Trabajo de Fin de Máster
"Máster Universitario en Microelectónica: Diseño y Aplicaciones de Sistemas Micro/Nanométricos"

Uso de herramientas de código abierto para el diseño de circuitos y sistemas integrados analógicos, digitales y de señal mixta

Paula González Bautista
Jose M. de la Rosa
27/10/2023

Agradecimientos

Agradecer encarecidamente a Jose M. de la Rosa por introducirme en este magnífico mundo y brindarme su ayuda y apoyo. También agradecer a los buenos docentes que me han acompañado hasta este momento, por que gracias a ellos he hecho de la microelectrónica una pasión más para mí. A mis familiares, quienes me han dado su comprensión y motivación y han hecho posible este camino. Y por último, a la comunidad Open Source por la gran labor que están llevando a cabo.

Resumen

Este trabajo de fin de master realiza una investigación de proyectos, entornos y herramientas de relevancia en el mundo del hardware libre. Gracias a ello se establece una base sólida de herramientas y entornos disponibles para el diseño de circuitos y sistemas integrados analógicos, digitales y de señal mixta. Haciendo uso de dichas herramientas, se instaura un flujo de trabajo de diseño digital desde las especificaciones del diseño hasta el nivel de layout completamente en código abierto, dando lugar a la evaluación de dichos recursos en el ámbito del diseño de circuitos y sistemas integrados con el fin de ser una guía de introducción a este metodología de diseño y contribuir a un diseño colaborativo de circuitos integrados.

Índice

Índice de figuras

Notación

Abreviatura	Definición
OSS	Open Source Software
OSH	Open Source Hardware
HDL	Hardware Description Language
FPGA	Field Programmable Gate Array
ASIC	Circuito Integrado para aplicaciones específicas
SoC	System on Chip
IC	Integrated Circuits
OpenMPW	Open Multi Project Wafer
OSIC	Open-source-based integrated circuit
FOSS	Free And Open Source Software
LGPL	Licencia Pública General Reducida de GNU
IIC	Institute for Integrated Circuits
RTL	Register-Transfer Level
PDK	Process Development Kit
IHP	Innovations for high performance, microelectronics
FDSOI	Fully Depleted Silicon on Insulator

1. Introducción

Para indagar en el mundo del diseño hardware en código abierto es necesario conocer sus inicios y el entorno que lo engloba. Código abierto (en inglés, open source) se refiere a la práctica de permitir que el código fuente de un software o un proyecto esté disponible de manera pública y sea modificable por cualquier persona. La idea de código abierto se remonta a varias décadas atrás, destacando sobre todo en el ámbito del software. Fue en la década de 1990 cuando el término "código abiertoz la filosofía detrás de él comenzaron a tomar forma y ganar reconocimiento significativo en la comunidad de desarrollo de software y en la industria de la tecnología, conocido como "Open Source Software"(OSS)[1].

El movimiento del hardware de código abierto, conocido como "Open Source Hardware"(OSH) o hardware libre, se ha desarrollado en paralelo al del software, aunque con un inicio menos definido y un menor impacto. Esto se debe a que la naturaleza del software es diferente a la del hardware. A diferencia del software de código abierto, el hardware de código abierto comenzó a ganar impulso a lo largo de las décadas de 2000 y 2010[2] y dado que el hardware tiene asociados a él costos variables directos, ninguna definición de software libre se puede aplicar directamente sin modificación al término hardware libre. Generalmente el término hardware libre o OSH se ha usado principalmente para reflejar el uso del software libre con el hardware y el lanzamiento libre de la información con respecto al hardware, a menudo incluyendo el lanzamiento de los diagramas esquemáticos, diseños, tamaños y otra información acerca del hardware.

El objetivo del movimiento OSH es ampliar los horizontes de la colaboración en diseño de hardware al aplicar los principios del software de código abierto al diseño de hardware. Al liberar diseños de hardware bajo licencias de código abierto, OSH intenta fomentar la transparencia, la colaboración y la innovación. Haciendo posible que cualquier persona pueda acceder, modificar y utilizar diseños libres para desarrollar productos personalizados y soluciones innovadoras. Esto no solo aceleraría el proceso de desarrollo, sino que también impulsaría la creación de una comunidad global de diseñadores y fabricantes que puedan construir sobre los logros de otros.

Con el auge de los dispositivos de lógica programable reconfigurables, el compartir los diseños lógicos es también una práctica de hardware libre. En lugar de compartir los diagramas esquemáticos, el código HDL es compartido. Esto difiere del software libre. Las descripciones HDL son usadas comúnmente para integrar en sistemas SoC (System-on-chip) que incluyan FPGA o directamente en diseños ASIC.

En el vertiginoso mundo de la tecnología, la innovación en el diseño de circuitos integrados y hardware desempeña un papel fundamental en la creación de productos electrónicos cada vez más avanzados y eficientes y los desafíos económicos y técnicos que los acompañan hacen que la mayoría de los diseños queden en manos de las grandes compañías, dificultando el acceso de diseñadores independientes, startups y comunidades a la creación de chips personalizados.

Para abordar esta cuestión, también han surgido conceptos como efabless y "Open Multi Project Wafer"(Open MPW)[3], que están revolucionando la forma en que se aborda el diseño colaborativo y el acceso a la tecnología de hardware. En el corazón de esta revolución se encuentra efabless[4], una innovadora plataforma que tiene como objetivo democratizar el mundo del diseño de chips. Al ofrecer un espacio en línea donde diseñadores de chips de todas

partes del mundo pueden colaborar en proyectos conjuntos, efabless elimina las barreras tradicionales que dificultan la entrada a este campo. Al facilitar la colaboración y el intercambio de conocimientos, efabless permite que diseñadores independientes y pequeñas empresas participen en el proceso de diseño de chips, lo que anteriormente estaba reservado principalmente para grandes corporaciones con recursos significativos. Para aquellos que buscan reducir los costos de fabricación de chips, la iniciativa Open MPW, colaborativa con efabless, emerge como una solución innovadora. Open MPW reúne varios proyectos de diseño de chips diferentes en una sola oblea de silicio compartida. Esta consolidación de esfuerzos permite a múltiples proyectos compartir los costos de producción, haciendo que el diseño y la fabricación de chips sean más asequibles para todos los involucrados. Pequeñas empresas, startups y diseñadores individuales pueden beneficiarse enormemente al acceder a la tecnología de fabricación de chips sin incurrir en los altos costos asociados.

En conjunto, la convergencia de efabless, Open MPW y el movimiento OSH está redefiniendo la forma en que se aborda el diseño de chips y hardware. Estas iniciativas promueven la democratización, la colaboración y la accesibilidad en un campo que alguna vez fue exclusivo y costoso. A medida que la tecnología continúa avanzando, estas tendencias seguramente seguirán siendo pilares clave en la evolución de la industria electrónica y del diseño de hardware.

1.1. Objetivos

Este proyecto tiene como objetivo principal la investigación de las herramientas de código abierto, entornos y documentación disponibles para diseños digitales, analógicos y de señal mixta.

Como subobjetivo se pretende establecer un flujo de trabajo de diseño digital utilizando exclusivamente herramientas de código abierto, que abarcará desde el nivel RTL hasta el proceso de layout, para corroborar los entornos y herramientas estudiadas.

1.2. Metodología

La metodología que se seguirá en este proyecto se divide en varias etapas clave:

1. **Investigación Inicial**: En esta fase, se llevará a cabo una investigación exhaustiva sobre las herramientas de código abierto disponibles para el diseño en código abierto de circuitos integrados. Se identificarán las mejores opciones disponibles y se evaluarán en función de su eficiencia y capacidad para satisfacer las necesidades del diseño.

2. **Estudio de Entornos Disponibles**: En esta fase, se llevará a cabo una investigación acerca de los entornos de desarrollo código abierto creados por la comunidad de OSH. Se identificarán las mejores opciones disponibles y se evaluarán en función de su eficiencia y capacidad para satisfacer las necesidades del diseño.

3. **Selección de Herramientas**: Una vez completada la investigación, se seleccionarán las herramientas de código abierto para comprobar funcionalidades de las mismas. Se tendrán en cuenta factores como la disponibilidad

de características necesarias, la comunidad de usuarios, la documentación y la facilidad de uso.

4. **Definición del Flujo de Trabajo**: Con las herramientas seleccionadas, se procederá a definir el flujo de trabajo que abarcará desde el nivel RTL hasta el placement. Se establecerán los pasos específicos que se seguirán en el proceso de un diseño digital y se documentarán detalladamente.

5. **Implementación del Diseño Digital**: En esta etapa, se llevará a cabo la implementación real del ejemplo. Se utilizarán las herramientas de código abierto seleccionadas para crear el diseño, teniendo en cuenta los estándares y las mejores prácticas del campo.

6. **Pruebas y Validación**: Una vez completada la implementación, se realizarán pruebas exhaustivas para asegurarse de que el diseño cumple con los requisitos y especificaciones establecidos. Se identificarán y corregirán posibles errores o problemas.

7. **Documentar y Compartir Resultados**: Se documentará todo el proceso, desde la investigación inicial hasta la implementación y las pruebas. Los resultados y los diseños creados se compartirán en la comunidad de código abierto, siguiendo los principios del movimiento OSH y contribuyendo al conocimiento compartido.

8. **Evaluación de Resultados y Conclusiones**: Se evaluará el éxito del proyecto en función de los objetivos establecidos. Se extraerán conclusiones sobre la viabilidad y eficacia de utilizar herramientas de código abierto en el diseño, así como las lecciones aprendidas durante el proceso.

2. Estado del arte

En la actualidad hay muchas áreas clave relacionadas con los circuitos integrados en código abierto, como son los kits de desarrollo de procesos, las herramientas disponibles para poder cumplimentar un proceso completo del flujo de diseño, los entornos disponibles, empresas colaborativas, proyectos importantes... Por ello, realizaremos en primer lugar un análisis de todo lo que nos ofrece actualmente esta metodología.

2.1. Proyectos de Hardware Libre

Una de las áreas clave que ha propulsado los diseños de hardware libre ha sido el éxito de proyectos notables relacionados con arquitecturas de procesadores en el mundo del open source, algunos de ellos son:

2.1.1. OpenRISC

OpenRISC es un diseño de CPU RISC de especificación libre, realizado por OpenCores y publicado bajo la licencia LGPL. La Licencia Pública General Reducida de GNU, o más conocida por su nombre en inglés "GNU Lesser General Public License", o simplemente por su acrónimo del inglés GNU LGPL, es una licencia de software creada por la Free Software Foundation que pretende garantizar la libertad de compartir y modificar el software cubierto por ella, asegurando que el software es libre para todos sus usuarios.

El diseño está implementado en el lenguaje de descripción de hardware verilog y se ha fabricado con éxito tanto como circuito integrado ASIC como implementado mediante entornos FPGA[5].

La GNU toolchain ha sido portada a OpenRISC para permitir el desarrollo en distintos lenguajes. Linux y uCLinux han sido también portados a este procesador.

2.1.2. RISC-V

RISC-V es una arquitectura de conjunto de instrucciones (ISA) de hardware libre basado en un diseño de tipo RISC (conjunto de instrucciones reducido).

A diferencia de la mayoría de los conjuntos de instrucciones, el de RISC-V es libre y abierto y se puede usar sin regalías para cualquier propósito, lo que permite que cualquiera diseñe, fabrique y venda chips y software de RISC-V. Si bien no es la primera ISA de arquitectura abierta, es significativa porque está diseñada para ser útil en una amplia gama de dispositivos. El conjunto de instrucciones también tiene un cuerpo sustancial de software de soporte, que evita una debilidad habitual de los nuevos conjuntos de instrucciones.

El proyecto comenzó en 2010 en la Universidad de California en Berkeley, pero muchos colaboradores son voluntarios y trabajadores de la industria fuera de la universidad.

El conjunto de instrucciones se ha diseñado pensando en implementaciones pequeñas, rápidas y de bajo consumo para el mundo real, pero sin una sobreingeniería excesiva que buscase una microarquitectura concreta.

En mayo de 2017, estaba cerrada la versión 2.2 del conjunto de instrucciones del espacio de usuario. El conjunto de instrucciones privilegiadas estaba disponible como borrador en la versión 1.10[6].

2.2. Plataformas, comunidades y empresas colaborativas con el código abierto

Hay variedad de iniciativas y organizaciones que promueven la colaboración y la accesibilidad en el diseño de hardware de código abierto, con el objetivo de democratizar el mundo del diseño de chips y sistemas digitales. Estas iniciativas son las principales responsables del impulso de la creación de diseños de silicio personalizados y fomento de la innovación en el campo de la electrónica.

2.2.1. FOSSi (Free and Open Source Silicon)

FOSSi se refiere a componentes y sistemas dentro de dispositivos de silicio (chips) que son de código abierto y libres. Su objetivo es que los componentes básicos utilizados en diseños de hardware digital sean accesibles y modificables por cualquier persona. Está dirigido a una amplia audiencia que incluye desde expertos de la industria hasta aficionados y académicos. La comunidad FOSSi se basa en la colaboración y la reutilización de bloques de construcción abiertos para el diseño de sistemas digitales[7].

2.2.2. LowRISC

LowRISC es una organización independiente que se centra en proyectos relacionados con la tecnología de código abierto y el silicio de código abierto. Colabora en proyectos de tecnologías precompetitivas y proporciona un entorno neutral para la colaboración en la industria, la academia y las fundaciones de código abierto. Busca eliminar las barreras para producir diseños de silicio y reduce la dependencia de los proveedores. Además, enfatiza la calidad, la simplicidad y la protección de la propiedad intelectual a través de licencias permisivas[8].

2.2.3. Open Source Silicon

Open Source Silicon se enfoca en facilitar la creación de diseños de silicio personalizados y la colaboración en proyectos de código abierto. Permite la reutilización de bloques de construcción existentes para construir sistemas digitales propios. Además de la comunidad de aficionados y académicos, cada vez más empresas utilizan bloques de IP de código abierto y desarrollan código abierto para involucrar a las personas en torno a sus diseños[9].

2.2.4. Efabless

efabless es una plataforma y una comunidad en línea que se centra en la creación colaborativa de circuitos integrados personalizados o chips de silicio. Permite a diseñadores de circuitos, ingenieros y entusiastas de todo el mundo colaborar en el diseño, verificación y producción de chips de silicio a través de una plataforma en línea[10].

Esta plataforma ofrece una serie de características y servicios que incluyen:

1. Marketplace de IP y diseño: efabless proporciona un mercado en línea donde los diseñadores pueden acceder a bloques de propiedad intelectual (IP) reutilizables y colaborar en proyectos de diseño de circuitos integrados.

2. Herramientas de diseño basadas en la nube: Los usuarios pueden acceder a herramientas de diseño en la nube que les permiten crear y verificar circuitos

integrados sin la necesidad de inversiones costosas en infraestructura y herramientas de diseño.

3. Colaboración en proyectos: efabless permite a los diseñadores colaborar en proyectos de chips, lo que fomenta el trabajo en equipo y la creación de diseños más complejos y avanzados.

4. Acceso a servicios de producción de chips: Los proyectos de diseño que se desarrollan en efabless pueden pasar a la etapa de producción. efabless colabora con fundiciones y proveedores de fabricación de chips para llevar los diseños a la producción.

