

UNIVERSIDAD NACIONAL DE CÓRDOBA
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES
CARRERAS DE INGENIERÍA EN COMPUTACIÓN E INGENIERÍA
ELECTRÓNICA



**PROYECTO INTEGRADOR PARA LA OBTENCIÓN DEL
TÍTULO DE GRADO DE INGENIERO EN COMPUTACIÓN E
INGENIERO ELECTRÓNICO**

**“DISEÑO Y TEST AUTOMÁTICOS DE UN MEZCLADOR
PARA UN TRANSCPECTOR ANALÓGICO Y MICRO
ELECTRÓNICO ORIENTADO A NOCS: DEFINICIONES
CONCEPTUALES Y DESARROLLOS EXPLORATORIOS”**

Córdoba, República Argentina

Resumen

En este Proyecto Integrador se buscó adquirir conocimientos sobre el desarrollo y testeo de un chip. Para ello, se utilizó la plantilla Caravan, ofrecida por Efabless, y se emplearon las herramientas recomendadas por la comunidad de Efabless, tales como Xschem, Magic, KLayout y LTSpice. El proyecto se dividió en dos grandes partes: la parte analógica, diseñada con las herramientas mencionadas, y la parte digital, que utilizó las mismas herramientas, pero de manera automática a través de OpenLane. Finalmente, ambas partes se unieron con Magic.

La parte digital se pudo testear mediante un procesador RISC-V incluido en la plantilla Caravan, el cual contaba con comunicación UART para la salida del resultado de la prueba. Para utilizar el procesador, se codificó un programa en C y se generó un archivo hexadecimal con el compilador de RISC-V. Posteriormente, se empleó Icarus Verilog para realizar el testbench de todo el chip, incluyendo la parte digital, desarrollada en Verilog. Cabe mencionar que Efabless ofrecía en la plantilla Caravan un contenedor con las herramientas necesarias para utilizar todo el flujo para la parte digital. Para la parte analógica, se debieron instalar localmente las herramientas.

Área Temática

Las Áreas Temáticas son: Computación e Informática, Digitales, Analógica.

Asignaturas

Respecto a Ingeniería en Computación: 1º Arquitectura de computadoras, 2º Sistemas operativos I y II, 3º Electrónica Digital II y III.

Respecto a Ingeniería Electrónica: 1º Electrónica Analógica II y III, 2º Teoría de las Comunicaciones, 3º Electrónica Física.

Palabras Claves

Diseño analógico en caravel_user_project_analog, Diseño digital en caravel_user_project_analog, Caravan, Efabless, Mixer.

Abstract

In this integrated project, the goal was to acquire knowledge about the development and testing of a chip. For this purpose, the Caravan template offered by Efabless was used, along with the tools recommended by the Efabless community, such as Xschem, Magic, KLayout, and LTSpice. The project was divided into two main parts: the analog part, designed with the aforementioned tools, and the digital part, which used the same tools but in an automated manner through OpenLane. Finally, both parts were integrated with Magic.

The digital part could be tested using a RISC-V processor included in the Caravan template, which featured UART communication for outputting the test results. To use the processor, a program was coded in C and a hexadecimal file was generated using the RISC-V compiler. Subsequently, Icarus Verilog was employed to perform the testbench for the entire chip, including the digital part, which was developed in Verilog. It is worth mentioning that Efabless provided a container with the necessary tools to use the entire flow for the digital part in the Caravan template. For the analog part, the tools had to be installed locally.

Thematic Area

The thematic areas are: Computing and Informatics, Digital, Analog.

Subjects

Regarding Computer Engineering: 1º Computer Architecture, 2º Operating Systems I and II, 3º Digital Electronics II and III.

Regarding Electronic Engineering: 1º Analog Electronics II and III, 2º Communication Theory, 3º Physical Electronics.

Key Words

Analog Design in caravel_user_project_analog, Digital Design in caravel_user_project_analog, Caravan, Efabless, Mixer.

Zusammenfassung

In diesem integrierten Projekt war es das Ziel, Kenntnisse über die Entwicklung und das Testen eines Chips zu erwerben. Zu diesem Zweck wurde die von Efabless angebotene Caravan-Vorlage verwendet und die von der Efabless-Community empfohlenen Werkzeuge wie Xschem, Magic, KLayout und LTSpice eingesetzt. Das Projekt wurde in zwei Hauptteile unterteilt: den analogen Teil, der mit den genannten Werkzeugen entworfen wurde, und den digitalen Teil, der die gleichen Werkzeuge, aber automatisiert über OpenLane verwendete. Schließlich wurden beide Teile mit Magic zusammengeführt.

Der digitale Teil konnte mit einem in der Caravan-Vorlage enthaltenen RISC-V-Prozessor getestet werden, der über eine UART-Kommunikation zur Ausgabe der Testergebnisse verfügte. Um den Prozessor zu nutzen, wurde ein Programm in C kodiert und eine Hexadezimaldatei mit dem RISC-V-Compiler erzeugt. Anschließend wurde Icarus Verilog verwendet, um das Testbench für den gesamten Chip durchzuführen, einschließlich des in Verilog entwickelten digitalen Teils. Es ist erwähnenswert, dass Efabless in der Caravan-Vorlage einen Container mit den notwendigen Werkzeugen zur Verfügung stellte, um den gesamten Ablauf für den digitalen Teil zu nutzen. Für den analogen Teil mussten die Werkzeuge lokal installiert werden.

Themenbereich

Die Themenbereiche sind: Informatik, Digital, Analog.

Fächer

In Bezug auf Computertechnik: 1° Computerarchitektur, 2° Betriebssysteme I und II, 3° Digitalelektronik II und III.

In Bezug auf Elektrotechnik: 1° Analoge Elektronik II und III, 2° Kommunikationstheorie, 3° Physikalische Elektronik.

Schlüsselwörter

Analoges Design in caravel_user_project_analog, Digitales Design in caravel_user_project_analog, Caravan, Efabless, Mixer.

Índice

Capítulo 1: Introducción.....	1
Antecedentes breves del problema.....	1
Relevancia de trabajo.....	1
Motivación para la elección del tema.....	3
Formulación del problema.....	3
Objetivo General y Objetivos Específicos.....	3
Metodología utilizada para lograr los objetivos propuestos.....	4
Orientación al lector de la organización del texto.....	4
MARCO TEÓRICO.....	5
Capítulo 2: Funcionamiento del <i>Mixer</i>.....	6
2.1. Introducción.....	6
2.2. Desarrollo del concepto de diseño.....	7
2.2.1. Descripción general de las partes del <i>Mixer</i>	8
2.3. Enfoque del diseño.....	13
2.3.1. Restricciones del diseño.....	13
2.3.2. Análisis del diseño.....	14
2.3.3. Implementación del diseño.....	15
Capítulo 3: Integración del Proyecto.....	23
MARCO METODOLÓGICO.....	26
Capítulo 4: Proceso de fabricación de un chip.....	27
4.1. PDK.....	29
4.1.1. Descripción.....	29
4.2 PDK de SkyWater.....	30
Capítulo 5: Proceso del diseño.....	33
5.1. Circuitos digitales.....	33
5.1.1. Introducción.....	33
5.1.2. Configuración en C (Firmware).....	34
5.1.3. Configuración del testbench.....	38
5.1.4. Configuración del RTL.....	45
5.1.5. Observación en Gtkwave.....	53
5.2. Circuitos analógicos.....	56
5.2.1. Introducción.....	56
5.2.2. Inverter.....	56
5.2.3. Mixer.....	67
5.2.4. Schmitt Trigger.....	77

Capítulo 6: Efabless Platform.....	82
6.1. Introducción.....	82
6.2. Integración del circuito digital.....	87
6.2.1. <i>OpenLane</i>	89
6.3. Integración del circuito analógico.....	96
RESULTADOS Y CONCLUSIONES.....	109
Resultados.....	110
Conclusiones.....	113
Trabajos futuros.....	114
Videos relacionados.....	115
Referencias.....	116
Anexos.....	119
Anexo A.....	119
Anexo B.....	127
Anexo C.....	133

Lista de Tablas

Tabla 1 Parámetros importantes a considerar para el diseño del Mixer (Fuente: Repositorio [13]).	15
Tabla 2 Tamaños de los transistores para el circuito Mixer (Fuente: Repositorio [13]).....	16
Tabla 3 Entradas al Mixer (Fuente: Repositorio [13]).....	21
Tabla 4 Salidas del Mixer (Fuente: Repositorio [13]).....	21
Tabla 5 Conexiones del Mixer (Fuente: Propia).....	104
Tabla 6 Conexiones del Schmitt Trigger (Fuente: Propia).....	104
Tabla 7 Pines Caravan (Fuente: Repositorio [55]).....	122

Lista de Figuras

Figura 1 Características del procesador PicoRV32 CPU (Fuente: PicoRV32 [39]).....	2
Figura 2 Abstracción del procesador (Fuente: PicoRV32 [39]).....	2
Figura 3 Chip Caravel Harness Chip (SoC) (Fuente: Caravel [2]).....	3
Figura 4 Esquema básico de un mezclador en el dominio del tiempo y frecuencia (Fuente: Propia).6	
Figura 5 Mixer de Radiofrecuencia (Fuente: Propia).....	7
Figura 6 Diseño modular para el mezclador seleccionado (Fuente: Repositorio [13]).....	8
Figura 7 Circuito general del Mixer (Fuente: Propia).....	9
Figura 8 Transistores saturados (Fuente: Propia).....	10
Figura 9 Etapa de switching para el circuito mezclador, detalle de la Figura 6 (Fuente: Propia)....	11
Figura 10 Entradas diferenciales (VLo) (Fuente: Propia).....	11
Figura 11 Etapa para el manejo de corriente del circuito mezclador (Fuente: Propia).....	11
Figura 12 Etapa de salida diferencial del circuito mezclador, detalle Figura 8 (Fuente: Propia)....	12
Figura 13 Etapa de cascode para el circuito mezclador, detalle Figura 6 (Fuente: Propia).....	12
Figura 14 Disposición de los transistores formando el Mixer (Fuente: Propia).....	17
Figura 15 Dispositivos para la simulación (Fuente: Propia).....	18
Figura 16 Señal de radiofrecuencia de 20khz (Fuente: Propia).....	19
Figura 17 Señal del oscilador local de 1Mhz (Fuente: Propia).....	19
Figura 18 Señal de salida del Mixer de radiofrecuencia (Fuente: Propia).....	20
Figura 19 Salida Ampliada out- del Mixer de Radiofrecuencia (Fuente: Propia).....	20
Figura 20 FFT de la señal de frecuencia Intermedia (Fuente: Propia).....	21
Figura 21 Diagrama descriptivo del proyecto (Fuente: Propia).....	24
Figura 22 Tamaño del user_project_area (Fuente: caravel [7]).....	25
Figura 23 Ruta de trabajo (Fuente: Propia).....	28
Figura 24 Skywater SKY130nm (Fuente: SkyWater130 [23]).....	31
Figura 25 Unidad bajo prueba (Fuente: Propia).....	34
Figura 26 Serie de Figuras 26 de secciones de código (Fuente: Propia).....	35
Figura 27 Serie de Figuras 27 de secciones de código (Fuente: Propia).....	39
Figura 28 Instancia del módulo test_mixer (Fuente: Propia).....	46
Figura 29 Máquina de estados (Fuente: Propia).....	47
Figura 30 Serie de Figuras 11 de secciones de código (Fuente: Propia).....	48
Figura 31 gtkwave (Fuente: Propia).....	54
Figura 32 gtkwave ampliado (Fuente: Propia).....	55
Figura 33 Xschem con nodo Sky130 (Fuente: Propia).....	57
Figura 34 Inverter en Xschem (Fuente: Propia).....	58
Figura 35 Netlist (Fuente: Propia).....	59
Figura 36 Magic con nodo Sky130A (Fuente: Propia).....	60
Figura 37 Inverter en Magic (Fuente: Propia).....	61
Figura 38 Salida del comando ‘drc fin’ (Fuente: Propia).....	63
Figura 39 Cell postlayout.spice (Fuente: Propia).....	65
Figura 40 Verificación LVS Inverter (Fuente: Propia).....	67
Figura 41 Directorios Mixer (Fuente: Propia).....	68
Figura 42 Mixer con Xschem (Fuente: Propia).....	69
Figura 43 Mixer.spice importado en Magic (Fuente: Propia).....	70
Figura 44 Observación de conexiones con Magic (Fuente: Propia).....	71
Figura 45 Fin de conexiones con Magic (Fuente: Propia).....	72
Figura 46 Salida de la consola de Magic luego de los comandos de extracción (Fuente: Propia)....	73
Figura 47 Salida de la consola de Magic luego del comando drc (Fuente: Propia).....	74
Figura 48 Fin de extracciones con Magic (Fuente: Propia).....	75
Figura 49 Fragmento Post Layout (Fuente: Propia).....	76
Figura 50 Verificación LVS (Fuente: Propia).....	77

Figura 51 Descripción funcionalidad del Schmitt Trigger (Fuente: Propia).....	78
Figura 52 Schmitt Trigger en Xschem (Fuente: Propia).....	79
Figura 53 Schmitt Trigger en LTSPICE (Fuente: Propia).....	80
Figura 54 Schmitt Trigger Simulación en LTSPICE (Fuente: Propia).....	80
Figura 55 Schmitt Trigger en Magic (Fuente: Propia).....	81
Figura 56 Repositorio de Efabless (Fuente: Propia).....	83
Figura 57 Repositorio caravel_user_project_analog (Fuente: Propia).....	85
Figura 58 make setup (Fuente: Propia).....	86
Figura 59 Directorio verilog en repositorio Caravan (Fuente: Propia).....	87
Figura 60 caravel_user_project_analog (Fuente: Propia).....	88
Figura 61 Caravel Harness Chip (Fuente: Caravel [2]).....	89
Figura 62 Flow OpenLane.v (Fuente: OpenLane [21]).....	90
Figura 63 Resumen de userDefines.v (Fuente: Propia).....	91
Figura 64 Directorio OpenLane Caravan (Fuente: Propia).....	92
Figura 65 Ubicación pines en user_project_wrapper_empty.gds (Fuente: Propia).....	93
Figura 66 Salida luego del comando make test_mixer (Fuente: Propia).....	93
Figura 67 Archivo .gds de la macro test_mixer con Klayout (Fuente: Propia).....	94
Figura 68 Archivo test_mixer.lef (Fuente: Propia).....	94
Figura 69 Archivo .gds de la macro user_analog_project_wrapper con Klayout (Fuente: Propia)	96
Figura 70 Path (Fuente: Propia).....	97
Figura 71 user_analog_project_wrapper_empty.gds (Fuente: Propia).....	98
Figura 72 Descripción del proyecto (Fuente: Efabless [6]).....	100
Figura 73 Figura abstracta del objetivo final (Fuente: Propia).....	101
Figura 74 Macro test_mixer en user_analog_project_wrapper (Fuente: Propia).....	102
Figura 75 Mixer, Schmitt Trigger y contador digital (test_mixer) (Fuente: Propia).....	103
Figura 76 Mixer, test_mixer, user_analog_project_wrapper Klayout (Fuente: Propia).....	105
Figura 77 Directorio Caravan/gds con caravan.gds (Fuente: Propia).....	106
Figura 78 Chip Caravan con MIXer, Schmitt Trigger y parte Digital (Fuente: Propia).....	107
Figura 79 Chip Final (Fuente: Efabless [6]).....	108
Figura 80 Salida del chip caravan observada en GTKWave (Fuente: Propia).....	110
Figura 81 Schmitt Trigger Simulación en LTSPICE (Fuente: Propia).....	111
Figura 82 Simulacion en LTSpice del Mixer (Fuente: Propia).....	111

Glosario

- Antenna Rule Violations
 - During manufacturing, a static charge can build up on the currently- topmost metal layer and destroy the chip if there is no path to the substrate for this charge to bleed off during layer deposition. The Antenna Rule ensures that each metal layer has a route to diffusion.
- LVS
 - Layout Versus Schematic
Layout Versus Schematic (LVS) verification is the process of determining whether a particular integrated circuit layout corresponds to the original schematic or circuit diagram of the design.
- PEX
 - Parasitic Extraction
Parasitic extraction is calculation of the parasitic effects in both the designed devices and the required wiring interconnects of an electronic circuit. This includes all parasitic components (often called parasitic devices) including parasitic; capacitances, resistances, and inductances.
- RTL
 - Register Transfer Language
A source code format that describes the transitions that hardware registers take at the register transfer level, such as Verilog or VHDL.
- VLSI
 - Very Large Scale Integration
Producing an integrated circuit in the million+ transistor scale, with multiple functions on the same chip (such as compute, memory, ROM, and power regulation).
- .lef
 - LEF - Library Exchange Format
Abstract description of the layout for place and route.
- .lib
 - Liberty Models - Liberty Timing Models - Liberty Wire Load Models
 - Liberty Files are an IEEE Standard for defining: PVT Characterization, Relating Input and Output Characteristics, Timing, Power, Noise.
 - Wire Load Models estimate the parasitics based on the fanout of a net.
- ESD
 - Electro-Static Discharge (protection from)

Circuit elements, especially on I/O pins, intended to protect the circuit from the effects of electrostatic discharge.

- DRC

- Design Rule Check - Design Rule Checking

Design rule checking or check(s) is the process of determining whether the physical layout of a particular chip layout satisfies a series of required parameters called design rules.

- GDSII/GDS2

- Es un formato de archivo binario que representa las capas necesarias para producir un ASIC.
 - En el flujo de OpenLane, Magic se utiliza para "stream" los archivos GDS2 finales.
 - Todas las formas se asignan a una capa GDS2 específica, y cada capa termina siendo utilizada para crear una máscara, aunque a menudo esto puede implicar la combinación de una o más capas GDS2 para formar una máscara, y las formas a menudo se agrandan, se reducen o se fusionan, por lo que lo que termina en la máscara puede no ser lo mismo que lo dibujado por el diseñador. Cada máscara se utiliza luego en un paso fotolitográfico para producir el chip.
 - Se pueden usar otros programas o incluso bibliotecas de software para generar archivos GDS2 válidos. Aquí he usado gdspy para convertir el logo de Hackaday en GDS2.

- MACRO

- Las macros son propiedades intelectuales que se pueden usar en el diseño. No necesitan ser diseñadas por uno mismo. Por ejemplo, memorias, núcleos de procesador, PLL, etc. Una macro puede ser una macro dura o blanda.
 - Las macros blandas y las macros duras se categorizan como IP.

