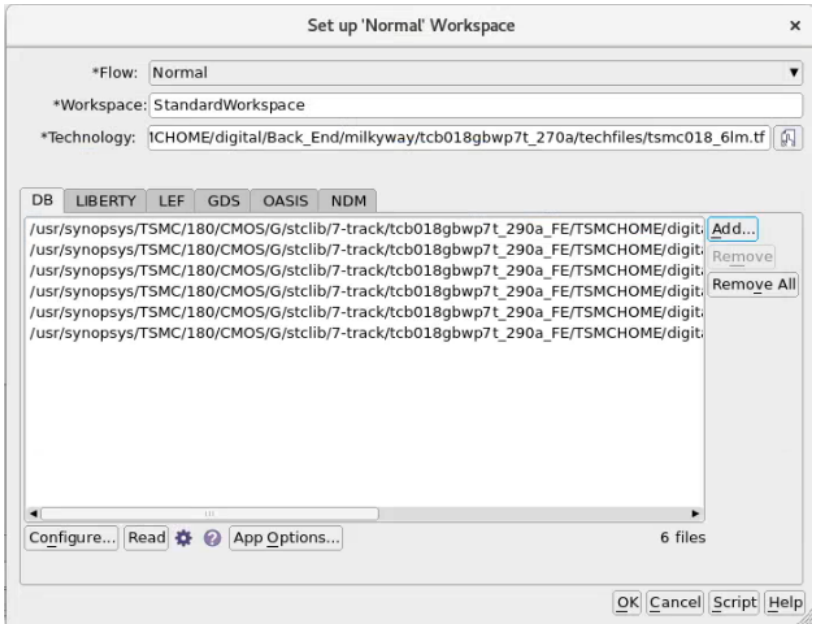

Figura 9: Interfaz gráfica de *Library Manager* para iniciar un nuevo flujo

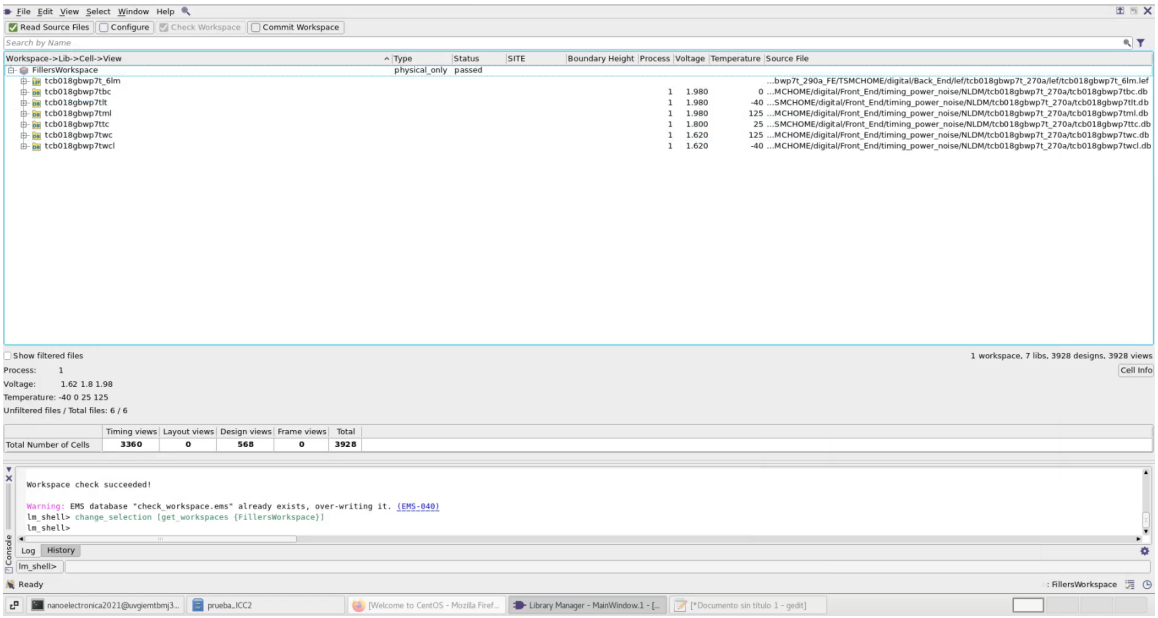

Figura 10: Interfaz gráfica de *Library Manager* luego de cargar archivos

Para finalizar, se debe de hacer un *check workspace* y luego un *commit workspace*. Estas se encuentran en la parte superior de la interfaz gráfica de **LM**. La primera acción realiza una revisión de las celdas que se hayan ingresado y reporta los errores que encuentra, además de

poder realizar una corrección de algunos. El segundo comando se encarga de crear la **NDM**, generando un archivo `.ndm` dentro de la carpeta en donde se está trabajando. Si ocurre algún error o *warning* en el proceso, también son mostrados en la consola de **LM**.

Con estos archivos cargados, es posible consultar información más detallada de las celdas en la interfaz gráfica. Como ejemplo, se puede observar la vista *frame* en la Fig. 11. También es posible consultar los errores que se generaron por medio de una opción al momento de correr el *check workspace* como se muestran en la Fig. 12. Otra de las funcionalidades de utilizar la interfaz gráfica es poder conocer los comandos que se ejecutan para realizar las acciones que se indican. Fue de esta manera en la que se obtuvieron los comandos para crear las librerías en la consola de **LM**.

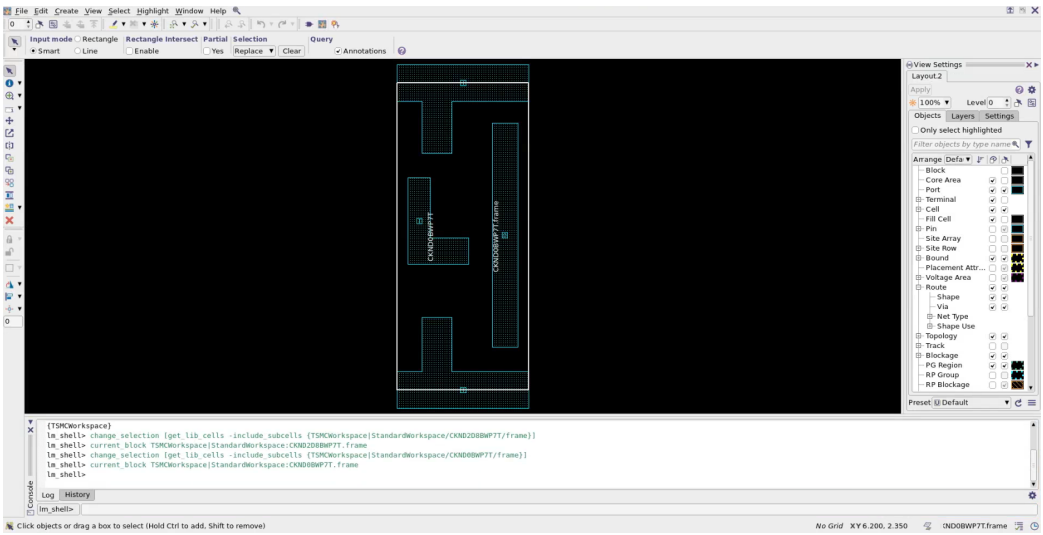

Figura 11: Vista *frame* de una celda en *Library Manager*

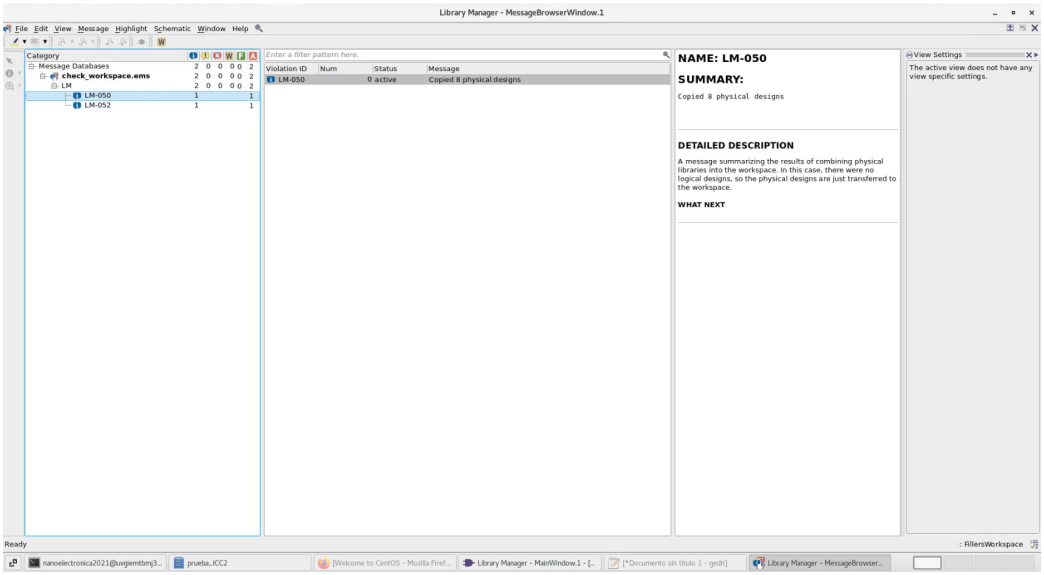

Figura 12: *Error browser* de *Library Manager*

Además de poder utilizar la interfaz gráfica también es posible utilizar los comandos que

se muestran en el anexo 16.1 para la creación de las librerías ingresando los comandos en la parte inferior de la interfaz gráfica en donde se muestra la consola de **LM** y el editor de *scripts*. Para iniciar el proceso, es necesario crear un *workspace* indicándole su nombre, el tipo de flujo a utilizar y el `.tf`.

```
create_workspace -flow tipo_de_flujo -technology /ruta/a/archivo/
techfile.tf nombre_workspace
```

Luego de haber generado el *workspace* y asociarle un *Technology File*, se puede proseguir a leer los archivos `.db` y `.lef` que contienen la información de las celdas. Esto se hace corriendo los siguientes comandos:

```
read_db {/ruta/a/archivo_1.db ... /ruta/a/archivo_n.db}
read_lef /ruta/a/archivo.lef
```

Luego de haber cargado los archivos, la ventana de **LM** debería se verse similar a la que se muestra en la Fig. 10.

Finalmente, se corren los comandos mostrados a continuación para finalizar el flujo. Los dos comandos `gui_create_window` y `open_ems_database` son opcionales. Estos permiten ver los *warnings* y errores que pudo haber encontrado y solucionado **LM** durante el proceso del chequeo de la librería. *workspace*.

```
current_workspace; check_workspace
gui_create_window -type MessageBrowserWindow
open_ems_database check_workspace.ems
current_workspace nombre_workspace; commit_workspace  -output
nombre_libreria.ndm
```

Al terminar este procedimiento, se debería de generar un directorio con extensión `.ndm` que contiene la librería creada. Este procedimiento se debe de repetir cuatro veces. Una vez es para obtener la librería de las celdas estándar, la segunda es para poder obtener la librería con los componentes *physical-only* de la librería de las celdas estándar, la tercera vez es para obtener la librería de los *pads* y la cuarta vez es para poder obtener la librería de los componentes *physical-only* instanciados en las librerías de los *pads*.

Después de obtener todas estas librerías, es posible poder consolidar todas en una sola. Es para esto que se utiliza el *aggregate flow*. Este tipo de flujo permite integrar varias librerías **NDM** dentro de una sola. La forma de hacer esto es similar a los otros flujos. El procedimiento es el siguiente:

```
create_workspace -flow aggregate nombre_workspace
read_ndm libreria_standard_cell.ndm
read_ndm libreria_physical_only_standard_cell.ndm
read_ndm libreria_pads.ndm
read_ndm libreria_physical_only_pads.ndm;
current_workspace; check_workspace
current_workspace nombre_workspace; commit_workspace  -output
nombre_libreria.ndm
```

El resultado de esta etapa son cinco librerías distintas, en donde cuatro contienen información de celdas específicas y la última es una colección de las otras. Las librerías creadas se pueden ver en la Fig. 13. `TSMCWorkspace.ndm` es la librería que contiene la información de las otras cuatro librerías creadas.

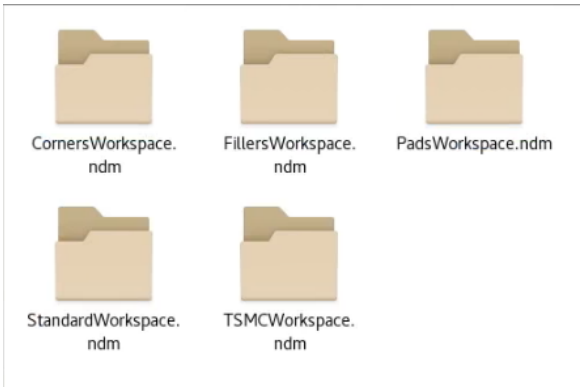

Figura 13: Librerías generadas por *Library Manager*

Como se mencionó anteriormente, existe la posibilidad para crear las librerías por medio de un *script*. Como los comandos mostrados en el anexo 16.1, es posible ingresarlos a un archivo `.tcl` que luego es cargado a la herramienta para poder ejecutar los comandos que están dentro del archivo. Esto es útil para not tener que correr los comandos línea por línea. Dentro de **LM** también es posible editar el *script* que se abre. Este *script* permite correr la creación de las librerías sin la necesidad de tener que abrir la interfaz gráfica de **LM**, facilitando el proceso de automatización del flujo de diseño del chip.

CAPÍTULO 8

Síntesis física en **ICC 2**

8.1. Interfaz gráfica de ICC 2

Luego de tener las librerías **NDM** con las celdas estándar y *pads* de TSMC, es posible comenzar a trabajar en la síntesis física de un circuito. Como entradas a esta parte del *design flow*, necesitamos los archivos resultantes de la síntesis lógica y la librería generada con LM. Para iniciar la interfaz gráfica de **ICC 2**, se corre el siguiente comando:

```
icc2_shell -gui
```

Luego de correr este comando, se abre una ventana como la que se muestra en la Fig. 5. En esta ventana es posible crear un nuevo bloque, modificar un bloque o ejecutar un *script*, en donde un bloque se refiere a la información lógica y física de un diseño específico [16]. La interfaz gráfica de **ICC 2** contienen muchas más herramientas que la de **ICC**, por lo que es mucho más amigable con el usuario. Una de las ventajas de utilizar la interfaz gráfica de **ICC 2** son los asistentes que proporciona para poder guiar el proceso de la síntesis física. Uno de estos asistentes es el *Task Assistant*. Este contienen distintas secciones que permiten la fácil ejecución de distintas partes de la síntesis física [17]. En la Fig. 14 se muestra el *Task Assistant* para la etapa de creación de bloque. El resto de etapas en las el *Task Assistant* puede ser de utilidad se muestra en la Fig. 15. Además de ser una guía para el usuario, este asistente también presenta ejemplos para algunas secciones del flujo de la síntesis física.

Al momento de comenzar a trabajar, la interfaz gráfica de **ICC 2** tiene distintos paneles. En la Fig. 16 se puede ver una síntesis física en proceso. La ventana se separa en tres regiones distintas. El recuadro negro de la interfaz gráfica muestra el estado actual del *layout* que se está generando. Mientras se vaya avanzando en el proceso de la síntesis, las celdas se van a

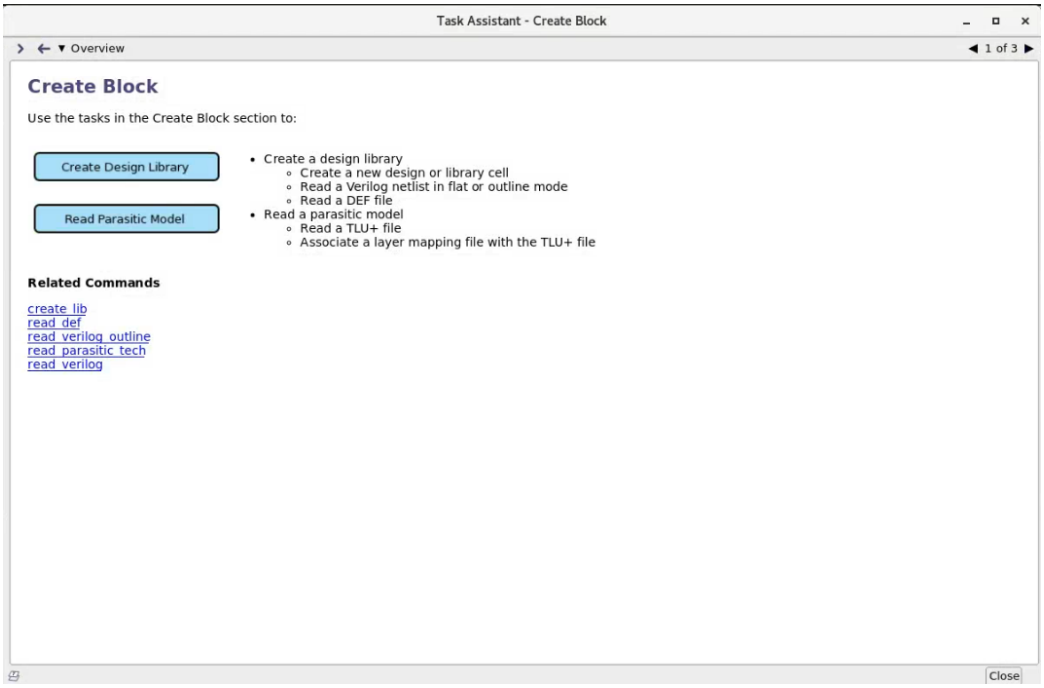

Figura 14: *Task Assistant* de **ICC 2**

ir posicionando y conectando dentro de este recuadro. En la parte derecha, se tienen varias pestañas que se pueden consultar. Las dos más importantes son las de *View Settings* y *Tasks*. Con la pestaña de *View Settings* es posible modificar los elementos que se quieran tener visibles en el cuadro del *layout*. Es posible filtrar metales, pines, celdas, etc. Es importante presionar *apply* luego de hacer los cambios. La pestaña de *Tasks* permite consultar las diferentes etapas que tienen el *Task Assistant*. En esta pestaña se presentan los pasos de una forma más detallada que los que se muestran en la Fig. 15. Finalmente, en la parte inferior se tiene la consola y el editor de *scripts* similar a los que se tienen en LM. En esta parte es posible ingresar los comandos para la síntesis física y poder consultar los errores que puedan surgir durante el proceso.