5. Comunidad y aprendizaje: La plataforma también fomenta la interacción y el aprendizaje entre sus miembros, lo que permite a los diseñadores y entusiastas de la electrónica mejorar sus habilidades y conocimientos en diseño de circuitos integrados. efabless cuenta con el apoyo de la industria de semiconductores:

2.2.5. ARM

ARM, una empresa británica de diseño de semiconductores, que ha estado involucrada en el mundo del código abierto en varias formas. Aunque ARM es conocida principalmente por sus diseños de arquitectura de procesadores, ha desempeñado un papel en la promoción y el apoyo al código abierto en diferentes niveles:

1. Diseño de arquitectura de procesadores: ARM ha proporcionado diseños de arquitectura de procesadores que se utilizan en una amplia variedad de dispositivos, incluyendo teléfonos inteligentes, tabletas y sistemas embebidos. Aunque ARM es una empresa que licencia sus diseños de hardware a otros fabricantes, ha permitido que estos diseños sean utilizados en sistemas basados en software de código abierto. Muchos sistemas operativos de código abierto, como Linux, Android y FreeBSD, se ejecutan en procesadores ARM.

2. Colaboración en proyectos de código abierto: ARM ha colaborado en varios proyectos de código abierto relacionados con el hardware y el software. Ha contribuido a proyectos como el kernel de Linux, el proyecto de código abierto RISC-V, y ha estado involucrado en la Fundación Linaro, que trabaja en la optimización de software para arquitecturas ARM.

3. Promoción de la educación y la innovación: ARM ha estado activo en la promoción de la educación y la formación en tecnología, especialmente en el ámbito de la informática y la electrónica. Ha apoyado iniciativas como Raspberry Pi, que utiliza sus diseños de procesadores y tiene como objetivo promover la programación y la informática entre los jóvenes.

4. Participación en estándares abiertos: ARM ha estado involucrado en la creación y promoción de estándares abiertos en la industria de semiconductores. Por ejemplo, ha sido un miembro activo en la promoción del estándar de interconexión AMBA (Advanced Microcontroller Bus Architecture) para sistemas en chip[11].

2.2.6. Google

Google ha estado involucrado en varias iniciativas de código abierto relacionadas con hardware a lo largo de los años. Algunas de las áreas en las que Google ha contribuido o participado en proyectos de código abierto de hardware incluyen:

1. **RISC-V**: Google ha sido un partidario de la arquitectura RISC-V, que es una arquitectura de conjunto de instrucciones (ISA) de código abierto. La compañía ha estado investigando y explorando la posibilidad de utilizar RISC-V en sus procesadores y ha colaborado en proyectos relacionados con RISC-V[6].

2. **OpenTitan**: Google ha liderado el proyecto OpenTitan, una iniciativa de código abierto que se centra en el diseño y la verificación de chips de seguridad de hardware. El objetivo es crear una plataforma de seguridad de hardware de código abierto que pueda utilizarse en una variedad de aplicaciones, desde servidores hasta dispositivos integrados[12].

3. **Android Open Source Project (AOSP)**: Aunque Android es principalmente un sistema operativo de código abierto para dispositivos móviles, también incluye componentes de hardware relacionados, como controladores de dispositivos y bibliotecas. Google ha contribuido al desarrollo de AOSP y ha fomentado la adopción de Android en una amplia variedad de dispositivos[13].

4. **Tensor Processing Unit (TPU)**: Aunque no se trata de un proyecto de código abierto, Google ha desarrollado su propia unidad de procesamiento de tensores (TPU) para acelerar el aprendizaje automático y la inteligencia artificial y ha compartido detalles de diseño y arquitectura de TPU en conferencias técnicas, fomentando la investigación relacionada con TPU[14].

5. **Colaboración en la industria**: Google ha colaborado con otras empresas y organizaciones en la promoción de estándares abiertos y hardware de código abierto en la industria. Por ejemplo, ha sido un miembro activo en la Open Compute Project (OCP), una iniciativa que promueve el diseño de hardware de centros de datos de código abierto.

2.2.7. SkyWater Technology

SkyWater produce chips semiconductores utilizando tecnología de proceso de 90 nanómetros en equipos diseñados para manejar obleas de silicio de 200 milímetros trabajando en las industrias de consumo, industrial, militar y de defensa, y automotriz. Actualemente, SkyWater ha colaborado con efabless y Google para crear el primer programa de fabricación de chips de código abierto generándose así el PDK de código abierto de SkyWater en colaboración con Google, además de recursos relacionados, que se pueden utilizar para crear diseños fabricables en las instalaciones de SkyWater[15].

2.3. Kits de Desarrollo de Proceso

El PDK (Process Development Kit) es esencial en el diseño de OSIC. Estos kits proporcionan herramientas, bibliotecas y recursos necesarios para adaptar el diseño de los circuitos al proceso de fabricación de semiconductores específico. Aunque la mayoría de los PDK son propietarios, algunos proyectos de hardware de código abierto han desarrollado PDK abiertos.

2.3.1. GF 180

El GF180MCU PDK de código abierto es una colaboración entre Google y GlobalFoundries que proporciona un kit de diseño de procesos (PDK) completo de código abierto y recursos relacionados. Este kit permite la creación de diseños que pueden ser fabricados en las instalaciones de GlobalFoundries utilizando su tecnología de proceso MCU de 0,18 µm y 3,3 V/6 V[16]. Este PDK se considera una vista previa experimental, y se advierte que no está diseñado para entornos de producción en este momento. Se sugiere su uso para chips de prueba y verificación del diseño. Mientras tanto se están llevando a cabo diseños de validación internos, que incluyen la validación de silicio y datos publicados, con planes de publicar estos resultados. Se prevé que el PDK se etiquetará con una versión de producción cuando esté listo para su uso en diseño de producción. Los recursos más recientes se pueden encontrar en GitHub en los repositorios de foss-eda-tools[16][17]. Para trabajar con este PDK, es necesario tener Git 2.35+ y Python 3.6+ instalados. GlobalFoundries ofrece soporte a través de un ecosistema de socios de mercado, y existe una lista de correo para que los usuarios se apoyen mutuamente. El GF180MCU PDK de código abierto representa un paso importante en la colaboración entre Google y GlobalFoundries para promover el diseño de semiconductores de código abierto[18].

2.3.2. IHP

El PDK de código abierto de IHP ("Innovations for high performance, microelectronics", en inglés) es un proyecto dedicado al diseño analógico, de señal mixta y de RF que se centra en una tecnología BiCMOS de 130 nm. El objetivo de este proyecto es proporcionar un Kit de diseño de procesos de código abierto y datos relacionados que se pueden utilizar para crear diseños en las instalaciones de IHP[19].

El estado actual del proyecto se encuentra en una etapa de vista previa. Aunque la tecnología y el PDK han sido utilizados con éxito en la fabricación de numerosos diseños, se advierte que no está destinado a la producción en este momento.

El nodo de proceso SG13G2 es una tecnología BiCMOS de alto rendimiento con un proceso CMOS de 0,13 µm. Ofrece dispositivos bipolares basados en SiGe:C npn-HBT con altas frecuencias de funcionamiento. El proceso incluye transistores NMOS, PMOS y NMOS aislados, así como otros componentes pasivos.

El contenido del PDK incluye un conjunto de celdas base con un conjunto limitado de celdas lógicas estándar, CDL, GDSII, LEF, SPICE Netlist, Liberty, Verilog, SRAM cellset, dispositivos primitivos, KLayout layer property, Pcells (conjunto limitado, solo para referencia), modelos HSPICE de dispositivos HBT y más.

El proyecto IHP Open Source PDK está abierto a contribuciones de la comunidad. Los interesados pueden bifurcar el repositorio en GitHub y colaborar en mejoras, correcciones de errores, adiciones de características y documentación.

IHP es un centro de investigación financiado por los gobiernos federal y estatal alemán. Se destaca en el campo de la electrónica de silicio/germanio y realiza investigaciones en tecnología de semiconductores, diseño de circuitos de alta frecuencia y soluciones de sistemas.

El PDK de código abierto de IHP se publica bajo la licencia Apache 2.0, con derechos de autor por parte de los Autores de PDK de IHP.

Este proyecto representa un avance significativo en la colaboración de IHP en el diseño de semiconductores de código abierto[20].

2.3.3. SKY 130

SKY130 es un kit de diseño de procesos (PDK, por sus siglas en inglés) de código abierto que resulta de una colaboración entre Google y SkyWater Technology Foundry. Este PDK proporciona un conjunto de herramientas y bibliotecas esenciales para diseñar circuitos integrados (CI) utilizando un proceso híbrido que combina tecnologías de 180 nm y 130 nm. A continuación, se resumen algunos aspectos clave de SKY130[15]:

1. Tecnología de Proceso: SKY130 es un proceso de fabricación de chips que ofrece una transición suave entre tecnologías de 180 nm y 130 nm. Esto significa que se pueden diseñar circuitos utilizando reglas de diseño y características propias de ambas tecnologías en un solo proceso.

2. Voltaje de Operación: La tensión central de operación en SKY130 es de 1.8 voltios (V). Aunque es posible utilizar tensiones más altas, en muchas aplicaciones se utiliza este valor estándar.

3. Interconexiones y Capas Metálicas: El proceso SKY130 ofrece varias capas de interconexión metálica (capas de Aluminio, Al) y una capa de interconexión local (capa de TiN) para enrutar las señales dentro del chip.

4. Capacitores: Los diseñadores pueden implementar capacitores en SKY130 utilizando capas específicas llamadas Metal-Insulator-Metal (MiM) layers.

5. Dispositivos MOSFET: El proceso SKY130 permite el uso de transistores MOSFET con una longitud mínima de 150 nm. Además, existen restricciones en el ancho de estos dispositivos dependiendo del tipo de celda y su aplicación.

6. Bibliotecas de Diseño: SKY130 proporciona bibliotecas que contienen celdas lógicas digitales de alta densidad, así como celdas primitivas analógicas (por ejemplo, MOSFET parametrizables, diodos, resistencias, capacitores) para el diseño de circuitos digitales y analógicos.

Estas bibliotecas permiten a los diseñadores crear circuitos complejos utilizando herramientas de diseño de chips y simulaciones para garantizar que los dispositivos cumplan con los requisitos específicos de su aplicación. SKY130 es una plataforma de código abierto que promueve la innovación y la colaboración en el diseño de circuitos integrados[21].

2.3.4. SKY 90

La tecnología SKY90FD de SkyWater vuelve a ser una colaboración entre Google y SkyWater Technology Foundry. De nuevo se proporciona un PDK de código abierto y recursos relacionados para permitir el diseño de dispositivos en

la tecnología SKY90FD de 90 nm FDSOI. Actualemente SkyWater nos proporciona una versión experimental.

La tecnología SKY90-FD es un proceso de 90 nm FDSOI (Fully Depleted Silicon on Insulator) que presenta una capa delgada de material aislante entre el sustrato y la capa superior de silicio. Esto permite que los transistores sean significativamente más delgados que en procesos convencionales, lo que reduce la fuga de corriente parasitaria y mejora el rendimiento de velocidad y potencia[22].

2.4. Entornos disponibles de código abierto

Dada la particularidad del open source de diseñar en comunidad hay muchos entornos ya disponibles, creados para facilitar la gran variedad de herramientas.

2.4.1. FOSS-ASIC-TOOLS

El proyecto "foss-asic-tools"[17] es un conjunto de herramientas de código abierto diseñadas específicamente para el desarrollo de Circuitos Integrados de Aplicación Específica (ASIC). FOSS-ASIC-TOOLS es un contenedor con las herramientas necesarias para el diseño basado en SKY130 tanto analógico como digital.

2.4.2. IIC-OSIC-TOOLS

El proyecto "iic-osic-tools"[23] es otro conjunto de herramientas de código abierto destinado al diseño de Circuitos Integrados de Código Abierto (OSIC) basado en el anterior proyecto, "foss-asic-tools". Estas herramientas son fundamentales para la creación de hardware de código abierto y permiten a los diseñadores trabajar en proyectos de OSIC de manera eficiente y colaborativa. iic-osic-tools ha contribuido al crecimiento y la innovación en el campo de los OSIC de la mano del Instituto de Circuitos Integrados (IIC) de la Universidad Johannes Kepler.

Este repositorio es un ejemplo concreto de cómo la comunidad académica y de investigación, en este caso, representada por el Instituto de Circuitos Integrados (IIC) de la Universidad Johannes Kepler (JKU), está contribuyendo a la mejora y colaboración en el campo de los circuitos integrados. Al igual que efabless y Open MPW, el repositorio se basa en el enfoque de código abierto, lo que significa que las herramientas y recursos proporcionados son accesibles para una amplia gama de personas, desde diseñadores hasta investigadores y estudiantes.

Este repositorio aborda específicamente la necesidad de contar con herramientas eficientes y versátiles para el diseño y análisis de circuitos integrados. Proporciona una solución integral en forma de un contenedor Docker que contiene una serie de herramientas y bibliotecas preconfiguradas para facilitar tanto el diseño analógico como el digital. Al igual que en el caso de Open MPW, donde varios diseñadores colaboran en un proyecto de diseño de un ASIC, el repositorio "iic-osic-toolsreúne una variedad de herramientas que permiten a los diseñadores trabajar en un entorno unificado y colaborativo.

En resumen, el repositorio "iic-osic-tools"se alinea con los principios de colaboración y código abierto de las iniciativas mencionadas anteriormente. Proporciona una plataforma integral para el diseño y análisis de circuitos integrados,

promoviendo la colaboración, la accesibilidad y la mejora continua en el campo de la electrónica y la tecnología de semiconductores. Al igual que efabless y Open MPW, esta iniciativa busca empoderar a los diseñadores y la comunidad en general para avanzar en el campo de los circuitos integrados y garantizar su seguridad y confiabilidad en un mundo cada vez más interconectado.

Herramientas En el diseño de Circuitos Integrados de Código Abierto (OSIC), se utilizan diversas herramientas de software para llevar a cabo tareas clave como la síntesis, simulación, verificación y diseño de circuitos. Muchas de estas herramientas son de código abierto y se han convertido en estándares en la comunidad de OSIC. A continuación, se describen algunas de estas herramientas:

- **align**: Una herramienta de generación de diseños de circuitos analógicos que ofrece características de alineación automática (solo en el contexto del Kit de Desarrollo de Proceso, PDK) para el proceso SkyWater sky130[24].
- **amaranth**: Una cadena de herramientas de diseño de hardware (HDL) basada en Python que permite el desarrollo de circuitos digitales[25].
- **cocotb**: Una biblioteca de simulación que permite escribir bancos de pruebas VHDL y Verilog en Python para verificar el diseño de hardware[26].
- **covered**: Una herramienta de cobertura de código para Verilog que ayuda a evaluar la eficacia de las pruebas[27].
- **cvc**: Un comprobador de validez de circuitos que verifica la corrección de un diseño[28].
- **edalize**: Una biblioteca de abstracción en Python que simplifica la interacción con herramientas de diseño asistido por computadora (EDA)[29].
- **fault**: Una solución de diseño para pruebas (DFT) que ayuda a diseñar circuitos para su fácil verificación y diagnóstico[30]
- **fusesoc**: Un administrador de paquetes y herramientas de construcción para sistemas en chip (SoC)[31].
- **gaw3-xschem**: Una herramienta para trazar formas de onda en xschem, un editor esquemático[32].
- **gdsfactory**: Una biblioteca en Python para generar ficheros GDS utilizados en la fabricación de circuitos integrados[33].
- **gdspy**: Un módulo en Python para la creación y manipulación de ficheros GDS (Stream de Datos Generales)[34].
- **gds3d**: Una herramienta de visualización en 3D para ficheros GDS[35].
- **gf180mcu**: Un kit de desarrollo de proceso (PDK) de GlobalFoundries para tecnología CMOS de 180 nm[36].
- **ghdl**: Un simulador de VHDL[37].
- **gtkwave**: Una herramienta para trazar formas de onda en simulaciones digitales[38].