- Macros Blandas:

Las macros blandas se utilizan en implementaciones de SOC. Las macros blandas están en forma de RTL sintetizable, y son más flexibles que las macros duras en términos de re-configurabilidad. Las macros blandas no son específicas de ningún proceso de fabricación y tienen la desventaja de ser impredecibles en términos de tiempo, área, rendimiento o potencia. Las macros blandas conllevan mayores riesgos de protección de IP porque el código fuente RTL es más portátil y, por lo tanto, menos fácilmente protegido que una lista de redes o datos de diseño físico. Las macros blandas son editables y pueden contener celdas estándar, macros duras u otras macros blandas.

- Macros Duras:

Las macros duras están dirigidas a una tecnología de fabricación de circuitos integrados específica. Son diseños a nivel de bloque que están optimizados para potencia, área o tiempo y han sido probados en silicio. Al realizar el diseño físico, es posible acceder solo a los pines de las macros duras, a diferencia de las macros blandas que nos permiten manipular el RTL. Una macro dura es un bloque que se genera mediante una metodología diferente a la de "place and route" y se importa a la base de datos del diseño físico como un archivo GDS2.

- MPW
 - Un Multi Proyecto en Oblea se utiliza para reducir los costos de fabricación de ASIC al compartir los costos de Ingeniería no Recurrente (NRE) asociados con la creación del juego de máscaras entre todos los participantes, mediante la inclusión de diversos diseños en una misma oblea.
- SoC
 - Sistem on Chip describe la tendencia cada vez más frecuente de usar tecnologías de fabricación que integran todos o gran parte de los módulos que componen un computador o cualquier otro sistema informático o electrónico en un único circuito integrado o chip.

Capítulo 1: Introducción

Antecedentes breves del problema

Este proyecto es un primer paso hacia la obtención del know how del proceso de desarrollo de System On Chips^[36] (SoCs) mixtos (analógicos y digitales) con auto testeo incluido en el diseño. El enfoque está en desarrollos exploratorios y definiciones conceptuales necesarias para dicho proceso, más que en los dispositivos y métodos de testeo específicos. El dispositivo bajo estudio (DUT) elegido es un mezclador para un transceptor, y el método de testeo se basa en un conteo indirecto de frecuencias de salida del mezclador. Es importante destacar que no existen antecedentes de trabajos con diseños mixtos a nivel microelectrónico en la universidad.

Relevancia de trabajo

La integración de señales analógicas de alta frecuencia y digitales en un SoC^[36] es un desafío tecnológico significativo, especialmente cuando se superan los 60 MHz. Este proyecto contribuye al conocimiento y la experiencia en el uso de herramientas de código abierto como Caravel y Caravan, proporcionadas por Efabless, para desarrollar SoCs mixtos. La relevancia se amplía al tratar de integrar tecnologías analógicas y digitales de manera efectiva en un mismo chip.

El SoC proporcionado por Efabless incluye un procesador PicoRV32^[39], ampliamente reconocido y utilizado en la comunidad de diseño de sistemas embebidos por su eficiencia y rendimiento fiable. Este procesador ofrece versatilidad para una variedad de aplicaciones y cuenta con las características que se muestran en la Figura 1.

En las Figura 2 y la Figura 3 se muestra un esquema del diseño del chip de base propuesto por *Efabless*^[46].

En efecto, las imágenes en dichas figuras ilustran la composición del SoC. El término "System on Chip" hace referencia a un diseño de circuito integrado donde la totalidad o la mayoría de los componentes requeridos para el funcionamiento de un sistema completo están integrados en un solo chip semiconductor. Se enfatiza aquí nuevamente que el mezclador es un dispositivo de RF complejo que requiere otros módulos analógicos asociados para verificar su funcionamiento y que dichos módulos se mantienen externos en este SoC para mantener el alcance y el tiempo de finalización de este proyecto, acotados y bajo control.

Module	Description
<code>picorv32</code>	The PicoRV32 CPU
<code>picorv32_axi</code>	The version of the CPU with AXI4-Lite interface
<code>picorv32_axi_adapter</code>	Adapter from PicoRV32 Memory Interface to AXI4-Lite
<code>picorv32_wb</code>	The version of the CPU with Wishbone Master interface
<code>picorv32_pcpi_mul</code>	A PCPI core that implements the <code>MUL[H SU U]</code> instructions
<code>picorv32_pcpi_fast_mul</code>	A version of <code>picorv32_pcpi_fast_mul</code> using a single cycle multiplier
<code>picorv32_pcpi_div</code>	A PCPI core that implements the <code>DIV[U]/REM[U]</code> instructions

Figura 1 Características del procesador PicoRV32 CPU (Fuente: PicoRV32 [39])

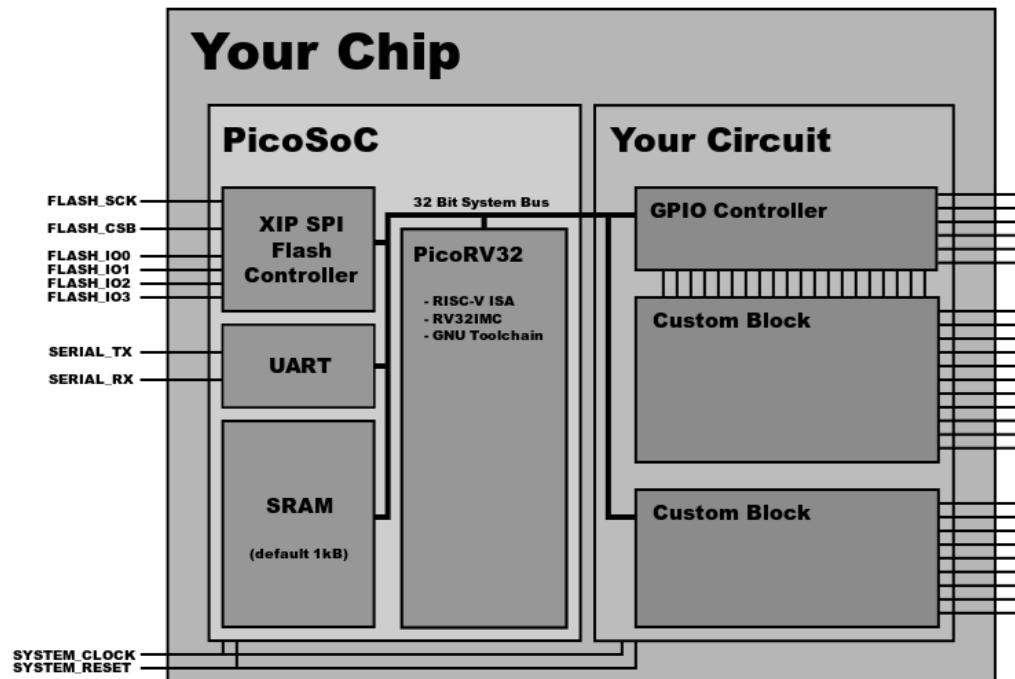


Figura 2 Abstracción del procesador (Fuente: PicoRV32 [39])

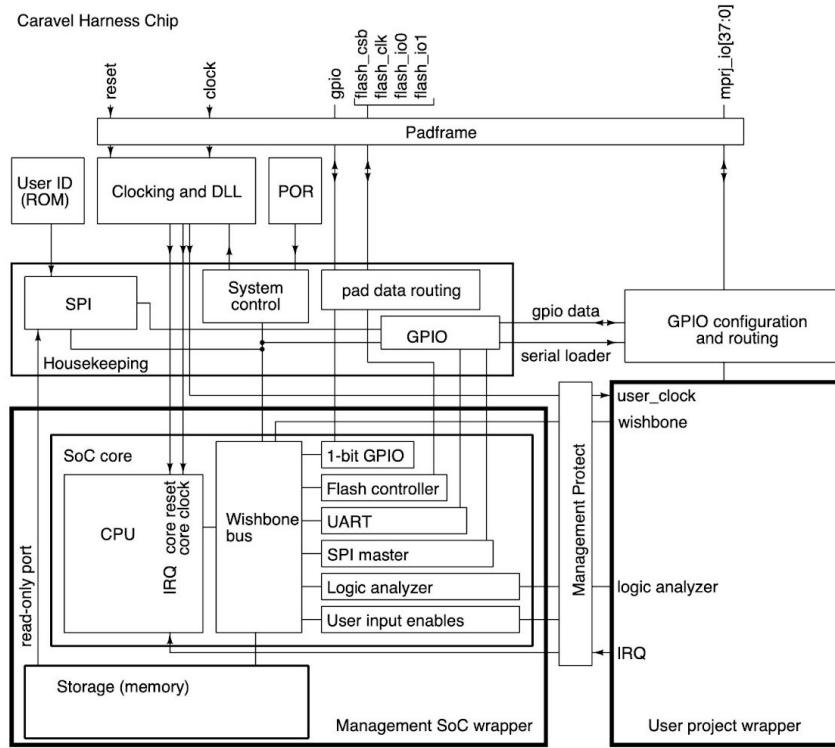


Figura 3 Chip Caravel Harness Chip (SoC) (Fuente: Caravel [2])

Motivación para la elección del tema

La motivación para elegir este tema radica en la necesidad de adquirir el know how del proceso de desarrollo de SoCs mixtos, dado que no existen precedentes claros y completos en la comunidad *Efabless* sobre la integración de partes digitales y analógicas en Caravan^[2]. Además, la constante evolución de las herramientas de código abierto y la posibilidad de fallos en las primeras etapas del chip hacen que este proyecto sea un campo de estudio y desarrollo esencial para futuras innovaciones.

Formulación del problema

El problema radica en cómo integrar de manera efectiva señales analógicas de alta frecuencia y digitales en un SoC utilizando herramientas de código abierto, y cómo desarrollar un proceso de auto testeo fiable para un mezclador de RF complejo que necesita módulos asociados externos para su funcionamiento.

Objetivo General y Objetivos Específicos

Objetivo General:

- Obtener el know how del proceso de desarrollo de SoCs mixtos con auto testeo incluido en el diseño.

Objetivos Específicos:

1. Desarrollar y definir conceptos necesarios para la integración de señales analógicas y digitales en un SoC.
2. Utilizar herramientas de código abierto como Caravel^[7] y Caravan^[9] para el diseño y verificación del SoC.
3. Implementar un método de testeo basado en el conteo indirecto de frecuencias de salida del mezclador.
4. Realizar numerosas pruebas y correcciones para asegurar la fiabilidad del diseño.

Metodología utilizada para lograr los objetivos propuestos

La metodología se basa en un enfoque exploratorio utilizando herramientas de código abierto para el diseño, simulación y verificación del SoC. Se emplean herramientas como Magic^[20], Xschem^[27], Netgen^[19], OpenLane^[10] y KLayout^[14] para diferentes etapas del proceso. Se realiza una investigación continua y se ejecutan pruebas iterativas para identificar y corregir fallos, optimizando así el diseño del SoC. Hoy en día, *Efabless* está experimentando un crecimiento notable y se está acercando cada vez más a su objetivo principal: permitir que cualquier persona pueda fabricar un chip, probarlo y, en caso de que no funcione correctamente, volver a realizar el proceso, lo que se conoce en el ámbito del software como '*troubleshooting*'^[44]. Uno de los desafíos que enfrentan estas tecnologías es la escasez de personas que tomen iniciativas durante el transcurso del proyecto. Existen varias empresas o universidades que están respaldando este tipo de iniciativas.

Orientación al lector de la organización del texto

El texto se organiza de la siguiente manera:

1. Introducción: Presenta el contexto y la importancia del proyecto.
2. Marco teórico desarrollo del mixer y Schmitt-trigger.
3. Marco metodológico Integración de los dispositivos digitales y analógicos en la utilizando la plantilla de efalbess^[6].

PARTE I

MARCO TEÓRICO

Capítulo 2: Funcionamiento del Mixer

En este capítulo se mostrará los fundamentos teórico y prácticos para llevar a cabo un mixer (mezclador de señales), en una primera instancia se mostrará el fundamento teórico y luego se procede a llevarlo a cabo con herramientas open source como LTSpice, Xschem, Magic.

2.1. Introducción

El *Mixer* se define como un dispositivo que combina dos o más señales en una o dos salidas compuestas por estas señales. Este dispositivo realiza la conversión de energía de radiofrecuencia (RF) a una frecuencia diferente en la salida, lo cual simplifica el procesamiento de señales. La conversión de frecuencia permite la amplificación de la señal recibida a una frecuencia distinta de la RF. Esta función es esencial en diversas aplicaciones, como la modulación de señales, la amplificación de señales débiles y la mezcla de frecuencias para generar nuevas señales. El funcionamiento del *Mixer* se puede representar como en la Figura 4.

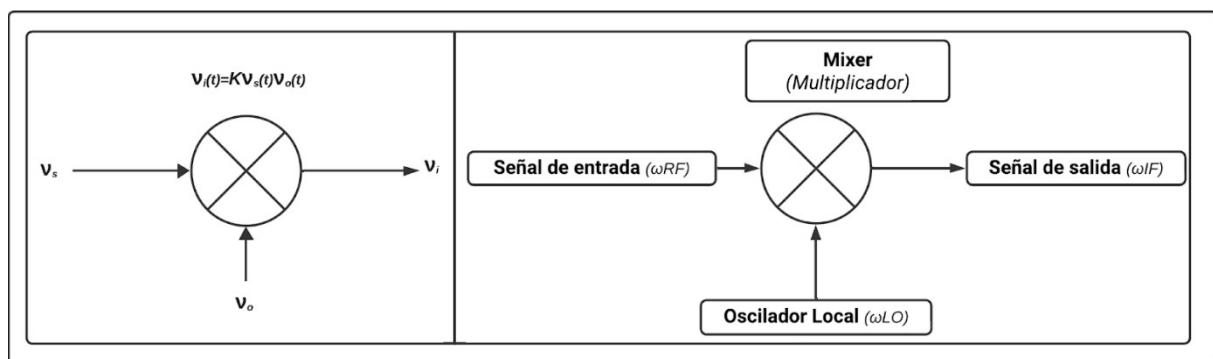


Figura 4 Esquema básico de un mezclador en el dominio del tiempo y frecuencia (Fuente: Propia)

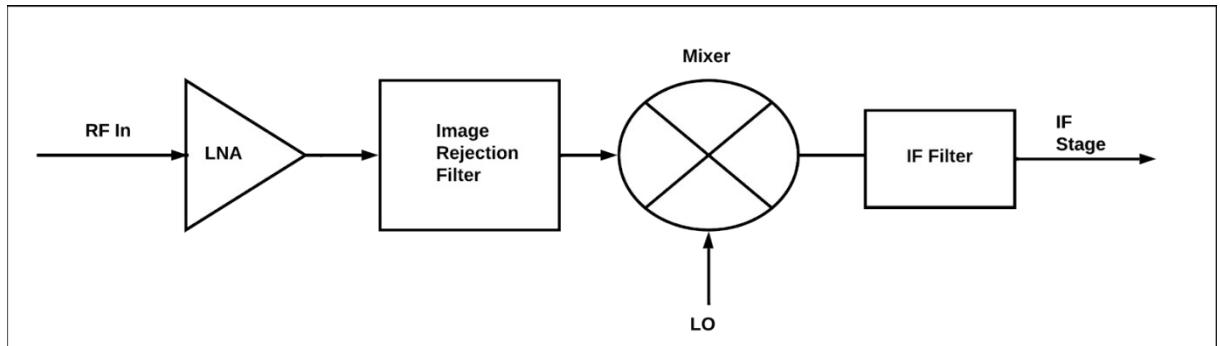


Figura 5 Mixer de Radiofrecuencia (Fuente: Propia)

2.2. Desarrollo del concepto de diseño

Esta sección del proyecto se basa en el paper "Diseño de un mezclador de frecuencias con topología de Gilbert para espectroscopia médica por impedancia eléctrica en tecnología CMOS de $0.13\text{ }\mu\text{m}$ " de la Universidad de Costa Rica^[13].

A partir de la propuesta seleccionada en el capítulo anterior, la Figura 6 muestra el concepto general de diseño que incluye dos bloques principales para la implementación funcional del *Mixer*. Estos bloques son: uno encargado de realizar la polarización para los nodos descritos en el circuito del *Mixer*, y el segundo bloque corresponde al circuito base funcional del *Mixer* de frecuencias. El diagrama de diseño modular se presenta en la Figura 6.



Figura 6 Diseño modular para el mezclador seleccionado (Fuente: Repositorio [13])

2.2.1. Descripción general de las partes del *Mixer*

El *Mixer* está compuesto por seis etapas principales, descritas a continuación junto con sus respectivas figuras.