8.2. Comandos para la síntesis física

Una vez dentro de la interfaz gráfica, se puede proceder a correr los comandos para iniciar la síntesis física. El primer comando a correr es el que se utiliza para poder crear una nueva librería en donde se van a guardar las características físicas y lógicas del diseño. A este comando se le debe de indicar el nombre que queremos para la librería, el *Technology File* que se va a utilizar y la librería de referencia que se va a utilizar. El archivo `.tf` es el mismo que se utilizó para la creación de las librerías en **LM**. La librería que se utiliza como referencia es la que librería consolidada que se generó en la creación de librerías en **LM**.

```
create_lib nombre_libreria.ndm -technology /ruta/a/archivo.tf -ref_libs
/ruta/a/libreria.ndm
```

Luego de crear la librería en donde se van a guardar las características del diseño, es

Figura 15: Opciones del *Task Assistant* de **ICC 2**

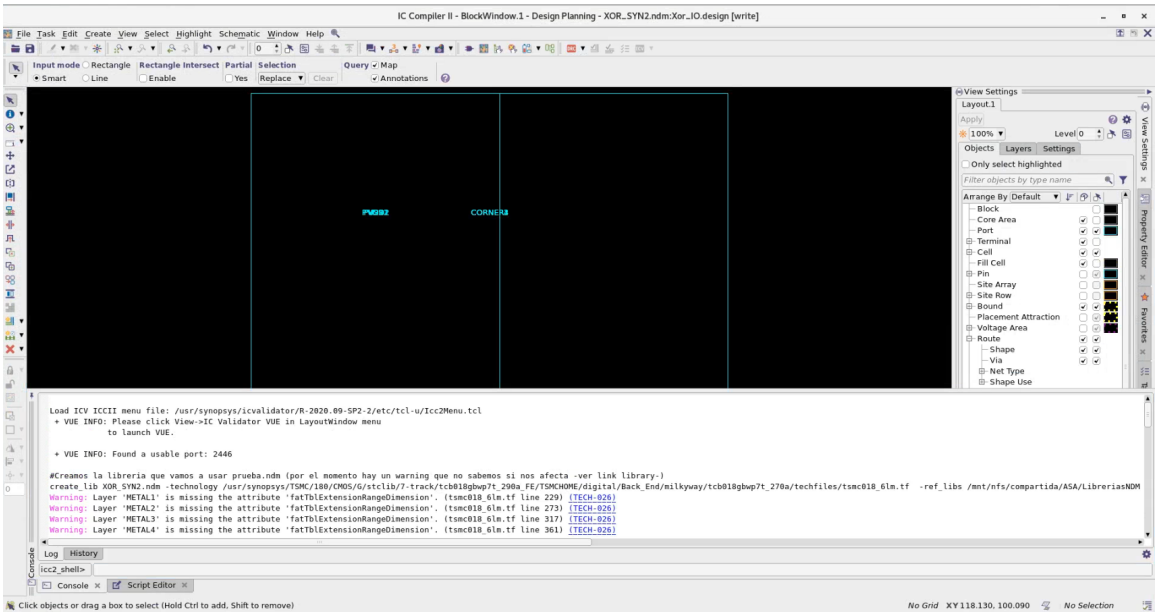

Figura 16: Interfaz gráfica de **ICC 2** durante la síntesis física

necesario leer el archivo *verilog* `.v` y el archivo `.sdc` que son producto de la etapa de síntesis física. Luego de haber leído los archivos, la interfaz gráfica se verá de forma similar a la que se muestra en la Fig. 17.

```
read_verilog /ruta/a/archivo.v
read_sdc -echo /ruta/a/archivo.sdc
```

Además de cargar los archivos de la síntesis física, también se puede cargar el archivo *TLUPLUS* que contiene la información parasítica de los metales. Para hacer esto, es necesario indicar el archivo `.tluplus` y un *layermap* para que **ICC 2** pueda realizar un mapeo de las capas del diseño a las características que se indican en el archivo *TLUPLUS*.

```
read_parasitic_tech -tlup /ruta/a/archivo.tluplus -layermap /ruta/a/
archivo/layermap.map
```

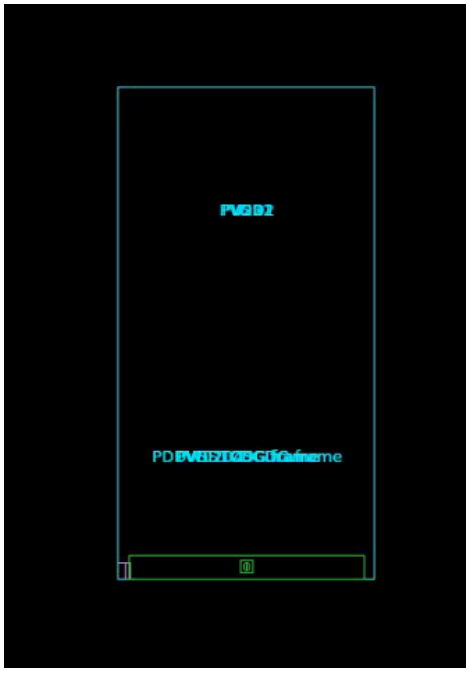

Figura 17: Interfaz gráfica de **ICC 2** luego de haber leído los archivos `.v` y `.sdc`

Estos archivos se pueden encuentran en la siguiente dirección:

```
TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/
Back_End/milky_way/tcb018gbwp7t_270a/techfiles/tluplus
```

Después de cargar estos archivos, se puede proceder a crear los *corners* y *pads* de entradas y salidas si el archivo de *verilog* no los contiene inicialmente. Para esto, es necesario indicar los nombres que se le quiere dar a los *corners* o *pads* entre llaves y luego indicar la celda de la librería que se quiere utilizar. Los *pads* seleccionados fueron los mismos que se utilizaron el trabajos anteriores.

```
create_cell {corner_1 corner_2 corner_3 corner_4} PCORNER
create_cell {pad_VDD} PVDD1CDG
create_cell {pad_VSS} PVSS1CDG
```

Luego de correr estos comandos, se agregan más celdas que pueden ser vistas en la interfaz gráfica.

Luego de tener creados los *pads* para VDD y VSS, es necesario generar las *nets* y crear las conexiones lógicas entre las celdas y los pads. Los primeros dos comandos crean las *nets* y el comando de `connect_pg_net` se encarga de realizar las conexiones entre las celdas. Estas conexiones son lógicas, por lo que no se ven reflejadas en la interfaz gráfica de **ICC 2**. Luego de correr el comando para conectar las *nets*, **ICC 2** genera un resumen de las conexiones que realizó. Un ejemplo de esto, se puede ver en la Fig. 18. En esta imagen se indican la cantidad de conexiones de de VDD y VSS que se tenían antes y después de correr el comando. En este ejemplo, se tenían 0 conexiones antes y 9 conexiones para VDD y VSS dentro del diseño luego correr `connect_pg_net`.

```
create_net -power VDD
```

```
create_net -ground VSS
connect_pg_net -automatic
```

```
icc2_shell> connect_pg_net -automatic
****************************************
Report : Power/Ground Connection Summary
Design : Xor_IO
Version: R-2020.09-SP3
Date   : Sun Aug 15 21:47:40 2021
****************************************
P/G net name                P/G pin count(previous/current)
--------------------------------------------------------------------------------
Power net VDD               0/9
Ground net VSS              0/9
--------------------------------------------------------------------------------
Information: connections of 18 power/ground pin(s) are created or changed.
```

Figura 18: Resultado de **ICC 2** luego del comando `connect_pg_net`

En el comando de `connect_pg_nets` se puede ver que se tiene la opción `-automatic`. Lo que hace esto es generar las conexiones entre VSS y VDD de manera automática, sin embargo, es posible tener un mayor control sobre las conexiones que se quieran realizar entre las *nets* de alimentación y tierra [18]. Un ejemplo de esto es con los siguientes comandos:

```
connect_pg_net -net VDD [get_pins -physical_context *VDD]
connect_pg_net -net VSS [get_pins -physical_context *VSS]
```

Con estos comandos se especifica la *net* que se quiere conectar y luego se le indica que se quieren conectar todos los pines que contengan VDD o VSS al final de su nombre. El `*` funciona como una *wildcard* para indicar cualquier combinación de letras dentro del nombre. Es por esto que el nombre del pin puede iniciar de cualquier forma, mientras termine en VDD o VSS, **ICC 2** será capaz de detectar el pin y realizar la conexión deseada.

En los comandos anteriores, se puede observar que es posible anidar comandos para utilizar el *output* de un comando como *input* del otro. En este caso, `get_pins` es el encargado de devolver el listado de pines que se encuentran dentro del diseño para que estos sean conectados.

Para revisar las conexiones realizadas, se puede utilizar el comando `report_cells -power` para consultar las conexiones de VDD y VSS que se hayan hecho. Un ejemplo de este reporte se puede observar en la Fig. 19. Con este comando, es posible ver las conexiones que se realizaron para alimentación y tierra para cada una de las celdas involucradas en el diseño.

```
Power data for cell 'U6':
Corner: default


Power pins:

    Pin Name       Type      PG Type   Power Rail  Netlist Net    UPF Supply Net Early Voltage  Late Voltage
-------------------------------------------------------------------------------------------------------------
    VDD            power     primary   <default_rail>
                                                   VDD            VDD            3.30 *         3.30 *
    VSS            ground    primary   <default_gnd>
                                                   VSS            VSS            -- *           -- *
```

Figura 19: Resultado de `report_cells -power` para una celda U6

Después de tener todas las celdas necesarias, es posible proseguir con la creación del

floorplan para el *chip*. Para esto se utilizó la ayuda del *Task Assistant*. La ventana en donde se encuentran el asistente para el *floorplan* se encuentra en *Task Assistant* → *Design Planning* → *Floorplan Initialization*. En esta ventana es posible indicar la forma que va a tener el *floorplan* dependiendo de los parámetros que se especifiquen. El comando que se utilizó para generar el *floorplan* final fue el siguiente:

```
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
{100 100} -core_offset {125}
```

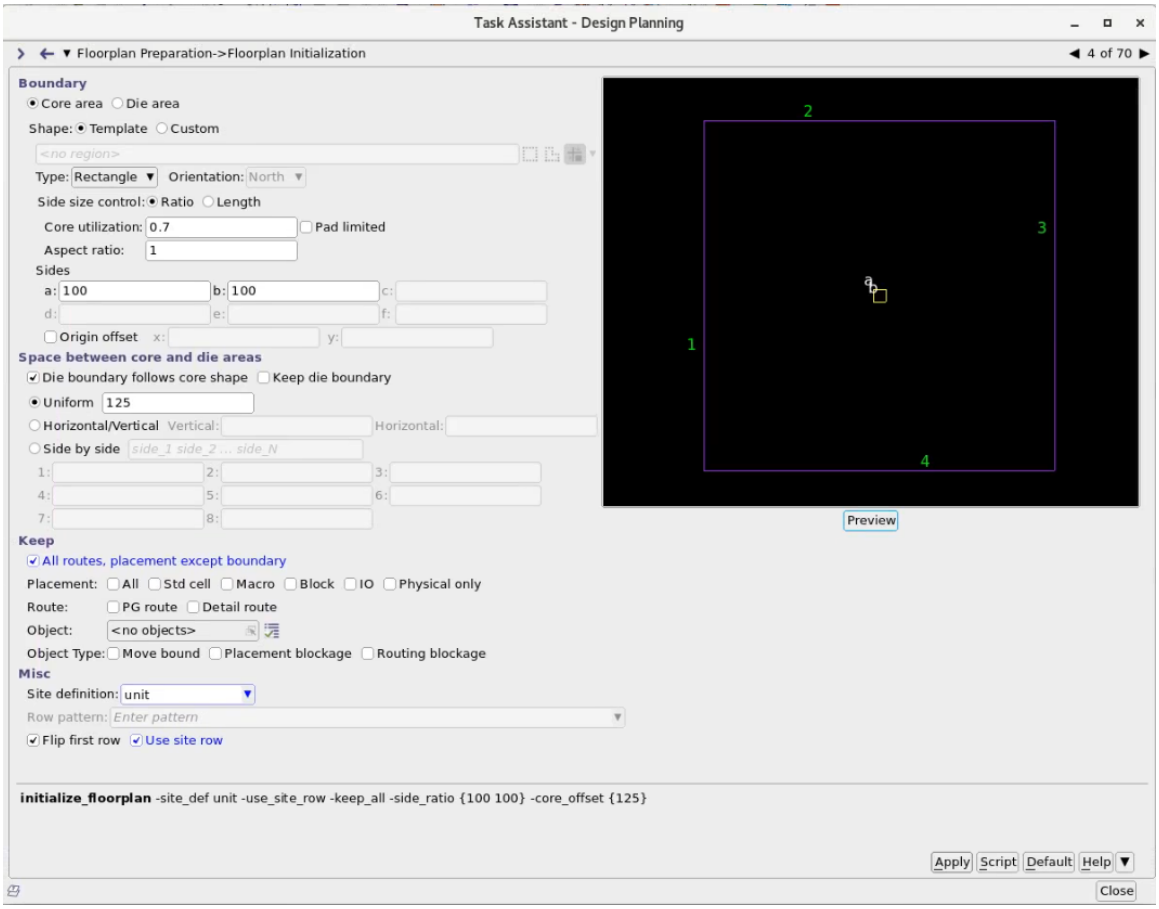

Figura 20: Asistente de **ICC 2** para la inicialización del *floorplan*

En este comando se le indica el `-site def` que se refiere a las características para las celdas que van a ocupar el *core* y las filas dentro del *core*. Luego, la opción `-use_site_row`, indica que se utilicen filas en el core. La opción `-keep_all` es utilizado para que mantenga los cambios que se han realizado como ruteos, colocación de celdas, o bloqueos, etc. También se indica el tamaño del *core* y el offset con el borde exterior del *chip*. Este comando contiene muchos más parámetros que se pueden utilizar para seguir personalizando el *floorplan* que se desea. Todas las opciones disponibles se pueden consultar en [18]. De igual forma, los campos también se pueden ver en el *Task Assistant* de **ICC 2**. Luego de correr el comando para el *floorplan*, la interfaz gráfica de **ICC 2** debería de verse de manera similar a la que se muestra en la Fig. 21.

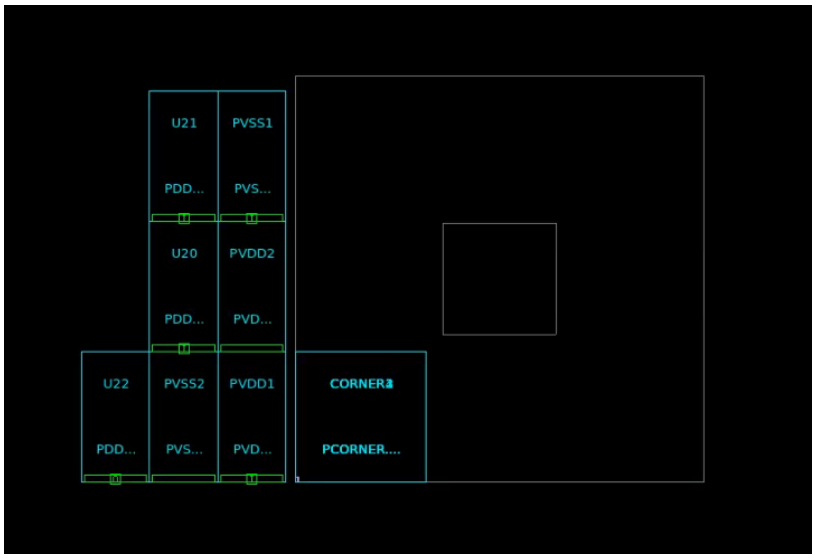

Figura 21: Interfaz gráfica de **ICC 2** luego de inicializar el *floorplan*

Luego de eso, se prosigue a generar un *IO ring*. Un anillo de entradas y salidas se refiere a crear ubicaciones en donde puedan ser colocados los *pads* de entradas y salidas por medio de *IO guides*. El comando genera cuatro guías, uno en cada lado del *chip*. El comando que genera el *IO ring* es el siguiente:

```
create_io_ring -name nombre_anillo_io -corner_height 115
```

El parámetro `-corner_height` es utilizado para indicar el tamaño de los *corners* utilizados. En el caso de las librerías de **TSMC**, estos tienen un tamaño de 115 μm. Esto es utilizado para poder acomodar el tamaño de los *pads* y *corners* en el *chip*. Las medidas del de los *corners* se pueden obtener a partir de la regla que se puede utilizar en **ICC 2**. A partir de este comando, se generan los siguientes *IO guides*:

```
nombre_anillo_io.top
nombre_anillo_io.right
nombre_anillo_io.bottom
nombre_anillo_io.left
```

Además de utilizar la línea de comandos, también es posible realizarlo con el *Task Assistant* de **ICC 2**. Este se encuentra en *Task Assistant* → *Floorplan Preparation* → *Flip-Chip I/O*. En la Fig. 22 se puede ver que se pueden indicar más opciones para la creación del anillo de entradas y salidas, sin embargo, las únicas opciones que se llenaron fueron el nombre y el tamaño de los *corners*. El resto de opciones son para crear un anillo que contenga *IO guides* creados con anterioridad, crear el anillo dentro de uno existente, los *pads* a incluir en el anillo, el tamaño que va a tener y el *offset* que va a tener. Todas las opciones se pueden consultar en [18].

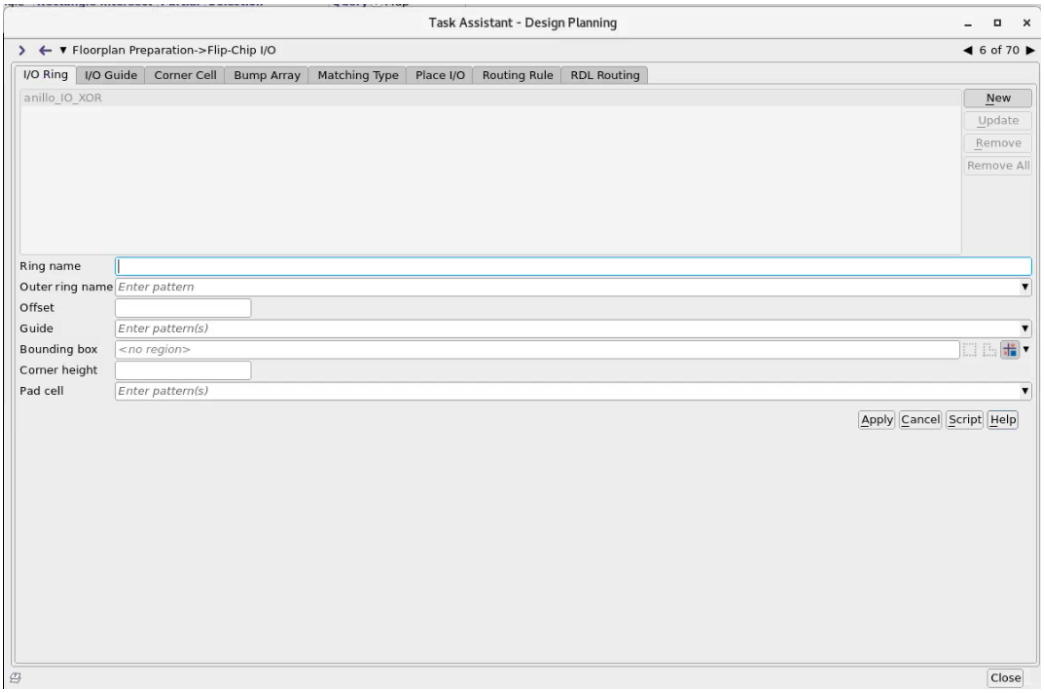

Figura 22: Asistente de **ICC 2** para la creación de *IO rings*

Luego de haber creado el anillo de entradas y salidas, se asignan los *pads* a una guía en específico. Esto se hace para tener un mejor control de las ubicaciones de los *pads* de VDD y VSS dentro del *chip*. En el caso de la síntesis física que se realizó, se colocaron los *pads* de alimentación y tierra en la izquierda y derecha del *core*. En este caso, se utilizó `*` nuevamente para indicar todos los *pads* que iniciaran con el nombre PVDD o PVSS y también se utilizó para poder obtener las guías que se encuentran a la izquierda y derecha sin importar el nombre que se coloca para el *IO ring*. De forma similar al comando de `get_pins`, se obtienen los *guides* por medio de `get_io_guides` como entrada al comando de `add_to_io_guide`.

```
add_to_io_guide [get_io_guides *left] PVDD*
add_to_io_guide [get_io_guides *right] PVSS*
```

Con los comandos anteriores ejecutados, se puede proceder a colocar los *pads* de IO y *corners*. Esto se hace con el siguiente comando: `place_io`. Esta instrucción coloca los *pads* y *corners* alrededor del *core*, generando un resultados como el que se muestra en la Fig. 23. `place_io` coloca los *pads* de una manera en la que **ICC 2** crea que sea más conveniente, sin embargo, con los comandos que se ejecutaron anteriormente para asignar los *pads* a un *IO guide* específico se fijan a una posición dentro del anillo. Si no se hubiera especificado de esta manera, no se sabría las posiciones en las que se colocarían los *pads*.