- **ihp-sg13g2**: Un PDK parcial para la tecnología SiGe:C BiCMOS de 130 nm de IHP Microelectronics[39].
- **irsim**: Un simulador digital de nivel de interruptores[40].
- **iverilog**: Un simulador de Verilog[41].
- **klayout**: Un visor y editor de diseño para ficheros GDS y OASIS[42].
- **magic**: Un editor de diseño con reglas de diseño y extracción de parámetros[43].
- **netlistsvg**: Una herramienta que genera gráficos SVG a partir de descripciones de netlists en formato JSON[44].
- **ngspice**: Un simulador SPICE para circuitos analógicos y de señal mixta[45].
- **ngspyce**: Un conjunto de enlaces de Python para el simulador ngspice[46].
- **nvc**: Un simulador y compilador de VHDL[47].
- **open pdks**: Scripts de configuración para preparar PDKs de código abierto[48].
- **openlane**: Un flujo de diseño digital de RTL a GDS[49].
- **openlane2**: Una reescritura de OpenLane en Python, la segunda generación del flujo de diseño digital[50].
- **openram**: Una biblioteca en Python para el diseño de memorias SRAM[51].
- **openroad**: Un motor de RTL a GDS utilizado por OpenLane y OpenLane2[52].
- **osic-multitool**: Una colección de scripts y documentación útiles para el diseño de OSIC[53].
- **padring**: Una herramienta para la generación de disposiciones de pads[54].
- **pyopus**: Un corredor de simulación y herramienta de optimización para circuitos analógicos[55].
- **pyrtl**: Una colección de clases para el diseño de hardware de nivel de registro en Python[56].
- **pyspice**: Una interfaz para usar el simulador SPICE desde Python[57].
- **pyverilog**: Un conjunto de herramientas en Python para el análisis y la generación de código Verilog[58].
- **RF toolkit**: Un conjunto de herramientas para diseño de RF, incluyendo FastHenry2, FasterCap y openEMS[59].
- **qucs-s**: Un entorno de simulación con un enfoque en RF (radiofrecuencia)[60].

- **rggen**: Una herramienta de generación de código para registros de configuración y estado[61].
- **spyci**: Una herramienta para analizar, trazar y generar datos con Python para simulaciones[62].
- **slang**: Una herramienta para el análisis y la traducción de SystemVerilog[63].
- **vlog2verilog**: Una herramienta para la conversión de ficheros Verilog[64].
- **volare**: Un administrador de versiones (y constructor) para PDK de código abierto[65].
- **risc-v toolchain**: Un conjunto de herramientas de compilación de GNU para núcleos RISC-V RV32I[66].
- **siliconcompiler**: Un sistema modular para la construcción de hardware[67].
- **sky130**: El PDK de tecnología CMOS de 130 nm de SkyWater Technologies[68].
- **verilator**: Un simulador rápido de Verilog[69].
- **xschem**: Un editor esquemático[70].
- **xyce**: Un simulador SPICE rápido y paralelo, incluyendo una herramienta de conversión de netlists[71].
- **yosys**: Una herramienta de síntesis de Verilog (con complemento GHDL para síntesis VHDL)[72].

2.5. Workflow

Una vez estudiadas las capacidades del diseño en open source se puede establecer un flujo de diseño concreto conociendo las posibilidades y limitaciones que nos ofrece este entorno para probar algunos de los entornos y herramientas más demandadas hasta ahora. Para ello se partirá del flujo de diseño común de cualquier circuito ASIC digital y a lo largo del desarrollo del proyecto se irán concretando las herramientas y simulaciones que se han ofrecido.

Figura 1: Workflow diseño digital

3. Descripción del flujo de diseño digital en código abierto

Después de explorar las tecnologías, herramientas y entornos de desarrollo, nos enfocaremos en establecer el proceso requerido para llevar a cabo un diseño digital utilizando las herramientas OSIC disponibles basándonos en el workflow definido anteriormente. Para ello partiremos de unas especificaciones sencillas de un diseño digital:

- Diseño de Multiplexor 2 a 1 en Verilog.
- Función: El módulo actúa como un multiplexor 2 a 1, seleccionando una de las dos entradas como salida basándose en la señal de control.
- Entradas:
  - ‘a‘: Entrada de datos 1.
  - ‘b‘: Entrada de datos 2.
  - ‘select‘: Señal de control que determina la selección.
- Salida:
  - ‘y‘: Salida seleccionada.

Una vez definidas las especificaciones debemos adaptarnos al entorno en el que se encuentran las herramientas haciendo uso de contenedores:

- Clonar los repositorios necesarios.
- Instalar Docker: El sistema de software de TI llamado Docker es la tecnología de organización en contenedores que posibilita la creación y el uso de los contenedores de Linux. El objetivo básicamente con Docker es poder utilizar los contenedores como máquinas virtuales muy livianas y modulares, y obteniendo la flexibilidad necesaria para crearlos, implementarlos, copiarlos y trasladarlos de un entorno a otro, lo cual permite optimizar las aplicaciones para la nube.
- Iniciar los contenedores Docker basados en las distintas imágenes de las herramientas o entornos[73].

3.1. RTL y banco de pruebas

Descritas las especificaciones del diseño y preparado el entorno de trabajo llegamos al nivel RTL, en esta capa se utiliza en los lenguajes de descripción de hardware (HDL), como Verilog y VHDL.

Figura 2: Nivel RTL

3.1.1. Elección de herramientas

Después de llevar a cabo un detallado estudio de las herramientas de código abierto más avanzadas y completas en el ámbito del diseño digital, nos enfrentamos a la elección de la herramienta más adecuada para nuestro proyecto. En particular, al considerar el diseño digital a nivel RTL, se han evaluado diversas opciones, como ghdl, iverilog, py verilog y verilator, entre otras.

Uno de los factores clave en esta elección radica en los diferentes lenguajes de descripción hardware que estas herramientas ofrecen, siendo los más prominentes VHDL y Verilog. A la luz de los criterios establecidos en la metodología del proyecto, que involucran la selección de herramientas de código abierto que mejor se adapten a nuestros requisitos, hemos optado por Verilog como nuestra elección principal.

Esta decisión se basa en varios factores que influyen en la eficiencia y la efectividad del proceso de diseño. Verilog, alineado con nuestra investigación, ha demostrado ser una opción robusta y bien respaldada en la comunidad de diseño digital. Además, su sintaxis y su enfoque en el modelado RTL son altamente compatibles con nuestros objetivos.

Sin embargo, es importante destacar que, en caso de que nuestras necesidades específicas requieran el uso de VHDL en lugar de Verilog, nuestra segunda opción sería ghdl, una herramienta igualmente valiosa y sólida en el ámbito del código abierto para descripción hardware.

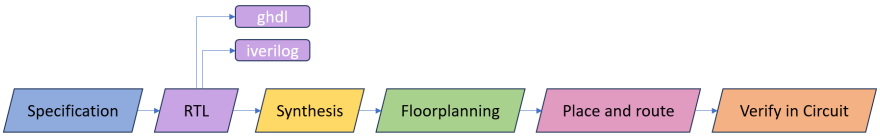

Figura 3: Herramientas HDL

La herramienta Icarus Verilog (iverilog) es una herramienta de simulación de hardware de código abierto ampliamente utilizada para la verificación de circuitos digitales. Con soporte para Verilog, ofrece simulación de nivel de comportamiento y puerta, siendo una elección popular académicamente para el diseño y prueba de circuitos digitales. Disponible bajo la Licencia Pública General de GNU (GPL), cuenta con una comunidad activa y su repositorio de GitHub[74]. Si en lugar de iverilog quisiéramos contemplar el uso de VHDL se recomienda el uso de ghdl, simulador de hardware de código abierto que permite la simulación y el análisis de circuitos digitales descritos en lenguajes de descripción de hardware (HDL) como VHDL. Desarrollado por la comunidad de código abierto, GHDL ofrece una solución valiosa para la verificación y validación de diseños digitales[75].

Para tener un conjunto de herramientas completo en el primer paso del diseño digital, utilizaremos gtk wave, herramienta que nos ayudará con la visualización de gráficos. GTKWave es una poderosa herramienta de visualización de simulaciones que permite observar gráficamente cómo se comportan las señales y los registros en un diseño digital a lo largo del tiempo. Esta capacidad de visualización es fundamental para comprender el funcionamiento del diseño y detectar posibles problemas o anomalías durante la simulación[76].

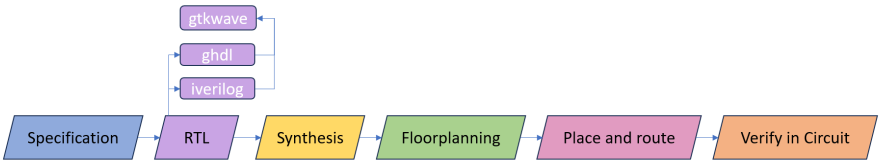

Figura 4: Herramientas HDL+GTKwave

3.1.2. Diseño con iverilog

Concluida la investigación y elección de herramientas, se comenzará un breve proceso de diseño para poner en marcha la metodología seguida.

Es recomendable comprobar que las herramientas están instaladas en el entorno en el que nos encontremos. Si usamos el docker de IIC-OSIC-TOOLS estas herramientas ya vienen instaladas.

```
$ iverilog --version
  ghdl --version
  gtkwave --version
```

En primer lugar, ejecutaremos el testbench, que instancia a nuestro diseño:

```
$ iverilog -o design multiplexor_2to1.v multiplexor_2to1_tb.v
$ vvp design
```

Y ahí está, el programa ha sido ejecutado. El comando 'iverilog', lee e interpreta el archivo fuente, luego genera un resultado compilado. Como el formato 'vvp' es el predeterminado, se ejecuta en la segunda línea interpretando el resultado compilado en "design".

Los comandos 'iverilog' y 'vvp' son los comandos más importantes disponibles para usuarios de Icarus Verilog. El comando 'iverilog' es el compilador, y el comando 'vvp' es el motor de tiempo de ejecución de simulación[77].

Hacemos uso de la herramienta GTKwave para la visualización de las señales:

```
$ gtkwave testbench.vcd
```

```
/foss/designs > iverilog -o design multiplexor_2to1.v multiplexor_2to1_tb.v
/foss/designs > vvp design
VCD info: dumpfile testbench.vcd opened for output.
Prueba 1 pasada
Prueba 2 pasada
multiplexor_2to1_tb.v:44: $finish called at 200 (1us)
/foss/designs > gtkwave testbench.vcd

GTKWave Analyzer v3.4.0 (w)1999-2022 BSI

[0] start time.
[200] end time.
```

Figura 5: Comandos terminal

Y obtenemos los siguientes resultados, haciendo posible la visualización del banco de pruebas gracias a GTKWave:

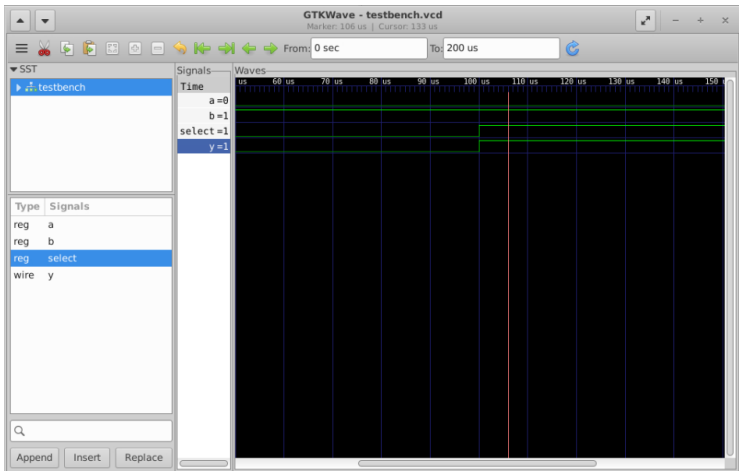

Figura 6: Test bench

3.2. Flujo de RTL a GDSII

3.2.1. Elección de herramientas

Una vez comprobada la funcionalidad de las especificaciones con el banco de pruebas se realizara el flujo de RTL a GDSII. Para ello se hará uso de OpenLane.

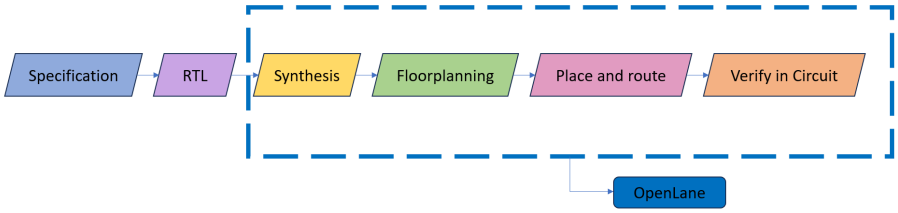

Figura 7: OpenLane

El uso de la herramienta OpenLane se debe a que es un flujo automatizado de diseño de circuitos integrados (ASIC) de nivel RTL a GDSII basado en varios componentes, que incluyen OpenROAD, Yosys, Magic, Netgen, CVC, SPEF-Extractor, KLayout y una serie de scripts personalizados para la exploración y optimización del diseño. Este flujo realiza todos los pasos de implementación de un ASIC, desde la descripción RTL de alto nivel hasta la generación de los archivos en formato GDSII listos para la fabricación. OpenLane tiene como objetivo proporcionar una solución completa de diseño de ASIC, lo que significa que puede llevar un diseño desde la descripción RTL hasta la síntesis, ubicación,

enrutamiento y verificación física para generar un archivo GDSII listo para la fabricación. Esta automatización es fundamental para proyectos de diseño de ASIC, ya que simplifica el proceso y reduce el potencial de errores humanos.

La naturaleza de código abierto de OpenLane fomenta la colaboración y la innovación dentro de la comunidad de diseño de hardware. Permite a los diseñadores personalizar y ampliar la cadena de herramientas para satisfacer sus necesidades específicas. Es especialmente beneficioso para proyectos académicos, de investigación, proyectos de aficionados y pequeñas startups que desean crear diseños de silicio personalizados sin invertir en costosas herramientas propietarias.

El proyecto OpenLane se mantiene activo y se puede acceder a él en GitHub, donde se encuentra el código fuente, documentación y discusiones de la comunidad relacionadas con el proyecto. Si estás interesado en utilizar OpenLane para el diseño de ASIC, la documentación en ReadTheDocs es un recurso valioso para comenzar y comprender cómo utilizar la herramienta de manera efectiva. Además, la integración de OpenLane con GitHub Actions facilita la configuración de flujos de diseño automatizados como parte de tu proceso de desarrollo de hardware.

Etapas de diseño Openlane El flujo de OpenLane consta de varias etapas. De forma predeterminada, todos los pasos del flujo se ejecutan en secuencia. Cada etapa puede constar de múltiples subetapas, aunque OpenLane también puede ejecutarse paso a paso.

Síntesis

yosys/abc Realiza la síntesis RTL y la asignación tecnológica. Yosys es una de las herramientas de síntesis de código abierto más populares en la comunidad de diseño digital y tiene varias características que hacen que sea una elección sólida y apreciada por muchos diseñadores.