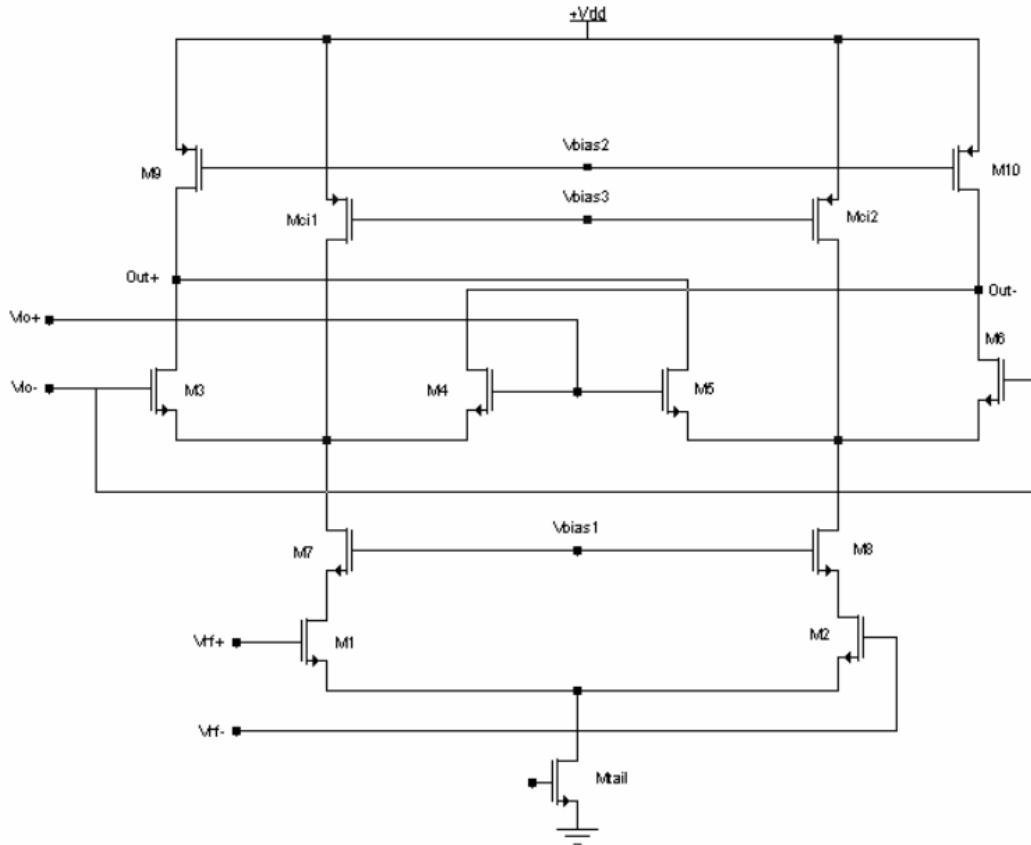


Figura 7 Circuito general del Mixer (Fuente: Propia)

Etapa de Ganancia

Esta etapa se encuentra en la parte inferior de la representación esquemática del *Mixer* (Figura 7, transistores M1 y M2). Debe poseer una alta linealidad para manejar la potencia del amplificador operacional del circuito. Los transistores en la Figura 8 deben mantenerse en la región de saturación para asegurar una alta corriente. Esta etapa convierte el voltaje en corriente (convertidor V-I). Sin embargo, debido a la tecnología utilizada, estos transistores operan en su zona lineal, logrando aun así una alta corriente, aunque la impedancia vista por el resto del circuito no es óptima y se inserta variabilidad, por ejemplo, con respecto a la frecuencia. La ecuación para un correcto funcionamiento es:

$$V_{GS} - V_T \geq 200 \text{ mV}$$

La ganancia de conversión se puede aumentar manteniendo el largo del canal "L" en el mínimo y aumentando el ancho "W" o incrementando la corriente.

$$G_C = \frac{2}{\pi} g_m R_L = \frac{2}{\pi} R_L \left(\mu_n C_{ox} \frac{W_1}{L_1} I_{tail} \right)$$

Como se observa en la ecuación anterior, la ganancia de conversión del mezclador se puede ver aumentada de dos maneras:

- 1) Manteniendo el largo del canal "L" como el mínimo y aumentando el ancho "W" hasta alcanzar la ganancia deseada.
- 2) Al mismo tiempo notando que la ganancia de la etapa es proporcional a g_m , se puede hacer que al aumentar la corriente se logre un aumento en la ganancia como se muestra en la siguiente ecuación:

$$g_m = \frac{2 I_{D_s}}{V_{GS} - V_T}$$

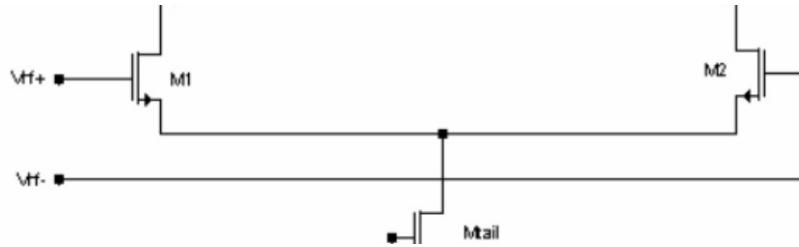


Figura 8 Transistores saturados (Fuente: Propia)

Etapa de Switching

En esta etapa, los transistores que manejan las señales del oscilador local (Figura 9) deben mantenerse siempre en saturación cuando sea necesario. El voltaje de modo común de las señales LO debe garantizar la correcta conmutación del *Mixer* y reducir el ruido en señales grandes de LO. Un aumento excesivo de la señal LO puede generar picos de voltaje que afectan la velocidad de conmutación y realimentan la salida IF, reflejándose en los puertos RF. Sin embargo, una señal diferenciada con estructura doblemente balanceada cancela estos efectos. Cuando un par de transistores está en saturación, el otro está apagado, eliminando ruidos en la salida. La condición de encendido del transistor ($V_{GS} - V_T$) debe acercarse a cero, permitiendo que, cuando la señal diferencial sea positiva, un par de transistores se encienda y el otro se apague, y viceversa en el ciclo negativo (Figura 10).

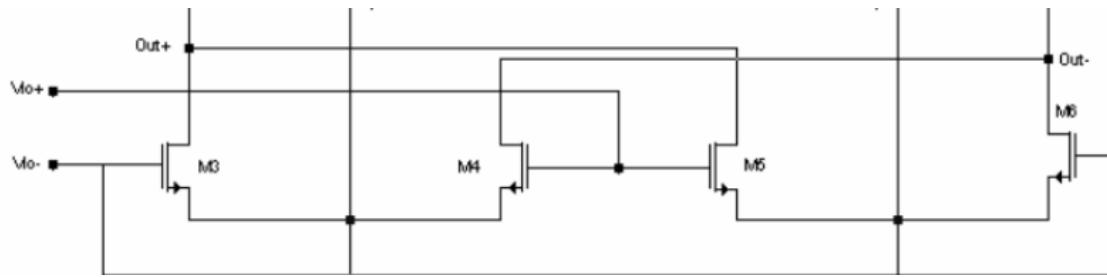


Figura 9 Etapa de switching para el circuito mezclador, detalle de la Figura 6 (Fuente: Propia)

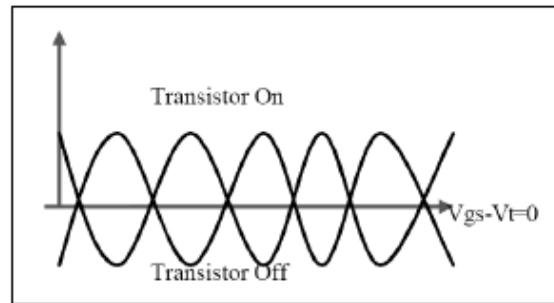


Figura 10 Entradas diferenciales (V_{Lo}) (Fuente: Propia)

Etapa para el manejo de corriente

Compuesta por tres transistores operando como fuentes de corriente (Figura 11). El transistor inferior (M_{tail}) proporciona corriente de polarización, mientras que los dos transistores superiores (M_{c1} y M_{c2}) implementan una inyección de carga mediante la "técnica de sangrado", mejorando el rendimiento del *Mixer*. Estos transistores deben mantenerse en la región de saturación configurando su ancho (W) y largo (L) adecuadamente, así como sus tensiones de polarización de V_{tail} y V_{bias3} .

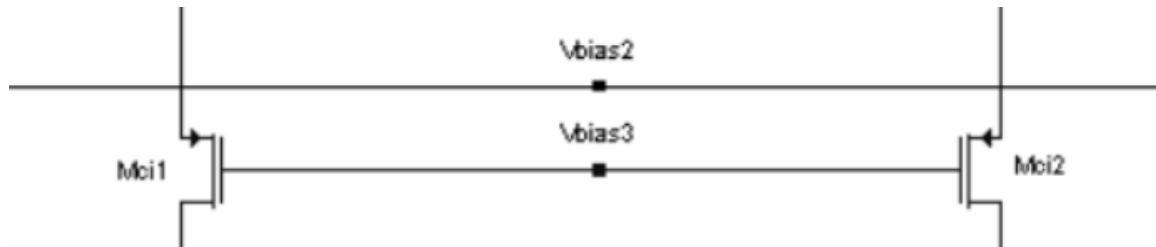


Figura 11 Etapa para el manejo de corriente del circuito mezclador (Fuente: Propia)

Etapa de salida diferencial

Compuesta por dos transistores PMOS (uno en cada salida, out^+ y out^-) que actúan como cargas para aumentar la ganancia de conversión de voltaje del *Mixer*. Esta configuración explota la simetría para eliminar las señales no deseadas de RF y LO realimentadas hacia la salida mediante cancelación (Figura 12).

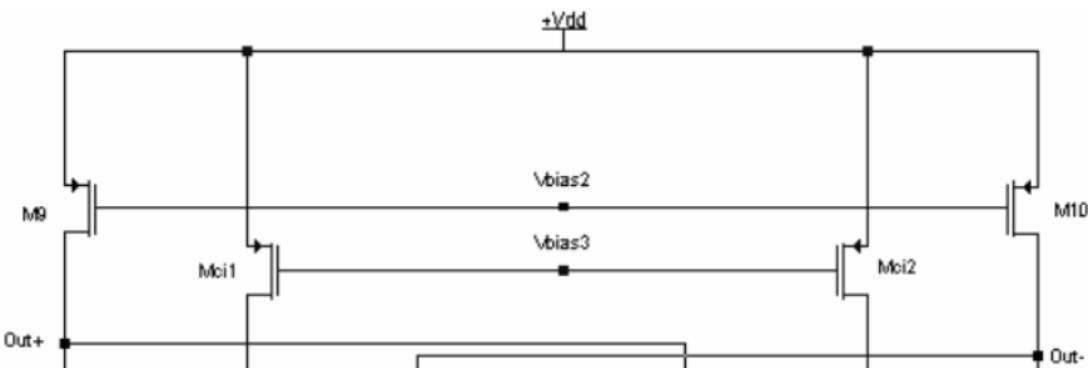


Figura 12 Etapa de salida diferencial del circuito mezclador, detalle Figura 8 (Fuente: Propia)

Etapa de cascodo

Esta etapa incluye dos transistores NMOS que funcionan como dispositivos cascodos para el circuito del *Mixer* (Figura 13). Estos transistores deben mantenerse en saturación para incrementar el aislamiento entre los puertos de entrada RF y LO.

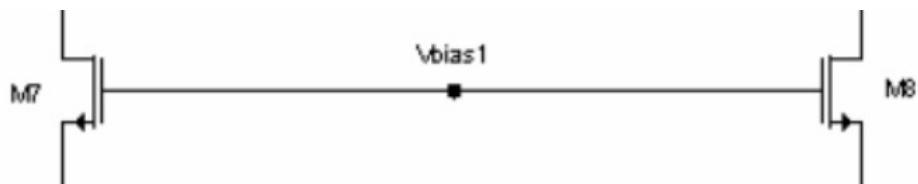


Figura 13 Etapa de cascodo para el circuito mezclador, detalle Figura 6 (Fuente: Propia)

Operación del Mixer

La célula de Gilbert está compuesta por dos etapas: la etapa de entrada o de radiofrecuencia y la etapa de salida o de frecuencia intermedia. La etapa de entrada, compuesta por un par diferencial, actúa como un amplificador de transconductancia cuya función es la conversión con ganancia de tensión a corriente. La etapa de salida, compuesta por dos pares diferenciales con salidas cruzadas, constituye el núcleo del *Mixer*, ya que se encarga de la multiplicación.

De la Figura 8 se pueden hacer varias observaciones sobre el funcionamiento general del *Mixer*. Los transistores M1 y M2 proporcionan el convertidor de voltaje-corriente proveniente de la señal RF diferencial, evitando que el transistor Mtail se vea afectado y manteniendo el circuito polarizado.

Cuando el voltaje en el puerto LO es suficientemente alto para polarizar los transistores M3 y M6, manteniéndolos en saturación, y la tensión en LO+ es suficientemente baja para apagar los transistores M4 y M5, los transistores M3 y M6 actúan como interruptores cerrados. Cada uno de estos transistores tiene un transistor PMOS como carga (M9 y M10, respectivamente), lo que resulta en una configuración diferencial a la salida con IF+ e IF- (OUT+ y OUT-).

En el siguiente ciclo, los roles se invierten: el voltaje LO+ es suficientemente alto para polarizar los transistores M4 y M5, mientras que LO- es suficientemente bajo para cortar los transistores M4 y M5. Estos transistores ahora actúan como interruptores cerrados con M4 cargado por M10 y M5 cargado por M9, resultando en una configuración diferencial a la salida con out+ y out- con la misma amplitud del ciclo anterior, pero con una fase diferente.

Para evitar problemas con la etapa que maneja la señal LO, los transistores Mc1 y Mc2 garantizan que la etapa superior del Mixer esté polarizada, asegurando que al menos uno de los pares de transistores que reciben las señales LO+ y LO- esté encendido y en saturación (M3-6). Finalmente, los transistores M7 y M8 deben mantenerse siempre en saturación para incrementar el aislamiento entre los puertos de entrada RF y LO.

2.3. Enfoque del diseño

2.3.1. Restricciones del diseño

El diseño del *Mixer* de Radiofrecuencia se ajusta a las siguientes restricciones, definidas según el área del proyecto y la aplicación específica:

- **Voltaje de alimentación:** $V_{DD}=1.2\text{ V}$
- **Tecnología:** Se utiliza la tecnología $0.13\text{ }\mu\text{m}$ PDK, por lo que el ancho de los transistores no puede exceder los $20\text{ }\mu\text{m}$.
- **Ancho de banda:** El Mixer debe operar en el rango de frecuencias de 1 MHz a 7 GHz , con la Celda de Gilbert utilizando tecnología CMOS.
- **Ancho de banda de entrada RF:** Debe ser de 1 GHz a 7 GHz sin atenuación significativa de la señal de salida.
- **Ancho de banda de salida IF:** Debe alcanzar de 100 MHz a 800 MHz sin atenuación significativa.
- **Conversión a bajas frecuencias:** Se requiere que la mayoría de los transistores funcionen en la región de saturación cuando estén activos.
- **Tensiones de polarización (VBIAS):** Deben ser lo más bajas posibles para minimizar el consumo de potencia y el área utilizada.
- **Fuentes de corriente:** Los transistores que actúan como fuentes de corriente deben diseñarse para comportarse como fuentes de corriente ideales con alta resistencia.
- **Largo de los transistores:** Aunque el mínimo es $0.13\text{ }\mu\text{m}$, para mantener una resistencia r_o constante y homogénea entre todos los transistores, no se utiliza este valor mínimo. Se requiere un largo mínimo de $0.5\text{ }\mu\text{m}$ o ligeramente superior para mantener una tensión V_T constante para todos los transistores.

2.3.2. Análisis del diseño

- **Corriente del circuito:** Debe estar en el orden de μA , aunque podría aumentar dependiendo de las corrientes de polarización requeridas para obtener un mayor rango de frecuencias para la entrada y salida.
- **Largo de los transistores:** Se establece en $0.5\text{ }\mu\text{m}$ para todos, excepto el transistor Mtail, cuyo largo es de $0.65\text{ }\mu\text{m}$ debido a su función de polarización. La compuerta de Mtail debe alimentarse con 0.4 V .
- **Tensión de entrada:** Para los puertos RF y LO del *Mixer*, se debe garantizar una tensión de modo común que permita la transición de saturación a corte y viceversa, minimizando el consumo de potencia por conmutación. Las tensiones de modo común se establecen en aproximadamente 0.6 V para los puertos RF y 0.7 V para los puertos LO.

Los parámetros listados en la Tabla 1 se consideran cruciales para el diseño del Mixer.

Parámetros	Valor
Tensión de alimentación (Vdd)	1.2 V
Tensión de “encendido” (Vt)	~0.3 V
Resistencia NMOS	1,93 kΩ.µm
Resistencia PMOS	5,48 kΩ.µm
VBIAS1	0.6 V
VBIAS2	0.6 V
VBIAS3	0.6 V
VTAIL	0.6 V
VCMLO	0.6 V
VCMRF	0.6 V

Tabla 1 Parámetros importantes a considerar para el diseño del Mixer (Fuente: Repositorio [13])

Los valores de los anchos de los transistores se obtuvieron a partir de valores conocidos para una Celda de Gilbert y se ajustaron para el diseño específico. Se realizaron análisis en DC para asegurar que los transistores funcionen en saturación. Se determinó que se necesitaba una fuente externa de 0.6 V tanto para VBIAS1 como para VBIAS2 para un alto manejo de corriente, mientras que VBIAS3 se ajustó a entre 0.85 V y 0.90 V para aumentar la ganancia y mantener la tensión de salida de modo común en 0.6 V.

2.3.3. Implementación del diseño

Después de realizar simulaciones en DC, se determinaron los siguientes valores para los transistores del circuito *Mixer*, los cuales se observan en la Tabla 2. Los modelos de transistores utilizados son nfet_rf y pfet_rf, diseñados específicamente para aplicaciones RF.

Transistores	W(μm)	L(μm)
M1-2	9.75	0.5
M3-6	9.50	0.5
M7-8	9.75	0.5
M9-10	8.70	0.5
MC1-2	2.00	0.5
Mtail	15.15	0.65

Tabla 2 Tamaños de los transistores para el circuito Mixer (Fuente: Repositorio [13])

El circuito esquemático final se construyó en LTspice, como se muestra en la Figura 4 y se puede observar en la Figura 14.