Después de haber colocado los *pads* alrededor del *core*, se hicieron los anillos de alimentación y tierra que rodean al *core* para poder realizar las conexiones a las celdas que se colocarán dentro del *chip* de una forma más eficiente.

```
create_pg_ring_pattern ring_pattern -horizontal_layer METAL2
-horizontal_width {2} -horizontal_spacing {2}
```

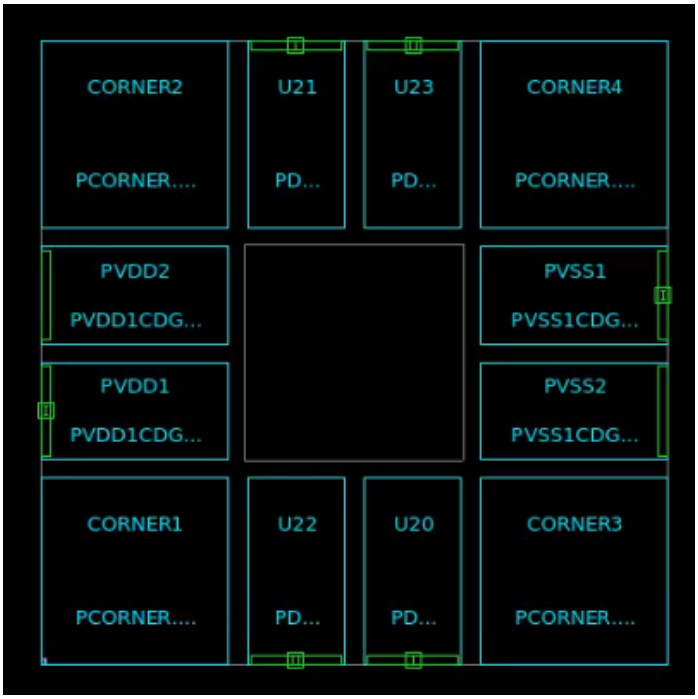

Figura 23: Interfaz de **ICC 2** luego de colocar los *pads* de IO y *corners*

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

El proceso para generar el anillo de alimentación y tierra inicia con la creación de un patrón que indica la forma que tendrá. En este caso, se crea un patrón llamado `ring_pattern` en donde los *tracks* horizontales del anillo tendrán un ancho de 2, un espaciado de 2 y estarán en la segunda capa de metal. Para el caso de los *tracks* verticales, se tienen las mismas características para el ancho y el espaciado, pero estarán en la capa 3 de metal. El *spacing* de este comando se refiere a la distancia que se tendrá entre *tracks* paralelos que pertenezcan a este patrón.

Luego de tener el patrón, se establece una estrategia. Esta estrategia se encarga de asociar un patrón a las *nets* que se quieran conectar. En este caso, se especifica el nombre de `core_ring` y se le asocia el patrón que se generó con el comando de `create_pg_ring_pattern`. Con la opción de `-pattern`, se especifica el nombre del patrón a utilizar, las *nets* que se quieren asociar y el offset al área en donde se genere el anillo. En el caso de las *nets*, es posible repetir valores para ir creando más anillos. En el caso de esta síntesis física, solamente es necesario un anillo para VDD y otro para VSS. Finalmente, se le indica que esta estrategia aplica para el *core* del diseño.

Finalmente, para insertar el anillo de VDD y VSS, se utiliza el comando `compile_pg`. En este caso, se especifica la estrategia que se desea insertar en el diseño. Luego de correr estos comandos, la interfaz gráfica de **ICC 2** se debería de ver de una manera similar a la que se muestra en la Fig. 24.

Para generar este anillo, también es posible utilizar el *Task Assistant*. Primero se crea una estrategia en la sección *Task Assistant* → *PG Planning* → *Ring/Hard Macro/Std Cell Pattern*. En esta sección se indica lo mismo que en el comando presentado. Luego, en la sección *Task Assistant* → *PG Planning* → *Strategy* se asocian patrones y *nets*. Finalmente, en la sección *Task Assistant* → *PG Planning* → *Create PG* se inserta el anillo al diseño.

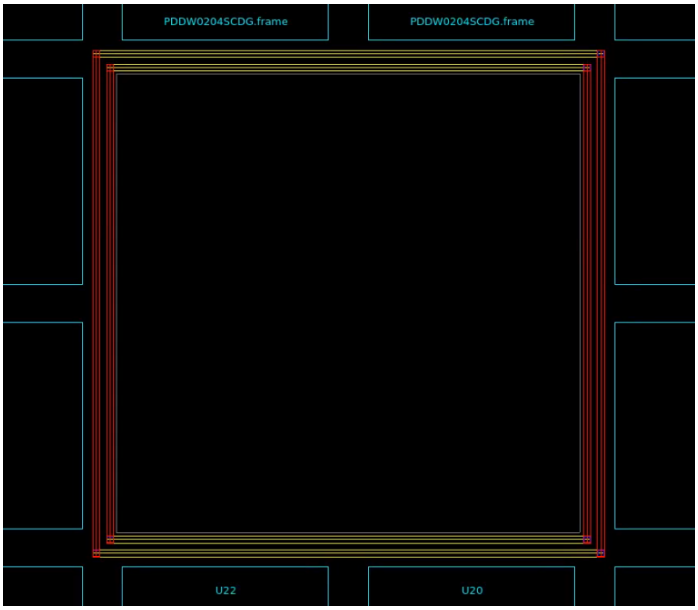

Figura 24: Interfaz de **ICC 2** luego de generar los anillos para VDD y VSS

Teniendo los anillos para VDD y VSS, es necesario realizar las conexiones entre los *pads* de VDD y VSS hacia el anillo. El proceso para realzar esto es similar al del anillo, en donde se debe de crear un patrón, asociar una estrategia y finalmente compilarlo. Para realizar esto, se utilizó como base el ejemplo llamado *Pad Pin Connection* que se encontraba en la sección de *Task Assistant* → *PG Planning* → *Examples* → *Overview*. Se adaptaron los comandos que se ejecutaban en este comando para realizar las conexiones que se deseaban.

```
create_pg_macro_conn_pattern hm_pattern -pin_conn_type scattered_pin
-layers {METAL2 METAL3} -nets {VDD VSS} -pin_layers {METAL2}
set_app_options -name plan.pgroute.treat_pad_as_macro -value true
set_pg_strategy macro_conn -macros [get_cells {PVDD* PVSS*}]
-pattern {{name: hm_pattern} {nets: {VDD VSS}}}
set_pg_strategy_via_rule macro_conn_via_rule -via_rule {{{{strategies:
macro_conn}}{{existing: all} {layers: METAL3}} {via_master: default}}
{{intersection: undefined}{via_master: NIL}}}
compile_pg -strategies macro_conn -via_rule macro_conn_via_rule
```

El patrón a generar en este caso es uno relacionado a las conexiones de celdas macro. Las celdas macro son celdas que tienen un mayor tamaño a las celdas estándar. Este patrón llamado `hm_pattern`, se le indica el tipo de pin que tiene los *pads* como primer parámetro. Estos pueden ser `long_pin`, `ring_pin` o `scattered_pin` [18]. Para este caso, se optó por utilizar los `scattered_pin` ya que esta clasificación era la más apropiada a los *pads* utilizados. El parámetro de `-layers` es utilizado para indicar las capas que se pueden utilizar

para generar los *straps* hacia el anillo de VDD y VSS. Luego de eso, se especifican los *nets* a conectar. Finalmente, la opción de `-pin_layers` es utilizado para indicar las capas de metal en donde se deben de encontrar los pines para generar las conexiones. En el caso de los *pads* utilizados, se inspeccionaron en **LM** para poder conocer en qué capa se encontraban.

Después de crear el patrón, es necesario cambiar una configuración de **ICC 2** con el comando de `set_app_options`. Con esta opción se le indica a la herramienta que trate los *pads* como macros para que el patrón generado anteriormente aplique a estos elementos [19].

Seguido de eso, se crea una estrategia para asociar el patrón. En este caso, se incluye una nueva opción que es `-macros` para indicar los macros o, en este caso, los *pads* para el cual aplica esta estrategia. De forma similar al anillo de alimentación y tierra, se le indica las *nets* que se quieren conectar, el nombre del patrón a utilizar y el nombre de la estrategia.

También se genera una estrategia para las vías a utilizar. Este indica cómo es que se van a generar las vías entre las estrategias creadas y las geometrías existentes dentro del *chip*. A esta estrategia se le dio el nombre de `macro_conn_via_rule`. Con la opción de `-via_rule`, se le indican los parámetros para realizar las conexiones por medio de vías. Este parámetro se puede separar en dos partes distintas:

```
{{{strategies: macro_conn}}{{existing: all} {layers: METAL3}} {
via_master: default}}
{{intersection: undefined}{via_master: NIL}}
```

La primera parte se le especifica a la regla para vías que se creen vías con los parámetros por defecto para cualquier intersección entre la estrategia `macro_conn` con geometrías en la capa 3 de metal existentes. La segunda parte del parámetro, indica que no se generen vías para cualquier otra intersección que no esté definida. En [18], se especifica esta estructura para los argumentos que se utilicen con la opción de `-via_rule`:

```
{{set_1}{filter_1}}{{set_2}{filter_2}}{via_def}
```

En las partes de *set*, se pueden indicar las estrategias, geometrías o pines a los que se quiere aplicar esta regla. Con *filter*, es posible indicarle específicamente ciertas *nets*, metales o *tracks* en un rango de anchura en donde se aplicará la regla. Finalmente, en la parte de `via_def` se especifica la vía que se quiere utilizar en estas intersecciones.

Por último, se compila la estrategia para insertarla al diseño con las reglas para las vías establecidas. El resultado se puede ver en la Fig. 25 en donde se observan *straps* saliendo de los pines del *pad* hacia el anillo creado anteriormente.

Después de crear estas conexiones, se prosigue con generar un *mesh* dentro del *core* para luego poder conectar las celdas estándar. El *mesh* generado consiste en varios *straps* verticales dentro del *core* que estaban conectados al anillo de VDD o de VSS. Al igual que las otras conexiones para alimentación y tierra, es necesario generar un patrón, una estrategia y luego compilarlo.

```
create_pg_mesh_pattern mesh_pattern -layers { {{vertical_layer: METAL3}
{width: 4.2} {pitch: 42} {spacing: interleaving}} }
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {259.960
265.280}} -pattern {{pattern: mesh_pattern}{nets: {VDD VSS}}} -blockage {
```

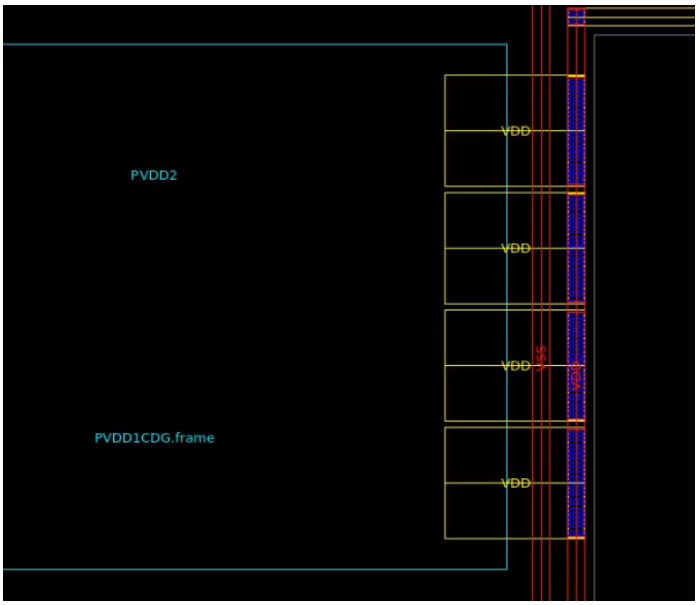

Figura 25: Interfaz de **ICC 2** luego de generar las conexiones de los *pads* y el anillo de VDD y VSS

```
macros: all}
 compile_pg -strategies mesh_strategy
```

En el patrón del *mesh*, es posible indicarle las capas en las que se va a ubicar y la dirección que se debe de tener. En este caso, únicamente se necesitaban los *straps* verticales del mesh. Además de eso, se especifica el ancho de los mismos con el parámetro de *width* y la distancia entre los *tracks* que pertenecen a la misma *net* con el *pitch*. Finalmente, se indica que el espaciado que se quiere tener entre los *tracks* de una misma capa de metal con el parámetro de *spacing*. *Interleaving* se utiliza para mantener un esapaciado uniforme entre todos los tracks, mientras que *minimum* utiliza el valor especificado en el *Technology File*. También es posible indicar un listado de valores.

Luego de crear el patrón, se le asocia nuevamente una estrategia. En este caso, se especifica una opción de *blockage* en donde se le indica a **ICC 2** que no se desea que se creen *tracks* sobre celdas macro. Además de eso, se le indica un área en donde se debe de generar la estrategia con la opción de *polygon*. Esta región se escogió ya que llega hasta la parte superior e inferior del anillo de VDD y VSS, lo cual permite hacer en los siguientes pasos las conexiones a las celdas. Finalmente, se compila la estrategia para generar un resultado como el que se observa en la Fig. 26

Continuando con las conexiones de alimentación y tierra, es necesario generar un último patrón. Este consiste en líneas horizontales dentro del *core* que permitirán realizar las conexiones entre las celdas que se coloquen dentro de él. Esto es conocido como el *standard cell connection pattern*. Los comandos son los siguientes:

```
 create_pg_std_cell_conn_pattern std_cell_pattern
 set_pg_strategy std_cell_strategy -core -pattern {{pattern:
std_cell_pattern}{nets: {VDD VSS}}}
 compile_pg -strategies std_cell_strategy
```

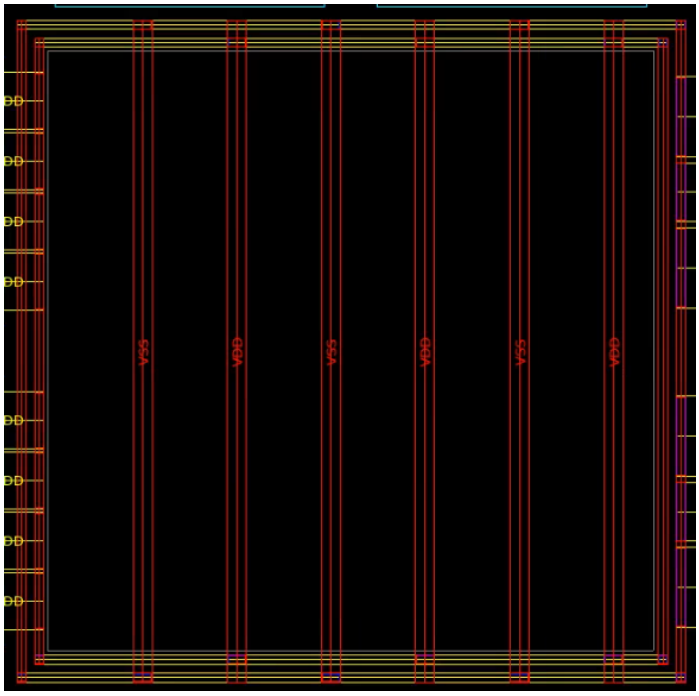

Figura 26: Interfaz de **ICC 2** luego de generar los *straps* dentro del *core*

En este caso, se genera la estrategia para el *core*. El resultado de este comando se puede ver en la Fig. 27. Se puede notar que estas conexiones horizontales están conectados a los *straps* creados con anterioridad, por lo que estos funcionan como rieles de alimentación para las celdas a colocar.

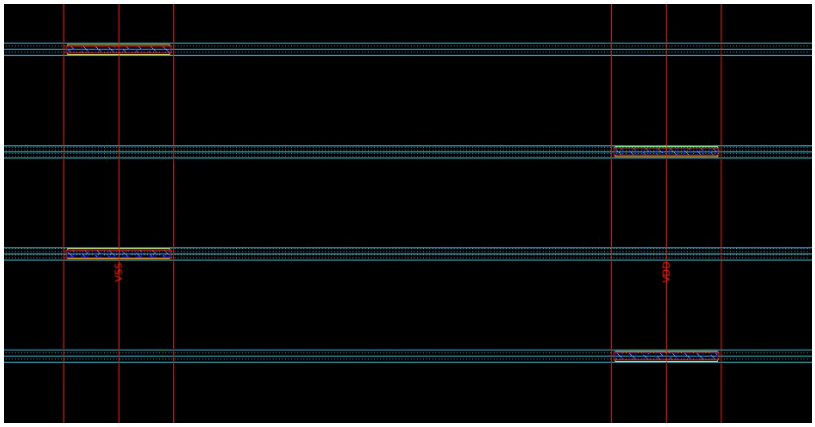

Figura 27: Detalle de las conexiones entre los *straps* y los rieles para las celdas estándar

Para generar estos patrones de alimentación y tierra, se utilizaron nuevamente los ejemplos que se encuentran en el *Task Assistant* de **ICC 2**. Estos ejemplos son *Simple Mesh* y *PG Standard Cell Rails*.

Cabe mencionar que el comando de `compile_pg`, puede ser utilizado sin el argumento de `-strategies` para compilar todas las estrategias que ya están generadas. Al especificar una

estrategia es posible tener un mayor control sobre las estrategias que se están implementando en el *layout*.

Con esto se finaliza la creación de la red de alimentación y tierra para el *chip* y se puede proceder con el posicionamiento de las celdas dentro del *core*. Para esto, se utilizó la sección de *Placement* en el *Task Assistant*. En la opción de *Create Placement*, es posible especificar varias opciones. En la Fig. 28 se puede ver en azul los parámetros que se cambiaron. En este caso, se generaron los siguientes comandos:

```
set_app_options -name place.coarse.fix_hard_macros -value false
set_app_options -name plan.place.auto_create_blockages -value auto
create_placement -floorplan -timing_driven -congestion -effort high -
congestion_effort high
```

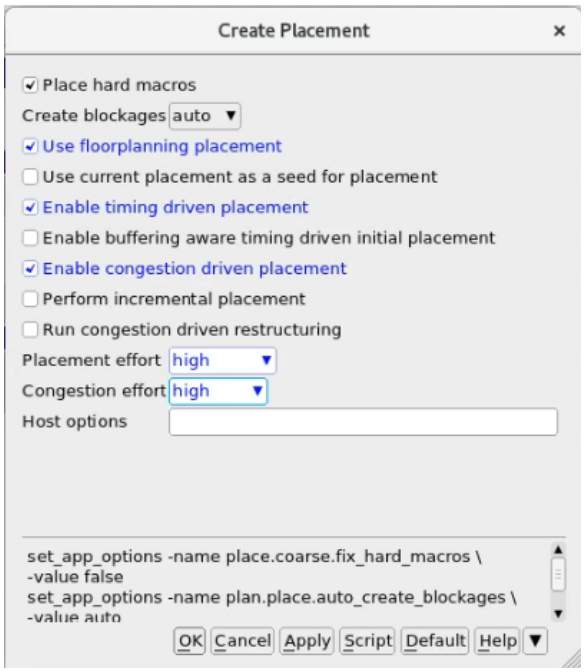

Figura 28: Asistente de **ICC 2** para la colocación de celdas en el *core*

Los comandos ejecutados cambian algunas opciones de **ICC 2**. Estas son para que realice el *placement* de celdas de tipo *hard macro* y para que genere áreas de bloqueo de ruteo de forma automática. Finalmente, se le indica que genere el *placement* en tomando en cuenta el *floorplan*, el *timing* de las señales y la congestión dentro del *core*. Finalmente, se le indica que haga un mayor esfuerzo para crear el *placement* y tomar en cuenta la congestión. El resultado de eso, se puede ver en la Fig. 29. En este resultado se puede ver que las celdas no concuerdan con los rieles creado, sin embargo, esto es debido a que el comando de `create_placement` únicamente las ubica de manera general.