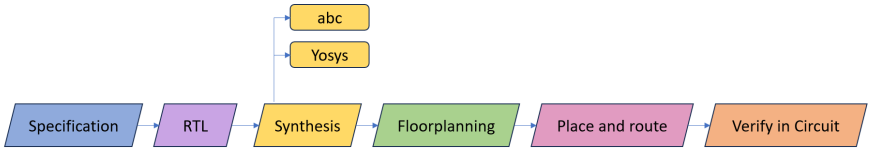

Figura 8: Yosys

Yosys es un marco de síntesis RTL de Verilog que goza de amplia compatibilidad con Verilog-2005 y ofrece un conjunto esencial de algoritmos de síntesis para diversas aplicaciones. Entre sus características destacadas y aplicaciones típicas se encuentran:

1. Procesamiento de diseños sintetizables en Verilog-2005.
2. Conversión de Verilog a diversos formatos, como BLIF, EDIF, BTOR, SMT-LIB, Verilog RTL simple, entre otros.

3. Integración de métodos formales para verificar propiedades y equivalencias.

4. Asignación a bibliotecas de celdas estándar ASIC en formato Liberty.

5. Mapeo a FPGA, incluyendo las series Xilinx 7-Series y Lattice iCE40 y ECP5.

6. Utilización como base y/o front-end para flujos de síntesis personalizados.

Yosys es altamente adaptable y puede llevar a cabo una amplia gama de tareas de síntesis al combinar los pases existentes (algoritmos) mediante scripts de síntesis y agregar pases adicionales según sea necesario, extendiendo el código base de Yosys en C++. Además, Yosys funciona como un backend para varias herramientas que emplean métodos formales para analizar diseños. Por ejemplo, se integra con SBY para la comprobación formal de propiedades basada en solucionadores SMT y con MCY para evaluar la calidad de los bancos de pruebas mediante métricas de cobertura de mutaciones. Yosys se distribuye como software libre y se licencia bajo la licencia ISC, una licencia GPL compatible que se asemeja en términos a la licencia MIT o la licencia BSD de 2 cláusulas. Es relevante destacar que Yosys es el primer software de código abierto completo para la síntesis en Verilog HDL y ofrece soporte para la mayoría de Verilog-2005, habiéndose probado ampliamente con diseños reales en los dominios ASIC y FPGA. A su vez, se introducen diferentes niveles de abstracción en circuitos digitales, desde el nivel de sistema hasta el nivel de puerta física, con definiciones claras y ejemplos para cada uno. Yosys se destaca por su capacidad de leer y procesar la mayoría de código Verilog-2005 moderno y llevar a cabo diversas operaciones en netlist, optimizaciones lógicas y mapeo de puertas con la ayuda de ABC1. Sin embargo, es importante señalar que Yosys no es apto para procesar lenguajes de alto nivel como C, C++, o SystemC, ni para crear diseños físicos mediante el proceso de place and route. En un flujo típico, Yosys se combina con una herramienta de implementación de bajo nivel, como Qflow2, especialmente en diseños ASIC. Los scripts de síntesis controlan Yosys y se componen de tres tipos de comandos: Frontends para leer archivos de entrada (normalmente en Verilog), Passes para realizar transformaciones en el diseño en memoria, y Backends para escribir el diseño en memoria en diversos formatos disponibles, como Verilog, BLIF, EDIF, SPICE, BTOR, entre otros.

OpenSTA Realiza el análisis de temporización estática en la lista de red resultante para generar informes de temporización. El Analizador de Temporización Estática Parallax, conocido como OpenSTA, es una herramienta verificadora de temporización estática a nivel de puerta. Se trata de un ejecutable autónomo que se utiliza para verificar el tiempo de un diseño, haciendo uso de formatos de archivo estándar. Entre los elementos que OpenSTA maneja se incluyen:

- Lista de redes en Verilog.
- Biblioteca Liberty.
- Restricciones de tiempo en formato SDC.
- Anotaciones de retardo SDF.

- Información sobre parásitos en formato SPEF.

OpenSTA hace uso de un intérprete de comandos TCL para leer el diseño, especificar restricciones de tiempo y generar informes relacionados con la temporización. Algunos de los aspectos clave que se pueden abordar con OpenSTA incluyen:

- Análisis de relojes.
- Generación de latencias.
- Evaluación de la latencia de origen (retraso de inserción).
- Medición de incertidumbre.
- Comprobaciones de reloj utilizando compuertas.
- Gestión de relojes de frecuencia múltiple.
- Tratamiento de rutas de acceso de excepción, tales como rutas falsas y multiciclo.
- Cálculo de retardos mínimo y máximo en trayectos.
- Identificación de puntos de excepción a través de diversas opciones.

OpenSTA ofrece un motor de sincronización y puede ser fácilmente integrado con otras herramientas mediante un adaptador de red, lo que le permite acceder al host netlist sin duplicar datos innecesariamente. Ofrece capacidades de actualización incremental basadas en consultas de retrasos, llegadas y tiempos requeridos. También incluye un simulador para propagar constantes a partir de restricciones y valores de red de alto o bajo. Es importante destacar que OpenSTA tiene una doble licencia. Se publica bajo la Licencia Pública General de GNU (GPL) versión 3 como OpenSTA, y también cuenta con una licencia para aplicaciones comerciales otorgada por Parallax Software, la cual no impone los requisitos de la GPL.

Floorplanning

init fp Define el área central para el macro, así como las filas (utilizadas para la ubicación) y las pistas (utilizadas para el enrutamiento).

ioplacer Coloca las entradas y salidas del macro.

pdngen Genera la red de distribución de energía.

tapcell Inserta celdas welltap y decap en el diseño de la distribución.

Placement

RePLace Realiza la ubicación global.

Resizer Realiza optimizaciones opcionales en el diseño.

OpenDP Realiza la ubicación detallada para legalizar los componentes ubicados globalmente.

Diseño de Árbol de Reloj (CTS)

TritonCTS Sintetiza la red de distribución de reloj (árbol de reloj).

Routing

FastRoute Realiza el enrutamiento global para generar un archivo de guía para el enrutador detallado. TritonRoute Realiza el enrutamiento detallado.

OpenRCX Realiza la extracción SPEF. Extracción de parásitos El módulo de extracción de parásitos en OpenROAD () se basa en el método OpenRCX de código abierto, una herramienta de extracción parásita (PEX o RCX) que funciona en las API de diseño de OpenDB. Extrae diseños enrutados basados en el modelo de diseño LEF/DEF.rcx OpenRCX extrae tanto la resistencia como la capacitancia de los cables, basándose en el acoplamiento distancia al cable más cercano y el contexto de densidad de vía por encima y/o por debajo del cable de interés, así como celular Resúmenes. Las mediciones de capacitancia y resistencia se basan en ecuaciones de la distancia de acoplamiento interpolada en mediciones exactas de una calibración archivo, denominado archivo de reglas de extracción. El archivo de reglas de extracción (tecnología RC archivo) se genera una vez para cada nodo de proceso y esquina, utilizando una utilidad proporcionada para la generación de patrones de cables DEF y el modelado de regresión. OpenRCX almacena la resistencia, la capacitancia de acoplamiento y la capacitancia a tierra (es decir, conectada a tierra) en objetos OpenDB con punteros directos al cable asociado y a través de db Objetos. Opcionalmente, OpenRCX puede generar un archivo .spef.

Tapeout

Magic Genera el archivo de diseño GDSII final a partir del archivo def enrutado. Magic VLSI es una herramienta de diseño y verificación en el campo de la ingeniería de circuitos integrados (VLSI, por sus siglas en inglés). Es un popular software de código abierto que se utiliza para el diseño y la edición de circuitos integrados y dispositivos electrónicos a nivel de diseño físico. Aquí tienes algunas características y usos típicos de Magic VLSI:

1. Edición de Diseño: Magic VLSI proporciona una interfaz gráfica que permite a los ingenieros de circuitos integrados crear, editar y visualizar el diseño físico de un circuito integrado. Esto incluye la disposición de transistores, puertas lógicas, interconexiones y otros elementos en el chip.

2. Diseño a Nivel de Celdas Estándar: Magic VLSI se utiliza comúnmente para diseñar circuitos a nivel de celdas estándar, lo que permite a los diseñadores crear bloques funcionales que se pueden reutilizar en múltiples proyectos.

3. Verificación de Reglas de Diseño: La herramienta verifica automáticamente si el diseño cumple con las reglas de diseño y restricciones específicas, ayudando a prevenir errores antes de que se fabrique el chip.

4. Dibujo de Máscaras: Magic VLSI es capaz de generar máscaras que se utilizan en el proceso de fabricación de circuitos integrados. Las máscaras son patrones fotográficos que se utilizan para transferir el diseño al sustrato de silicio.

5. Integración con Flujos de Diseño: Puede integrarse con otros programas y flujos de diseño de circuitos integrados para realizar tareas adicionales, como simulación, extracción de parámetros, enrutamiento y más.

Magic VLSI es una herramienta ampliamente utilizada en la industria de semiconductores y en el ámbito académico para el diseño y la verificación de circuitos integrados. Proporciona una solución de código abierto que es especialmente útil para proyectos de diseño a pequeña y mediana escala en el campo de la VLSI.

KLayout Genera una copia de seguridad del archivo de diseño GDSII final a partir del archivo def enrutado. 25 Verificación Final KLayout es una popular herramienta de código abierto para la visualización y edición de diseños de circuitos integrados (CI) y máscaras utilizadas en el proceso de fabricación de semiconductores. Esta herramienta se utiliza principalmente en el campo de la ingeniería de circuitos integrados (VLSI) y es muy apreciada por los profesionales en este campo. Aquí tienes algunas características y usos clave de KLayout:

1. Visualización de Diseños: KLayout permite cargar y visualizar diseños de circuitos integrados en varios formatos, como GDSII, OASIS, CIF y otros formatos comunes utilizados en la industria de semiconductores. Los diseñadores pueden inspeccionar con detalle las estructuras y componentes de los circuitos integrados.

2. Edición de Diseños: La herramienta permite realizar ediciones en los diseños, como añadir, eliminar o modificar elementos. Esto es útil para ajustar o corregir detalles en el diseño.

3. Verificación de Reglas: KLayout puede verificar si un diseño cumple con las reglas de diseño específicas, lo que es crucial para garantizar que el circuito integrado funcione correctamente y se pueda fabricar con éxito.

4. Generación de Máscaras: La herramienta es capaz de generar máscaras que se utilizan en el proceso de fabricación de circuitos integrados. Las máscaras son patrones fotográficos que se utilizan para transferir el diseño al sustrato de silicio.

5. Automatización: KLayout es altamente personalizable y se puede utilizar en scripts para automatizar tareas de diseño y verificación.

6. Soporte para Lenguajes de Programación: KLayout admite scripts escritos en lenguajes de programación como Python y Ruby, lo que permite a los usuarios personalizar y extender su funcionalidad.

Signoff

Magic Realiza comprobaciones DRC y comprobaciones de antena.

KLayout Realiza comprobaciones DRC.

Netgen Realiza comprobaciones LVS. Netgen es una herramienta de software de código abierto utilizada en el campo del diseño de circuitos integrados (IC) y en particular en el proceso de diseño de circuitos digitales. Su función principal es realizar la extracción de redes desde un diseño de circuito en un formato específico y generar una representación del mismo en un formato más adecuado para el análisis, la simulación y la verificación. Aquí tienes algunos puntos clave sobre Netgen:

1. Extracción de Redes: Netgen se utiliza para extraer información de la topología del diseño de circuito, incluyendo conexiones entre componentes, pines, y nodos, y luego representar esta información en un formato comprensible y útil.
2. Formatos de Entrada y Salida: Puede aceptar diseños en varios formatos de entrada comunes, como Verilog o VHDL, y generar representaciones en formatos de salida utilizados por herramientas de simulación y verificación, como SPICE.
3. Verificación: Netgen es fundamental para la verificación de diseños de circuitos digitales, ya que permite detectar problemas de conexión, cortocircuitos y otros errores que podrían afectar el rendimiento del circuito.
4. Simulación: Después de la extracción de redes, la representación generada por Netgen se puede utilizar en herramientas de simulación para evaluar el comportamiento del circuito y garantizar su funcionamiento correcto.
5. Flujo de Diseño de Circuitos Integrados: Netgen es parte del flujo de diseño de circuitos integrados y a menudo se integra con otras herramientas de diseño y verificación de circuitos para garantizar la consistencia y la precisión del diseño.

CVC Realiza comprobaciones de validez del circuito y ERC con reconocimiento de voltaje para listas de redes CDL. Funciones:

- El formato de entrada de la lista de redes es Calibre LVS CDL
- Comprueba listas de red con hasta 4 B de dispositivos.
- Parámetros de alimentación y dispositivo de Microsoft Excel
- Posibilidad de archivos de energía jerárquicos
- Capacidad de diferenciar modelos por parámetros
- Opción de configuración para listar modelos y redes eléctricas
- Todas las reglas están automatizadas. No es necesario escribir archivos de reglas.

- Analizador interactivo de netlist
- Ejecución de scripts disponible
- Creación automática de entornos de depuración de subcircuitos
- Interfaz gráfica de usuario para registrar los resultados de los análisis de errores.
- Aunque usaremos OpenLane de manera directa, antes de ello estudiaremos en profundidad e independiente al flujo OpenLane la herramienta Yosys dada su relevancia en el Open Source.

3.2.2. Síntesis con Yosys

Después de haber realizado el diseño RTL (Registro de Transferencia de Nivel) de un circuito digital y elegido OpenLane para cumplimentar el resto del flujo de diseño dedicaremos una sección a la herramienta Yosys, usada como herramienta backend en OpenLane, aunque también de gran relevancia de manera independiente.