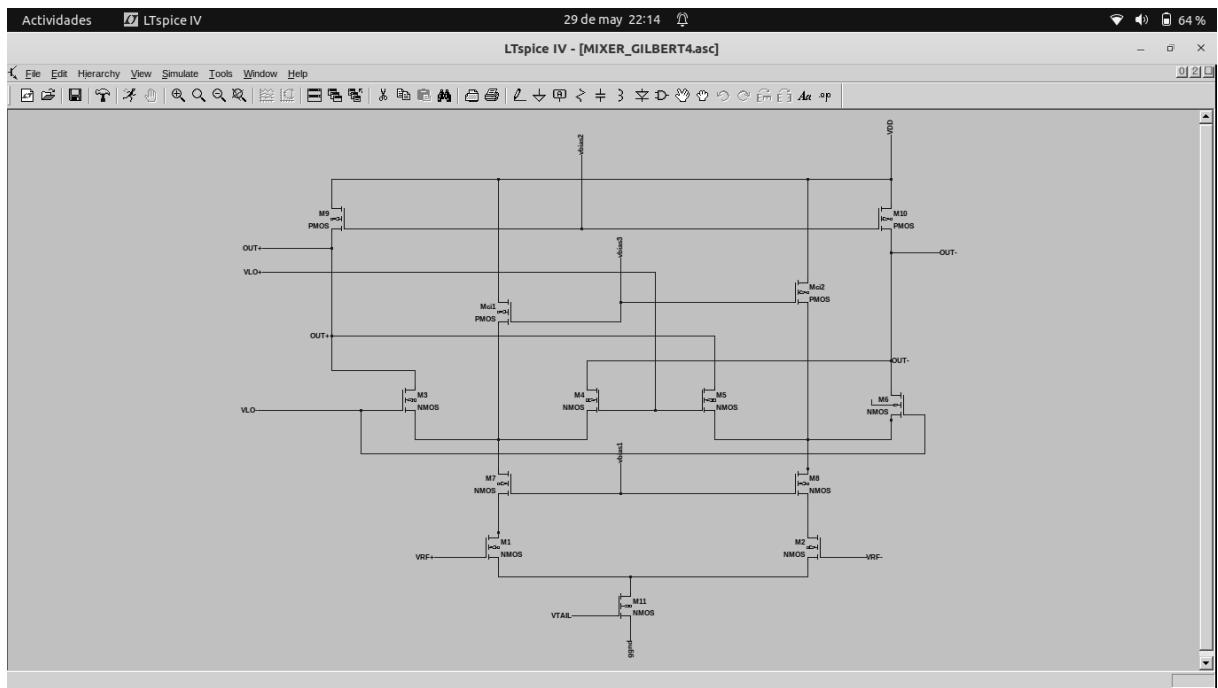


Figura 14 Disposición de los transistores formando el Mixer (Fuente: Propia)

La Figura 15 muestra la realización de la simulación usando un símbolo del Mixer de Radiofrecuencia

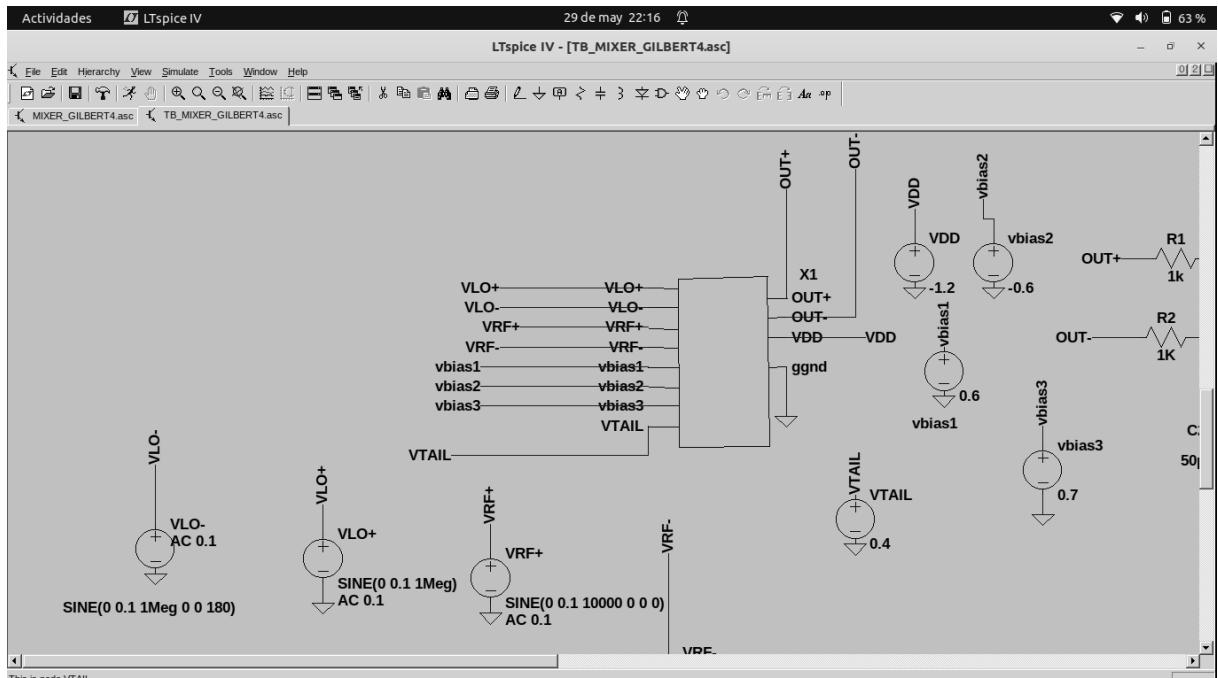


Figura 15 Dispositivos para la simulación (Fuente: Propia)

En las Figuras 16 y 17 se observan dos señales a distintas frecuencias. Posteriormente, en la Figura 18 se muestra la salida del *Mixer*, mientras que la salida invertida se puede ver en la Figura 19. Finalmente, en la Figura 20 se presenta la transformada rápida de Fourier de la señal de frecuencia intermedia a 1 MHz, con una ganancia de -80 dB y una amplitud de 0.26 mV.

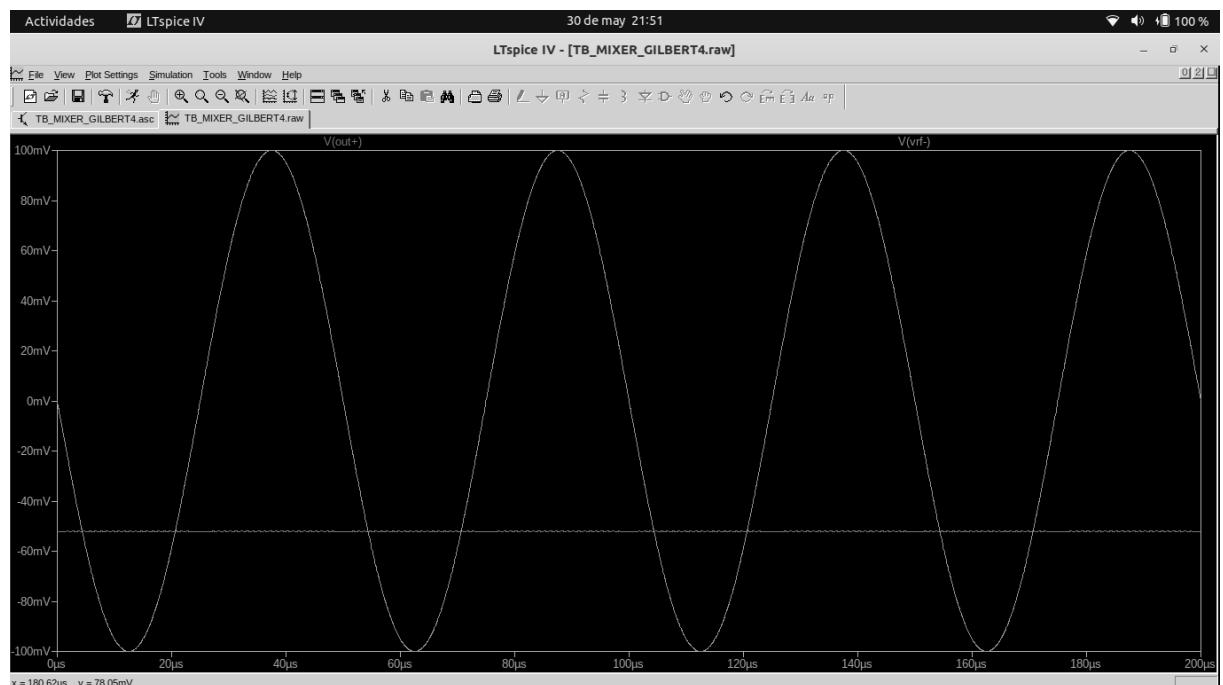


Figura 16 Señal de radiofrecuencia de 20khz (Fuente: Propia)

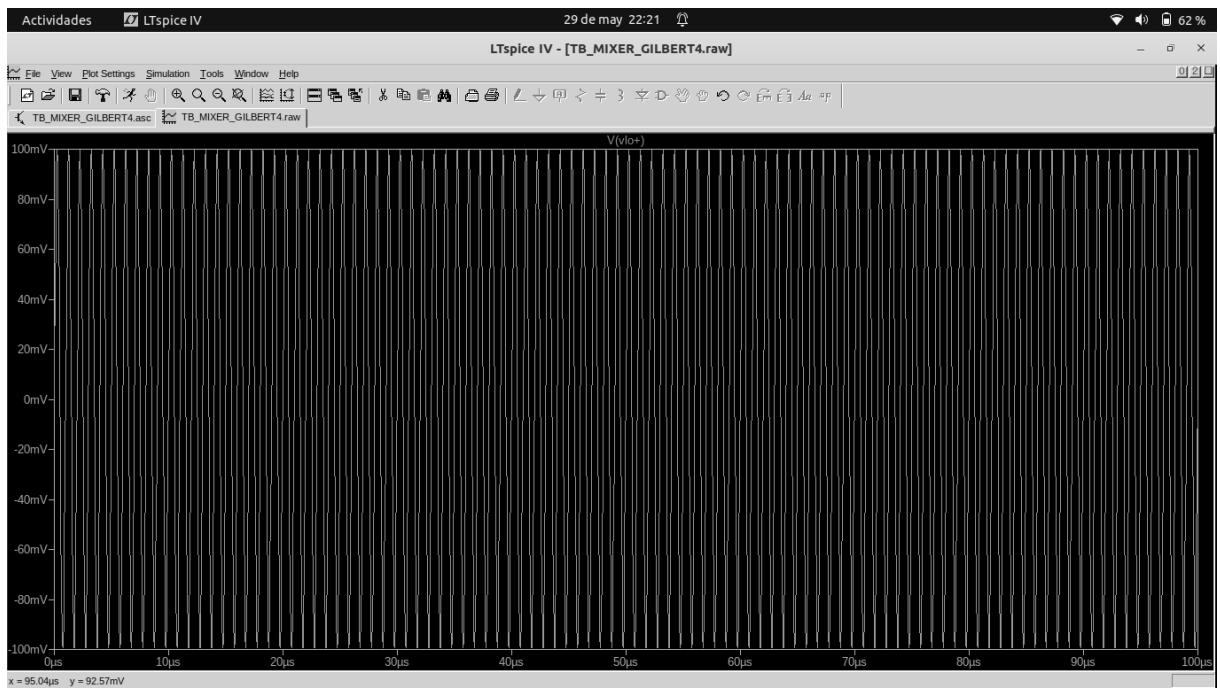


Figura 17 Señal del oscilador local de 1Mhz (Fuente: Propia)

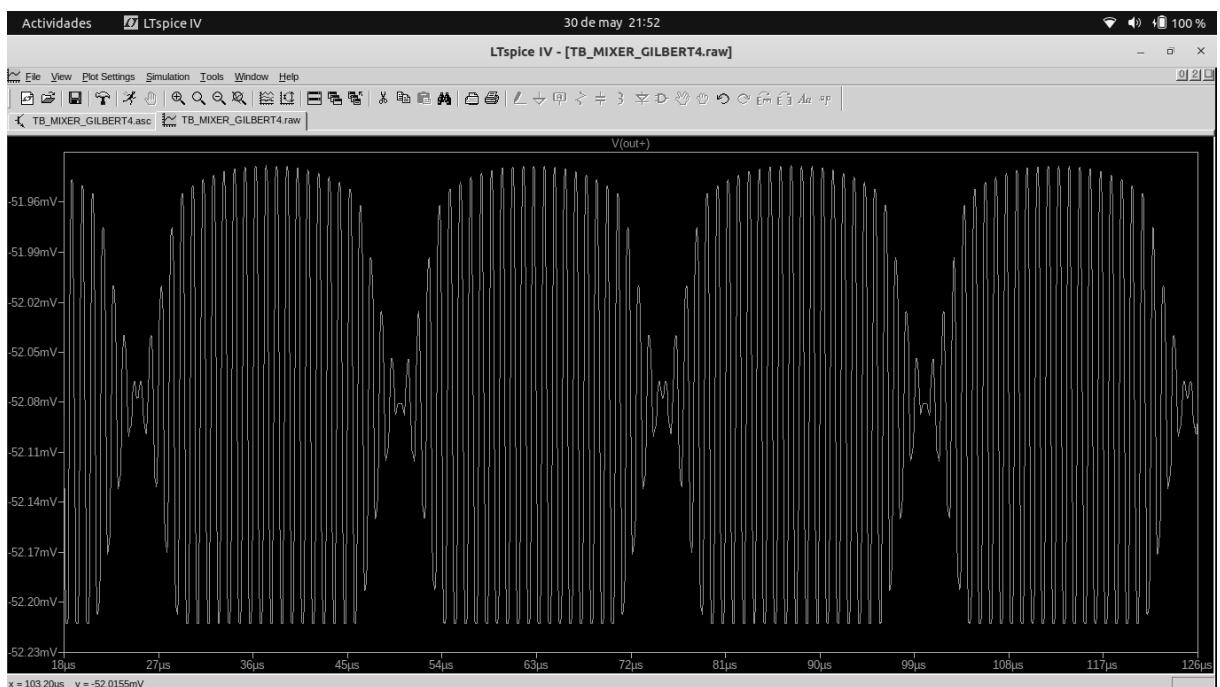


Figura 18 Señal de salida del Mixer de radiofrecuencia (Fuente: Propia)

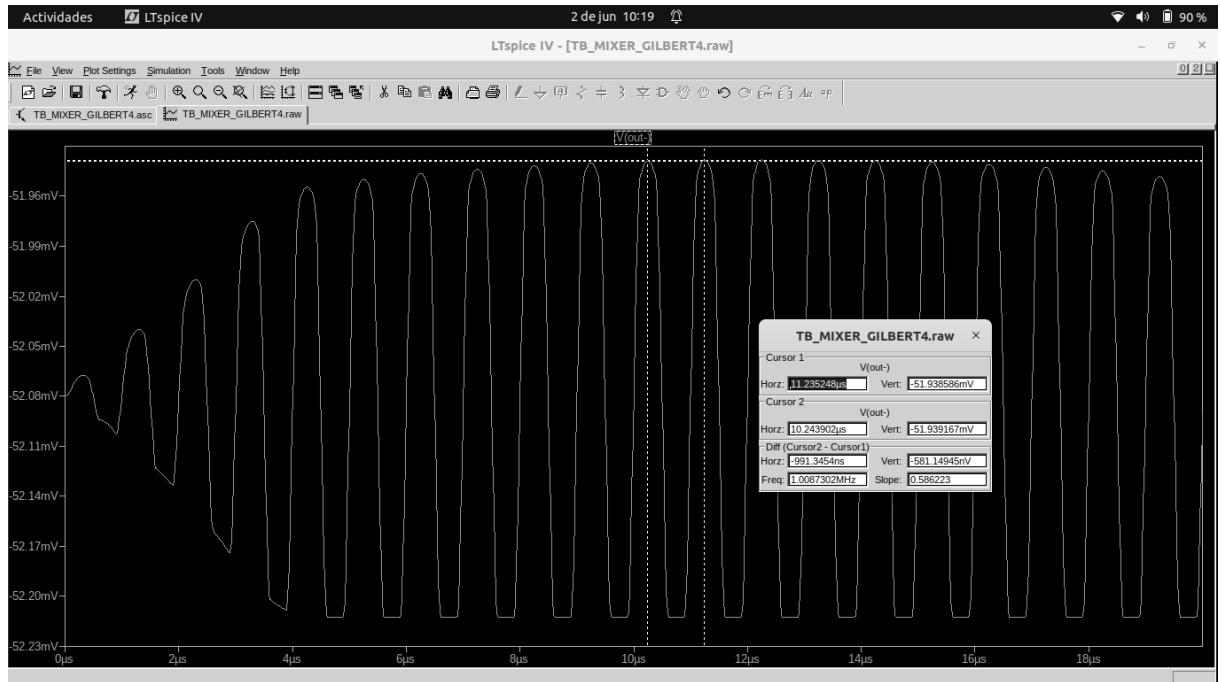


Figura 19 Salida Ampliada out- del Mixer de Radiofrecuencia (Fuente: Propia)

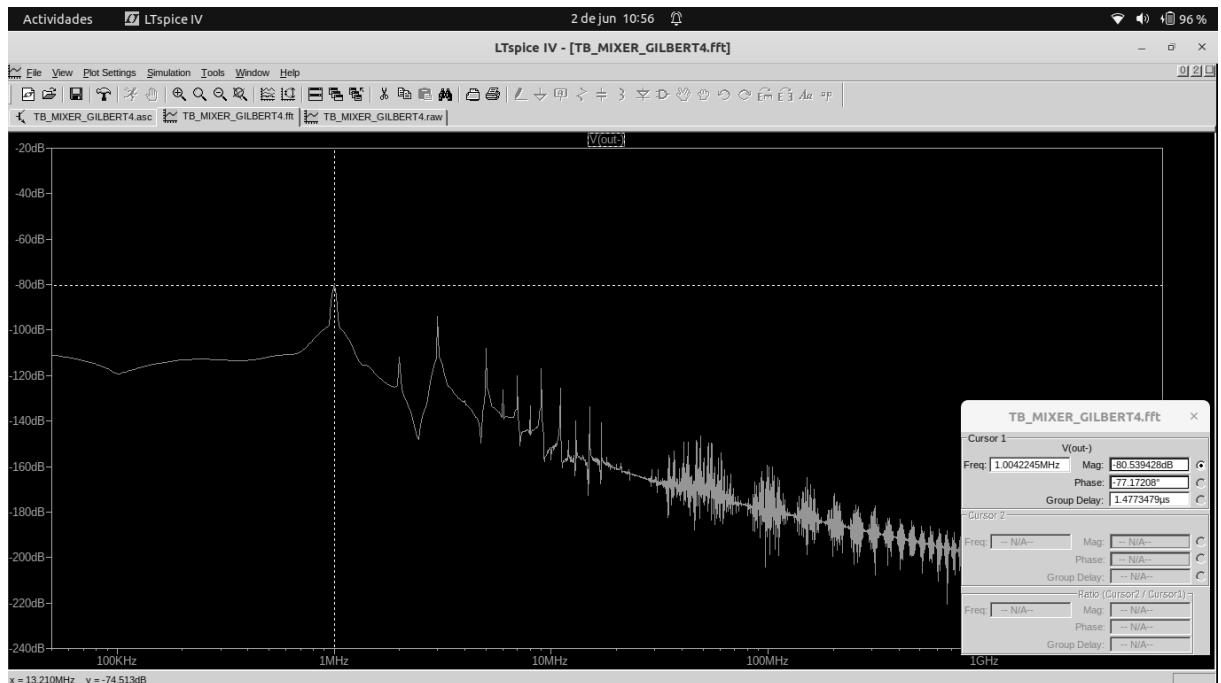


Figura 20 FFT de la señal de frecuencia Intermedia (Fuente: Propia)

Parámetro	Oscilador Local	Radiofrecuencia
-----------	-----------------	-----------------

Frecuencia	1 Mhz	10 KHz
Amplitud (mV)	100	100

Tabla 3 Entradas al Mixer (Fuente: Repositorio [13])

Parámetro	Frecuencia Intermedia	Ganancia
Frecuencia	1 Mhz	-80db
Amplitud(mv)	0.26	

Tabla 4 Salidas del Mixer (Fuente: Repositorio [13])

Los resultados observados en *Ltspice* fueron los esperados estos resultados son los expuestos en las Tablas 3 y la Tabla 4, coincidiendo con la multiplicación de las señales que ingresaron al *Mixer*. Posteriormente, se procedió a implementar el diseño en el chip Caravan, asegurando que los parámetros de funcionamiento fueran óptimos y cumplieran con las especificaciones requeridas. Esta etapa incluyó pruebas adicionales y ajustes finos para garantizar la integración exitosa del circuito en el chip.