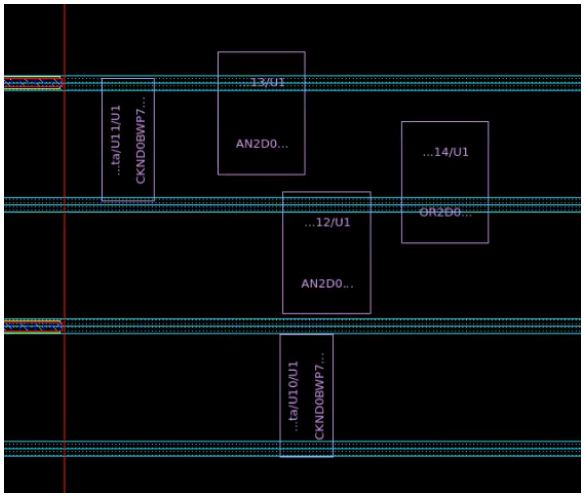

Figura 29: Grupo de celdas colocadas luego de correr `create_placement`

Para finalizar el *placement*, es necesario legalizarlo para que haga una colocación de las celdas más detallada. Esto se hace con el siguiente comando:

```
legalize_placement
```

El resultado se puede ver en la Fig. 30.

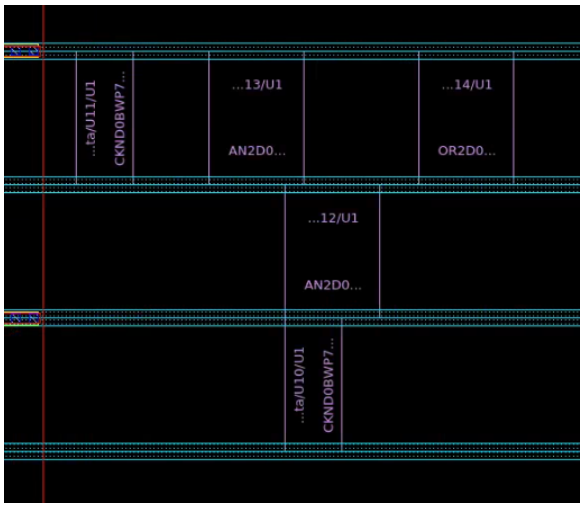

Figura 30: Mismo grupo de celdas de Fig. 29 luego de correr `legalize_placement`

Ya con las celdas ubicadas dentro del *core*, es posible realizar el ruteo. Para esto, es necesario realizar dos verificaciones antes de poder interconectar las celdas. El primero es utilizado para poder determinar si existen problemas que puedan impactar el desempeño de los comandos para realizar ruteo. En este caso, se quiere verificar si se tiene algún puerto bloqueado. El segundo se usa para poder realizar un *mega-check* de distintos parámetros del diseño. De encontrar un error, este se mostrará en **ICC 2**. Finalmente, se realiza el ruteo automático de las celdas. El resultado del ruteo se puede observar en la Fig. 31. Estos comandos fueron obtenidos a partir del *Task Assistant* en la sección de *Routing*.

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

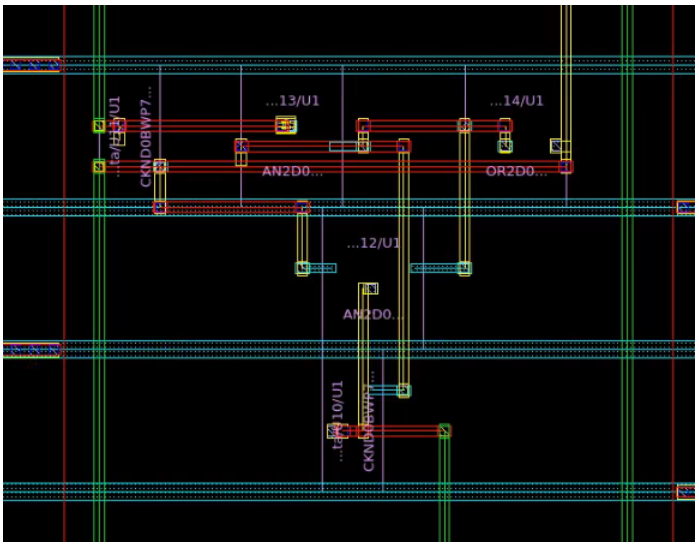

Figura 31: Detalle de ruteo para un grupo de celdas en **ICC 2**

Luego del ruteo, se procede a generar los *fillers* dentro de todo el *die*. Esto quiere decir que se colocarán *fillers* tanto en *core* como en el anillo de IO. Esto se hace utilizando los siguientes comandos:

```
create_io_filler_cells -io_guides [get_io_guides {anillo_IO_XOR.top
anillo_IO_XOR.right anillo_IO_XOR.left anillo_IO_XOR.bottom}] -
reference_cells {PFILLER1 PFILLER5 PFILLER05 PFILLER0005 PFILLER10
PFILLER20}
create_stdcell_fillers -lib_cells [get_lib_cells {TSMCWorkspace|
FillersWorkspace/FILL64BWP7T TSMCWorkspace|FillersWorkspace/FILL32BWP7T
TSMCWorkspace|FillersWorkspace/FILL16BWP7T TSMCWorkspace|FillersWorkspace
/FILL8BWP7T TSMCWorkspace|FillersWorkspace/FILL4BWP7T TSMCWorkspace|
FillersWorkspace/FILL2BWP7T TSMCWorkspace|FillersWorkspace/FILL1BWP7T}]
```

El primer comando es utilizado para la creación de *fillers* para el anillo de entradas y salidas. En este caso, se le indican los *IO guides* en donde se van a generar los *fillers* además de los *fillers* a utilizar. La diferencia entre estos *fillers* es el tamaño que tienen, por lo que se decidió incluir todos los que se tenían en las librerías. De igual forma, se crean los *fillers* para el *core*. Para este comando también se indicaron todos los *fillers* que estaban en la librería. Los resultados de los comandos se pueden observar en las figuras 32 y 33.

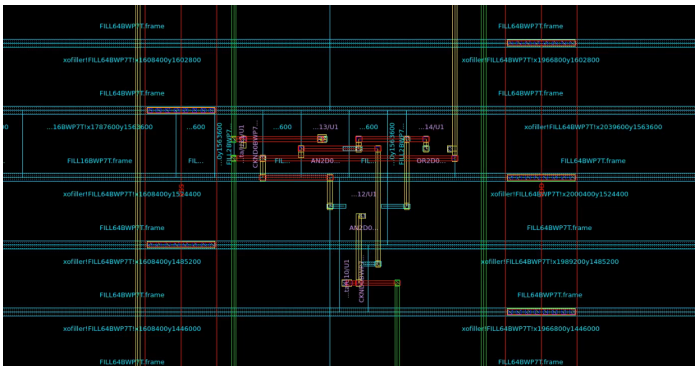

Figura 32: Detalle de los *fillers* en el core

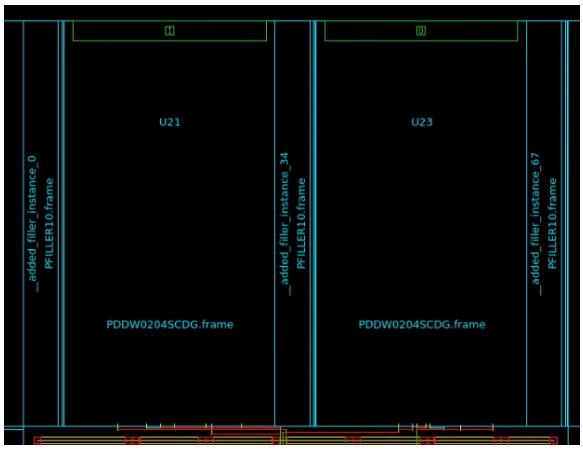

Figura 33: Detalle de los *fillers* en el *IO ring*

Luego de eso, es necesario correr nuevamente el comando para conectar los pines de VDD y VSS. Además de eso, se eliminan los *fillers* que pueden estar violando alguna regla. Finalmente, se realiza una verificación de la legalidad del *placement* que se ha realizado.

```
connect_pg_net -automatic
remove_stdcell_fillers_with_violation
check_legality
```

El comando de `check_legality` es utilizado para poder verificar las siguientes características dentro del *layout* [18]:

- Celdas que no estén en las filas.
- Celdas que se encuentren encima de otras.
- Celdas que se encuentren encima de áreas de *blockage*.
- Celdas con la orientación incorrecta.
- Celdas que no concuerden con la definición de *site*.

- Celdas con violaciones de *straps* de alimentación.
- Celdas con violaciones de tecnologías avanzadas (procesos de 20nm y menores).
- Celdas con violaciones de espaciamiento.

Con esto, se llega a un *layout* finalizado para el circuito. Finalmente, es necesario guardar la librería creada. Esto se hace de la siguiente manera:

```
save_block nombre_libreria.ndm:nombre_topcell
```

Se le debe de indica que se desea guardar el *topcell* en la librería que se generó. El *topcell* es el módulo en el archivo de *verilog* que contiene todas conexiones de las celdas y los pads de entradas y salidas generados por la síntesis física. Este genera un directorio `.ndm` en donde se está trabajando que contiene el *layout* generado.

Al igual que en **LM**, es posible correr *scripts* dentro de **ICC 2**. Esto se hace con un archivo de extensión `.tcl` que se puede cargar en la interfaz gráfica y luego ejecutar. Un *script* de ejemplo utilizado en la síntesis física de un circuito se muestra en el anexo 16.2. Este *script* posibilita la automatización de la síntesis física de un circuito ya que tampoco es necesario abrir la interfaz gráfica del **ICC 2**.

CAPÍTULO 9

Generación del **GDS**

Para la creación de un archivo `.gds` se encontró el comando `write_gds` en **ICC 2** para poder realizar esto. Este comando, en su forma más sencilla, toma el bloque que estamos trabajando y genera un archivo `.gds` con el nombre que se le indique. En **ICC 2** es posible observar todas las opciones que tiene este comando, estas se muestran en la Fig. 34. Para nuestro caso, se utilizó también el argumento de `lib_cell_view` para que utilizara la vista *frame* con las que se generaron las librerías **NDM** utilizadas para la síntesis. Además de esta opción, también es necesario seleccionar la opción de `all` para el argumento de `hierarchy`. Este comando se utiliza para poder indicarle a **ICC 2** que se quieren tomar las celdas de diseño y también las librerías de referencia que se utilizaron. El comando que se genera para crear el archivo es el siguiente:

```
write_gds -library nombre_libreria.ndm -design nombre_diseno -hierarchy
all -lib_cell_view frame nombre_layout.gds
```

Esto resultó en un archivo `.gds` con el que intentamos ejecutar las verificaciones de **LVS**, **ERC** y antena. Los resultados de estas validaciones fueron exitosas como se pueden observar en la sección 11.

A pesar de poder realizar la generación del archivo por medio de **ICC 2**, también se utilizó *Custom Compiler* para crear el `.gds`. Para poder realizar este procedimiento es necesario cargar los archivos **PDK** que provee **TSMC**. Estos archivos se encuentran en la carpeta de **TSMC** siguiente ruta:

```
TSMC/180/CMOS/G/IO3.3V/pdk
```

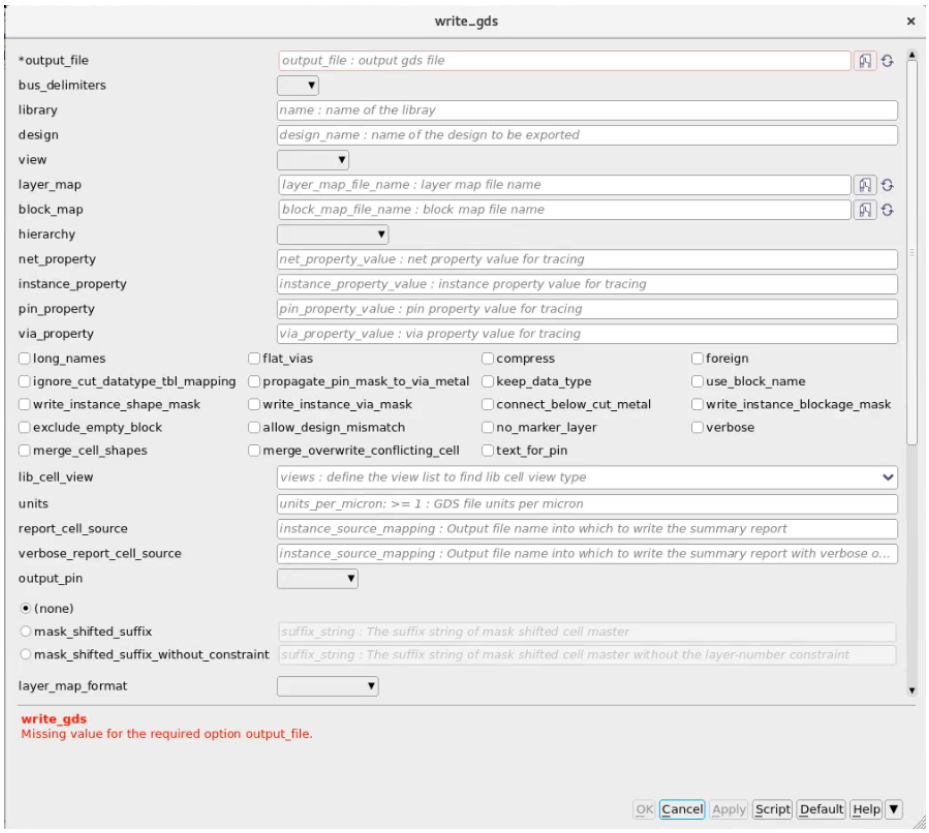

Figura 34: Opciones para el comando `write_gds`

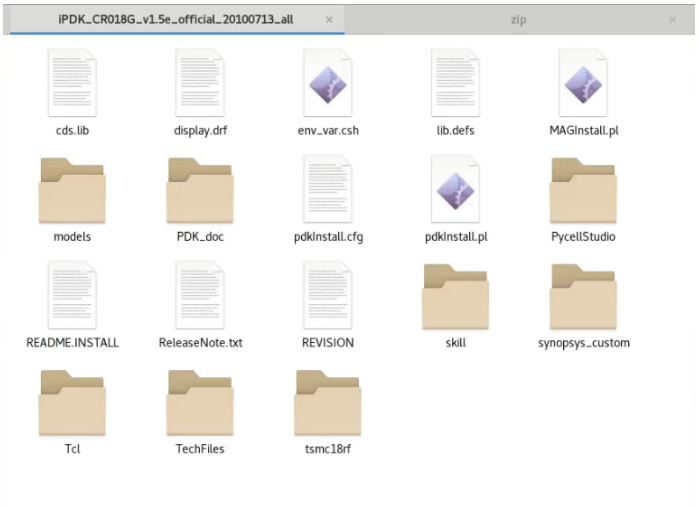

Figura 35: Carpeta final del **PDK** de **TSMC**

Dentro de esta carpeta habrán varios archivos comprimidos. Todos pueden ser descomprimidos, sin embargo, la carpeta de interés es la llamada "`T-018-MM-SP-001-W1_1_5E.zip`". Dentro de esta, también se tienen más carpetas comprimidas que tiene tiene que ser descomprimidas. Se recomienda que todos estos cambios se hagan sobre una copia de estos

archivo. La carpeta final debe de verse similar a la que se muestra en la Fig. 35. Con todos los archivos descomprimidos, se debe de ejecutar el comando `custom_compiler` para abrir la herramienta con la librería **PDK** cargada. *Custom Compiler* deberá de tener cargada una librería cargada en la sección de *Library Manager* con el nombre de "tsmc18ref".

Luego de haber cargado estos archivos, se procede a importar a *Custom Compiler* la librería **NDM** generada en **ICC 2**. Para hacer esto, se debe ir a la opción de *File* → *Add ICC 2 Library*. Esto abre una ventana en donde se debe de especificar la librería **NDM** que se quiere agregar. La ventana se puede ver en la Fig. 36. Es importante que en la pestaña de *Attributes* se le indique que se debe de cargar la librería utilizando como *Tech Library* la que se tiene disponible por el **PDK**. Luego de esto, se puede ver la vista de *layout* generada en **ICC 2**. Esta se verá similar a la que se muestra en la Fig. 37.

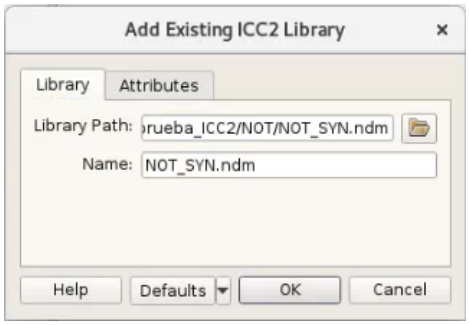

Figura 36: Ventana para cargar el archivo **NDM** a *Custom Compiler*

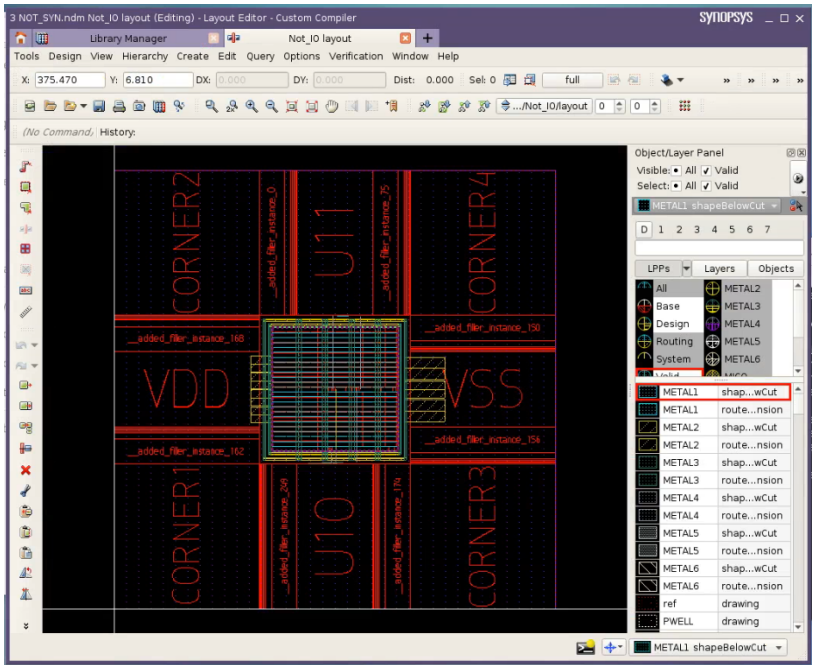

Figura 37: Vista de *layout* en *Custom Compiler*

Con la librería **NDM** cargada, se puede crear el archivo `.gds`. Esto se hace regresando al la pestaña de *Library Manager* y yendo a la opción de *File* → *Export* → *Stream...*. Este abre

una ventana en donde se le debe indicar cuál es la librería que se quiere exportar, la celda de esta librería y el nombre del archivo de salida. La ventana mostrada se verá de forma similar a la de la Fig. 38. El resultado de esto será un archivo `.gds` dentro de la carpeta en donde se está trabajando.