Llevándolo a la práctica, en el entorno de IIC-OSIC-TOOLS realizaremos la síntesis lógica de nuestro diseño:

1. En primer lugar, iniciamos yosys:

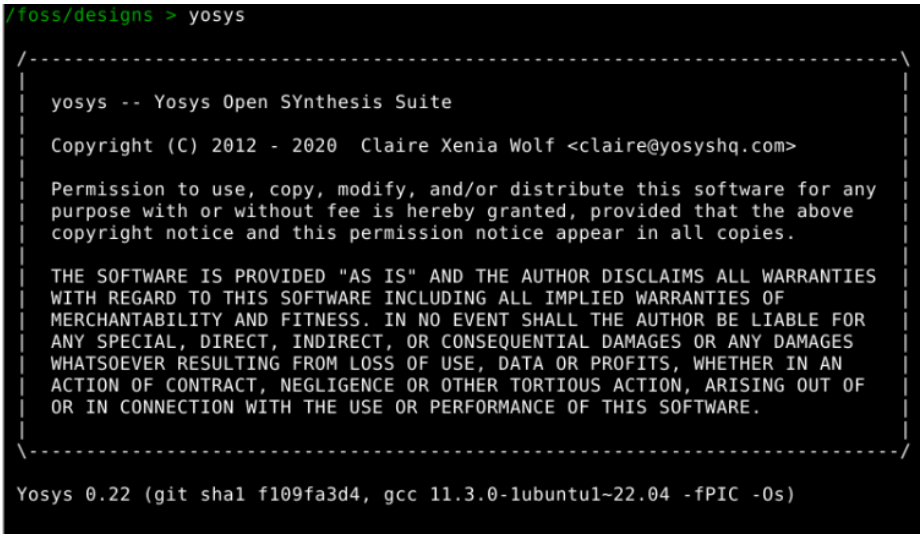

Figura 9: yosys en terminal

2. En segundo lugar, leemos nuestro diseño, seleccionamos la tecnología y generamos los archivos de síntesis:

```
$ read_verilog multiplexor_2to1.v
$ show
$ techmap
$ show
```

```
$ write_verilog synth1.v
$ write_verilog -noattr synth1.v
$ abc -liberty sky130_fd_sc_hd__ff_100C_1v65.lib
$ write_verilog -noattr synth2.v
```

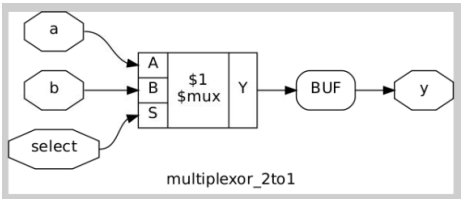

Figura 10: Tech map

Obteniendo así los siguientes archivos de síntesis, synth1.v:

```
/* Generated by Yosys 0.22 (git sha1 f109fa3d4, gcc
    11.3.0-1ubuntu1~22.04 -fPIC -Os) */

module multiplexor_2to1(a, b, select, y);
  wire _0_;
  input a;
  wire a;
  input b;
  wire b;
  input select;
  wire select;
  output y;
  wire y;
  assign _0_ = select ? b : a;
  assign y = _0_;
endmodule
```

y synth2:

```
/* Generated by Yosys 0.22 (git sha1 f109fa3d4, gcc
    11.3.0-1ubuntu1~22.04 -fPIC -Os) */

module multiplexor_2to1(a, b, select, y);
  wire _0_;
  wire _1_;
  wire _2_;
  wire _3_;
  wire _4_;
  input a;
  wire a;
  input b;
  wire b;
  input select;
  wire select;
  output y;
  wire y;
  sky130_fd_sc_hd__mux2_1 _5_ (
    .A0(_0_),
    .A1(_1_),
    .S(_2_),
    .X(_3_)
  );
  assign y = _4_;
  assign _0_ = a;
  assign _1_ = b;
  assign _2_ = select;
  assign _4_ = _3_;
endmodule
```

3.2.3. Diseño con Openlane

Una vez comprobadas las especificacioens de diseño con el banco de pruebas, estudiado el flujo de diseño OpenLane y Yosys en profundidad el siguiente paso es iniciar el diseño con OpenLane. Para ello se deben seguir los pasos del *Anexo C* para su instalación o bien hacer uso de OpenLane desde el entorno de IIC-OSIC-TOOLS.

Para ello se seguira con el proceso de diseño del multiplexor: En primer lugar, usaremos OpenLane de manera secuencial, generando el flujo de diseño al completo:

```
cd OpenLane
make mount
OpenLane Container (903a86a):/openlane$ flow.tcl -design
    ↪ multiplexor_2to1 -src multiplexor_2to1/src -
    ↪ init_design_config

OpenLane Container (903a86a):/openlane/openlane$ flow.tcl -design
    ↪  multiplexor_2to1 -tag first_run -overwrite
```

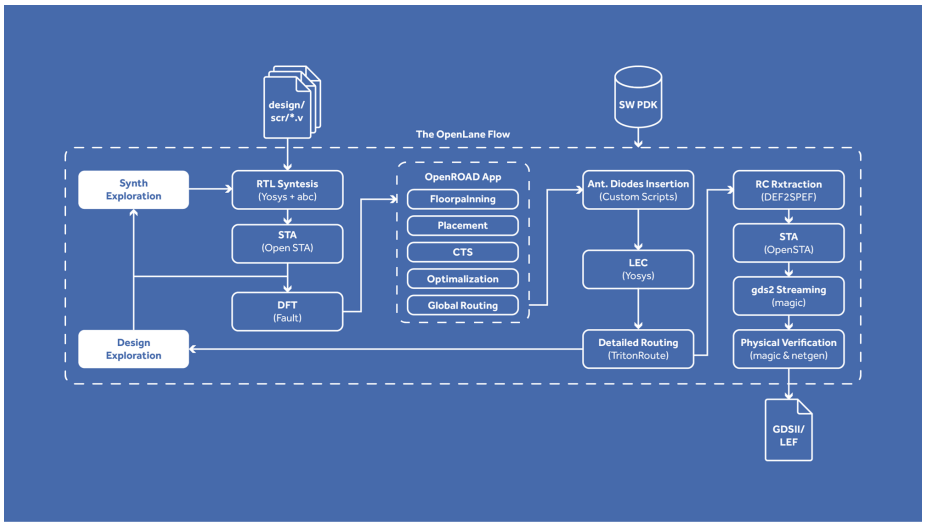

Figura 11: OpenLane Flow

Generándose así el archivo de configuración (config.json o config.tcl) de forma predeterminada:

```
{
    "DESIGN_NAME": "multiplexor_2to1",
    "VERILOG_FILES": "dir::src/*.v",
    "RUN_CTS": false,
    "CLOCK_PORT": clock,
    "CLOCK_PERIOD": clock,
}
```

Este archivo nos ayudará a adaptarnos a las condiciones de nuestro diseño. Por lo que es necesario conocer las variables de flujo configurables por el usuario y sus valores predeterminados.

Variable	Descripción
PDK	Especifica el kit de diseño de procesos (PDK). (Por defecto: sky130A)
DESIGN_NAME	El nombre del módulo de nivel superior del diseño
VERILOG_FILES	La ruta de acceso de los archivos Verilog del diseño, proporcionada como una matriz de archivos en JSON o una lista de archivos delimitada por espacios en blanco en Tcl. Los archivos se evalúan en orden, es decir, si el archivo B depende del archivo A, el archivo A debe aparecer primero.
CLOCK_PERIOD	El período de reloj utilizado para los relojes en el diseño, en nanosegundos.
CLOCK_PORT	El nombre del puerto de reloj del diseño.
CLOCK_NET	El nombre de la entrada neta al búfer de reloj raíz. (Por defecto: CLOCK_PORT)
STD_CELL_LIBRARY	Especifica la biblioteca de células estándar que se va a utilizar en el PDK especificado. (Por defecto: sky130_fd_sc_hd)
STD_CELL_LIBRARY_OPT	Especifica la biblioteca de celdas estándar que se utilizará durante las optimizaciones del cambio de tamaño. (Por defecto: STD_CELL_LIBRARY)
PDK_ROOT	Especifica la ruta de acceso de la carpeta del PDK.
DIODE_PADDING	Número de sitios en el panel izquierdo durante la colocación detallada. (Predeterminado: sitios)
MERGED_LEF	Apunta a , que es una fusión de varios archivos LEF, incluida la tecnología lef, cells lef, cualquier lefs personalizado y lefs de E/S.
NO_SYNTH_CELL_LIST	Especifica el archivo que contiene la lista "don't-use-cell"que se va a excluir del archivo liberty durante la síntesis.
DRC_EXCLUDE_CELL_LIST	Especifica el archivo que contiene la lista "don't-use-cell-list"que se va a excluir del archivo liberty durante la síntesis y las optimizaciones de temporización.
BASE_SDC_FILE	Especifica el archivo SDC base que se utiliza durante el flujo.
PNR_SDC_FILE	Especifica el archivo SDC utilizado durante todas las fases de implementación (PnR).

Variable	Descripción
VERILOG_FILES_BLACKBOX	Archivos Verilog de caja negra en los que se ignora la implementación. Útil para macros pre-endurecidas que incorporas a tu diseño, utilizadas durante la síntesis y opensta. Se puede agregar a un archivo para omitir ese archivo mientras se hace sta. Esto hará una caja negra de todos los módulos definidos dentro de ese archivo. Se recomienda proporcionar una lista de redes a nivel de puerta siempre que sea posible para hacer el stato completo./// sta-blackbox
EXTRA_LEFS	Especifica los archivos LEF de macros preprotegidas que se utilizan en el diseño actual, que se utilizan en la colocación y el enrutamiento.
EXTRA_LIBS	Especifica los archivos LIB de macros preprotegidas que se utilizan en el diseño actual, que se utilizan durante el análisis de temporización. (Opcional)
EXTRA_GDS_FILES	Especifica los archivos GDS de macros preprotegidas que se utilizan en el diseño actual, que se utilizan durante la salida de cinta.

Además de las variables de configuración de PDK:

Variable	Descripción
FP_PDN_RAIL_OFFSET	Define el desfase de carril para met1 utilizado en PDN. (Ejemplo: 0)
FP_PDN_HSPACING	El espaciado entre el par horizontal de alimentación/tierra (predeterminado: 1.7)
FP_PDN_VSPACING	El espaciado entre el par vertical de alimentación/-tierra (predeterminado: 1.7)
FP_PDN_VOFFSET	El desplazamiento de las regletas verticales en la capa metálica 5 de la red de distribución de energía (predeterminado: 16.32)
FP_PDN_VPITCH	El paso de las regletas verticales en la capa metálica 4 de la red de distribución de energía (predeterminado: 153.6)
FP_PDN_HOFFSET	El desplazamiento de las regletas horizontales en la capa metálica 5 de la red de distribución de energía (predeterminado: 16.65)
FP_PDN_HPITCH	El paso de las regletas horizontales en la capa metálica 5 de la red de distribución de energía (predeterminado: 153.18)
FP_PDN_VWIDTH	Define el ancho de correa para la capa vertical utilizada en PDN. (Ejemplo: 1.6)
FP_PDN_HWIDTH	Define la anchura de la correa para la capa horizontal utilizada en PDN. (Ejemplo: 1.6)
FP_PDN_CORE_RING_VWIDTH	Define la anchura vertical de la capa vertical utilizada para crear el anillo central en la PDN. (Ejemplo: 20)
FP_PDN_CORE_RING_HWIDTH	Define la anchura horizontal de la capa horizontal utilizada para crear el anillo central en la PDN. (Ejemplo: 20)
FP_PDN_CORE_RING_VSPACING	Define el espaciado de la capa vertical utilizada para crear el anillo central en la PDN. (Ejemplo: 5)

Variable	Descripción
FP_PDN_CORE_RING_HSPACING	Define el espaciado de la capa horizontal utilizada para crear el anillo central en la PDN. (Ejemplo: 5)
FP_PDN_CORE_RING_VOFFSET	Define el desplazamiento de la capa vertical utilizada para crear el anillo central en la PDN. (Ejemplo: 20)
FP_PDN_CORE_RING_HOFFSET	Define el desplazamiento de la capa horizontal utilizada para crear el anillo central en la PDN. (Ejemplo: 20)
GRT_LAYER_ADJUSTMENTS	Reducciones específicas de la capa en la capacidad de enrutamiento de los bordes entre las celdas en el gráfico de enrutamiento global, delimitadas por comas. Los valores oscilan entre 0 y 1. (Ejemplo: 0.99,0,0,0,0,0)
RT_MIN_LAYER	La capa metálica más baja sobre la que se va a enrutar. (Ejemplo: met1)
RT_MAX_LAYER	La capa metálica más alta sobre la que se va a enrutar. (Ejemplo: met5)
WIRE_LENGTH_THRESHOLD	Un valor en micras por encima del cual las longitudes de cable generan advertencias y, si se establece, el flujo se producirá un error. Si un PDK no establece este valor, se considera que el valor es infinito. (Opcional)

En este ejemplo concreto modificamos el archivo de configuración con las siguientes variables, dado que el ejemplo carece de puerto de reloj:

```
{
{
    "DESIGN_NAME": "multiplexor_2to1",
    "VERILOG_FILES": "dir::src/*.v",
    "RUN_CTS": false,
    "CLOCK_PORT": null,
    "PL_RANDOM_GLB_PLACEMENT": true,
    "FP_SIZING": "absolute",
    "DIE_AREA": "0 0 34.5 57.12",
    "PL_TARGET_DENSITY": 0.75,
    "FP_PDN_AUTO_ADJUST": false,
    "FP_PDN_VPITCH": 25,
    "FP_PDN_HPITCH": 25,
    "FP_PDN_VOFFSET": 5,
    "FP_PDN_HOFFSET": 5,
    "DIODE_INSERTION_STRATEGY": 3
}
}
```

Una vez generado, se obtiene el flujo de diseño completo, generándose así los siguientes archivos:

Directorio de trabajo:

Figura 12: Config

Dentro de la carpeta runs ubicamos el primer flujo completo de diseño generado (runs/first_run):

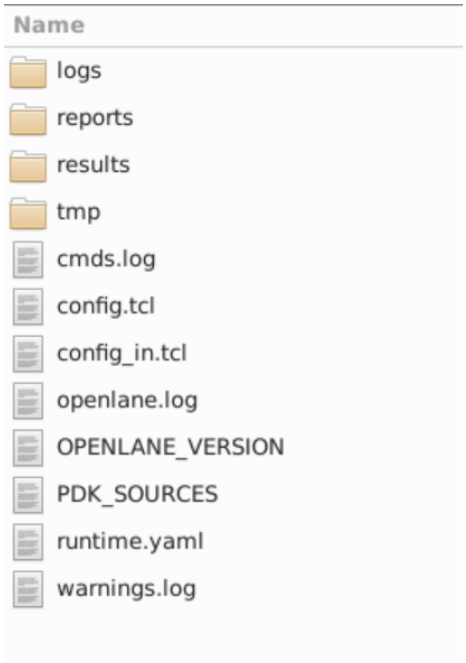

Figura 13: First run

Y se pueden apreciar los distintos resultados obtenidos de cada bloque del flujo de diseño definido:

Figura 14: Results

Ya generados, se analizarán los distintos resultados comenzando por el primer bloque del flujo. Centrándonos en la síntesis se analizaran los resultados obtenidos a través de las herramientas yosys y OpenSTA: Se generan dos archi-

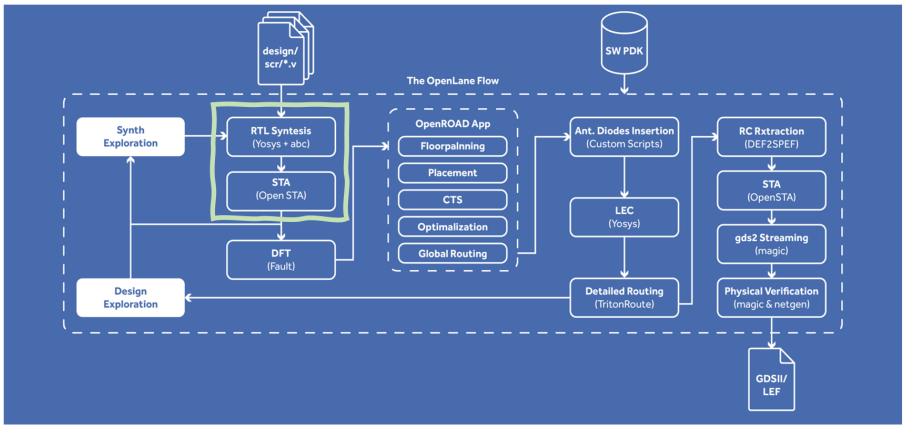

Figura 15: Síntesis

vos que contienen información sobre las características de la síntesis y el diseño del ejemplo inicial (.sdf y .v). Algunas observaciones clave sobre cada uno de los archivos: Archivo SDF (Standard Delay Format):

1. SDFVERSION: indica la versión del formato SDF utilizado.

2. DESIGN: muestra el nombre del diseño, que es “multiplexor_2to1”.

3. DATE: registra la fecha en la que se generó el archivo SDF.

4. VENDOR especifica el nombre del proveedor de la herramienta utilizada para generar el archivo (en este caso, “Parallax”).