Capítulo 3: Integración del Proyecto

En esta sección, se indaga con mayor detenimiento el funcionamiento del proyecto completo. Como ya se mencionó, se abordan la creación del *Mixer*, el *Schmitt Trigger* y el contador. A continuación, se procede a mostrar cómo se combinan todos estos elementos en el chip. En la Figura 21, se observa el flujo de trabajo de las señales.

Se considera el siguiente escenario: la señal a analizar ingresa al SoC por un pin especificado previamente. Luego, esa señal es analizada por el *Mixer*, el cual modula o multiplica la señal. Posteriormente, la señal sale del chip y entra a una placa exterior que demodula la señal. Este demodulador, demodula la señal en la frecuencia intermedia, que es la salida del *Mixer*.

El demodulador recupera la señal de información original de la señal modulada. Dependiendo del tipo de modulación utilizada (AM, FM, PM), se utilizará un tipo de demodulador^[50] específico.

La salida del demodulador es la señal de información original, que luego puede ser procesada, amplificada y utilizada según sea necesario. Una vez demodulada, la señal ingresa de nuevo al chip, pero esta vez se comporta de manera similar a una señal senoidal, y luego la señal puede ser analizada por el *Schmitt Trigger*.

La salida del Schmitt Trigger es una señal cuadrada que ingresa a un proceso *always* del módulo digital, el cual cuenta cada flanco de subida de la señal. Luego, cuando el módulo recibe una señal desde el procesador del chip, comienza a registrar los valores en un registro acumulador, esperando que el procesador envíe la señal de registro. De esta manera, se contabiliza la cantidad de pulsos de la señal. La transmisión entre el RISC-V y el módulo se realiza mediante un bus denominado *Wishbone*. Además, el módulo contiene un pulso que se envía en el momento en que se recibe el valor desde el RISC-V.

Esquema general del proyecto:

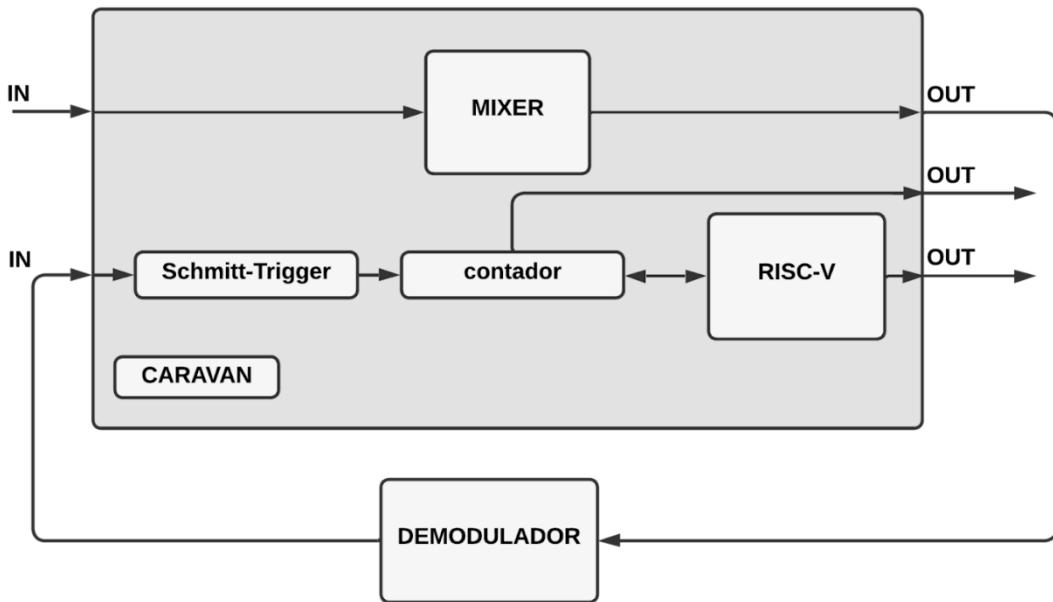


Figura 21 Diagrama descriptivo del proyecto (Fuente: Propia)

En la imagen de la Figura 21, se puede observar que el contador tiene una salida "out". Esta salida representa un pulso que, posteriormente, envía el valor al procesador RISC-V. Cabe recordar que el RISC-V está conectado al área de trabajo mediante el bus Wishbone, permitiendo así la comunicación entre los diferentes dispositivos del chip.

Este procedimiento permite observar el comportamiento conjunto del *Mixer*, el contador y el RISC-V. Se pueden estimar los valores esperados según la frecuencia de trabajo del mixer y compararlos con la salida del chip.

PicoRV32 es un procesador que provee *Efabless* basado en Risc-V ubicado en el repositorio *Caravan* donde se puede escribir un código en C y luego compilarlo con las herramientas necesarias^[40] y obtener el archivo *.hexa* el cual luego se ejecuta en el procesador. Para cargar el programa en el chip, se utiliza la conexión SPI realizando lo que se denomina housekeeping^[47].

User Project Area (área disponible en el chip)

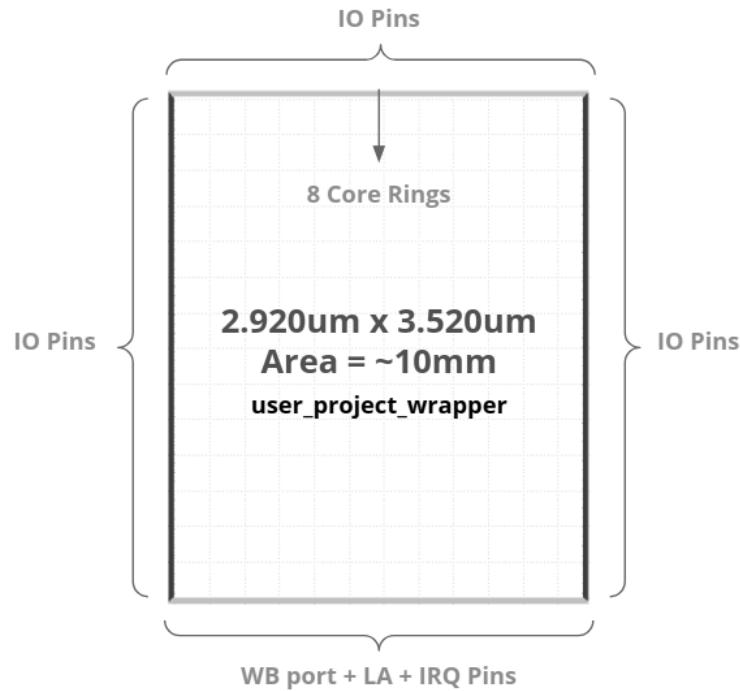


Figura 22 Tamaño del *user_project_area* (Fuente: caravel [7])

Como se observa en la Figura 22, el SoC tiene un espacio del usuario de silicio limitada de 2.92 mm x 3.52 mm, así como un número fijo de 38 pines de entrada/salida y 4 pines de alimentación. El espacio del usuario tiene acceso a las siguientes utilidades proporcionadas por el SoC de gestión:

- 38 puertos de E/S
- 128 sondas de analizador lógico
- Conexión al bus Wishbone del SoC de gestión.

PARTE II

MARCO METODOLÓGICO

Capítulo 4: Proceso de fabricación de un chip

En el siguiente capítulo se explora el proceso de fabricación del chip, incluyendo la selección del nodo tecnológico y los pasos necesarios para llevarlo a cabo. Es importante tener en cuenta que este procedimiento a menudo requiere regresar a los pasos iniciales para realizar mejoras y ajustar el dispositivo a las características específicas del chip.

La Figura 23 muestra de manera abstracta cómo se llevará a cabo el proyecto.

1. Diseño y especificaciones:

- o Definición de las funcionalidades y características del chip.
- o Especificaciones técnicas, incluyendo requisitos de rendimiento, consumo de energía, tamaño, etc.

2. Diseño de esquemático y simulación:

- o Creación del diseño esquemático, que representa la conexión lógica de los componentes del circuito.
- o Simulación del diseño esquemático para verificar que el circuito cumple con las especificaciones y funciona correctamente bajo diversas condiciones.

3. Diseño Layout:

- o Conversión del diseño esquemático a un layout físico.
- o Colocación de componentes y trazado de rutas de interconexión en el chip, asegurando la optimización del espacio y el rendimiento eléctrico.

4. Verificación: DRC y LVS:

- o **DRC (Design Rule Check):** Verificación de que el diseño cumple con las reglas de fabricación establecidas por la tecnología del proceso (espaciado mínimo, tamaño de las vías, etc.).
- o **LVS (Layout Versus Schematic):** Comparación entre el layout y el esquemático para asegurar que el diseño físico corresponde exactamente al diseño lógico.

5. Extracción de corrientes parásitas:

- o Identificación y modelado de capacitancias, inductancias y resistencias parásitas que surgen debido a la proximidad de componentes y rutas de interconexión en el layout.
- o Herramientas de extracción parasitaria se utilizan para generar un modelo más preciso del circuito incluyendo estos efectos.
- o **Si se detectan problemas:** Volver al paso 2 para modificar el diseño esquemático y la simulación, luego actualizar el layout en el paso 3, y repetir la verificación DRC y LVS en el paso 4.

6. **Simulación:**
 - o Simulación del diseño completo, incluyendo los efectos parasitarios, para asegurar que el rendimiento del chip cumple con las especificaciones.
 - o **Si se detectan problemas durante la simulación:** Volver a los pasos 2 y 3 para realizar los ajustes necesarios y luego repetir la extracción de corrientes parásitas y las verificaciones.
7. **Tapeout:**
 - o Preparación de los datos finales del diseño para la fabricación.
 - o Envío de los archivos de diseño a efabless para la creación y la producción del wafer^[54].
8. **Testing:**
 - o Pruebas funcionales y de rendimiento en chips encapsulados.
 - o Identificación de chips defectuosos y análisis de fallas para mejorar futuras iteraciones del diseño.

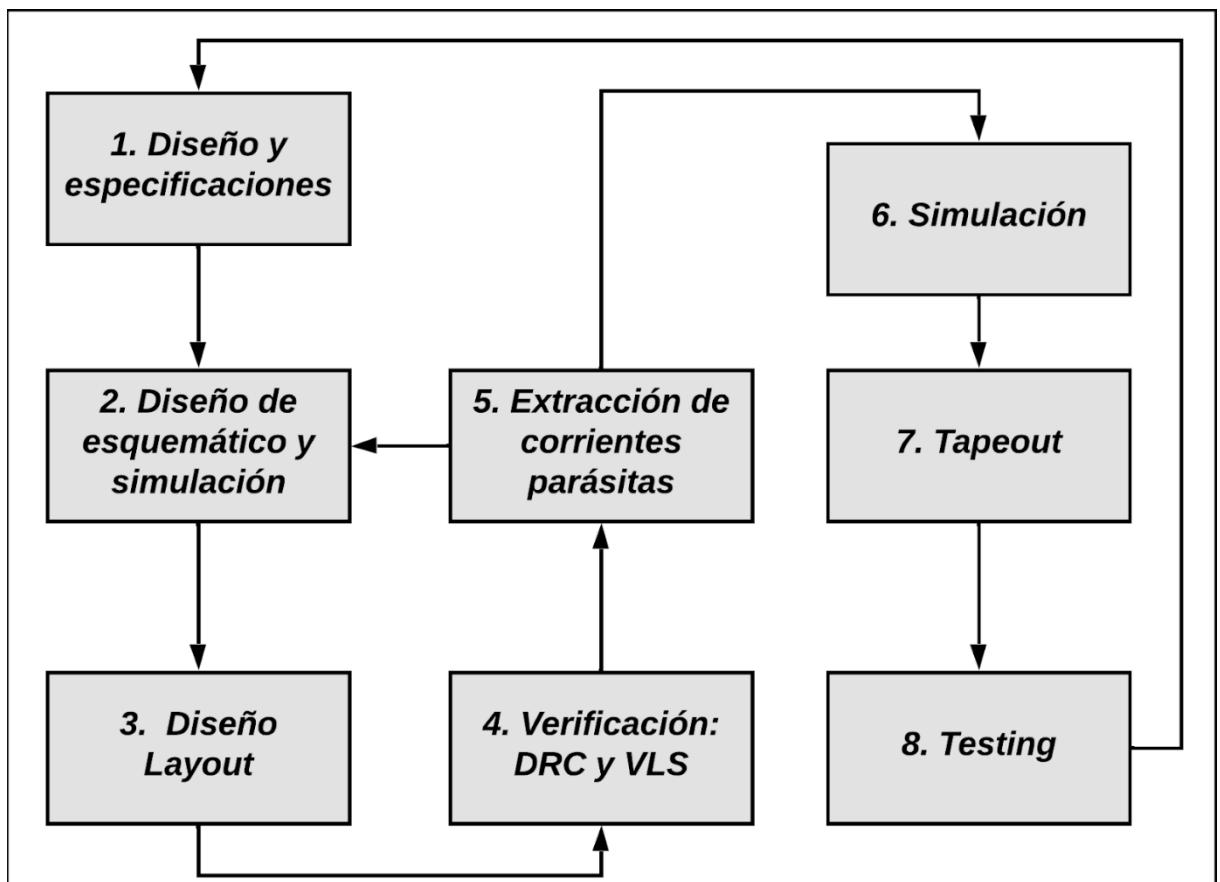


Figura 23 Ruta de trabajo (Fuente: Propia)

4.1. PDK

Un kit de diseño de procesos^[34] (PDK) es un conjunto de archivos empleados en la industria de semiconductores para modelar un proceso de fabricación y las herramientas de diseño necesarias para crear un circuito integrado. El PDK es desarrollado por la empresa o instalación que fabrica los circuitos integrados, definiendo una variación tecnológica específica para sus procesos. Posteriormente, se entrega a los clientes para su utilización en el proceso de diseño. Estos clientes pueden mejorar el PDK, adaptándolo a sus estilos de diseño y a los mercados específicos en los que operan.

Los diseñadores utilizan el PDK para diseñar, simular, dibujar y verificar el diseño antes de devolverlo a la empresa o instalación que fabrica los circuitos integrados para la producción de chips. Los datos del PDK son específicos de la variación del proceso de fundición y se seleccionan al inicio del proceso de diseño, influenciados por los requisitos del mercado para el chip. Un PDK preciso incrementa las probabilidades de que el silicio tenga éxito en la primera fase de producción.

4.1.1. Descripción

Las diferentes herramientas en el flujo de diseño tienen distintos formatos de entrada para los datos del PDK. Los ingenieros de PDK deben decidir qué herramientas soportarán en los flujos de diseño y crear las bibliotecas y conjuntos de reglas que respalden esos flujos.

Un PDK típico contiene:

- Una biblioteca de dispositivos primitivos
- Símbolos
- Parámetros de dispositivos
- PCells (células paramétricas)
- Verificaciones de diseño
 - Comprobación de Reglas de Diseño
 - Comparación de Layout versus Esquemático
 - Verificación de reglas de antena y eléctricas
- Extracción física
- Datos tecnológicos

- o Capas, nombres de capas, pares capa/propósito
- o Colores, rellenos y atributos de visualización
- o Restricciones del proceso
- o Reglas eléctricas
- o Archivos de reglas
- o LEF (Library Exchange Format)
- o Formatos de reglas dependientes de la herramienta
- Modelos de simulación de dispositivos primitivos (SPICE o derivados de SPICE)
 - o Transistores (típicamente SPICE)
 - o Capacitores
 - o Resistores
 - o Inductores
- Manual de Reglas de Diseño
 - o Una representación amigable de los requisitos del proceso

Un PDK también puede incluir bibliotecas de celdas estándar de la fundición, de un proveedor de bibliotecas o desarrolladas internamente:

- Formato LEF de datos de diseño abstractos
- Símbolos
- Archivos de biblioteca (.lib)
- Datos de diseño GDSII

También se tienen bloques de IP (propiedad intelectual) los cuales están optimizados para ser utilizados con el proceso de fabricación específico descrito por el PDK y proporcionan a los diseñadores componentes prediseñados y verificados que pueden integrar en sus diseños, lo cual acelera el proceso de desarrollo y reduce el riesgo de errores.

4.2 PDK de SkyWater

El PDK de SkyWater, fruto de la colaboración entre Google y SkyWater Technology Foundry, marca un hito en la industria al ofrecer un kit de diseño de procesos (PDK) de código abierto[23]. Este kit permite a los diseñadores trabajar con bibliotecas y configuraciones compatibles con herramientas de código abierto, facilitando la fabricación de diseños reales en las instalaciones de SkyWater. Desde el año 2020, este repositorio[24] se ha centrado en el nodo de proceso SKY130, caracterizado por tener una longitud mínima de característica de 130 nanómetros. Además, se ha desarrollado un nodo de 90 nanómetros, el cual se prevé que sea lanzado como código abierto en el futuro, representando un avance significativo en la tecnología de fabricación.

La Figura 24 se muestra una aproximación de los distintos niveles de metales y aislantes del nodo, así como la distancia entre cada nivel.

Características del nodo

- Soporte para 1.8V interno con I/Os de 5.0V (operable a 2.5V)
 - 1 nivel de interconexión local
 - 5 niveles de metal
 - Capacidad de inductor
 - Posee un resistor de polisilicio con alta resistividad superficial (sheet rho)
 - Capacitores MiM opcionales
 - Incluye celda reducida SONOS
 - Nodo de 130 nm
 - Compatible con suministro regular de 10V
 - NMOS y PMOS con drenaje extendido de alta tensión (Hv extended-drain).

(Diagram not to scale!)