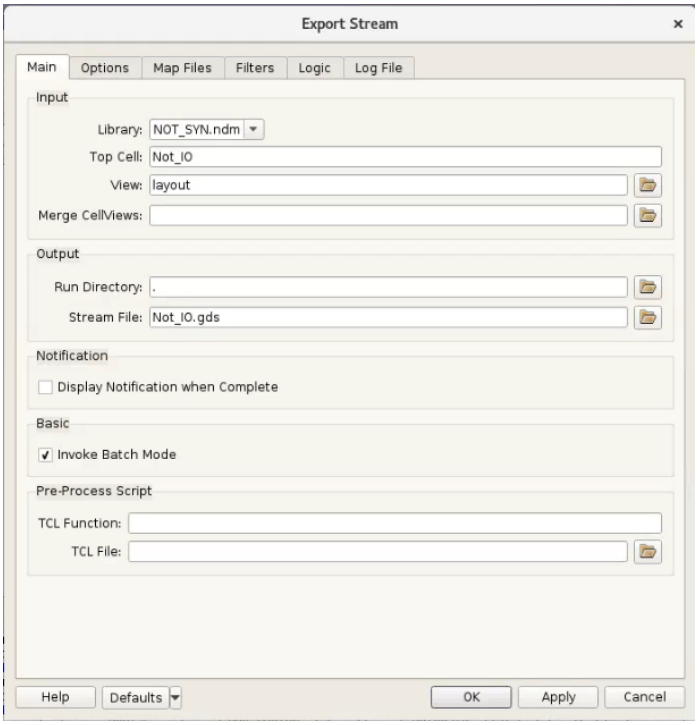

Figura 38: Opciones para exportar la librería a un `.gds`

A pesar de poder utilizar *Custom Compiler*, se sugiere utilizar **ICC 2** para continuar con los trabajos ya que de esta manera se eliminar la necesidad de introducir al flujo un programa distinto. La otra ventaja es que se pueden utilizar *scripts* en **ICC 2** para automatizar el proceso, lo que no es posible en *Custom Compiler*.

CAPÍTULO 10

Proceso de *Electrical Rule Check*

Para realizar el *Electrical Rule Check*, son necesarios los archivos `.gds`, `.icv` y el *runset* como se explica en el trabajo [5]. Se realizaron los cambios que se indicaron el *runset* además de modificar las rutas en donde ser encuentran los archivos `.gds` y `.icv`. Este cambio es importante ya que este *runset* es el mismo que se utiliza para realizar el **LVS**, por lo que las rutas de los archivos no serán iguales. Para ubicar esta sección del *runset* se puede utilizar `ctrl-f` y buscar "ENVIRONMENT SETUP". En esta misma sección se encuentran las líneas a descomentar para habilitar las opciones de **ERC**.

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

library(
        library_name = "/home/nanoelectronica2021/Documentos/prueba_erc/FA/FA.gds",
        cell = "fulladd_io",
        format = GDSII
);

SCHEMATIC_TOPCELL : string = "fulladd_io";              // Set schematic top cell name here
sch_db = schematic(
        schematic_file = {{"/home/nanoelectronica2021/Documentos/prueba_erc/FA/FA.icv", ICV}},
        schematic_library_file = {{"/usr/synopsys-old/TSMC/SCRIPTS_NUEVOS/20191128-124344/unit.cdl", SPICE}},
        expand_multiple_devices = true,
        spice_settings = {slash_is_space = false}
);
//#define USER_EQUIV_FILE                       // Turn on for user-specified equivalent point file flow
#ifdef USER_EQUIV_FILE
#include "./user.equiv"                         // Set equivalent point file here
#endif
```

Figura 39: Rutas dentro del *runset*

Además de cambiar la ruta de estos archivos, también se puede notar que se debe de indicar el nombre de la *top cell*. Además de eso, también es necesario el archivo `.cdl` que se encuentra en la ruta mostrada en la Fig. 39.

```
/* EDIT: The following section contains all of the runset variables for RC extraction tools. */
//#define RC_DECK                         // Turn on for LPE/RC extraction
//#define CROSS_REFERENCE                 // Turn on for source cross reference in LPE extraction
//#define ZERO_NRS_NRD                    // Turn off when this deck would calculate NRS and NRD
//#define FILTER_DGS_TIED_MOS             // Turn on to filter MOS with D, G and S tied together (default filter MOS with all pins tied)