5. PROGRAM muestra el nombre del programa que generó el archivo SDF (“STA”se refiere a Static Timing Analysis).

6. VERSION: indica la versión de la herramienta utilizada.

7. VOLTAGE y PROCESS: muestran información sobre la tensión y el proceso utilizado.

8. TEMPERATURE: especifica la temperatura de funcionamiento.

9. TIMESCALE: establece la escala de tiempo utilizada en el archivo SDF (1ns en este caso).

A continuación, se define la información de retraso para las celdas del diseño, incluyendo las rutas de interconexión y sus características de retraso.

Archivo de Módulo Verilog:

El archivo Verilog muestra la descripción del diseño en el lenguaje de programación HDL (Hardware Description Language). Aquí algunas observaciones:

1. El diseño se llama “multiplexor_2to1τ se define como un módulo con entradas a, b, select, y una salida y.

2. Se definen cables (wires) _0_, a, b, select, y y para conectar las entradas y salidas.

3. Se instancian dos celdas de diseño del tipo “sky130_fd_sc_hd__mux2_2τ “sky130_fd_sc_hd__buf_1çon nombres _1_ y _2_. Estas celdas son parte del diseño y se conectan a las entradas y salidas del módulo.

4. El multiplexor (MUX) “sky130_fd_sc_hd__mux2_2”se conecta a las entradas a, b, y select, y su salida se conecta al cable _0_.

5. El buffer sky130_fd_sc_hd__buf_1 se conecta a _0_ y su salida se conecta a la salida y.

En resumen, el archivo SDF proporciona información detallada sobre los retrasos y características de temporización del diseño, mientras que el archivo de módulo Verilog describe la estructura del diseño y cómo se interconectan las celdas. Estos archivos son esenciales para el proceso de diseño y análisis de circuitos digitales.

A continuación, seguimos con el floorplaning donde obtenemos dos archivos (.def y .odb):

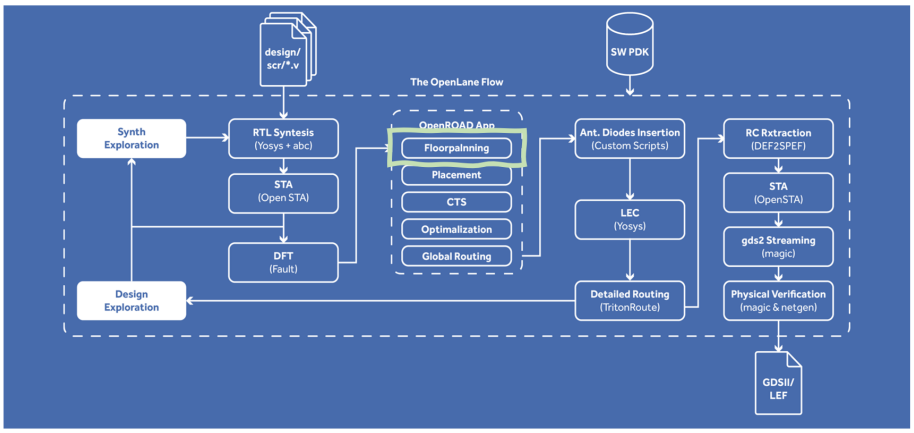

Figura 16: Floorplanning

El archivo .def generado es un archivo de descripción de piso (floorplan). Este archivo se utiliza para describir la organización física del diseño en un chip o placa y contiene información sobre las ubicaciones y tamaños de las diferentes celdas y elementos en el diseño.

A continuación, desglosaré algunos de los elementos clave en el archivo de descripción de piso:

1. VERSION 5.8: Esto indica la versión del formato del archivo.
2. DIVIDERCHAR /": Define el carácter utilizado como separador.
3. BUSBITCHARS "[]": Define los caracteres utilizados para representar buses.
4. DESIGN multiplexor_2to1: Especifica el nombre del diseño que se está describiendo.
5. UNITS DISTANCE MICRONS 1000: Establece la unidad de distancia en micrones.
6. DIEAREA: Define el área total del chip o placa en coordenadas (0,0) para la esquina inferior izquierda y (34500, 57120) para la esquina superior derecha.
7. ROW: Describe las filas en el diseño, con detalles como nombre, orientación (N para norte, FS para flip-south, etc.), origen y dimensiones. Las filas se utilizan para organizar y posicionar las celdas del diseño.
8. TRACKS: Define las pistas (tracks) en el diseño para diferentes capas (layers), especificando la dirección, el espaciado y la capa utilizada.
9. VIAS: Enumera las vías utilizadas en el diseño, junto con detalles sobre el tamaño, ubicación y restricciones.
10. COMPONENTS: Enumera las celdas (componentes) utilizadas en el diseño junto con sus detalles, como nombre, tipo y posición.

11. PINS: Describe los pines utilizados en el diseño, especificando su nombre, dirección (entrada, salida, entrada/salida) y ubicación en las diferentes capas.

12. SPECIALNETS: Enumera las redes especiales utilizadas en el diseño junto con detalles sobre su nombre, pines asociados y conexiones.

13. NETS: Enumera las redes (conexiones) entre componentes, especificando los nombres de las redes y las conexiones de los pines.

En general, el archivo de descripción de piso es fundamental para el diseño físico y la disposición de las celdas en el chip o placa. Contiene información sobre la organización y la ubicación de las celdas, las capas utilizadas, las vías, las filas y otros elementos que son esenciales para la fabricación y el diseño de chips electrónicos.

Seguidamente con el placement: En la etapa de placement (colocación) en el

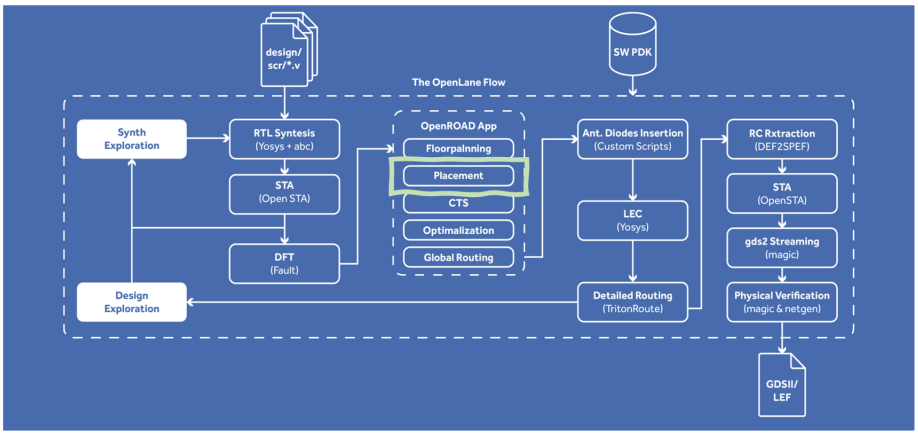

Figura 17: Placement

diseño de circuitos integrados, se pueden encontrar varios tipos de archivos que desempeñan roles específicos:

1. .def y .odb del floorplan: conteniendo la información descrita anteriormente.

2. Archivo `nl.v` (Netlist Lógico):

   - El módulo `multiplexor_2to1` representa un multiplexor de 2 a 1, que toma tres entradas (`a`, `b`, `select`) y produce una salida (`y`).
   - Utiliza componentes lógicos estándar de SkyWater 130 nm, como el multiplexor `sky130_fd_sc_hd__mux2_1`.
   - También incluye buffers de reloj `sky130_fd_sc_hd__clkbuf_1` para acondicionar las señales, así como componentes de desacoplamiento `sky130_fd_sc_hd__decap_3` para gestionar la energía de la capa física.
   - El módulo `multiplexor_2to1` no tiene información específica sobre la ubicación de estos componentes en el chip en este archivo.

3. Archivo `pnl.v` (Post-Place and Route Netlist):

   - El módulo `multiplexor_2to1` es idéntico al del archivo `nl.v`, pero con la adición de dos entradas, `VPWR` y `VGND`, que representan las conexiones de suministro de energía y tierra (VDD y VSS).
   - Utiliza los mismos componentes lógicos y estructura que el archivo `nl.v`, pero con conexiones físicas y suministro de energía específicos para la ubicación física de estos componentes en el chip.
   - Los componentes de desacoplamiento `sky130_fd_sc_hd__decap__3` se configuran para reflejar su ubicación y conexiones en la disposición física.
   - Este archivo `pnl.v` es esencial para la generación de máscaras y la fabricación del circuito integrado, ya que refleja la ubicación física de los componentes.

   En resumen, ambos archivos describen el mismo módulo lógico `multiplexor_2to1`, pero el archivo `pnl.v` proporciona detalles específicos de ubicación y conexión física para permitir la fabricación del circuito integrado, mientras que el archivo `nl.v` se enfoca en la descripción lógica abstracta del diseño.

Routing y optimización, dado que el CTS esta desactivado:

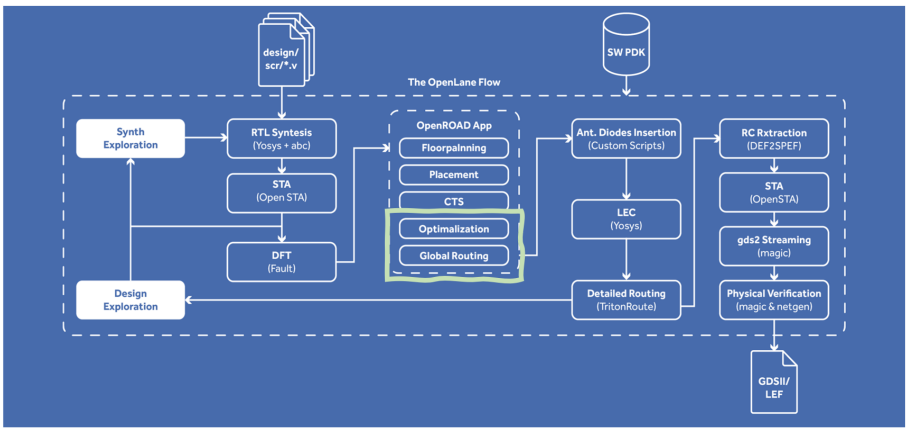

Figura 18: Routing

En la etapa de enrutamiento del diseño de circuitos integrados, se encuentran varios archivos y carpetas críticos para garantizar la calidad y el rendimiento del circuito. Estos incluyen los archivos anteriores y además, archivos relacionados con las condiciones de *process corner*, específicamente *max process corner*, *min process corner*, y *nom* (condiciones máximas, mínimas y nominales).

- **Process Corner**: Los archivos correspondientes a *process corner* representan condiciones extremas de fabricación, ya sea en su peor momento (*max process corner*) o en su mejor situación (*min process corner*). Estos archivos se utilizan para verificar que el diseño sea robusto y funcional bajo condiciones desafiantes o favorables, respectivamente.
- **Nom** (Nominales): Estos archivos describen el diseño bajo condiciones de funcionamiento normales o promedio. Se utilizan para asegurarse de que el circuito integrado funcione de manera confiable en situaciones típicas, sin condiciones extremas.

Además, en la etapa de enrutamiento también se manejan los archivos de sincronización y características parasitarias:

- **SDF** (Standard Delay Format): El archivo SDF detalla los retardos temporales en el diseño y proporciona información esencial para analizar y garantizar la sincronización adecuada de las señales en el circuito. Ayuda a verificar que las señales se sincronicen adecuadamente.
- **SPEF** (Standard Parasitic Exchange Format): El archivo SPEF describe las características parasitarias, como resistencias y capacitancias, que surgen de la disposición física del chip. Estos archivos son esenciales para comprender cómo los componentes físicos afectan el rendimiento del diseño y garantizar que las señales se propaguen de manera eficiente.

En conjunto, estos archivos y carpetas desempeñan un papel crucial en la caracterización y verificación del diseño en diversas condiciones. Aseguran que

el circuito integrado sea funcional, eficiente y confiable, ya sea bajo condiciones extremas, típicas o en términos de sincronización y características físicas. Su gestión adecuada es fundamental para el éxito del proyecto de diseño de CI.

Por último, en la etapa de signoff: Dentro de la carpeta de "signoff", se

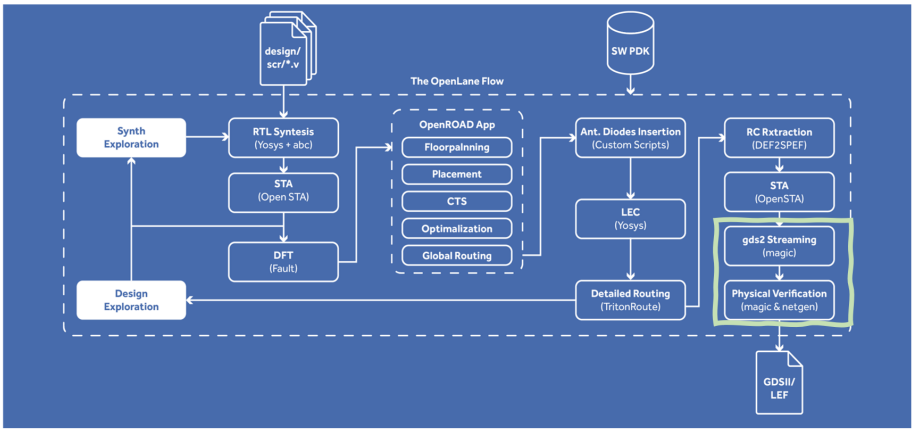

Figura 19: Signoff

encuentran varios archivos críticos que son fundamentales para la fase de verificación y preparación del diseño de circuitos integrados:

- **Archivos .mag (Magic)**: Los archivos .mag son específicos del software de diseño de circuitos Magic. Estos archivos contienen información sobre el diseño del circuito en un formato propietario. Magic es una herramienta ampliamente utilizada para el diseño de circuitos integrados y, en esta etapa, se pueden utilizar para verificar y analizar el diseño en detalle.
- **Archivos GDS (Graphic Data System)**: Los archivos GDS son archivos que contienen datos gráficos del diseño del circuito integrado en formato estándar GDSII. Estos archivos son esenciales para la creación de máscaras y la fabricación del circuito integrado. Proporcionan información detallada sobre la disposición física de los componentes y las conexiones.
- **Archivos SDF (Standard Delay Format)**: Los archivos SDF, que mencionamos anteriormente, son fundamentales para describir los retardos temporales en el diseño. Estos archivos son utilizados para garantizar la sincronización y el rendimiento adecuado de las señales en el circuito integrado.
- **Archivos SPICE**: Los archivos SPICE contienen descripciones de circuitos en formato SPICE (Simulation Program with Integrated Circuit Emphasis). Estos archivos se utilizan para realizar simulaciones y análisis de circuitos. Proporcionan información detallada sobre el comportamiento eléctrico de los componentes del diseño.
- **Archivos Kalayout**: Los archivos Kalayout pueden contener información específica de diseño relacionada con la herramienta KLayout, que es una herramienta de diseño de circuitos integrados y visualización de disposición

física. Estos archivos son útiles para verificar y analizar la disposición física del diseño.

Estos archivos desempeñan un papel crítico en la fase de "signoff", donde se verifica y valida que el diseño del circuito integrado cumple con todos los requisitos y está listo para la fase de fabricación. La correcta gestión y análisis de estos archivos son esenciales para garantizar un circuito integrado funcional y de alta calidad.