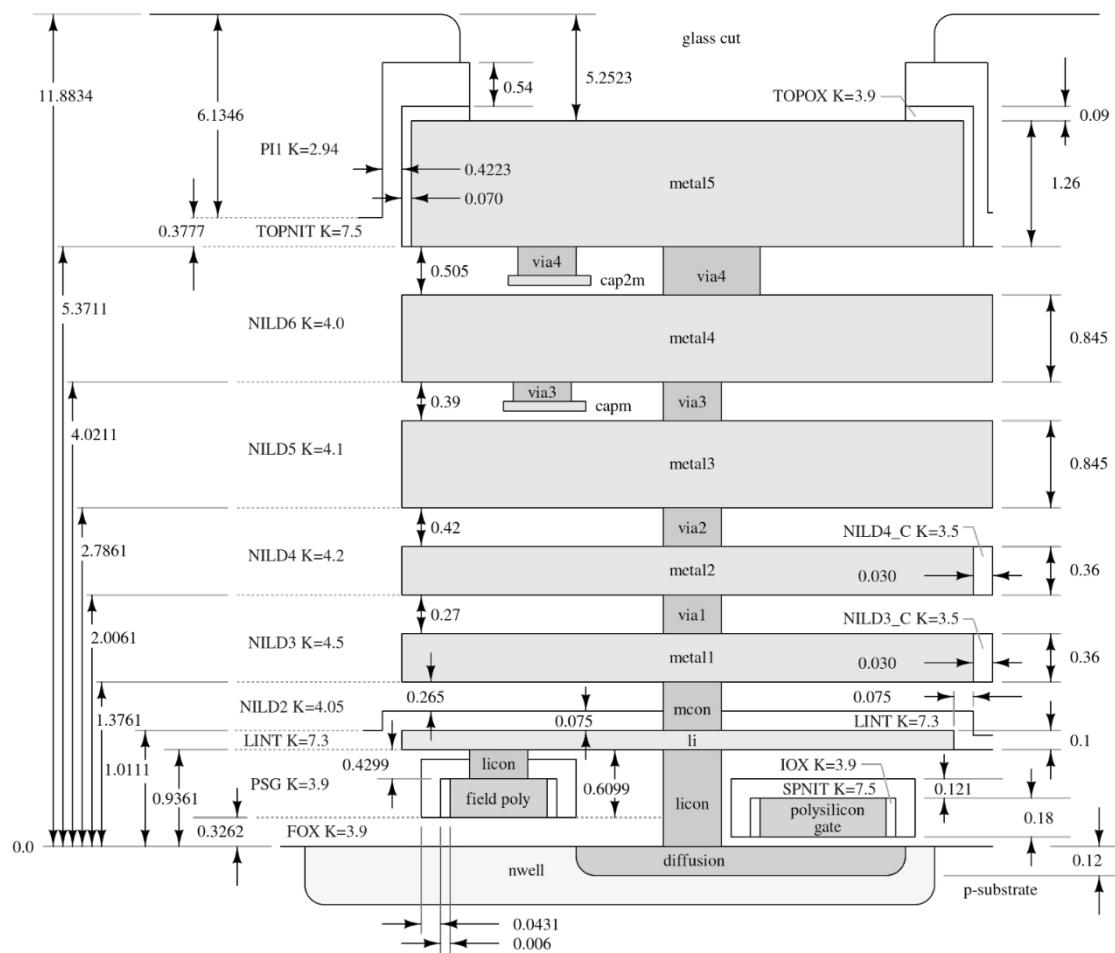


Figura 24 Skywater SKY130nm (Fuente: SkyWater130 [23])

Para más información acerca de esta tecnología, en la bibliografía se encuentran los enlaces a la documentación oficial de SkyWater130^[23].

Capítulo 5: Proceso del diseño

Para emprender un proyecto en el SoC Caravel^[1-2], es esencial considerar lo siguiente: en primer lugar, determinar si el objetivo del proyecto es de naturaleza digital o analógica. Aunque ambas versiones de Caravel ofrecen acceso a pines analógicos y digitales, la versión Caravan destaca por su idoneidad para proyectos analógicos de alta frecuencia, mientras que la versión estándar de Caravel^[2] es más adecuada para proyectos digitales o analógicos de baja frecuencia. Sin embargo, es importante tener en cuenta que la versión estándar de Caravel carece de pines con protección ESD. Dado que el proyecto en consideración implica una componente principal analógica con una pequeña parte digital, se recomienda evaluar la versión Caravan^[9] para su implementación.

La protección ESD^[29] (*Electrostatic Discharge*, o Descarga Electroestática) en los pines se refiere a medidas diseñadas para proteger los componentes electrónicos de daños causados por descargas electrostáticas. Estas descargas pueden ocurrir cuando dos objetos con cargas electrostáticas diferentes entran en contacto, generando una corriente momentánea que puede dañar o destruir los componentes sensibles de un circuito electrónico.

La plataforma *Efabless* ofrece dos enfoques para trabajar con el *harness*^[31]. Uno de estos métodos es más adecuado para el diseño digital, es decir, se puede diseñar el circuito digital utilizando el lenguaje de descripción de hardware Verilog y luego integrarlo en el *wrapper* mediante *OpenLane*^[10]. Para lograr esto, es importante que el circuito esté relacionado con el *wrapper*, lo que implica que los pines digitales o analógicos del *wrapper* deben coincidir con los del circuito diseñado en la subsección 5.1.4 se describe el flujo de trabajo para llevar a cabo la parte digital.

5.1. Circuitos digitales

5.1.1. Introducción

En el diseño de circuitos digitales, el nivel de transferencia de registros (RTL) sirve como una abstracción que modela un circuito digital síncrono al representar el flujo de señales digitales (datos) entre registros de hardware y las operaciones lógicas realizadas en esas señales. Este nivel

de abstracción se utiliza comúnmente en lenguajes de descripción de hardware (HDL) como *Verilog* y *VHDL* para crear representaciones de alto nivel de circuitos.

A partir de estas representaciones, se pueden derivar detalles de nivel inferior y el cableado real del circuito. Diseñar a nivel de RTL es una práctica estándar en el diseño digital moderno. La modelización a nivel de transacción representa un nivel superior de diseño de sistemas electrónicos, proporcionando una vista más abstracta del comportamiento del sistema.

En este proyecto, se utiliza *Verilog* debido a su amplio uso por parte de *Efabless* en numerosos módulos de construcción del chip. Para esta etapa específica del proyecto, se ha dividido el proceso en varias fases. Se aprovecha el repositorio de Caravan, que cuenta con secciones dedicadas a la inclusión de módulos *Verilog* y código en C, así como su posterior sometimiento a pruebas mediante *Icarus Verilog* (*iverilog*).

En la Figura 25 se proporciona una representación abstracta de cómo se lleva a cabo la configuración del test.

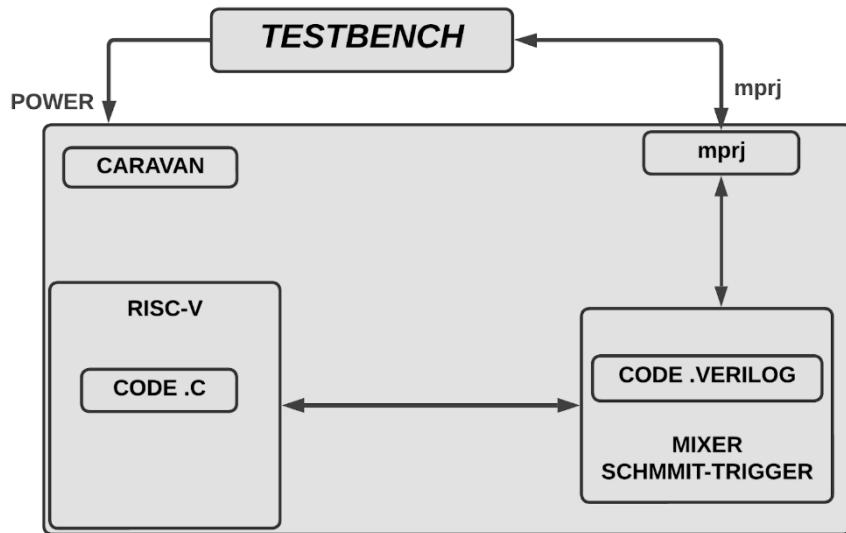


Figura 25 Unidad bajo prueba (Fuente: Propia)

Primero, se configura el programa en C, donde se establecen los pines a utilizar. Luego, se ajustan los módulos del *user_analog_project_wrapper* y, finalmente, se configura el *testbench* para definir cómo se manejará el chip desde el exterior. Para una comprensión más detallada de las configuraciones posibles, se deja el Anexo A a disposición.

5.1.2. Configuración en C (Firmware)

A continuación, se presenta la comunicación entre el *PicoRV32*^[39] (RISC-V^[22] integrado en el chip) y el área *user_analog_project_wrapper*. En el código comprendido en la serie de Figuras 26, se observa la transferencia de datos en el procesador PicoRV32^[39] y regiones de memorias utilizando punteros y el *bus* de comunicación *wishbone*, en la Figura 26 A línea 1 y 2 se definen las direcciones de las dos regiones de memoria en las cuales se escriben datos. En las líneas 29 y 33 de la Figura 26 B se escriben los datos que luego se leen en el *user_analog_project_wrapper*.

```

1 #define reg_set_value    (*(volatile uint32_t*)0x30000000)
2 #define reg_get_value    (*(volatile uint32_t*)0x30000004)
3
4 void UART_enableRX(bool is_enable);
5 void UART_enableTX(bool is_enable);
6 void UART_sendInt(int data);
7 void UART_sendChar(char c);
8 void delay(volatile uint32_t time);
9
10 void main()
11 {
12     uint32_t value;
13     int time;
14     reg_mprj_io_35 = GPIO_MODE_USER_STD_OUTPUT; // 8
15     reg_mprj_io_6  = GPIO_MODE_MGMT_STD_OUTPUT;
16     UART_enableTX(1);
17
18     reg_wb_enable=1;
19     reg_mprj_xfer = 1;
20
21     /* Apply configuration */
22     while (reg_mprj_xfer == 1);
23
24     time = 50;
25     reg_get_value = 0;
26
27     while(1)
28     {
29         reg_set_value = 0x07; // value of init
30
31         delay(time);

```

Figura 26 Serie de Figuras 26 de secciones de código (Fuente: Propia)

Figura 26 A Primera sección de código

```

10 void main()
11 {
12     uint32_t value;
13     int time;
14     reg_mprj_io_35 = GPIO_MODE_USER_STD_OUTPUT; // 8
15     reg_mprj_io_6 = GPIO_MODE_MGMT_STD_OUTPUT;
16     UART_enableTX(1);
17
18     reg_wb_enable=1;
19     reg_mprj_xfer = 1;
20
21     /* Apply configuration */
22     while (reg_mprj_xfer == 1);
23
24     time = 50;
25     reg_get_value = 0;
26
27     while(1)
28     {
29         reg_set_value = 0x07; // value of init
30
31         delay(time);
32
33         reg_set_value = 0x08;
34
35         while(reg_get_value > 0 ){
36             value = reg_get_value;
37             UART_sendInt(value);
38             delay(160);
39             reg_get_value = 0;
40             break;
41         };
42     }
43 }
```

Figura 26 B Segunda sección de código (Fuente: Propia)

```

45
46 void delay(volatile uint32_t time) {
47     while (time > 0) time--;
48 }
49
50 void UART_enableRX(bool is_enable){
51     if (is_enable){
52         reg_uart_enable = 1;
53     }else{
54         reg_uart_enable = 0;
55     }
56 }
57
58
59 void UART_enableTX(bool is_enable){
60     if (is_enable){
61         reg_uart_enable = 1;
62     }else{
63         reg_uart_enable = 0;
64     }
65 }
66
67 void UART_sendChar(char c){
68     while (reg_uart_txfull == 1);
69     reg_uart_data = c;
70 }

```

Figura 26 C Tercera sección de código (Fuente: Propia)

```

67 void UART_sendChar(char c){
68     while (reg_uart_txfull == 1);
69     reg_uart_data = c;
70 }
71
72 void UART_sendInt(int data){
73     for (int i = 0; i < 8; i++) {
74         // Extract the current 4-bit chunk
75         int chunk = (data >> (i * 4));
76         if (chunk == 0) {
77             break;
78         }
79         chunk = chunk & 0x0F;
80         char ch;
81         if (chunk >= 0 && chunk <= 9) {
82             ch = '0' + chunk; // Convert to corresponding decimal digit character
83         } else {
84             ch = 'A' + (chunk - 10); // Convert to corresponding hex character A-F
85         }
86         UART_sendChar(ch);
87     }
88     UART_sendChar('\n');
89 }

```

Figura 26 D Cuarta sección de código (Fuente: Propia)

Procedimiento:

Configuración de puertos:

- Se establece un puerto como salida del chip Figura 26A línea 14.

Habilitar el pin uart:

- Configurar el pin 6 como salida uart Figura 26A línea 15.

Activar comunicación:

- Activar la comunicación para transmisión función `UART_enableTX` (`reg_uart_enable = 1`) Figura 26A línea 16.

Habilitar la comunicación:

- Activar la comunicación wishbone Figura 26A línea 18 (`reg_wb_enable = 1`).

Configuración:

- Esperar a que la configuración esté lista en el chip (`reg_mprj_xfer = 1; while (reg_mprj_xfer == 1)`) Figura 26A línea 28.

5.1.3. Configuración del testbench

En esta sección se describe la configuración del testbench para el código en C mostrado en la subsección 5.1.2. Se establecen los registros utilizados por la *UUT* (Unidad Bajo Test) que incluyen la energía del chip, los GPIO y el reloj, entre otros^[53]. En el caso de este análisis, dado que se está interactuando con el *user_analog_project_wrapper*, se presta especial atención al registro *mpkj*, que es observado por el *testbench* para cambiar su estado de acuerdo con la configuración proporcionada.

Por ejemplo la variable *uart_pulse* (línea 21), el cual está asociada a un pin del registro *mpkj_io*(*mpkj_io* mismo *mpkj*), cambia de 0 a 1 con cada envío por uart desde el chip hacia un dispositivo externo en este caso se tiene una variable *pulse_counter* = 5 en la línea 74 Figura 27 C lo cual nos indica que la simulación se realiza para 5 envíos por uart del contador.

Existe una interacción directa entre el código en C y el módulo *user_analog_project_wrapper*.

```

1  `default_nettype none
2
3  `timescale 1 ns / 1 ps
4
5  module test_mixer_tb;
6      reg clock;
7      reg RSTB;
8      reg CSB;
9      reg power1, power2;
10     reg power3, power4;
11
12     wire gpio;
13     wire uart_rx;
14     wire [37:0] mprj_io;
15     wire uart_pulse;
16     reg toggle;
17     reg [3:0] pulse_counter;
18
19     assign uart_rx = mprj_io[6];
20     assign mprj_io[7] = toggle;
21     assign uart_pulse = mprj_io[35];
22
23     always #12.5 clock <= (clock === 1'b0); // Frecuencia del clock 40 MHz
24     always #500 toggle <= (toggle === 1'b0); // Frecuencia del pulso 1 MHz.
25
26     initial begin
27         clock = 0;
28         toggle = 0;
29         pulse_counter = 5;
30     end
31
32     initial begin

```

Figura 27 Serie de Figuras 27 de secciones de código (Fuente: Propia)

Figura 27 A Primera sección de código

```

5   module test_mixer_tb;
31
32     initial begin
33
34       $dumpfile("test_mixer.vcd");
35       $dumpvars(0, test_mixer_tb);
36
37       repeat (400) begin
38         repeat (1000) @(posedge clock);
39       end
40       $display("%c[1;31m",27);
41       $display ("Monitor: Timeout, Test Mega-Project IO Ports (RTL) Failed");
42       $display("%c[0m",27);
43       $finish;
44     end
45
46     initial begin
47       RSTB <= 1'b0;
48       CSB  <= 1'b1; // Force CSB high
49       #2000;
50       RSTB <= 1'b1; // Release reset
51       #170000;
52       CSB = 1'b0;    // CSB can be released
53     end
54
55     initial begin      // Power-up sequence
56       power1 <= 1'b0;
57       power2 <= 1'b0;
58       power3 <= 1'b0;
59       power4 <= 1'b0;
60       #100;
61       power1 <= 1'b1;

```

Figura 27 B Segunda sección de código (Fuente: Propia)

```

53      end
54
55      initial begin      // Power-up sequence
56          power1 <= 1'b0;
57          power2 <= 1'b0;
58          power3 <= 1'b0;
59          power4 <= 1'b0;
60          #100;
61          power1 <= 1'b1;
62          #100;
63          power2 <= 1'b1;
64          #100;
65          power3 <= 1'b1;
66          #100;
67          power4 <= 1'b1;
68      end
69
70
71      always @ (posedge uart_pulse) begin
72          pulse_counter <= pulse_counter - 1;
73          #1 $display("uart_pulse state = %b ", uart_pulse);
74          if (pulse_counter == 0)
75              begin
76                  wait(uart_pulse==1)
77                  wait(uart_pulse==0)
78                  wait(uart_pulse==1)
79                  $display("Fin Test");
80                  $finish;
81              end
82      end
83

```

Figura 27 C Tercera sección de código (Fuente: Propia)

```

84      wire flash_csb;
85      wire flash_clk;
86      wire flash_io0;
87      wire flash_iol;
88
89      wire VDD3V3 = power1;
90      wire VDD1V8 = power2;
91      wire USER_VDD3V3 = power3;
92      wire USER_VDD1V8 = power4;
93      wire VSS = 1'b0;
94
95      assign mprj_io[3] = 1; // Force CSB high.
96      assign mprj_io[0] = 0; // Disable debug mode
97
98      caravan uut (
99          .vddio      (VDD3V3),
100         .vssio      (VSS),
101         .vdda      (VDD3V3),
102         .vssa      (VSS),
103         .vccd      (VDD1V8),
104         .vssd      (VSS),
105         .vddal      (USER_VDD3V3),
106         .vdda2      (USER_VDD3V3),
107         .vssa1      (VSS),
108         .vssa2      (VSS),
109         .vccd1      (USER_VDD1V8),
110         .vccd2      (USER_VDD1V8),
111         .vssd1      (VSS),
112         .vssd2      (VSS),
113         .clock      (clock),
114

```

Figura 27 D Cuarta sección de código (Fuente: Propia)

```

97     assign mprj_io[0] = 0; // Disable debug mode
98
99     caravan uut (
100         .vddio      (VDD3V3),
101         .vssio      (VSS),
102         .vdda      (VDD3V3),
103         .vssa      (VSS),
104         .vccd      (VDD1V8),
105         .vssd      (VSS),
106         .vddal1    (USER_VDD3V3),
107         .vda2      (USER_VDD3V3),
108         .vssal1    (VSS),
109         .vssa2      (VSS),
110         .vccd1      (USER_VDD1V8),
111         .vccd2      (USER_VDD1V8),
112         .vssd1      (VSS),
113         .vssd2      (VSS),
114         .clock      (clock),
115         .gpio       (gpio),
116         .mprj_io   (mprj_io),
117         .flash_csb(flash_csb),
118         .flash_clk(flash_clk),
119         .flash_io0(flash_io0),
120         .flash_io1(flash_io1),
121         .resetb    (RSTB)
122     );
123
124     spiflash #(
125         .FILENAME("test_mixer.hex")
126     ) spiflash (
127         .csb(flash_csb),

```

Figura 27 E Quinta sección de código (Fuente: Propia)

```

99      caravan_uut (
116      .mprj_io  (mprj_io),
117      .flash_csb(flash_csb),
118      .flash_clk(flash_clk),
119      .flash_io0(flash_io0),
120      .flash_io1(flash_io1),
121      .resetb   (RSTB)
122 );
123
124     spiflash #(
125         .FILENAME("test_mixer.hex")
126     ) spiflash (
127         .csb(flash_csb),
128         .clk(flash_clk),
129         .io0(flash_io0),
130         .io1(flash_io1),
131         .io2(),           // not used
132         .io3()            // not used
133     );
134     tbuart tbuart (
135         .ser_rx(uart_rx)
136     );
137 endmodule
138 `default_nettype wire

```

Figura 27 F Sexta sección de código (Fuente: Propia)

En la Figura 27 A se observa en la línea 3 la definición de la escala de tiempo y la precisión de la simulación. Aquí, el tiempo se mide en nanosegundos y la precisión es de picosegundos. De la línea 6 a la 17 se definen varias señales de registro (reg) y cables (wire) que se utilizarán en la simulación. Luego, en las líneas 19-21, se aplica la generación de señales. La generación de relojes y pulsos se encuentra en las líneas 23-24 y la inicialización de variables en las líneas 27-29. El tiempo máximo de ejecución se define en las líneas 32-44 Figura 27 B. Se realiza una secuencia de reinicio y configuración del chip select (CSB) en las líneas 47-52 Figura 27 B, y una secuencia de encendido en las líneas 55-68 Figura 27 C. El manejo del pulso UART se encarga de contar los pulsos recibidos en *uart_pulse* y finalizar la simulación después de un número específico de pulsos, detallado en las líneas 71-82 Figura 27 C.