#define WELL_TO_PG_CHECK                  // Default is on. Turn on to highlight if nwell connects to ground or psub connects to power.
#define GATE_TO_PG_CHECK                  // Default is off. Turn on to highlight if a mos gate directly connects to power or ground.
#define PATH_CHECK                        // Default is off. Turn on to highlight if
                                          // (1) nodes have a path to power but no path to ground
                                          // (2) nodes have a path to ground but no path to power
                                          // (3) nodes have no path to power or ground
                                          // (4) nodes have no path to any label net
#define DS_TO_PG_CHECK                    // Default is on. Turn on to highlight if drain connects to power and source connects to ground.
#define FLOATING_GATE_CHECK                     // Default is on. Turn on to highlight if there are floating gates.
#define FLOATING_WELL_CHECK               // Default is on. Turn on to highlight if well does not connect to power or ground.
                                          // The nwell of moscaps and nwell-resistor are excluded
//#define NW_RING                         // Turn on to enable NW ring to separate the node from BULK
```

Figura 40: *Runset* con las opciones de **ERC** habilitadas

Con estos cambios realizados en el *runset* de **LVS**, es posible correr el mismo comando mostrado en la sección 6.4.2 para llamar al programa *IC Validator* para realizar el **ERC**. Luego de haber hecho esto, se generan los archivos mostrados en la Fig. 41. En el caso en el que la verificación haya sido exitosa, el archivo `.RESULTS` tendrá un encabezado como el que se muestra en la Fig. 42. En el caso en el que no se tenga uno un resultado satisfactorio, se podrá leer un *FAIL* para el resultado del **LVS** y un *NOT CLEAN* para el **DRC** y extracciones.

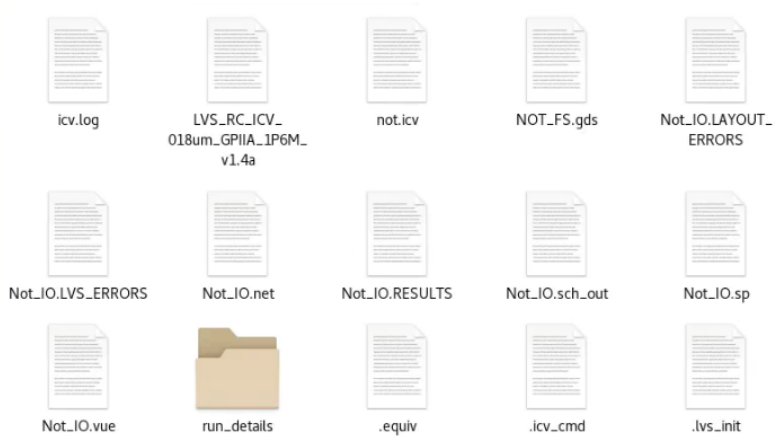

Figura 41: Archivos creados luego de ejecutar **LVS** y **ERC**

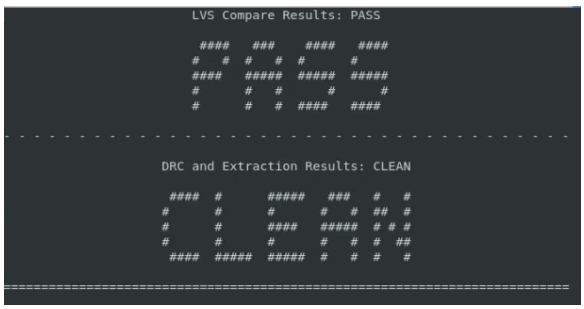

Figura 42: Resultado exitoso para **LVS** y **ERC**

CAPÍTULO 11

Resultados de la síntesis física y **ERC**

11.1. Compuerta NOT

Esta fue la primera compuerta que pasó el proceso de síntesis física en **ICC 2** y la validación **ERC**. Se decidió iniciar con un circuito sencillo para poder conocer de mejor manera el entorno de **ICC 2**. Debido a esto, esta fue la compuerta en la que se tuvo la mayor parte de descubrimiento en el proceso de traducción de comandos de **ICC** a **ICC 2**.

11.1.1. Síntesis física de la compuerta NOT

Lo primero que se realizó fue leer la documentación que se *Synopsys* provee para **ICC 2** y **LM**. En especial, se utilizó la guía de implementación [16], la guía de comandos [18], la guía de *design planning* [20] y la guía de implementación de *Library Manager* [15]. Toda esta documentación se puede encontrar en la página de Solvnet.

Al leer estos manuales, se encontró la necesidad de utilizar la herramienta de **LM** para poder generar las librerías **NDM** para poder iniciar el flujo de la síntesis física. Los primeros intentos para generar las librerías **NDM** a partir de los archivos proveídos por **TSMC** no fueron exitosos.

En [15] se indicaba un procedimiento para poder obtener la librería física a partir de las librerías *Milkyway*. Debido a que en las iteraciones pasadas se utilizaron estas librerías, se optó por intentar el método que se explicaba en el manual. Este consistía en utilizar **ICC** para generar un archivo comprimido en el cual se encontraban los archivos `.lef` necesarios

para la creación de la **NDM**. Se siguieron los pasos expuestos, sin embargo, al momento de hacer el *check workspace* se mostraba un a gran cantidad de errores. Este procedimiento se intentó varias veces con ligeras alteraciones en los comandos utilizados, sin embargo, no se obtuvieron los resultados esperados.

Luego de esos intentos fallidos, se prosiguió a investigar de manera más detenida la carpeta con todos los archivos de las librerías de **TSMC**. Dentro de esta, se pudieron encontrar los archivos necesarios para la librería **NDM** en las rutas mostradas en el Cuadro 2. Con estas, se generó la primera librería que contenía las celdas estándar de **TSMC**.

Después de generar esta primera librería, se inició el proceso de traducción de comandos de **ICC**. El siguiente problema que surgió durante esta síntesis fue que no se habían generado las librerías para los *pads* de entradas y salidas, por lo que no aparecían estas celdas luego de cargar el *verilog*. Esto se solucionó al momento de generar la otra librería necesaria para los *pads*. El siguiente problema que surgió fue al momento de crear los *corners*. Este se solucionó de manera similar en donde se creó la librería *physical only*.

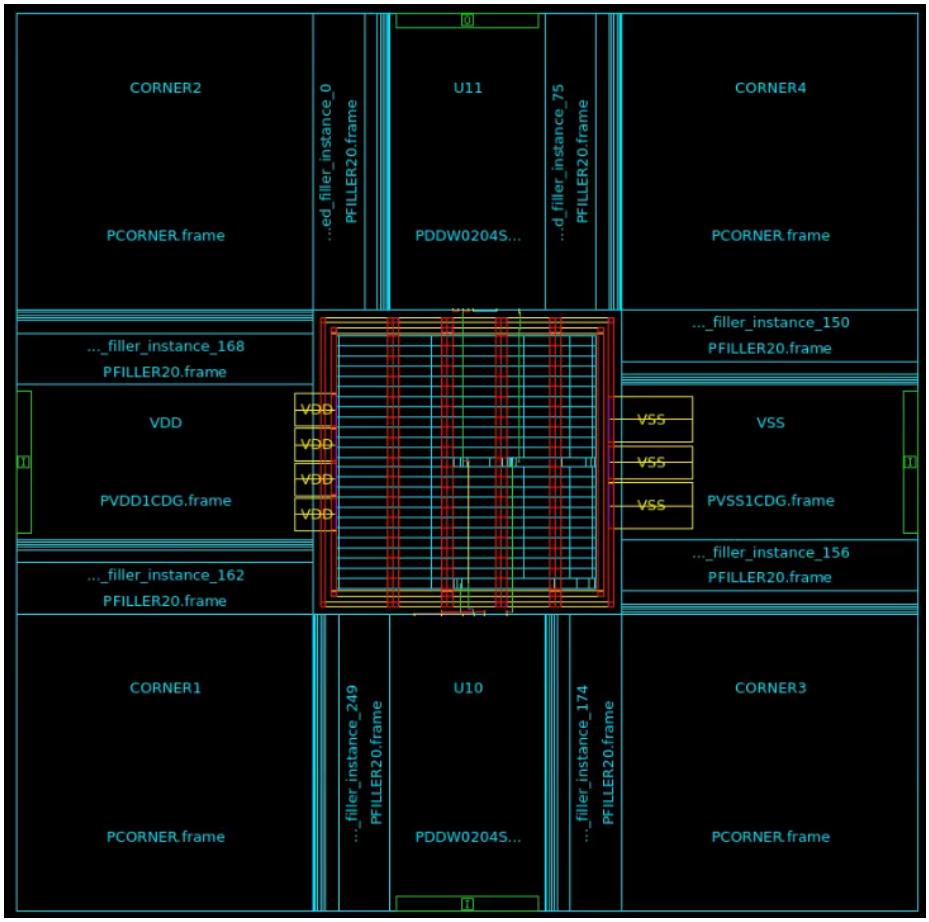

Figura 43: *Layout* final para la compuerta NOT

El siguiente problema que surgió fue la creación de los patrones para alimentación y tierra. En este caso, se encontró la opción de *Task Assistant*. Fue posible generar las conexiones necesarias para VDD y VSS utilizando esta herramienta. La principal complicación de esta etapa fue poder generar los distintos patrones sin que estos tuvieran inconvenientes

en sus conexiones. Aprovechando que los *pads* de alimentación se encontraban a los lados, se crearon los *straps* verticales de tal manera que se conectaran al lado superior e inferior del anillo de VDD y VSS.

El *placement* y ruteo de los componentes se pudo realizar sin ningún inconveniente con ayuda del *Task Assistant* y el manual de comandos [18].

Finalmente, al momento de generar los fillers del *core* tampoco se habían creado las librerías *physical only* que las contenían. Estas se crearon y se pudo finalizar el proceso. Estas fueron las cuatro librerías necesarias para la creación del layout de este circuito y los siguientes.

El resultado de la síntesis física de este circuito se puede observar en la Fig. 43.

11.1.2. ERC de la compuerta NOT

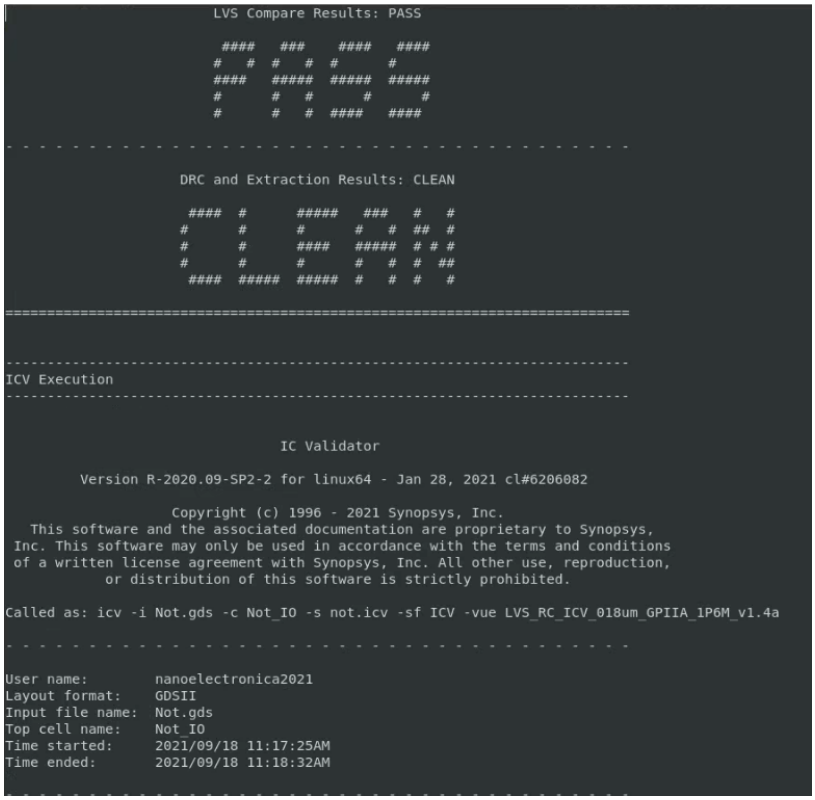

Figura 44: Resultado de **ICV** para la compuerta NOT

Luego de obtener los archivos necesario del proceso de **LVS**, se pudo correr la prueba de **ERC**. El resultado de esta prueba fue exitoso como se puede ver en la Fig. 44. No se tuvieron problemas al momento de correr las pruebas de **ERC**.

11.2. Compuerta XOR

El segundo circuito a probar para la síntesis física fue una compuerta XOR, sin embargo, esta compuerta estaba compuesta de compuertas AND, OR y NOT. Este fue un siguiente paso para utilizar más entradas y salidas para el circuito además de tener más celdas dentro del *core* del *chip*.

11.2.1. Síntesis física de la compuerta XOR

Para este circuito, se utilizó el mismo *script* generado para la compuerta NOT, sin embargo, surgió el problema con el comando para colocar los pads de entradas y salidas. Este colocaba los *pads* en ubicaciones aleatorias, por lo que no se sabía en que lado iban a estar. Es por esto, que se encontró el comando para colocar asociar los *pads* a un *IO guide* en específico que se muestra en la sección 8.2. Esto es importante por la manera en la que se hacen las conexiones de estos *pads* al anillo de VDD y VSS. El resultado final de esta síntesis física se puede observar en la Fig. 45. Se puede notar que dentro del *core* se tienen más celdas.

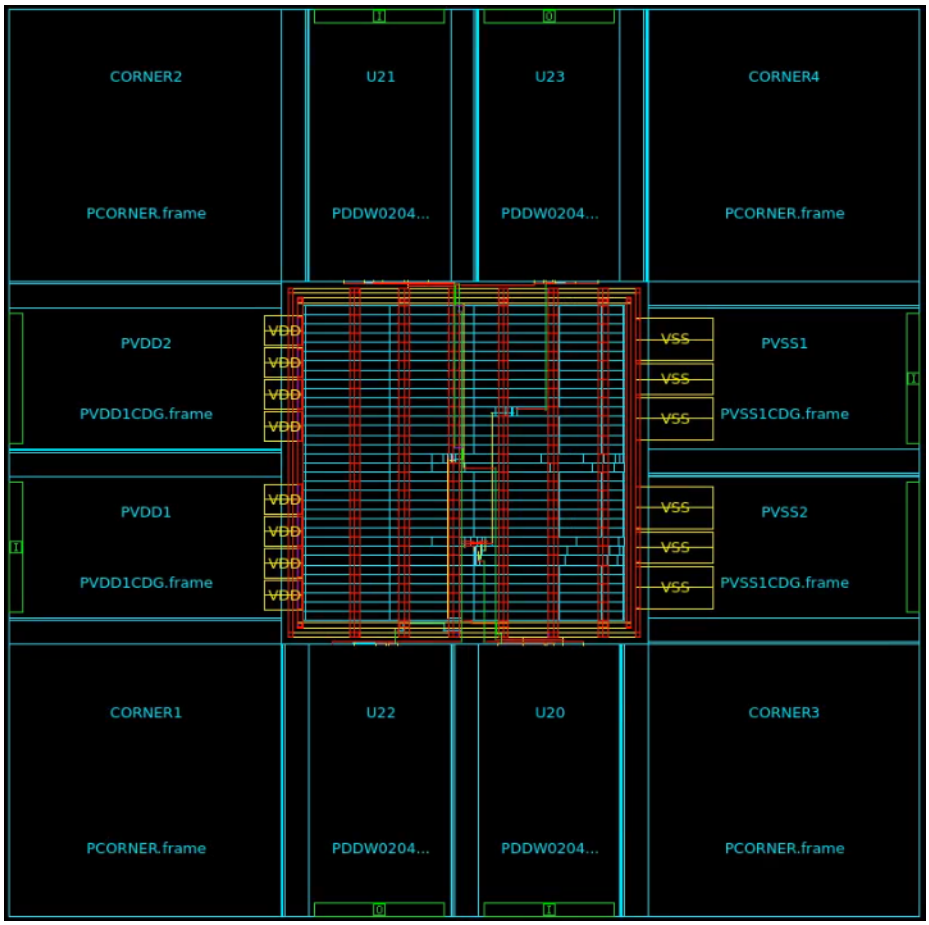

Figura 45: *Layout* final para la compuerta XOR

11.2.2. ERC de la compuerta XOR

De forma similar a la NOT, el proceso de **ERC** para la XOR fue exitoso. El resultado se puede ver en la Fig. 46. La única diferencia en este proceso de **ERC** es el archivo del *runset* de **LVS** en donde se instancias las otras celdas utilizadas. Sin embargo, dado que primero la celda pasa por el proceso de **LVS**, no fue necesario realizar los cambios.

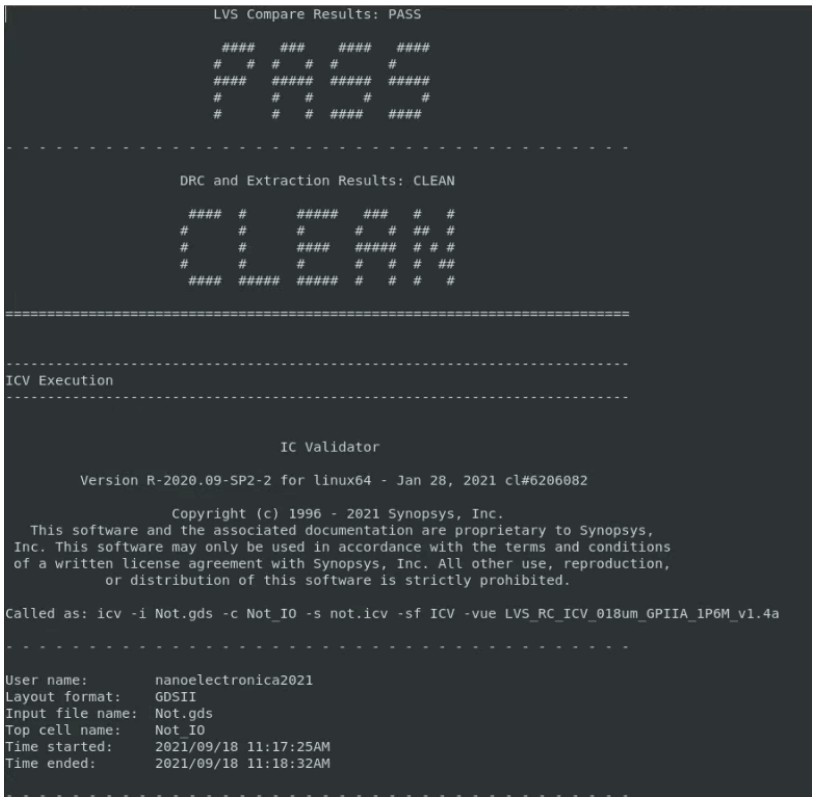

Figura 46: Resultado de **ICV** para la compuerta XOR

11.3. Circuito *full adder*

El siguiente circuito que fue sometido al proceso de síntesis física fue un *full adder* de 4 bits. Este circuito aumentó la cantidad de entradas y salidas que se tenían además de agregar más compuertas al *core*, incrementando la complejidad del circuito a sintetizar.

11.3.1. Síntesis física del circuito *full adder*

Para el proceso de la síntesis física, se pudo observar que el tamaño del *chip* que se estaba utilizando no era suficiente para poder acomodar todos los *pads* de entrada y salida. Para poder corregir esto, fue necesario ajustar el tamaño del *floorplan* que se estaba utilizando. Este se hizo más grande y se cambiaron las medidas para el patrón del *mesh* de VDD y VSS. El resultado del *full adder* se puede observar en la Fig. 47. Se puede observar que

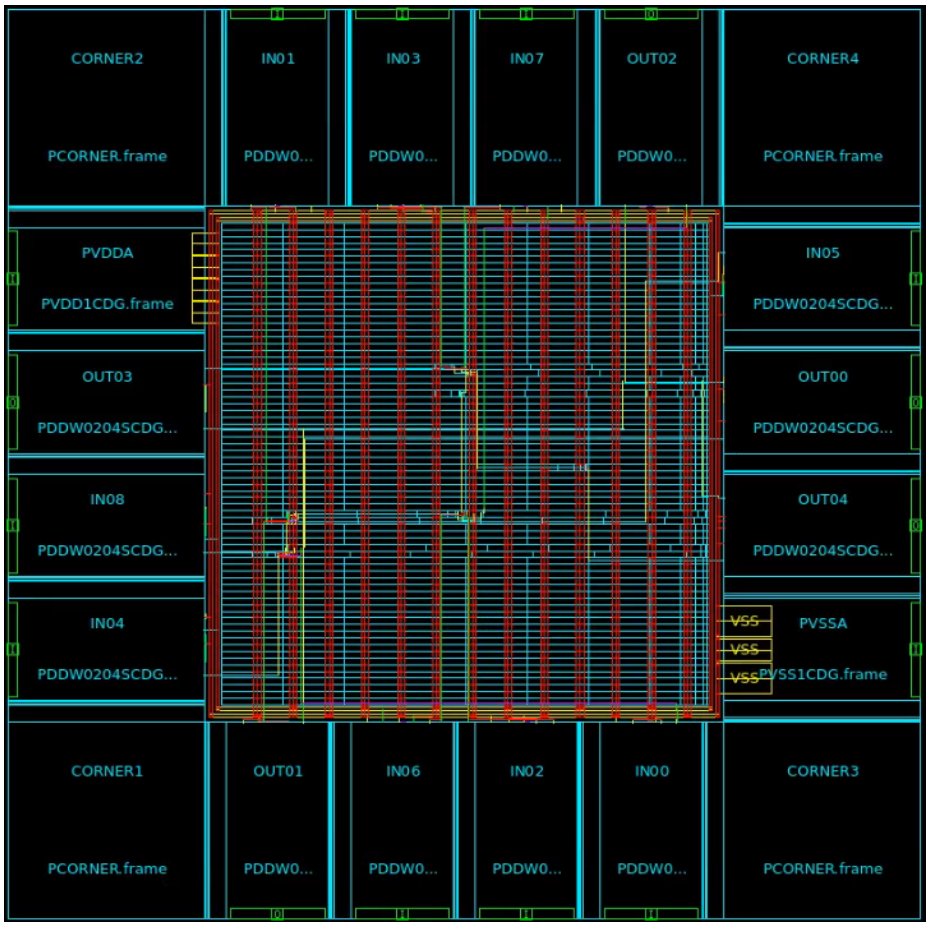

Figura 47: *Layout* final para el *full adder*

efectivamente el *chip* tiene mayores dimensiones, haciendo posible colocar 4 *pads* en cada lado del chip.

11.3.2. ERC del circuito *full adder*

Para el **ERC** de este circuito no se tuvieron problemas, generando un resultado exitoso al ejecutar el comando de **ICV**. El resultado se puede ver en la Fig. 48.

11.4. Circuito ALU

La **ALU** de 4 bits que se sintetizó era era capaz de realizar la siguientes operaciones:

- Suma
- Resta
- Multiplicación

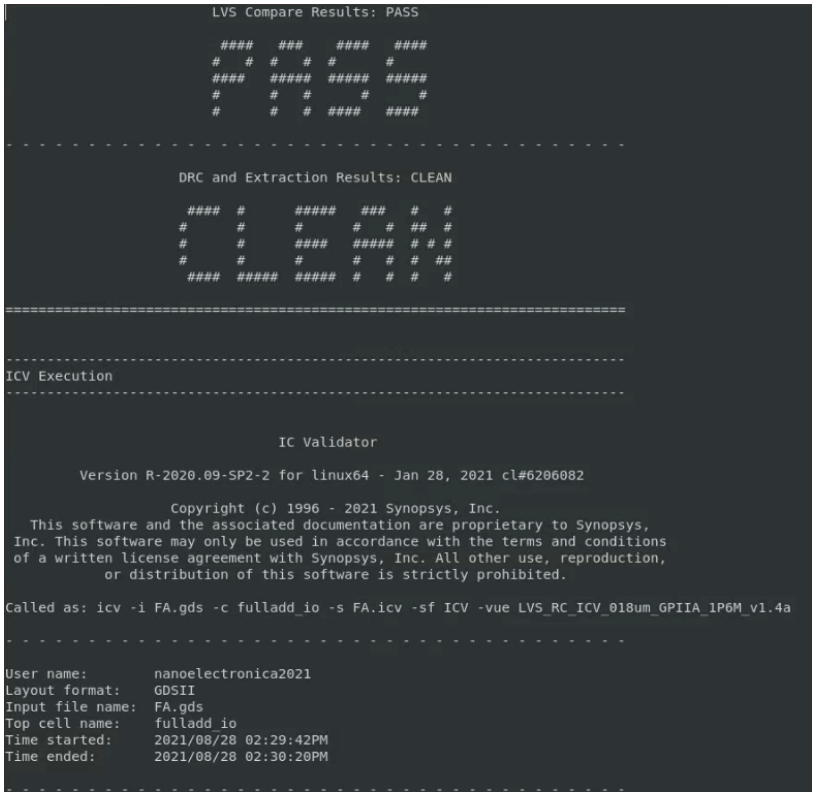

Figura 48: Resultado de **ICV** para el *full adder*

- AND
- OR
- NOT
- XOR
- XNOR

Para este circuito, se incrementó de manera considerable la cantidad de celdas presentes dentro del *core* para poder realizar todas las operaciones propuestas.

11.4.1. Síntesis física del circuito ALU

Para esta síntesis física, se incrementó el tamaño nuevamente ya que tiene una mayor cantidad de entradas y salidas. Al momento de correr el *script*, se obtuvo el resultado que se puede ver en la Fig. 49. En este *layout* se puede ver que todas las celdas se concentran en el centro del *core*. Debido a la cantidad de celdas por todas las operaciones y condiciones necesaria para el funcionamiento de la **ALU**, se necesita una gran cantidad de celdas e interconexiones. A diferencia de los otros *layouts* generados, es posible ver el ruteo a simple vista. En la Fig. 50, se puede ver el *core* con mayor detalle.

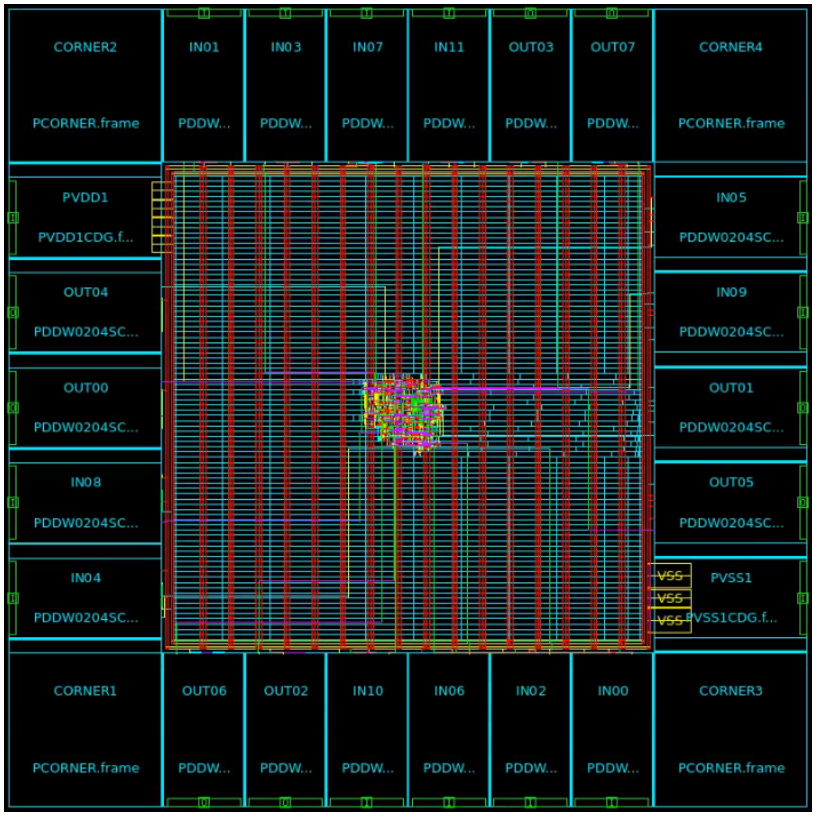

Figura 49: *Layout* final para la **ALU** de 4 bits

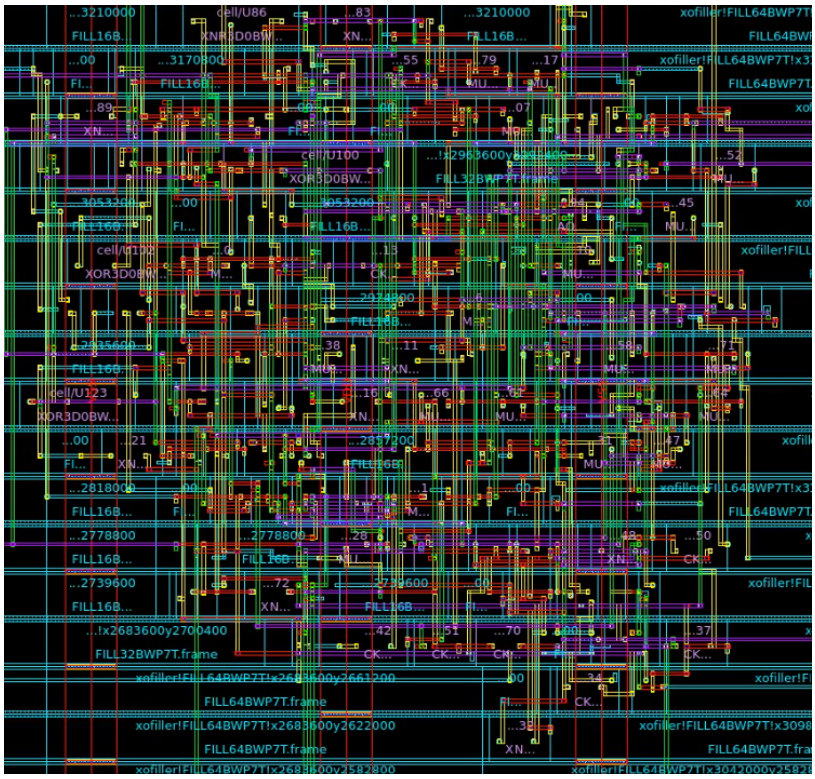

Figura 50: Detalle del *core* para la **ALU** de 4 bits

11.4.2. ERC del circuito ALU

El resultado de la verificación para este circuito se puede ver en la Fig. 51. Nuevamente, el resultado obtenido es satisfactorio.

Figura 51: Resultado de **ICV** para la **ALU** de 4 bits

11.5. Circuito contador de 4 bits

El siguiente circuito a probar fue un contador de 4 bits. A diferencia de los otros circuito sintetizados, este es un circuito con lógica secuencial. Esto significa que dentro del *chip* se tendrá una señal de *clock* además de la presencia de *flip-flops* dentro del core.

11.5.1. Síntesis física del circuito contador de 4 bits

La síntesis de este circuito es ligeramente distinta debido a la presencia de una señal de reloj dentro del circuito. En el *Task Assistant* existe una sección para poder sintetizar circuitos con *clocks*. Para realizar esto, se ejecutaron los siguientes comandos luego de colocar las celdas pero antes de realizar el ruteo de los componentes:

```
check_clock_trees -clocks clk
check_design -checks pre_clock_tree_stage
```

```
synthesize_clock_trees -clocks clk -postroute -routed_clock_stage
detail_with_signal_routes
clock_opt -list_only
check_design -checks cts_qor
```

El primer comando se encarga de revisar las conexiones de la señal de reloj `clk` para determinar si no se tienen problemas con el *netlist*, restricciones de *timing*, restricciones de la señal de reloj, restricciones de ruteo y restricciones adicionales que se hayan configurado. El segundo comando también realiza una verificación del diseño antes de haber realizado las conexiones de la señal de reloj dentro del diseño.

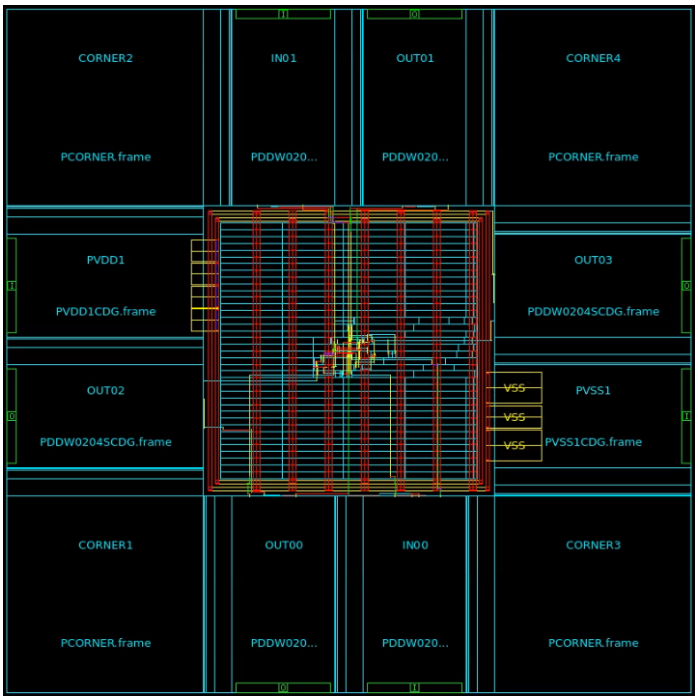

Figura 52: *Layout* final para el contador de 4 bits

Luego de realizar las verificaciones, es posible sintetizar la señal de reloj. El tercer comando se encarga de generar las conexiones del la señal `clk`. La opción de `-postroute` se encarga de realizar un a verificación luego de haber ruteado las señales que se le indiquen en la opción `-routed_clock_stage`. Esta última opción puede encargarse de rutear solamente el clock si su argumento es `detail` o realizar todas las conexiones si el argumento es `detail_with_signal_routes`.

Con el cuarto comando, se le indica a **ICC 2** que se desea optimizar la señal de reloj. Finalmente, se realiza una última verificación del circuito luego de haber ruteado las señales. El resultado final de la síntesis física se puede observar en la Fig. 52, mientras que en la Fig. 53 se puede ver el detalle de las celdas en el *core*.

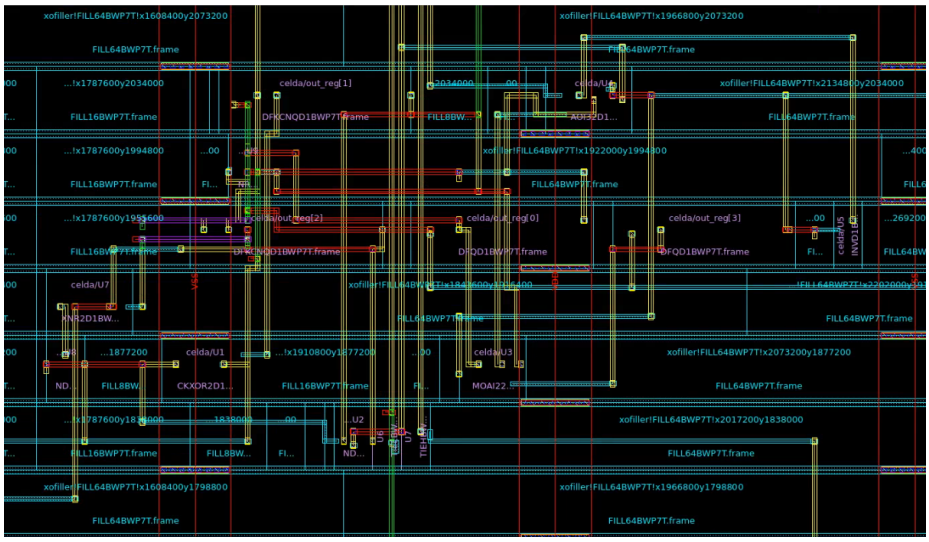

Figura 53: Detalle del *core* para el contador de 4 bits

11.5.2. ERC del circuito contador de 4 bits

A pesar de que el contador de 4 bits se un circuito secuencial, no se tuvieron problemas con la verificación de *ERC*. Los resultados se pueden consultar en la Fig. 54. Es posible ver que el resultado fue exitoso para este circuito.

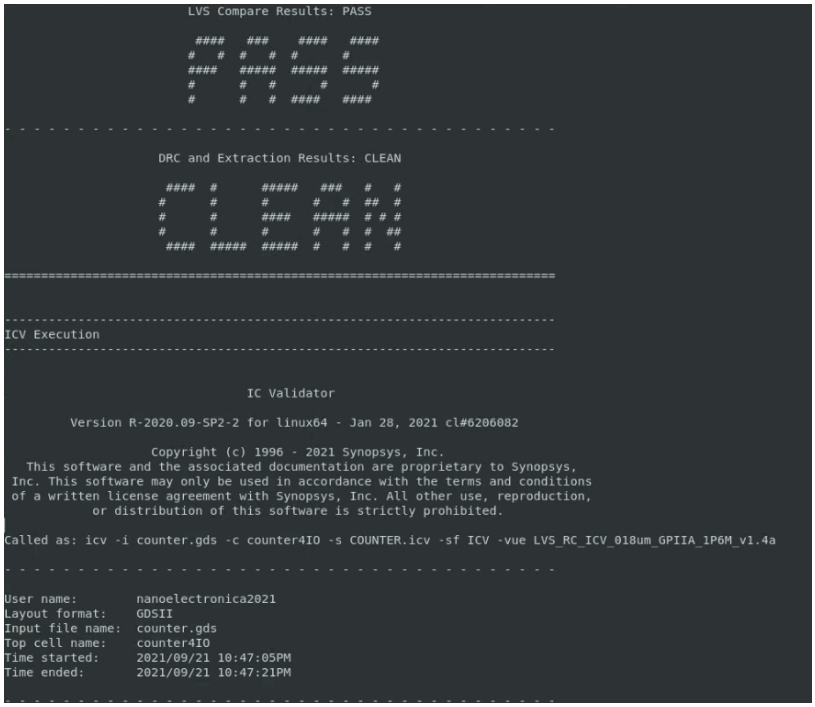

Figura 54: Resultado de **ICV** para el contador de 4 bits

11.6. El Gran Jaguar

El Gran Jaguar es el nombre que se le dio al chip diseñado por el grupo de trabajo de la UVG. La función de este circuito es mostrar un mensaje codificado en **ASCII** en 8 pines distintos de salida. Esta función fue implementada por medio de un contador, que va incrementando con cada flanco de reloj, que indica la letra que se tiene que estar transmitiendo. Este circuito luego será conectado a un microcontrolador que será capaz de interpretar estas señales en paralelo y transmitirlos a una computadora para que el mensaje pueda ser leído en voz alta por medio de algún programa *text-to-speech*. El texto que se definió para este circuito fue el siguiente:

> "Hola, soy El Gran Jaguar, el primer nanochip elaborado en Centroamerica por una Universidad. Desarrollado completamente en la Universidad del Valle de Guatemala por alumnos graduandos de Ingeniería Electrónica entre 2019 y 2021"

En el texto se pueden observar palabras sin acentos, esto es debido a que se utilizó codificación **ASCII** para poder representar los caracteres a enviar.

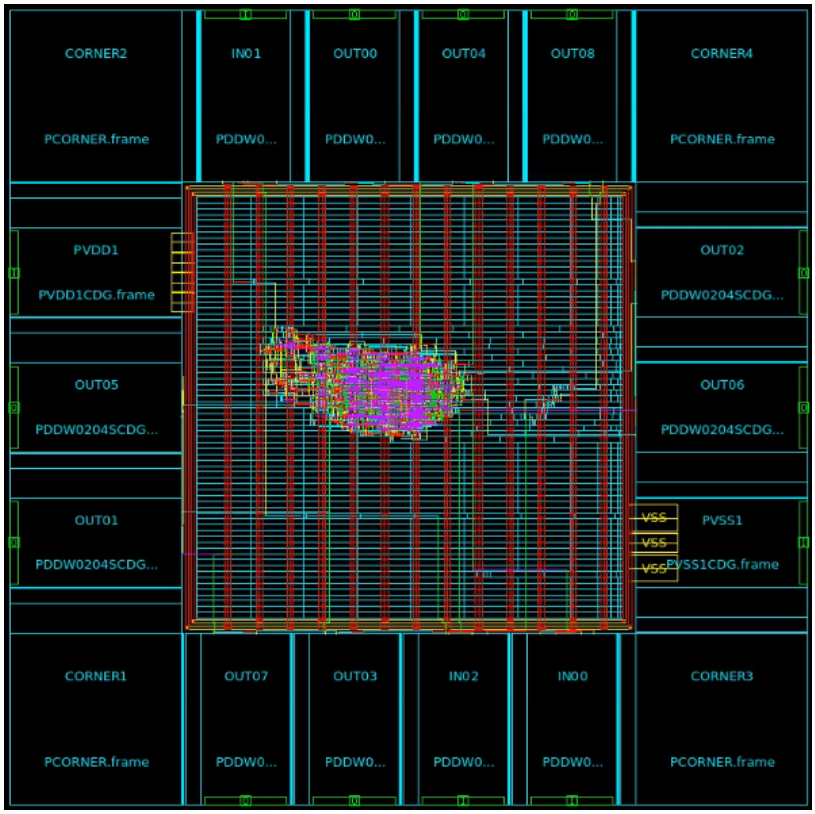

Figura 55: *Layout* final para la primera versión de El Gran Jaguar

Además de poder generar la cadena de texto, también se incluyó un *ring oscillator* dentro del diseño para generar una señal cuadrada.

11.6.1. Primera síntesis física de El Gran Jaguar

Para la síntesis física de El Gran Jaguar, se utilizaron de base los scripts utilizados para la síntesis de circuitos con señales de reloj. Para este caso se generó el *layout* mostrado en la Fig. 55. En esta se puede apreciar que el ruteo del diseño es mucho más complejo debido a la función que debe de realizar el chip. En la Fig. 56 se puede apreciar de mejor manera todas las conexiones que se realizan en el *core*. Se puede notar que la cantidad de salidas es de 9 debido a los 8 bits de para los caracteres y la salida del oscilador, mientras que de entradas se tienen las señales de reloj, *enable* y *reset* además de VDD y VSS.

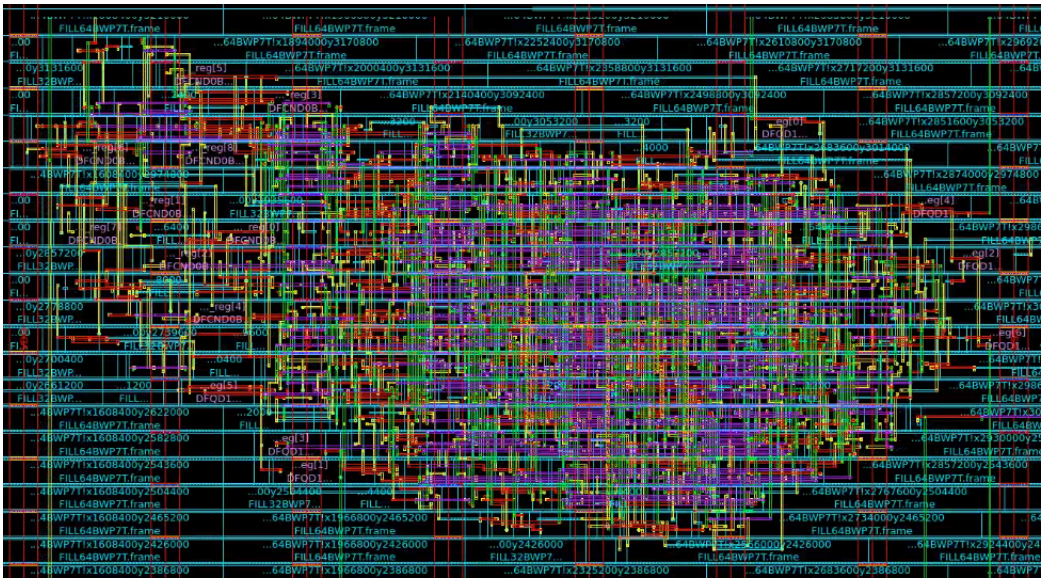

Figura 56: Detalle del *core* para la primera versión de El Gran Jaguar

11.6.2. Segunda síntesis física de El Gran Jaguar

Dado que se observó que aún se tenía espacio dentro del core del circuito, se procedió a complicar el diseño que se tenía planteado en un inicio. Para esta segunda iteración, se agregaron pines para poder seleccionar el texto a enviar. Se agregó el siguiente mensaje en donde se da crédito a todas las personas que estuvieron involucradas en este proyecto:

> "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"

El resultado de esta síntesis y el detalle del *core* se puede observar en las Fig. 57 y 58. Se puede notar que las celdas ocupan una mayor área del diseño a comparación de la primera versión del circuito.

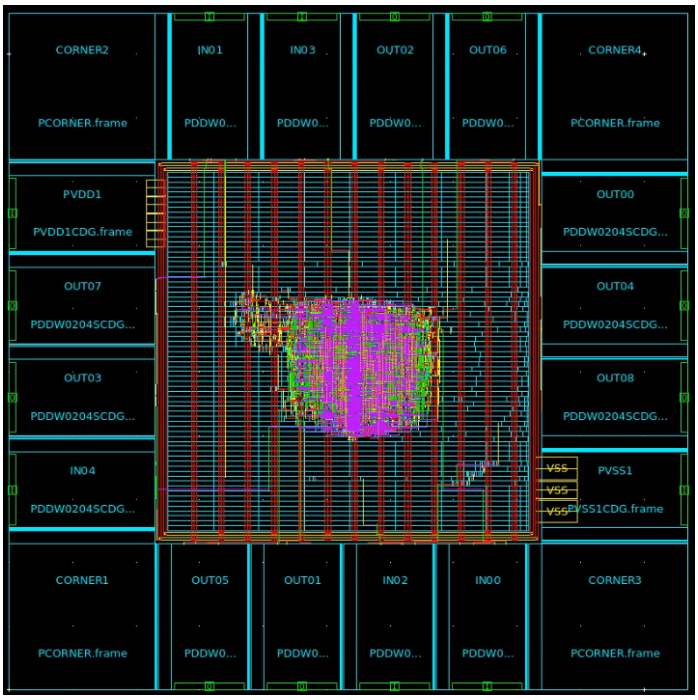

Figura 57: *Layout* final para la segunda versión de El Gran Jaguar

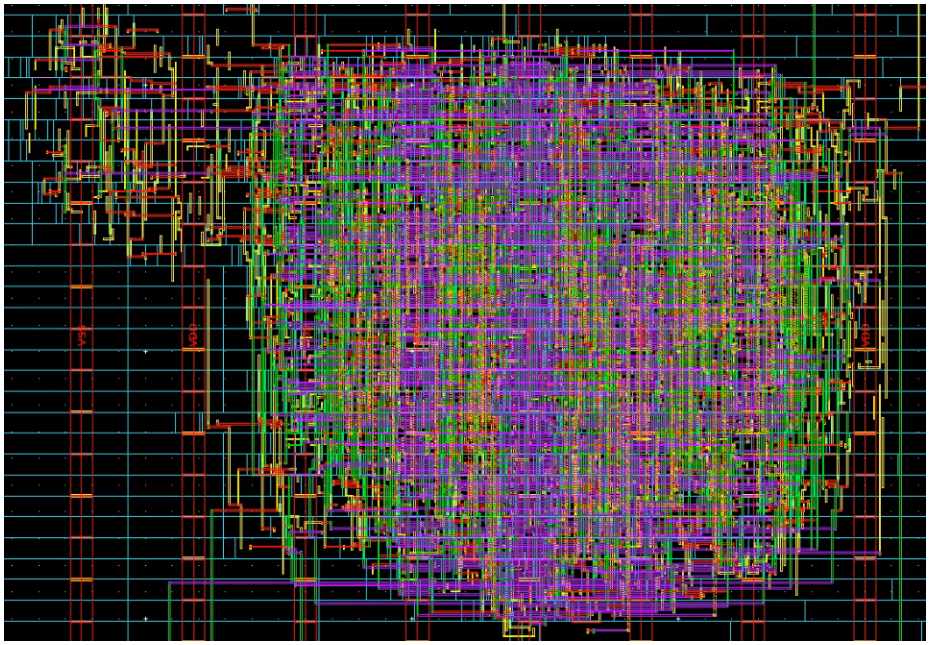

Figura 58: Detalle del *core* para la segunda versión de El Gran Jaguar

11.6.3. ERC de El Gran Jaguar

La prueba de **ERC** solamente se corrió para la segunda versión del diseño. A pesar de ser un circuito secuencial más complejo, la verificación de **ERC** fue satisfactoria como se

observa en la Fig. 59

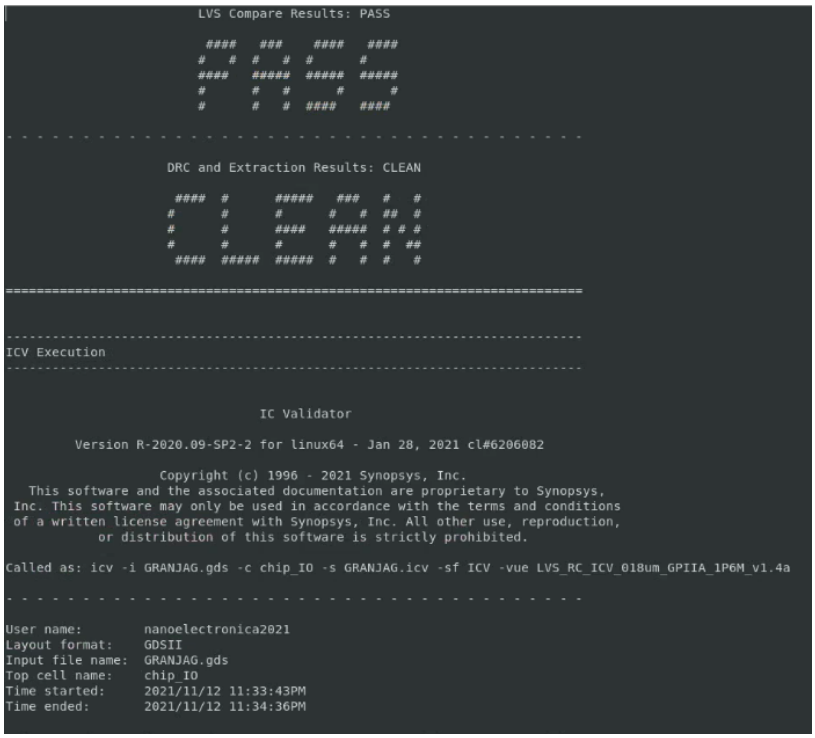

Figura 59: Resultado de **ICV** para El Gran Jaguar

CAPÍTULO 12

Otras verificaciones

Además de tener que cumplir con el **ERC**, las síntesis físicas realizadas debían de pasar satisfactoriamente las verificaciones de **DRC**, *antenna* y **LVS**. Estas verificaciones ayudaron a depurar el proceso de la síntesis realizada para disminuir la cantidad de errores que se generaban en cada uno. A pesar de que este trabajo no contempla el proceso de estas verificaciones, es necesario saber los resultados que se obtienen para poder mejorar el resultado de la síntesis física.

Se pudo encontrar que **ICC 2** tiene las capacidades de poder correr las verificaciones de **DRC** directamente sobre la librería que se está trabajando, por lo que no era necesario generar un `.gds`. Generar este archivo fue necesario únicamente para las validaciones que dependían de **ICV**.

12.1. *Design Rule Check*

Los resultados de los primeros *layouts* generados tenían una gran cantidad de errores. Estos eran debido a varios problemas que habían en el proceso de la síntesis física.

Los primeros errores que se encontraron fueron que se generaban vías encima de otras, causando que las reglas de **DRC** no se cumplieran. Estos se generaban en las conexiones entre los *pads* de alimentación y algunos *straps* del *mesh*. Fue por esto que se decidió dejar únicamente los *straps* verticales y los *pads* a los lados.

El siguiente error que se generaba eran que los *fillers* no se encontraban conectados a VDD y VSS. Inspeccionando las celdas de los *fillers* utilizados dentro del *core* fue posible

detectar que estos necesitaban estar conectados a los rieles de alimentación. Es por esto que se tuvo que realizar la conexión de estas celdas con el comando `connect_pg_net` luego de generar los *fillers*.

Esto disminuyó la cantidad de errores de forma considerable, dejando solamente errores de densidad que se muestran en la Fig. 60. Estos son debido a que no se está llegando al requerimiento mínimo de densidad de metal en todas las capas en las que se está trabajando el *chip*. Para corregir esto, se encontró en el *Task Assistant* una opción para poder generar metales *dummy* para llegar a este requisito. Sin embargo, para poder correr este comando se necesita un *runset* que no se tiene disponible. Este solicitó a **IMEC**, pero no se ha obtenido repuesta hasta el momento de finalizar este trabajo.

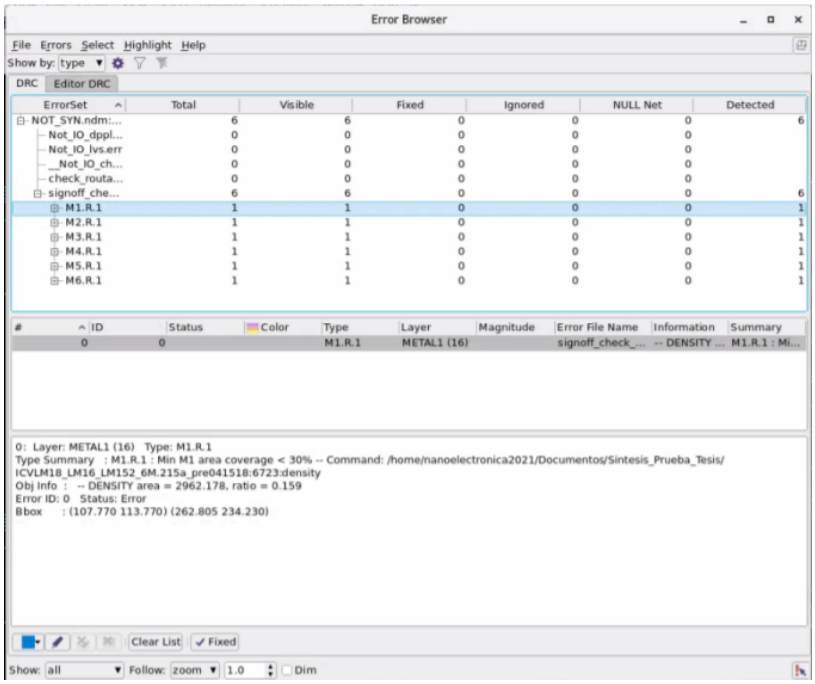

Figura 60: Errores de densidad luego de ejecutar el **DRC**

12.2. *Antenna Rule Check*

Los resultado de las verificaciones de antena fueron exitosos. Con los distintos circuitos sintetizados, fue posible correr el *runset* asociado al *Antenna Rule Check* para determinar si el *layout* generado cumplía con los requisitos. Un ejemplo de una de estas verificaciones exitosas se puede observar en la Fig. 61. Dado que esta validación no presentó errores, no fue necesario hacer cambios en el proceso de la síntesis física.

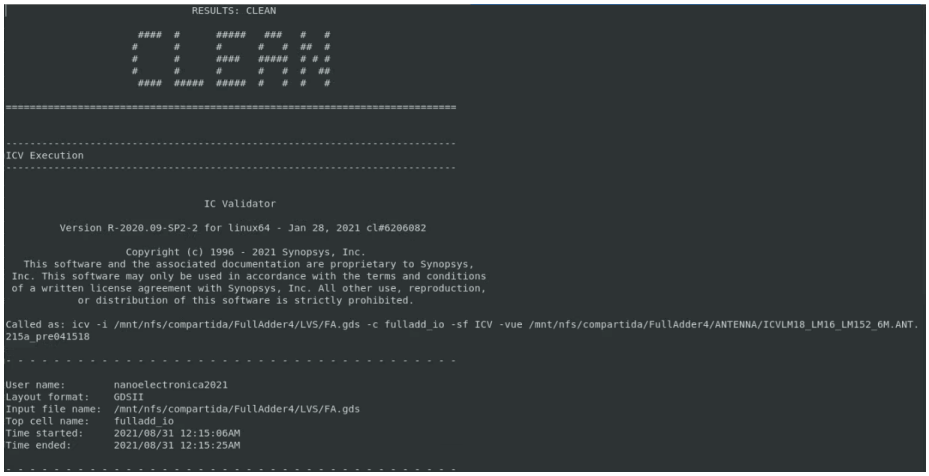

Figura 61: Resultados de una verificación de antena

12.3. *Layout Versus Schematic*

Los resultados de **LVS** para los circuitos sintetizados fueron satisfactorios. Uno de los resultados de esta prueba se puede ver en la Fig. 62. Además de eso, es posible ver que todos los circuitos funcionan ya que para poder correr la prueba de **ERC** se utiliza el mismo *runset* de **LVS**, por lo que todos los resultados mostrados en la sección 11.

Figura 62: Resultado de una verificación de *LVS*

capítulo 13

Conclusiones

- Se mejoró el proceso de la síntesis física al tener un proceso más robusto que funcionó con circuitos de distinta complejidad, tanto de naturaleza combinacional y secuencial.
- Fue posible realizar la migración de comandos de *IC Compiler* a *IC Compiler II* luego de haber leído la documentación de las herramientas para encontrar los equivalentes en la nueva herramienta utilizada para la síntesis física.
- Se pudo generar un archivo **NDM** y `.gds` para los circuitos por medio de *IC Compiler II* y se pudo eliminar la necesidad de utilizar *Custom Compiler* como parte del flujo de diseño.
- Para las verificaciones de **ERC** se obtuvieron los mismos resultados que en trabajos pasados, con una validación de reglas eléctricas exitosa para todos los circuitos trabajados.
- Los *layouts* generados por la síntesis física también tuvieron resultados exitosos para las verificaciones de **DRC**, antena y **LVS**, indicando que los resultados del proceso fueron correctos.
- Se crearon *scripts* para la generación de librerías **NDM** necesarias para el trabajo y para el proceso de síntesis física de un circuito para poder automatizar el proceso completo para el diseño de un *chip*.

capítulo 14

Recomendaciones

- Apoyarse de la documentación de las herramientas a utilizar para familiarizarse con las opciones que provee *IC Compiler II* y *Library Manager* para la síntesis física. De especial interés son los manuales [15], [16], [18] y [20].
- Continuar investigando los comandos que se pueden utilizar en *IC Compiler II* ya que el *script* creado aún necesita de ajustes manuales para obtener los tamaños a utilizar en el *chip*. Se encontraron comandos que eran capaces de obtener propiedades de las geometrías del *chip*, sin embargo, estos no regresaban los resultados esperados.
- Realizar pruebas con circuitos secuenciales más complejos ya que en estos es necesaria la síntesis de un reloj dentro del diseño para que dicte el cambio de los *flip-flops*. Puede que para circuitos más grandes surjan complicaciones que no existieron durante este trabajo.
- Mantener comunicación con todos los grupos involucrados en el proyecto. Esto es necesario para poder conocer el estado de las distintas etapas del flujo para la creación de un *chip* y comunicar los resultados pertinentes a cada integrante para poder proceder de manera eficaz. Además de la comunicación entre integrantes del grupo de trabajo de la **UVG**, también es necesario poder comunicarse con **IMEC** para poder obtener archivos que puedan ser necesarios para el proyecto.

CAPÍTULO 15

Bibliografía

[1] 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.* Tesis de licenciatura en Ingeniería Electrónica, Universidad del Valle de Guatemala, 2014.

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

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

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

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

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

[7] N. Weste y D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Estados Unidos: Addison-Wesley, 2011.

[8] Synopsys, *What is Design Rule Checking (DRC)? – Types of DRC*. dirección: `https://www.synopsys.com/glossary/what-is-design-rule-checking.html`.

[9] H. Kaeslin, *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Estados Unidos: Cambridge University Press, 2008.

[10] Synopsys, *What is Programmable Electrical Rules Checking?* Dirección: https://www.synopsys.com/glossary/what-is-programmable-electrical-rules-checking.html.

[11] H. Hegazy, *Programmable electrical rule checking*, oct. de 2008. dirección: https://www.eetimes.com/programmable-electrical-rule-checking/.

[12] N. Kaul, *A short introduction to IC Compiler II*, oct. de 2014. dirección: https://www.techdesignforums.com/practice/technique/guide-ic-compiler-ii/.

[13] Synopsys, *IC Validator Physical Verification*, Datasheet, 2019.

[14] J. R. Girón, *Etapa de verificación física de Diseño en Silicio vs. Esquemático (LVS) en el flujo de diseño para un chip a nanoescala*, Tesis de licenciatura en Ingeniería Electrónica, Universidad del Valle de Guatemala, 2020.

[15] Synopsys, *Library Manager User Guide. Version S-2021.06, June 2021*, Estados Unidos, 2021.

[16] ——, *IC Compiler II Implementation User Guide. Version S-2021.06, June 2021*, Estados Unidos, 2021.

[17] ——, *IC Compiler II Graphical User Guide User Guide. Version S-2021.06, June 2021*, Estados Unidos, 2021.

[18] ——, *IC Compiler II Tool Commands. Version S-2021.06, June 2021*, Estados Unidos, 2021.

[19] ——, *IC Compiler II Application Options and Attributes. Version S-2021.06, June 2021*, Estados Unidos, 2021.

[20] ——, *IC Compiler II Design Planning User Guide. Version S-2021.06, June 2021*, Estados Unidos, 2021.

capítulo 16

Anexos

16.1. Script para la creacion de librerías