En la carpeta final del proceso de diseño de circuitos integrados se encuentran los siguientes archivos:

- .def (Design Exchange Format)
- .gds (Graphic Data System)
- .lef (Library Exchange Format)
- .lib (Library)
- .mag (Magic)
- .maglef (Magic Layout Exchange Format)
- .sdc (Synopsys Design Constraints)
- .sdf (Standard Delay Format)
- .spef (Standard Parasitic Exchange Format)
- .spi (Signal Pin Information)
- Archivos Verilog

Se utilizan para representar la disposición física y lógica del diseño, las restricciones de diseño, la información de temporización, las capacidades y resistencias parasitarias, y la descripción del hardware. La visualización final del diseño se puede realizar a través de KLayout para verificar su integridad antes de la fase de fabricación.

```
# Open the spm.gds using KLayout with sky130 PDK
klayout -e -nn $PDK_ROOT/sky130A/libs.tech/klayout/tech/sky130A.
    ↪ lyt \
   -l $PDK_ROOT/sky130A/libs.tech/klayout/tech/sky130A.lyp \
   ./openlane/multiplexor_2to1/runs/first_run/results/final/gds/
       ↪ multiplexor_2to1.gds
```

Resultado final:

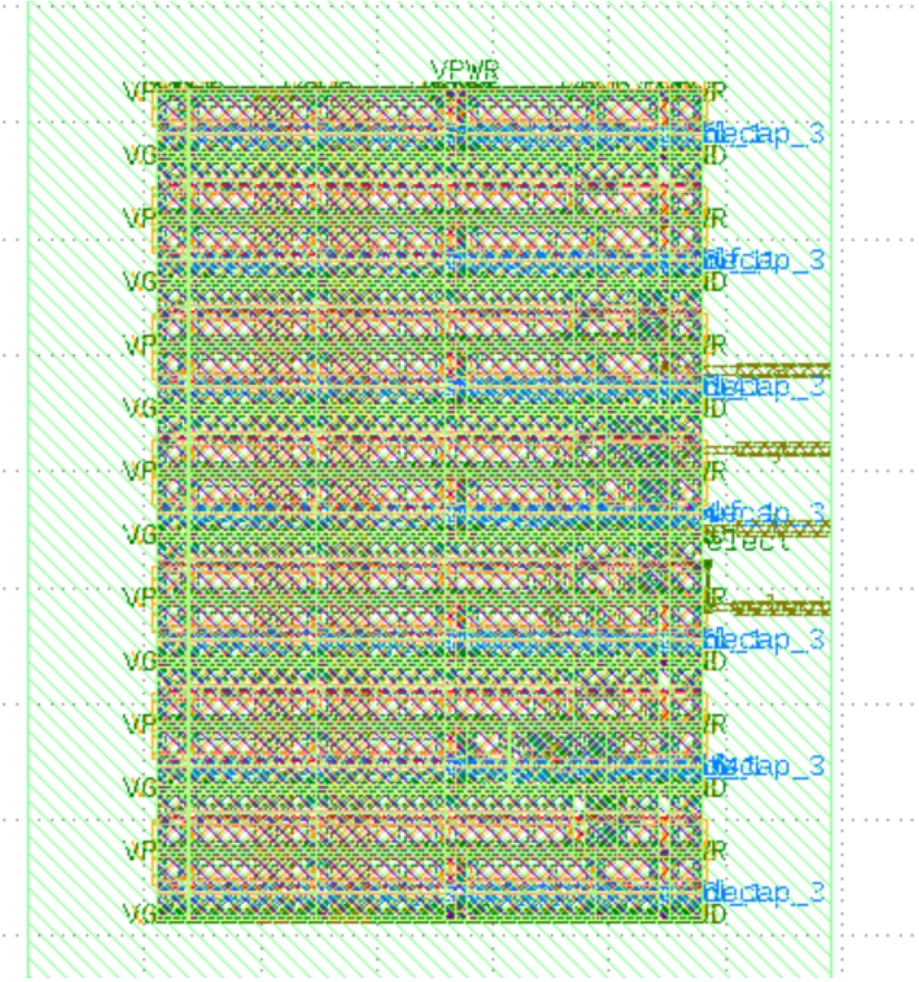

Figura 20: Final

4. Investigación de flujos de diseño analógicos en código abierto

Finalizado el diseño digital en código abierto se abren las puertas a la investigación de distintos flujos de diseño en el ámbito del diseño analógico y mixto. Para ello basándonos en la investigación inicial realizada y los entornos disponibles se establecerá un flujo de diseño para circuitos analógicos:

- Xschem para el diseño de los esquemáticos[70].
- Ngspice/Xyce para las simulaciones[45][71].
- Xschem/gaw3/gnuplot para la representación de las simulaciones[32].
- Magic para la implementación del diseño[43].
- Netgen para la validación mediante una comprobación de trazado contra esquema (LVS)[78].
- Magic y klayout para garantizar las reglas de diseño (DRC)[42].
- Magic para capacidades parásitas del circuito RC y para la generación de archivos .gds y .lef con magic.

Para el uso de estas herramientas se pueden usar de manera independiente, hacer uso del entorno de IIC-OSIC-TOOLS o bien utilizar OpenAnalogDesign. OpenAnalogDesign es un entorno de diseño analógico abierto que encapsularía una gama de herramientas de diseño de código abierto (véase Anexo D).

Para el diseño analógico nos encontramos con las siguientes herramientas:

- Xschem - Un editor esquemático para diseños personalizados VLSI/Asic/Analógico
- Xcircuit - Programa de dibujo [especialmente para esquemas de circuitos]
- klayout - Transmite el archivo de diseño final
- magic - Transmite el archivo de diseño final
- netgen Realiza comprobaciones de LVS
- Xyce - Simulador de circuitos analógicos
- ngspice - Simulador para circuitos eléctricos y electrónicos.

Respecto a los PDK compatibles se encuentra el Skywater 130 nm utilizando open_pdk instalación. Este entorno de diseño analógico se encuentra en desarrollo acutualmente[79]. Algunos ejemplos de este flujo de diseño se encuentran en el entorno de IIC-OSIC-TOOLS, en la ruta foss/examples/SKY130_ANALOG-BLOCKS teniendo en común ambos el uso de las herramientas xschem, ngspice, magic y klayout.

5. Conclusiones y líneas futuras

El mundo del diseño hardware en código abierto, se ha convertido en un movimiento revolucionario que busca democratizar el diseño de circuitos integrados y hardware. A lo largo de esta investigación, hemos explorado los inicios y el contexto que rodea a este movimiento, la gran comunidad que comparte estos principios y contribuye al auge de la misma y el sinfín de servicios que ofrece.

El proyecto ha tenido como objetivo principal investigar las herramientas, entornos y documentación disponibles para diseños digitales, analógicos y de señal mixta, con un subobjetivo de establecer un flujo de trabajo de diseño digital utilizando exclusivamente herramientas de código abierto. Durante este proceso, se ha identificado una amplia variedad de herramientas y entornos de desarrollo que están disponibles para diseñadores de hardware, lo que demuestra la creciente relevancia de la comunidad OSH en este campo.

Hemos observado que la convergencia de iniciativas de las distintas plataformas están derribando barreras tradicionales y permitiendo que diseñadores independientes, estudiantes, pequeñas empresas y comunidades participen en el proceso de diseño de chips, lo que antes estaba reservado principalmente para grandes corporaciones. La práctica de compartir diseños de circuitos integrados de manera abierta y accesible, conocida como .ºpen Source Integrated Circuit"(OSIC) y la cantidad de herramientas y material disponible ha conseguido promover la colaboración, el estudio, la modificación y el uso libre de diseños de circuitos integrados, fomentando aún más la transparencia y la innovación en este campo. Haciendo posible el desarrollo de un diseño digital desde el nivel RTL hasta GDSII, completando el flujo típico de cualquier diseño digital sin ningún problema en open source.

En resumen, el movimiento OSH está redefiniendo la forma en que se aborda el diseño de chips y hardware. Estas iniciativas promueven la democratización, la colaboración y la accesibilidad en un campo que alguna vez fue exclusivo y costoso. A medida que la tecnología continúa avanzando, estas tendencias seguramente seguirán siendo pilares clave en la evolución de la industria electrónica. Este proyecto propone como futuras líneas de investigación la consolidación del flujo de diseño analógico en open source y la aplicación del flujo de diseño digital descrito en aplicaciones reales. Este trabajo de fin de master finaliza con la intención de representar un paso más hacia un futuro más abierto, colaborativo e innovador en el diseño de circuito integrados.

Anexo A

Instrucciones instalación entorno IIC-OSIC-TOOLS:

Guía paso a paso sobre cómo usar la herramienta IIC-OSIC-TOOLS desde el principio:

1. Preparativos: Asegúrese de tener Docker instalado en su sistema. Puedes descargar Docker desde su sitio web oficial[73].
2. Clonar el Repositorio:
   - Abre una terminal en tu sistema.
   - Navega a la ubicación donde deseas clonar el repositorio. Puedes usar el comando 'cd' para cambiar de directorio.
   - Clona el repositorio de IIC-OSIC-TOOLS desde GitHub utilizando el siguiente comando:
   - git clone https://github.com/iic-jku/iic-osic-tools.git
3. Acceder al Contenedor:
   - Navega al directorio del repositorio clonado: cd iic-osic-tools
   - Ejecuta el script `start_vnc.bat` para iniciar el contenedor en modo noVNC:
   - Esto debería iniciar el contenedor y mostrar información sobre los puertos y configuraciones utilizados.
4. Acceder a la Interfaz Gráfica:
   - Abre tu navegador web y visita la siguiente dirección: http://localhost
   - Deberías ver la interfaz gráfica del entorno de desarrollo dentro del contenedor.
5. Iniciar Sesión en el Entorno:
   - Si se te solicita una contraseña, ingresa "abc123"(según la configuración predeterminada).
   - Una vez que hayas iniciado sesión, estarás dentro del entorno de desarrollo.
6. Utilizar las Herramientas:
   - Explora las herramientas y recursos disponibles en el entorno de desarrollo.
   - Puedes acceder a las herramientas de diseño, simulación y otras utilidades para trabajar en tus proyectos de diseño de circuitos integrados.
7. Trabajar en Tus Proyectos:
   - Utiliza las herramientas disponibles para crear, diseñar y simular circuitos integrados según tus necesidades.

8. Guardar Tus Proyectos:

    - Puedes guardar tus proyectos en la carpeta ‘designs‘ dentro del directorio del repositorio. Esta carpeta se encuentra dentro del contenedor y se mapea a tu sistema de archivos local.

9. Detener el Contenedor:

    - Cuando hayas terminado de trabajar, puedes detener el contenedor ejecutando el siguiente comando en la terminal: `docker stop iic-osic-tools_xvnc`

Estas son instrucciones generales para comenzar a utilizar la herramienta IIC-OSIC-TOOLS.

Anexo B

Descripción HDL módulo multiplexor:

```
module multiplexor_2to1 (
  input wire a,
  input wire b,
  input wire select,
  output wire y        // Salida
);

assign y=(select) ? b:a;

endmodule
```

Test bench módulo multiplexor:

```
‘timescale 10us/1us

module testbench();

  // Declaraci n de se ales de prueba
  reg a, b, select;
  wire y;

  // Instancia del multiplexor
  multiplexor_2to1 mux_inst (
    .a(a),
    .b(b),
    .select(select),
    .y(y)
  );

  // Inicializaci n de se ales de prueba
  initial begin
    $dumpfile("testbench.vcd"); // Archivo VCD para
        simulaci n
    $dumpvars(0, testbench); // Dump de variables

    // Prueba 1: Select = 0, a debe ser dirigido a la salida
    a = 1’b0;
    b = 1’b1;
    select = 1’b0;
    #10; // Espera 10 unidades de tiempo
    // Verificaci n del resultado
    if (y === a)
      $display("Prueba 1 pasada");
    else
      $display("Prueba 1 fallida");

    // Prueba 2: Select = 1, b debe ser dirigido a la salida
    a = 1’b0;
    b = 1’b1;
    select = 1’b1;
```

```
37      #10; // Espera 10 unidades de tiempo
38      // Verificaci n del resultado
39      if (y === b)
40        $display("Prueba 2 pasada");
41      else
42        $display("Prueba 2 fallida");
43
44      $finish; // Finalizar la simulaci n
45    end
46
47  endmodule
```

Síntesis:

```
(DELAYFILE
 (SDFVERSION "3.0")
 (DESIGN "multiplexor_2to1")
 (DATE "Fri Oct 20 16:28:56 2023")
 (VENDOR "Parallax")
 (PROGRAM "STA")
 (VERSION "2.4.0")
 (DIVIDER .)
 (VOLTAGE 1.800::1.800)
 (PROCESS "1.000::1.000")
 (TEMPERATURE 25.000::25.000)
 (TIMESCALE 1ns)
 (CELL
  (CELLTYPE "multiplexor_2to1")
  (INSTANCE)
  (DELAY
   (ABSOLUTE
    (INTERCONNECT a _1_.A0 (0.007:0.007:0.007)
        (0.003:0.003:0.003))
    (INTERCONNECT b _1_.A1 (0.007:0.007:0.007)
        (0.002:0.002:0.002))
    (INTERCONNECT select _1_.S (0.012:0.012:0.012)
        (0.005:0.005:0.005))
    (INTERCONNECT _1_.X _2_.A (0.000:0.000:0.000))
    (INTERCONNECT _2_.X y (0.000:0.000:0.000))
   )
  )
 )
 (CELL
  (CELLTYPE "sky130_fd_sc_hd__mux2_2")
  (INSTANCE _1_)
  (DELAY
   (ABSOLUTE
    (IOPATH A0 X (0.119:0.119:0.119) (0.237:0.237:0.237))
    (IOPATH A1 X (0.120:0.120:0.120) (0.239:0.239:0.239))
    (IOPATH S X (0.195:0.195:0.195) (0.306:0.306:0.306))
    (IOPATH S X (0.129:0.129:0.129) (0.243:0.243:0.243))
   )
  )
 )
 (CELL
  (CELLTYPE "sky130_fd_sc_hd__buf_1")
  (INSTANCE _2_)
  (DELAY
   (ABSOLUTE
    (IOPATH A X (0.319:0.320:0.320) (0.218:0.219:0.219))
   )
  )
 )
)
```

```
/* Generated by Yosys 0.30+48 (git sha1 14d50a176d5, gcc
    8.3.1 -fPIC -Os) */

module multiplexor_2to1(a, b, select, y);
  wire _0_;
  input a;
  wire a;
  input b;
  wire b;
  input select;
  wire select;
  output y;
  wire y;
  sky130_fd_sc_hd__mux2_2 _1_ (
    .A0(a),
    .A1(b),
    .S(select),
    .X(_0_)
  );
  sky130_fd_sc_hd__buf_1 _2_ (
    .A(_0_),
    .X(y)
  );
endmodule
```

Anexo C

Instrucciones instalación OpenLane: En este anexo se detalla el proceso de instalación de OpenLane en Windows utilizando Docker Desktop y WSL 2. Una vez completadas todas las etapas se llega a un entorno de desarrollo de OpenLane funcional[80].

Instalación de OpenLane en Windows con Docker

Requisitos previos

- Windows 10 o Windows 11
- Docker Desktop instalado y configurado.