- Declaración de Señales para la Memoria Flash

Se declaran cables (wire) para las señales de la memoria flash SPI líneas 85-88 Figura 27

D.

`flash_csb`: Chip Select Bar, controla la selección del chip de memoria.

`flash_clk`: Clock, proporciona la señal de reloj para la comunicación SPI.

`flash_io0` y `flash_io1`: Líneas de entrada/salida de datos para la comunicación SPI.

- Conexión de Fuentes de Alimentación

Se asignan cables a las señales de potencia del sistema líneas Figura 27 D líneas 90-94, donde:

- `VDD3V3` se asigna a `power1`, representa una fuente de alimentación de 3.3V.
- `VDD1V8` se asigna a `power2`, representa una fuente de alimentación de 1.8V.
- `USER_VDD3V3` se asigna a `power3`, otra fuente de 3.3V utilizada por el usuario.
- `USER_VDD1V8` se asigna a `power4`, otra fuente de 1.8V utilizada por el usuario.
- `VSS` se asigna a `0V`, representa la conexión a tierra (GND).

- Configuración de Pines del Proyecto

Se realizan asignaciones específicas para los pines del módulo de proyecto Figura 27 D líneas 96-97 (`mprj_io`):

- `mprj_io[3] = 1`: Fuerza el pin 3 del módulo de proyecto a un valor alto (1), lo que equivale a mantener la señal CSB alta, asegurando que el chip select está desactivado inicialmente.
- `mprj_io[0] = 0`: Fuerza el pin 0 del módulo de proyecto a un valor bajo (0), lo que deshabilita el modo de depuración.

La instanciación de módulos incluye:

- Instancia el módulo Caravan con sus conexiones de energía y señales línea 99-122 Figura 27 E.
- Instancia un módulo de memoria flash (`spiflash`) para cargar el archivo `test_mixer.hex` línea 124-133 Figura 27 F.
- Instancia un módulo UART (`tuart`) para manejar la recepción de datos UART línea 34-36 Figura 27 F.

Este testbench se realiza dentro de *Docker* utilizando la herramienta provista por *Eflabless Icarus Verilog*^[38].

5.1.4. Configuración del RTL

La etapa final del proceso implica la preparación del módulo que se integrará en el módulo *user_analog_project_wrapper*. Esta fase es crítica, ya que cualquier error no detectado podría resultar en complicaciones posteriores, una vez que el chip esté fabricado.

La conexión del módulo con el *user_analog_project_wrapper* con el submódulo *test_mixer* se observa en la *Figura 28*. Lo primero que se conecta es la alimentación al submódulo (líneas 128 a 129), luego la transmisión wishbone (líneas 132 a 141) y de las líneas 143 a 145 los pines utilizados por el submódulo *test_mixer*.

```
122  /*-----*/
123  /* User project is instantiated here */
124  /*-----*/
125
126  test_mixer test_mixer (
127  |`ifdef USE_POWER_PINS
128  |  .vccd1(vccd1),
129  |  .vssd1(vssd1),
130  `endif
131
132  .wb_clk_i(wb_clk_i),
133  .wb_rst_i(wb_rst_i),
134  .wbs_cyc_i(wbs_cyc_i),
135  .wbs_stb_i(wbs_stb_i),
136  .wbs_we_i(wbs_we_i),
137  .wbs_sel_i(wbs_sel_i),
138  .wbs_adr_i(wbs_adr_i),
139  .wbs_dat_i(wbs_dat_i),
140  .wbs_ack_o(wbs_ack_o),
141  .wbs_dat_o(wbs_dat_o),
142
143  .io_in (io_in),
144  .io_out(io_out),
145  .io_oeb(io_oeb)
146 );
```

Figura 28 Instancia del módulo test_mixer (Fuente: Propia)

En la Figura 29 se observa una máquina de estados que representa al submódulo *test_mixer*. Se parte de un *Reset* y luego el submódulo espera valores desde el procesador para iniciar la acumulación de la cuenta de los picos de la señal cuadrada que ingresa por un pin al *always*, Figura 30 E, línea 88.

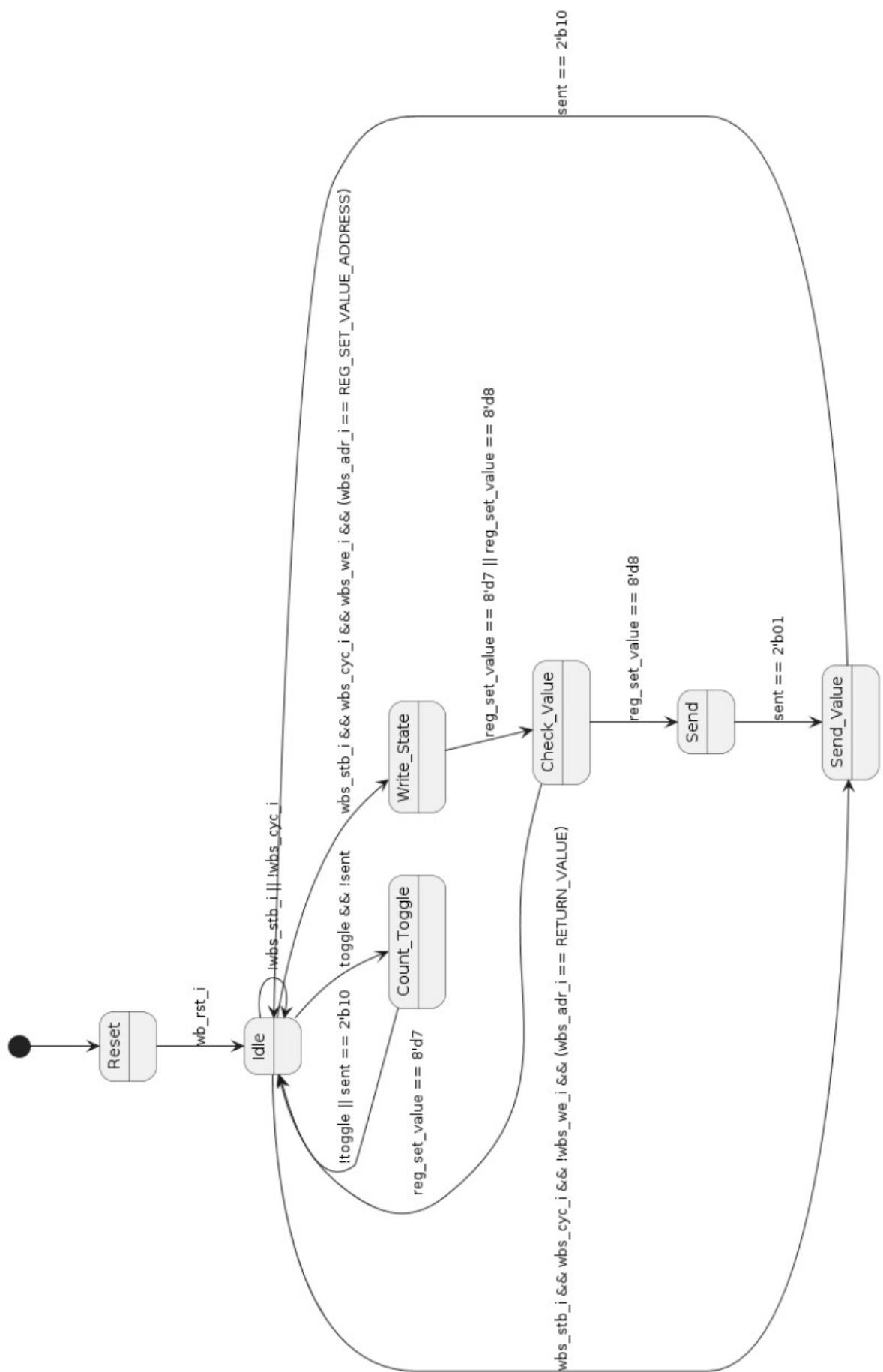


Figura 29 Máquina de estados (Fuente: Propia)

```

1 `default_nettype none
2
3 module test_mixer #(
4     parameter [31:0] REG_SET_VALUE_ADDRESS = 32'h3000_0000,
5     parameter [31:0] RETURN_VALUE         = 32'h3000_0004,
6     parameter     BITS_8 = 8
7 ) (
8     `ifdef USE_POWER_PINS
9         inout vccd1,
10        inout vssd1,
11     `endif
12
13     input wb_clk_i,
14     input wb_rst_i,
15     input wbs_stb_i,
16     input wbs_cyc_i,
17     input wbs_we_i,
18     input [3:0] wbs_sel_i,
19     input [31:0] wbs_dat_i,
20     input [31:0] wbs_addr_i,
21     output reg wbs_ack_o,
22     output reg [31:0] wbs_dat_o,
23
24     // IOs
25     input [`MPRJ_IO_PADS-`ANALOG_PADS-1:0] io_in,
26     output reg [`MPRJ_IO_PADS-`ANALOG_PADS-1:0] io_out,
27     output [`MPRJ_IO_PADS-`ANALOG_PADS-1:0] io_oeb
28
29 );
30

```

Figura 30 Serie de Figuras 11 de secciones de código (Fuente: Propia)

Figura 30 A Primera sección de código (Fuente: Propia)

```

31      reg [31:0] contador = 0;
32      reg [31:0] valor_final = 0;
33      reg [1:0]  sent;
34
35      wire  toggle;
36      assign toggle = io_in[7];
37
38      assign  io_oeb = {(`MPRJ_IO_PADS-`ANALOG_PADS){1'b0}};
39
40      initial
41      begin
42          |    io_out = 0;
43          |    sent = 2'b00;
44      end
45
46      reg [(BITS_8-1):0]reg_set_value;
47

```

Figura 30 B Segunda sección de código (Fuente: Propia)

```

48      always @(posedge wb_clk_i) begin
49          if(wb_rst_i)
50          begin
51              reg_set_value <= {BITS_8{1'b0}};
52              sent <= 0;
53          end
54          else if(wbs_stb_i && wbs_cyc_i && wbs_we_i)
55          case(wbs_adr_i)
56              REG_SET_VALUE_ADDRESS:
57              begin
58                  reg_set_value <= wbs_dat_i[(BITS_8-1):0];
59                  if (reg_set_value == 8'd7)
60                  begin
61                      //reset_values <= 1;
62                      reg_set_value <= {BITS_8{1'b0}};
63                      sent <= 2'b00;
64                  end
65                  if (reg_set_value == 8'd8)
66                  begin
67                      sent <= 2'b01;
68                      reg_set_value <= 0;
69                  end
70              end
71              default:
72                  wbs_dat_o <= 32'b0;
73          endcase

```

Figura 30 C Tercera sección de código (Fuente: Propia)

```

54         else if(wbs_stb_i && wbs_cyc_i && wbs_we_i)
55             case(wbs_adr_i)
56                 default:
57                     |     wbs_dat_o <= 32'b0;
58                 endcase
59             else if(wbs_stb_i && wbs_cyc_i && !wbs_we_i)
60                 case(wbs_adr_i)
61                     RETURN_VALUE:
62                         if (sent == 2'b01)
63                             begin
64                             |     wbs_dat_o <= contador;
65                             |     sent  <= 2'b10;
66                             end
67                         default:
68                             |     wbs_dat_o <= 32'b0;
69                 endcase
70
71             end
72
73         always @(posedge toggle) begin
74
75             if(!sent)
76                 begin
77                 |     contador <= contador + 1;
78             end
79             else if(sent == 2'b10)
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94

```

Figura 30 D Cuarta sección de código (Fuente: Propia)

```

85      |
86      end
87
88      always @ (posedge toggle) begin
89
90          if (!sent)
91          begin
92              contador <= contador + 1;
93          end
94          else if (sent == 2'b10)
95          begin
96              contador <= 0;
97          end
98      end
99
100     always @ (posedge wb_clk_i) begin
101         if (wb_rst_i) begin
102             io_out[24] <= 0;
103         end
104         else if (sent <= 2'b01)
105         begin
106             io_out[24] <= 1'b1;
107         end
108         else io_out[24] <= 1'b0;
109     end
110

```

Figura 30 E Quinta sección de código (Fuente: Propia)

```

99
100    always @ (posedge wb_clk_i) begin
101        if (wb_rst_i) begin
102            io_out[24] <= 0;
103        end
104        else if (sent <= 2'b01)
105        begin
106            io_out[24] <= 1'b1;
107        end
108        else io_out[24] <= 1'b0;
109    end
110
111    always @ (posedge wb_clk_i) begin
112        if (wb_rst_i)
113            wbs_ack_o <= 0;
114        else
115            wbs_ack_o <= (wbs_stb_i && (wbs_addr_i == REG_SET_VALUE_ADDRESS || wbs_addr_i == RETURN_VALUE));
116    end
117
118 endmodule
119
120

```

Figura 30 F Sexta sección de código (Fuente: Propia)

En el módulo de la Figura 30, se monitorea constantemente el cambio del *bus wishbone*, ya que este *bus* es el responsable de la comunicación entre el código en C en el procesador *PicoRV32* y el *user_analog_project_wrapper*.

Parámetros del Módulo Figura 30 A.

- REG_SET_VALUE_ADDRESS y RETURN_VALUE: Direcciones específicas en la interfaz Wishbone para establecer un valor de registro y para devolver el valor del contador.
- BITS_8: Parámetro que define un tamaño de 8 bits.

Entradas y Salidas del Módulo Figura 30 A.

- Entradas de la Interfaz Wishbone:
 - wb_clk_i: Señal de reloj.
 - wb_rst_i: Señal de reset.
 - wbs_stb_i: Señal de strobe.
 - wbs_cyc_i: Señal de ciclo.
 - wbs_we_i: Señal de escritura.
 - wbs_sel_i: Señal de selección.
 - wbs_dat_i: Datos de entrada (32 bits).
 - wbs_adr_i: Dirección de entrada (32 bits).
- Salidas de la Interfaz Wishbone: Figura 30 A.
 - wbs_ack_o: Señal de reconocimiento.
 - wbs_dat_o: Datos de salida (32 bits).
- Entradas y Salidas del Módulo: Figura 30 A.
 - io_in: Entrada de E/S (definida por MPRJ_IO_PADS y ANALOG_PADS).
 - io_out: Salida de E/S (definida por MPRJ_IO_PADS y ANALOG_PADS).
 - io_oeb: Buffer de salida de E/S (definida por MPRJ_IO_PADS y ANALOG_PADS).
- Señales Internas y Registros Figura 30 B.
 - contador: Registro de 32 bits que actúa como contador.
 - valor_final: Registro de 32 bits para almacenar un valor final.
 - sent: Registro de 2 bits para el estado del envío.
 - toggle: Señal de toggle derivada del bit 7 de io_in.
 - reg_set_value: Registro de 8 bits para almacenar valores establecidos.
- Bloques Principales
 1. Bloque Inicial Figura 30 B:
 - Inicializa io_out y sent a 0.
 2. Bloque Always (Posedge de wb_clk_i) Figura 30 C:

- Resetea reg_set_value y sent si wb_rst_i está activo.
 - Maneja la escritura y lectura de registros basándose en las direcciones REG_SET_VALUE_ADDRESS y RETURN_VALUE.
 - Si wbs_stb_i y wbs_cyc_i están activos y wbs_we_i está en alta, se escribe el valor en reg_set_value. Si el valor es 7, se resetea el registro; si es 8, se establece sent a 1.
 - Si wbs_stb_i y wbs_cyc_i están activos y wbs_we_i está en baja, se lee el valor del contador si sent es 1 y se establece sent a 2.
3. Bloque Always (Posedge de toggle) Figura 30 E:
 - Incrementa el contador si sent es 0.
 - Resetea el contador si sent es 2.
 4. Bloque Always (Posedge de wb_clk_i) Figura 30 F. linea 100:
 - Controla la salida io_out[24] en función del estado de sent.
 5. Bloque Always (Posedge de wb_clk_i) Figura 30 F. linea 111:
 - Maneja la señal de reconocimiento wbs_ack_o.

Resumen del Comportamiento

El módulo *test_mixer* implementa una interfaz de comunicación usando el protocolo Wishbone. A través de esta interfaz, se puede establecer un valor en un registro específico y leer el valor de un contador. El contador se incrementa con un pulso de toggle y se resetea según el estado de envío (sent). La señal de salida *io_out[24]* se usa para indicar el estado del proceso de envío.