```
#Importar el tech file
create_workspace -flow normal -technology usr/synopsys/TSMC/180/CMOS/G/
    stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/
    tcb018gbwp7t_270a/techfiles/tsmc018_6lm.tf NormalWorkspace

#Importar los db (no incluimos los IO)
read_db { usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/
    TSMCHOME/digital/Front_End/timing_power_noise/NLDM/tcb018gbwp7t_270a/
    tcb018gbwp7tbc.db usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/
    tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/timing_power_noise/NLDM/
    tcb018gbwp7t_270a/tcb018gbwp7tlt.db usr/synopsys/TSMC/180/CMOS/G/stclib
    /7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/
    timing_power_noise/NLDM/tcb018gbwp7tml.db usr/synopsys/TSMC/180/CMOS/G/
    stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/
    timing_power_noise/NLDM/tcb018gbwp7ttc.db usr/synopsys/TSMC/180/CMOS/G/
    stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/
    timing_power_noise/NLDM/tcb018gbwp7twc.db usr/synopsys/TSMC/180/CMOS/G/
    stclib/7-track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Front_End/
    timing_power_noise/NLDM/tcb018gbwp7twcl.db }

#Importar el LEF
read_lef /usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/tcb018gbwp7t_290a_FE/
    TSMCHOME/digital/Back_End/lef/tcb018gbwp7t_270a/lef/tcb018gbwp7t_6lm.lef

#Correr y desplegar el check
```

```
current_workspace; check_workspace
gui_create_window -type MessageBrowserWindow
open_ems_database check_workspace.ems

#Commit y save a la libreria
current_workspace NormalWorkspace; commit_workspace  -output
    StandardWorkspace.ndm

#Creamos el ndm de los pads
create_workspace -flow normal -technology /home/nanoelectronica2021/
    Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf PadsWorkspace
read_db { /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/
    tcb018gbwp7t/LM/tpd018nvtc.db }
read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
    tpd018nv_6lm.lef
current_workspace; check_workspace
current_workspace PadsWorkspace; commit_workspace  -output PadsWorkspace.ndm

#Creamos el ndm para los corners
create_workspace -flow physical_only -technology /home/nanoelectronica2021/
    Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf CornersWorkspace
read_db { /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/
    tcb018gbwp7t/LM/tpd018nvtc.db }
read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
    tpd018nv_6lm.lef
current_workspace; check_workspace
current_workspace CornersWorkspace; commit_workspace  -output
    CornersWorkspace.ndm

#Creamos el ndm para los fillers std cells
create_workspace -flow physical_only -technology /home/nanoelectronica2021/
    Documentos/tcb018gbwp7t_290a_FE/tf/tsmc018_6lm.tf PhysicalOnlyWorkspace
read_lef /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/lef/
    tcb018gbwp7t_6lm.lef
read_db { /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/
    tcb018gbwp7t/LM/tcb018gbwp7tbc.db /home/nanoelectronica2021/Documentos/
    tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/tcb018gbwp7tlt.db /home/
    nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/
    tcb018gbwp7tml.db /home/nanoelectronica2021/Documentos/
    tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/tcb018gbwp7ttc.db /home/
    nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/
    tcb018gbwp7twc.db /home/nanoelectronica2021/Documentos/
    tcb018gbwp7t_290a_FE/tcb018gbwp7t/LM/tcb018gbwp7twcl.db }
current_workspace; check_workspace
current_workspace PhysicalOnlyWorkspace; commit_workspace  -output
    FillersWorkspace.ndm

#Integramos los 4 NDM
```

```
create_workspace -flow aggregate TSMCWorkspace
read_ndm /mnt/nfs/compartida/ASA/LibreriasNDM/CornersWorkspace.ndm; read_ndm
     /mnt/nfs/compartida/ASA/LibreriasNDM/FillersWorkspace.ndm; read_ndm /mnt
    /nfs/compartida/ASA/LibreriasNDM/PadsWorkspace.ndm; read_ndm /mnt/nfs/
    compartida/ASA/LibreriasNDM/StandardWorkspace.ndm;
current_workspace; check_workspace
current_workspace TSMCWorkspace; commit_workspace  -output TSMCWorkspace.ndm
```

16.2. Script para la síntesis física

```
#Creamos la libreria que vamos a usar prueba.ndm (por el momento hay un
    warning que no sabemos si nos afecta -ver link library-)
create_lib XOR_SYN2.ndm -technology /usr/synopsys/TSMC/180/CMOS/G/stclib/7-
    track/tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/
    tcb018gbwp7t_270a/techfiles/tsmc018_6lm.tf \
 -ref_libs /mnt/nfs/compartida/ASA/LibreriasNDM/TSMCWorkspace.ndm

#Abrimos el verilog file sintetizado
read_verilog /mnt/nfs/compartida/XOR2/sintesis_logica/sintesis_cell_io/
    salidas/out_xor_io.v

read_sdc -echo -syntax_only /mnt/nfs/compartida/XOR2/sintesis_logica/
    sintesis_cell_io/salidas/out_xor_io.sdc

#Importamos las TLU+ y el map
read_parasitic_tech -tlup /home/nanoelectronica2021/Documentos/
    tcb018gbwp7t_290a_FE/tluplus/t018lo_1p6m_typical.tluplus \
-layermap /home/nanoelectronica2021/Documentos/tcb018gbwp7t_290a_FE/tluplus/
    star.map_6M

#Limpiamos las cosas de PG
remove_pg_strategies -all

#Agregamos las reglas de antenna
source -echo -verbose "/usr/synopsys/TSMC/180/CMOS/G/stclib/7-track/
    tcb018gbwp7t_290a_FE/TSMCHOME/digital/Back_End/milkyway/tcb018gbwp7t_270a
    /clf/antennaRule_018_6lm.tcl"

#Creacion de corners
create_cell {CORNER1 CORNER2 CORNER3 CORNER4} PCORNER

#Creacion de pads para  VDD y VSS
create_cell {PVDD1 PVDD2} PVDD1CDG
create_cell {PVSS1 PVSS2} PVSS1CDG

#Creacion de nets de VDD y VSS
```

```
resolve_pg_nets
create_net -power VDD
create_net -ground VSS
connect_pg_net -net VDD [get_pins -physical_context *VDD]
connect_pg_net -net VSS [get_pins -physical_context *VSS]
connect_pg_net -automatic
report_cells -power

#Floorplan inicial
initialize_floorplan -site_def unit -use_site_row -keep_all -side_length
    {135 135} -core_offset {125}

#Creacion del anillo IO
create_io_ring -name anillo_IO_XOR -corner_height 115

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

#Creacion del anillo de PG
create_pg_ring_pattern ring_pattern -horizontal_layer METAL2    -
    horizontal_width {2} -horizontal_spacing {2}    -vertical_layer METAL3 -
    vertical_width {2} -vertical_spacing {2}
set_pg_strategy core_ring    -pattern {{name: ring_pattern}    {nets: {VDD
    VSS}} {offset: {1 1}}} -core
compile_pg -strategies core_ring

#creamos conexion IO ejemplo 2
create_pg_macro_conn_pattern hm_pattern -pin_conn_type scattered_pin -layers
     {METAL2 METAL3} -nets {VDD VSS} -pin_layers {METAL2}
set_app_options -name plan.pgroute.treat_pad_as_macro -value true
set_pg_strategy macro_conn -macros [get_cells {PVDD* PVSS*}] -pattern {{name
    : hm_pattern} {nets: {VDD VSS}}}
set_pg_strategy_via_rule macro_conn_via_rule -via_rule { { { {strategies:
    macro_conn}} { {existing: all} {layers: METAL3} } {via_master: default} }
     {{intersection: undefined}{via_master: NIL}}}
compile_pg -strategies macro_conn -via_rule macro_conn_via_rule -tag test

#Creamos el mesh del circuito para VDD y VSS
connect_pg_net -automatic
create_pg_mesh_pattern mesh_pattern -layers { {{vertical_layer: METAL3} {
    width: 4.2} {pitch: 42} {spacing: interleaving}} }
set_pg_strategy mesh_strategy -polygon {{125.000 118.000} {259.960 265.280}}
     -pattern {{pattern: mesh_pattern}{nets: {VDD VSS}}} -blockage {macros:
    all}
create_pg_std_cell_conn_pattern std_cell_pattern
```

```
set_pg_strategy std_cell_strategy -core -pattern {{pattern: std_cell_pattern
    }{nets: {VDD VSS}}}
compile_pg

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

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

#Ruteamos
check_routability -check_pg_blocked_ports true
check_design -checks pre_route_stage
route_auto

#Creamos los filler del core y el IO ring
create_io_filler_cells -io_guides [get_io_guides {anillo_IO_XOR.top
    anillo_IO_XOR.right anillo_IO_XOR.left anillo_IO_XOR.bottom}] \
-reference_cells {PFILLER1 PFILLER5 PFILLER05 PFILLER0005 PFILLER10
    PFILLER20}
create_stdcell_fillers -lib_cells [get_lib_cells {TSMCWorkspace|
    FillersWorkspace/FILL64BWP7T TSMCWorkspace|FillersWorkspace/FILL32BWP7T
    TSMCWorkspace|FillersWorkspace/FILL16BWP7T TSMCWorkspace|FillersWorkspace
    /FILL8BWP7T TSMCWorkspace|FillersWorkspace/FILL4BWP7T TSMCWorkspace|
    FillersWorkspace/FILL2BWP7T TSMCWorkspace|FillersWorkspace/FILL1BWP7T}]
connect_pg_net -automatic
remove_stdcell_fillers_with_violation
check_legality

#Guardamos el bloque
save_block XOR_SYN2.ndm:Xor_IO
```

CAPÍTULO 17

Glosario

GDS: Las siglas pueden significar diferentes cosas: *Graphic Design System*, *Graphic Data Stream* o *Geometric Data Stream*. Archivos utilizados para el intercambio del diseño de un *layout* de un circuito. 12

HDL: Lenguaje descriptor de *hardware*. En este lenguaje se describen componentes físicos en vez de una función que se realiza. 11

***chip*:** Conjunto de transistores nanométricos que realizan una función, circuito integrado. 1

***core*:** Parte central del *layout* del diseño. En donde se ubican las celdas del circuito. 19

***corners*:** Celdas de relleno utilizadas exclusivamente en las esquinas del diseño. 19

***die*:** Bloque de material semiconductor utilizado para la elaboración de un circuito. 12

***fillers*:** Celdas de relleno utilizados entre las celdas dentro del *core* y alrededor del anillo de IO. 21

***layout*:** Representación física de las geometrías que conforman un circuito integrado. 6

***netlist*:** Archivo que contiene las componentes las conexiones de un circuito a nivel de nodos y componentes. 12

***place-and-route*:** Colocación e interconexión de componentes dentro de un circuito integrado. 16

***script*:** Conjunto de instrucciones o comandos que realizan una tarea de manera predefinida. 21

***track*:** Geometrías del *layout* que conectan las distintas celdas de un diseño. Análogo a un cable. 15

anillo de entradas y salidas: Parte que rodea el *core* del circuito. En donde se ubican las celdas que menejan los *inputs* y *ouputs* del diseño. 21

celda: Bloques básicos utilizados para la síntesis de circuitos que ejecutan funciones elementales como compuertas NOT, OR, etc. hasta *flip-flops* o multiplexores. 12

MOSFET: Transistores de efecto de campo metal-óxido-semiconductor. 11