Pasos

1. **Instalar WSL 2 (Windows Subsystem for Linux):**
   - Para instalar WSL 2, consulta la documentación oficial de Microsoft.
   - Asegúrate de que tu versión de Windows sea compatible con WSL 2.
2. **Instalar Docker Desktop en Windows:**
   - Sigue los pasos oficiales para instalar Docker Desktop en Windows desde la *página oficial de Docker*.
   - Asegúrate de habilitar la integración de WSL 2 con Docker Desktop siguiendo la configuración adecuada en "Settings-WSL Integration".
3. **Verificar la configuración de Docker Desktop:**
   - Asegúrate de que Docker Desktop esté habilitado para iniciar automáticamente cuando inicies sesión en Windows desde "Docker Desktop-Settings".
4. **Abrir Windows PowerShell:**
   - Haz clic en el icono de Windows y busca "Windows PowerShell", ábrelo.
5. **Instalar una distribución de Ubuntu en WSL:**
   - Ejecuta el siguiente comando para instalar una distribución de Ubuntu en WSL:

     ```
     wsl --install -d Ubuntu
     ```
6. **Verificar la versión de WSL:**
   - Para verificar que la instalación de WSL 2 se realizó correctamente, ejecuta el siguiente comando:

     ```
     wsl --list --verbose
     ```

- Deberías ver una salida similar a la siguiente:

```
NAME STATE VERSION
* Ubuntu Running 2
docker-desktop Running 2
docker-desktop-data Running 2
```

- Si ves 'Stopped' en lugar de 'Running', asegúrate de iniciar Docker Desktop desde el menú de inicio.

7. **Iniciar la distribución de Ubuntu en WSL:**
   - Desde el menú de inicio, inicia 'Ubuntu'.
8. **Instalar paquetes necesarios:**
   - Actualiza la base de datos de paquetes e instala paquetes necesarios con los siguientes comandos:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt install -y build-essential python3
    ↪ python3-venv python3-pip make git
```

9. **Comprobar la instalación de Docker:**
   - Ejecuta el siguiente comando para comprobar si Docker está funcionando correctamente en WSL:

```
docker run hello-world
```

   - Deberías recibir un mensaje indicando que la instalación de Docker funciona correctamente.
10. **Comprobar los requisitos de instalación:**
    - Verifica que las siguientes herramientas estén instaladas correctamente en tu entorno de WSL:

```
git --version
docker --version
python3 --version
python3 -m pip --version
make --version
python3 -m venv -h
```

    - Deberías ver versiones y detalles de cada herramienta, lo que indica que están correctamente instaladas.
11. **Descargar e instalar OpenLane:**
    - Clona el repositorio de OpenLane desde GitHub y compila el proyecto con los siguientes comandos:

```
git clone --depth 1 https://github.com/The-
    ↪ OpenROAD-Project/OpenLane.git
cd OpenLane/
make
make test
```

- Estos pasos descargarán y compilarán OpenLane y el PDK sky130. Una prueba se ejecutará para verificar la instalación.

12. **Opcional: Visualizar resultados del diseño de pruebas:**

- Abre el diseño final utilizando KLayout con los siguientes comandos:

```
make mount
klayout -e -nn $PDK_ROOT/sky130A/libs.tech/
    ↪ klayout/tech/sky130A.lyt \
   -l $PDK_ROOT/sky130A/libs.tech/klayout/tech/
       ↪ sky130A.lyp \
   ./designs/spm/runs/openlane_test/results/final
       ↪ /gds/spm.gds

# Salir de la sesion de Docker:
exit
```

Anexo D

Instrucciones instalación OpenAnalogDesign: Para instalar el entorno de diseño analógico OpenAnalogDesign, siga estos pasos en un sistema Linux compatible. En este ejemplo, se utilizará Ubuntu 20.04:

1. Abra una terminal en su sistema.
2. Asegúrese de que Docker esté instalado en su sistema. Si no lo tiene instalado, puede hacerlo ejecutando el siguiente comando:

```
sudo apt-get install docker.io
```

3. Clone el repositorio de OpenAnalogDesign en su sistema:

```
git clone https://github.com/The-OpenROAD-Project/
    ↪ OpenAnalogDesign.git
```

4. Ingrese al directorio del repositorio clonado:

```
cd OpenAnalogDesign
```

5. Luego, ejecute los comandos de instalación para OpenAnalogDesign y el PDK:

```
make open_analog
make pdk
```

6. Estos comandos descargarán e instalarán las dependencias necesarias y configurarán el entorno de diseño analógico.
7. Una vez que la instalación esté completa, puede utilizar el entorno ejecutando el comando:

```
make mount
```

   Este comando montará el entorno de diseño analógico en Docker y le permitirá acceder a las herramientas en diferentes plataformas.

   Nota: Tenga en cuenta que la imagen de Docker para OpenAnalogDesign es de aproximadamente 9 GB, por lo que necesitará suficiente espacio de almacenamiento en su sistema. Después de realizar estos pasos, debería tener el entorno OpenAnalogDesign configurado y listo para su uso en su sistema Ubuntu 20.04. Asegúrese de revisar la documentación del proyecto OpenAnalogDesign y cualquier otro recurso proporcionado en los enlaces externos mencionados para obtener más detalles sobre cómo utilizar estas herramientas en su diseño analógico.

Referencias

[1] Ming-Wei Wu; Ying-Dar Lin, IEEE, *Open source software development: an overview*, \https://ieeexplore.ieee.org/abstract/document/928619, Recurso en línea, 2001.

[2] Acosta, Roberto, S.M. Massachusetts Institute of Technology, *Open source hardware*, \https://dspace.mit.edu/handle/1721.1/55201, Recurso en línea, 2009.

[3] efabless, *Open Shuttle Program*, https://efabless.com/open_shuttle_program, Accedido en línea en 2023.

[4] efabless, *Sitio web de efabless*, https://efabless.com/, Accedido en línea en 2023.

[5] *OpenRISC 1200 - Archived Page*, https://web.archive.org/web/20070928162331/http://www.opencores.org/projects.cgi/web/or1k/openrisc_1200, 2023.

[6] *RISC-V*, https://riscv.org/, Recurso en línea.

[7] *FOSSi Foundation*, https://fossi-foundation.org/, Recurso en línea.

[8] *LowRISC - Open Silicon*, https://lowrisc.org/open-silicon/, Recurso en línea.

[9] *Open Source Silicon*, https://open-source-silicon.dev/, Recurso en línea.

[10] *efabless Website*, https://efabless.com/, Recurso en línea.

[11] "ARM - AMBA." Recurso en línea. ().

[12] *OpenTitan*, https://opentitan.org/, Recurso en línea.

[13] *Página de inicio de Android Open Source Project*, https://source.android.com/?hl=es-419, Consultado el 3 de noviembre de 2023.

[14] Wikipedia, *Tensor Processing Unit*, Recuperado el 3 de noviembre de 2023. dirección: https://en.wikipedia.org/wiki/Tensor_Processing_Unit.

[15] Google, *Google SkyWater PDK*, https://github.com/google/skywater-pdk, Recurso en línea.

[16] gdsfactory, *Repositorio de GitHub: gf180*, Recuperado el 3 de noviembre de 2023. dirección: https://github.com/gdsfactory/gf180.

[17] efabless, *FOSS ASIC Tools*, https://github.com/efabless/foss-asic-tools, Recurso en línea.

[18] *GF180MCU PDK Repository*, https://github.com/google/gf180mcu-pdk, Recurso en línea.

[19] I. GmbH, *IHP Open PDK*, https://github.com/IHP-GmbH/IHP-Open-PDK, Fecha de acceso, por ejemplo, Noviembre 4, 2023, Año de acceso, por ejemplo, 2023.

[20] *IHP Open Source PDK Repository*, https://github.com/IHP-GmbH/IHP-Open-PDK, Recurso en línea.

[21] iic-jku, *SKY130_SAR-ADC1 Repository*, Consultado en octubre de 2023, 2023. dirección: https://github.com/iic-jku/SKY130_SAR-ADC1.

[22] Google and SkyWater Technology Foundry, *SkyWater SKY90FD Open Source PDK*, https://github.com/google/sky90fd-pdk, Accedido en Noviembre 4, 2023, 2023.

[23] Institute for Integrated Circuits (IIC), Johannes Kepler University (JKU), *IIC OSIC Tools*, https://github.com/iic-jku/iic-osic-tools, Recurso en línea.

[24] A. A. Layout, *ALIGN-public*, https://github.com/ALIGN-analoglayout/ALIGN-public, 2023.

[25] A. Language, *Amaranth*, https://github.com/amaranth-lang/amaranth, 2023.

[26] cocotb, *cocotb*, https://github.com/cocotb/cocotb, 2023.

[27] iic-jku, *verilog-covered*, https://github.com/iic-jku/verilog-covered, 2023.

[28] d-m-bailey, *cvc*, https://github.com/d-m-bailey/cvc, 2023.

[29] olofk, *edalize*, https://github.com/olofk/edalize, 2023.

[30] AUCOHL, *Fault*, https://github.com/AUCOHL/Fault, 2023.

[31] olofk, *fusesoc*, https://github.com/olofk/fusesoc, 2023.

[32] S. Schippers, *xschem-gaw*, https://github.com/StefanSchippers/xschem-gaw, 2023.

[33] gdsfactory, *gdsfactory*, https://github.com/gdsfactory/gdsfactory, 2023.

[34] heitzmann, *gdspy*, https://github.com/heitzmann/gdspy, 2023.

[35] trilomix, *GDS3D*, https://github.com/trilomix/GDS3D, 2023.

[36] Google, *gf180mcu-pdk*, https://github.com/google/gf180mcu-pdk, 2023.

[37] ghdl, *ghdl*, https://github.com/ghdl/ghdl, 2023.

[38] gtkwave, *gtkwave*, https://github.com/gtkwave/gtkwave, 2023.

[39] IHP-GmbH, *IHP Open PDK*, https://github.com/IHP-GmbH/IHP-Open-PDK, 2023.

[40] rtimothyedwards, *irsim*, https://github.com/rtimothyedwards/irsim, 2023.

[41] steveicarus, *iverilog*, https://github.com/steveicarus/iverilog, 2023.

[42] KLayout, *KLayout*, https://github.com/KLayout/klayout, 2023.

[43] magic layout editor with DRC y PEX, *Magic Layout Editor*, https://github.com/rtimothyedwards/magic, 2023.

[44] netlistsvg draws SVG netlist from a yosys JSON netlist, *NetlistSVG*, https://github.com/nturley/netlistsvg, 2023.

[45] ngspice SPICE analog y w. O. s. mixed-signal simulator with, *NgSpice*, http://ngspice.sourceforge.net/, 2023.

[46] ngspyce Python bindings for ngspice, *NgSpyce*, https://github.com/ignamv/ngspyce, 2023.

[47] nvc VHDL simulator y compiler, *NVC*, https://github.com/nickg/nvc, 2023.

[48] open$_p$*dksPDKsetupscripts*, *Open$_P$DKs*, https://github.com/RTimothyEdwards/open_pdks, 2023.

[49] openlane digital RTL2GDS flow, *OpenLane*, https://github.com/The-OpenROAD-Project/OpenLane, 2023.

[50] 2. g. openlane2 rewrite of OpenLane in Python, *OpenLane2*, https://github.com/efabless/openlane2, 2023.

[51] openram OpenRAM Python library, *OpenRAM*, https://github.com/VLSIDA/OpenRAM, 2023.

[52] openroad RTL2GDS engine used by openlane y openlane2, *OpenRoad*, https://github.com/The-OpenROAD-Project/OpenROAD.git, 2023.

[53] osic-multitool collection of useful scripts y documentation, *OSIC-Multitool*, https://github.com/iic-jku/osic-multitool.git, 2023.

[54] padring padring generation tool, *Padring*, https://github.com/donn/padring, 2023.

[55] pyopus simulation runner y optimization tool for analog circuits, *PyOPUS*, https://fides.fe.uni-lj.si/pyopus/index.html, 2023.

[56] pyrtl collection of classes for pythonic RTL design, *PyRTL*, https://github.com/UCSBarchlab/PyRTL, 2023.

[57] pyspice interface ngspice y xyce from Python, *PySpice*, https://github.com/PySpice-org/PySpice, 2023.

[58] pyverilog Python toolkit for Verilog, *PyVerilog*, https://github.com/PyHDI/Pyverilog, 2023.

[59] F. RF toolkit with FastHenry2 y openEMS., *RF Toolkit*, https://github.com/ediloren/FastHenry2, 2023.

[60] qucs-s simulation environment with RF emphasis, *Qucs-S*, https://github.com/ra3xdh/qucs_s, 2023.

[61] rggen code generation tool for configuration y status registers, *RGGen*, https://github.com/rggen/rggen, 2023.

[62] spyci analyze/plot ngspice/xyce output data with Python, *SpyCI*, https://github.com/gmagno/spyci, 2023.

[63] slang SystemVerilog parsing y translation (e.g. to Verilog), *SLang*, https://github.com/MikePopoloski/slang, 2023.

[64] qflow digital synthesis y place-and-route tool, *Qflow*, https://github.com/RTimothyEdwards/qflow.git, 2023.

[65] volare version manager (and builder) for open-source PDKs, *Volare*, https://github.com/efabless/volare, 2023.

[66] risc-v toolchain GNU compiler toolchain for RISC-V RV32I cores, *RISC-V GNU Toolchain*, https://github.com/riscv/riscv-gnu-toolchain, 2023.

[67] siliconcompiler modular build system for hardware, *SiliconCompiler*, https://github.com/siliconcompiler/siliconcompiler, 2023.

[68] sky130 SkyWater Technologies 130nm CMOS PDK, *SkyWater PDK*, https://github.com/google/skywater-pdk.git, 2023.

[69] verilator fast Verilog simulator, *Verilator*, https://github.com/verilator/verilator, 2023.

[70] xschem schematic editor, *XSchem*, https://github.com/StefanSchippers/xschem.git, 2023.

[71] xyce fast parallel SPICE simulator, *Xyce*, https://github.com/Xyce/Xyce.git, 2023.

[72] yosys Verilog synthesis tool, *Yosys*, https://github.com/YosysHQ/yosys, 2023.

[73] Docker, *Docker*, [Online; accessed 2023-11-05]. dirección: https://www.docker.com/.

[74] Steve Williams, *Icarus Verilog*, https://github.com/steveicarus/iverilog, Repositorio de GitHub y fuente importante de recursos para simulación de hardware digital.

[75] GHDL Community, *GHDL: Open-source VHDL simulator*, https://github.com/ghdl/ghdl, Repositorio de GitHub y herramienta de simulación de hardware digital en código abierto para VHDL.

[76] *GTKWave*, https://gtkwave.sourceforge.net/, Recurso en línea.

[77] S. Icarus, *Icarus Verilog - Getting Started*, [Online; accessed 2023-11-05]. dirección: https://steveicarus.github.io/iverilog/usage/getting_started.html.

[78] netgen netlist comparison (LVS), *Netgen*, https://github.com/rtimothyedwards/netgen, 2023.

[79] M. Author, *get_docker_config.py - OpenAnalogDesign*, [Online; accessed 2023-11-05]. dirección: https://github.com/mabrains/OpenAnalogDesign/blob/main/scripts/get_docker_config.py.

[80] *OpenLane Installation on Windows*, https://armleo-openlane.readthedocs.io/en/merge-window-4/getting_started/installation_win.html, Recurso en línea.