5.1.5. Observación en Gtkwave

Tras la configuración previa mencionada, se procede a verificar el funcionamiento ejecutando el programa *iverilog*^[38], el cual está provisto por *Efabless* en el contenedor de *Docker*^[5]. Luego, se ejecuta mediante el comando "*make verify-test_mixer-rtl*" y, posteriormente, se obtiene el archivo *RTL-test_mixer.vcd* el cual se lee en la aplicación *gtkwave*^[25] para su análisis en la Figura 31.



Figura 31 gtkwave (Fuente: Propia)

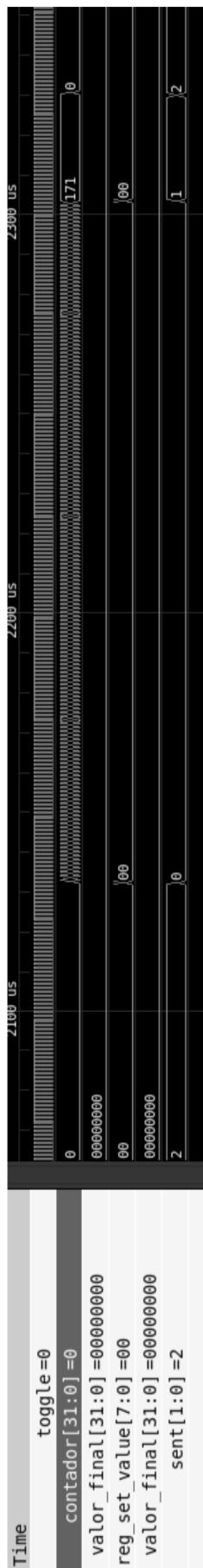


Figura 32 gtkwave ampliado (Fuente: Propia)

En la Figura 32 se puede observar la máquina de estado y los cambios de la misma. Es importante destacar que el tiempo de respuesta está directamente relacionado con la frecuencia del reloj que se esté utilizando.

Este proyecto proporciona una visión de cómo se lleva a cabo la comunicación entre el *user_analog_project_wrapper* y el *picorv32*. Sin embargo, existen otras formas de lograr esta comunicación, como el uso de la *Logic Analyzer Signals*^[8,33].

5.2. Circuitos analógicos

5.2.1. Introducción

En esta sección, se abordará el proceso de desarrollo de circuitos analógicos en microchips, centrándose específicamente en la creación del *Mixer*. Antes de adentrarse en este procedimiento, se describirán las herramientas utilizadas para llevar a cabo el proyecto, tomando como ejemplo la construcción de un *Inverter*.

5.2.2. Inverter

Para realizar un *Inverter*, tras examinar las características del nodo a emplear, se procederá a implementar una aplicación simple. El primer paso consistirá en configurar el entorno de trabajo, comenzando con la instalación de las herramientas *Magic* y *Xschem*^[28].

5.2.2.1. Inverter con Xschem

Para utilizar la herramienta Xschem, es importante tener en cuenta los siguientes comandos principales:

bash

```
cp /usr/local/share/pdk/sky130B/libs.tech/xschem/xschemrc .
xterm &
xschem
```

Al ejecutar estos comandos, se debería observar, al abrir la aplicación, que se está utilizando el nodo correspondiente, tal como se muestra en la Figura 33.

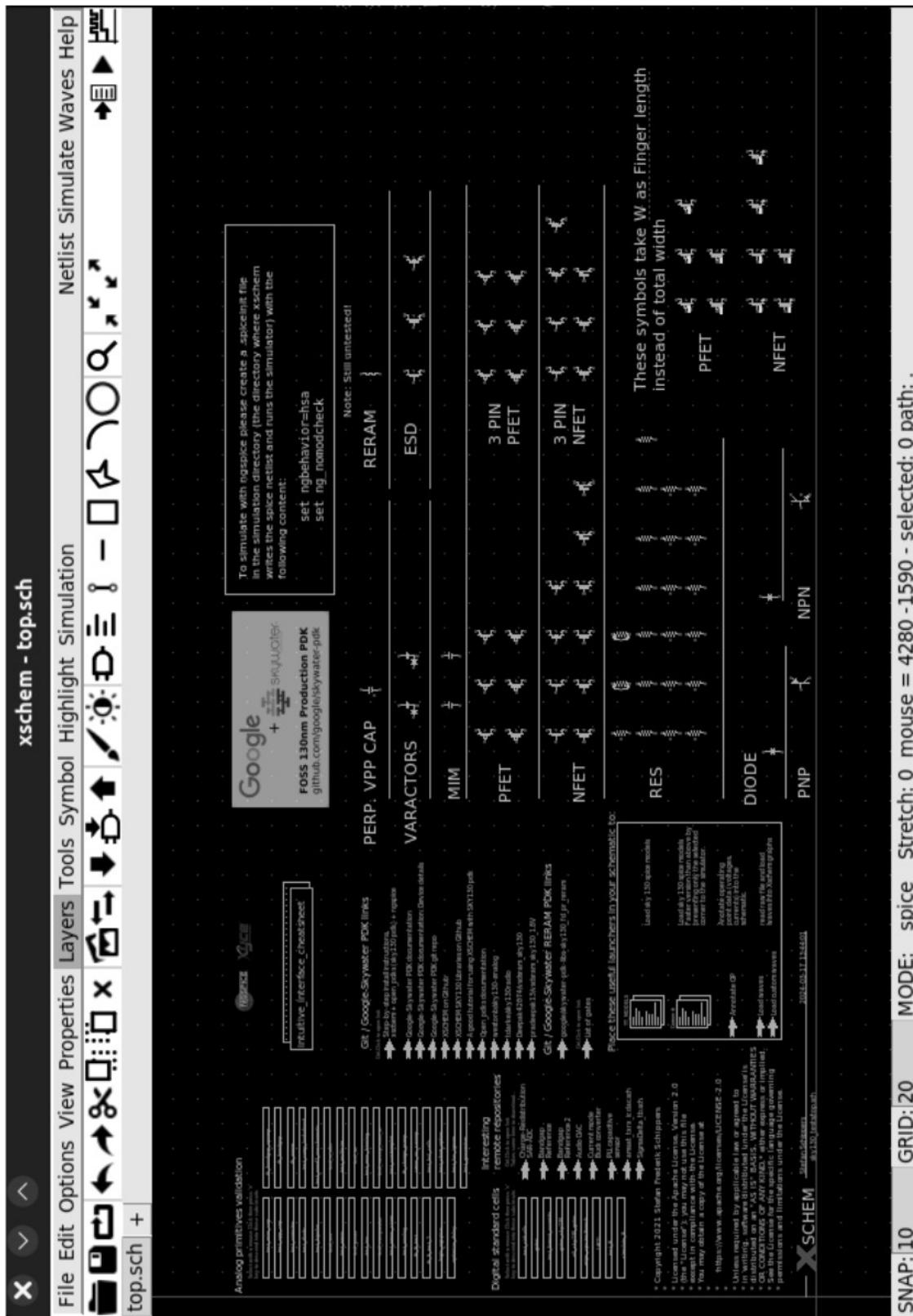


Figura 33 Xschem con nodo Sky130 (Fuente: Propia)

Después, se procede a desarrollar el *Inverter*, detallando los pasos a seguir^[18].

Idealmente, este proceso debería reflejarse de manera similar a lo ilustrado en la Figura 34.

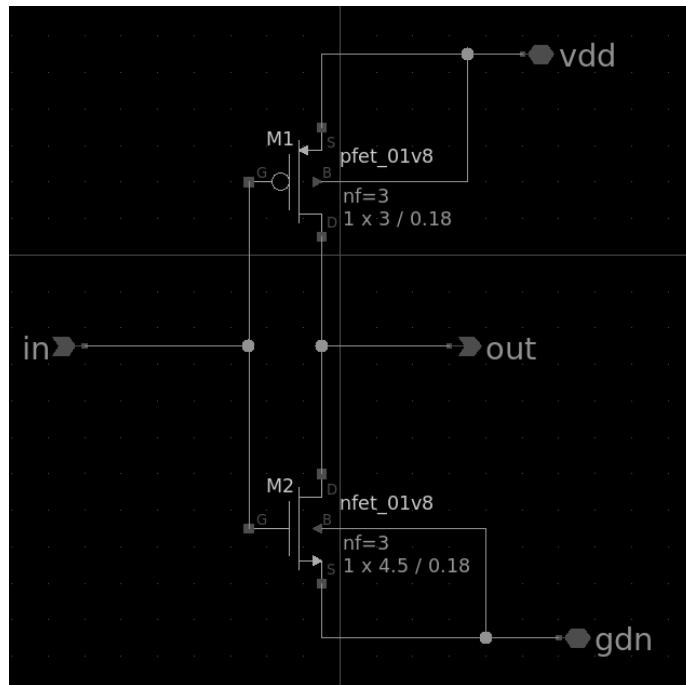


Figura 34 Inverter en Xschem (Fuente: Propia)

Para llevar a cabo este proceso, se procede a exportar el archivo **.spice**, que servirá como indicador de que el desarrollo del *Inverter* se realizó correctamente. Para lograrlo, es fundamental asegurarse de tener activadas las siguientes opciones en la pestaña de *Simulation*, Figura 35:

"Show netlist after netlist command"

"LVS netlist: Top level is a .subckt"

Una vez configuradas estas opciones, se procede a ejecutar el *Netlist*. En la Figura 35 se observa la salida **untitled.spice**, la cual se procede a guardar (*save*).

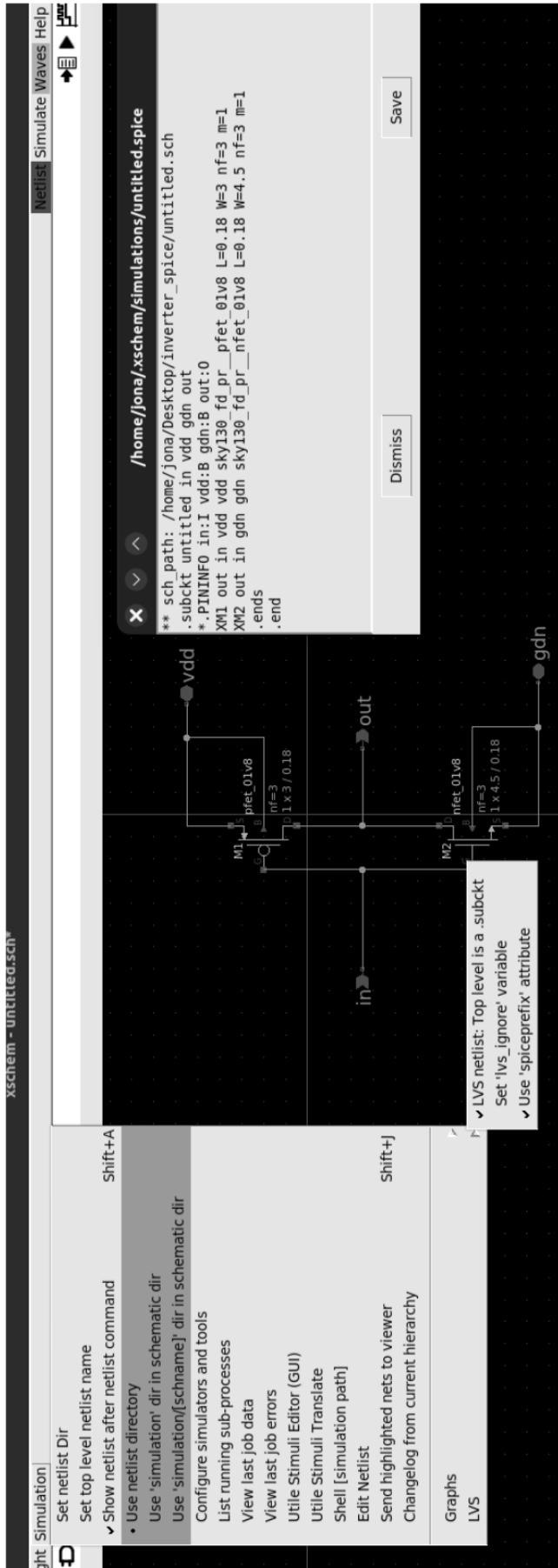


Figura 35 Netlist (Fuente: Propia)

5.2.2.2. Inverter con Magic

A continuación, se destacan los siguientes comandos como los más relevantes a utilizar:

bash

```
export PDK_ROOT=/usr/local/share/pdk/  
magic -d XR -rcfile $PDK_ROOT/sky130A/libs.tech/magic/sky130A.magicrc
```

Una vez ejecutados en una terminal, se observa la ventana de Magic como se muestra en la Figura 36, en la cual se tiene la tecnología cargada (Technology = sky130A) en el cuadro de color rojo y la terminal de Magic, donde se podrán ingresar los comandos propios de Magic y también observar los errores en caso de que ocurran.

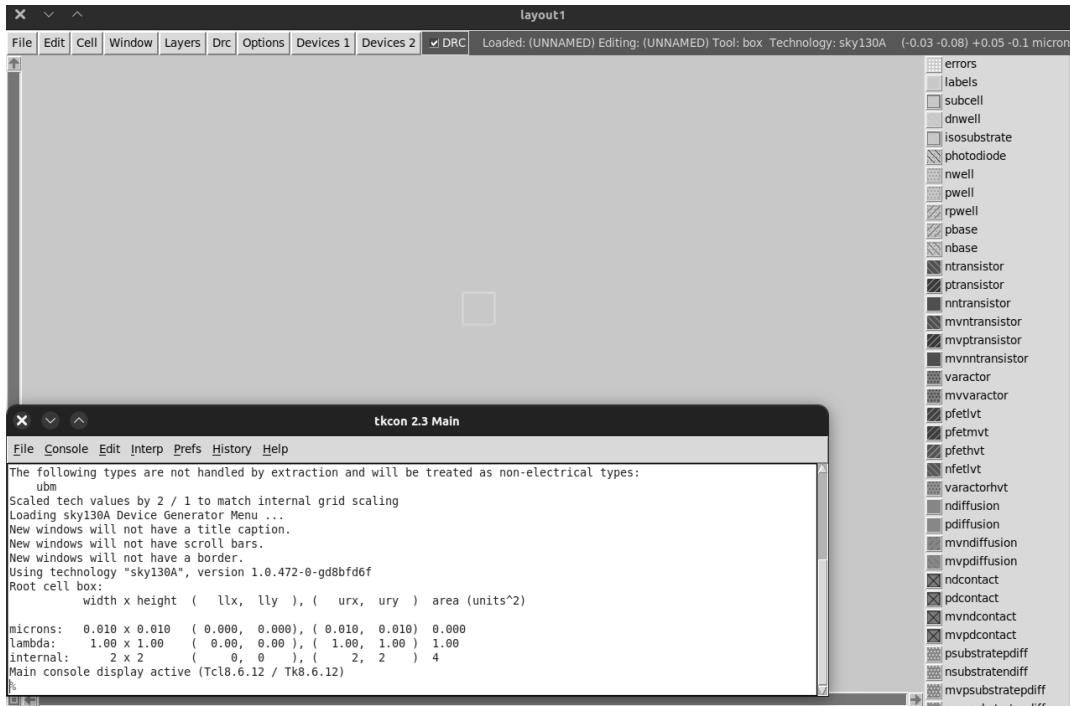


Figura 36 Magic con nodo Sky130A (Fuente: Propia)

Después de haber obtenido el archivo *.spice* con *Xschem*, se lo importa en *Magic* a través de la opción "*File -> Export SPICE*". Además, se realizaron las conexiones manualmente, quedando el diseño final como se observa en la Figura 37.

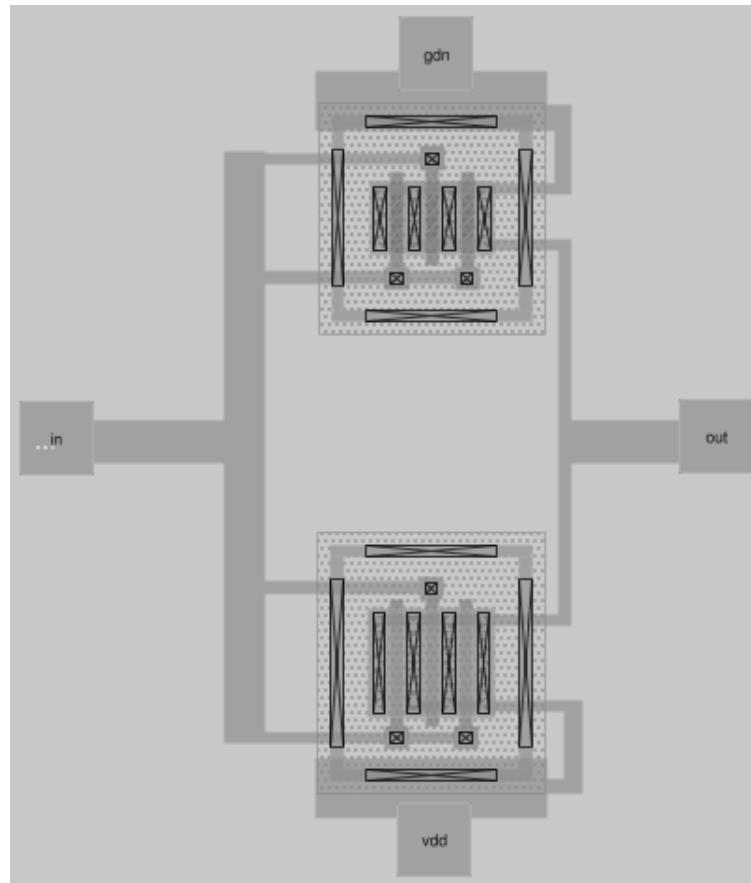


Figura 37 Inverter en *Magic* (Fuente: Propia)

5.2.2.2.1. Extracción de archivos *.ext .spice .mag* del Inverter

Para avanzar a la siguiente etapa de este ejemplo, luego de completar el diseño en *Magic*, es necesario ejecutar los siguientes comandos en la terminal del mismo:

extract do local:

Este comando indica a Magic que realice la extracción de todos los dispositivos y conexiones que están presentes en el diseño y que son locales alrededor del punto de inserción de la celda.

`extract all:`

Este comando le dice a Magic que realice una extracción completa de todos los dispositivos y conexiones en el diseño.

`ext2spice lvs:`

Este comando convierte el archivo de extracción de Magic a un formato que pueda ser utilizado por programas de simulación de circuitos (como SPICE) durante la verificación LVS.

`ext2spice:`

Similar al anterior, este comando convierte el archivo de extracción de Magic a un formato SPICE, pero sin especificar la verificación LVS.

Estos comandos son esenciales para preparar el diseño para la verificación LVS, asegurando que el diseño físico sea consistente con el diseño lógico y funcione correctamente.

5.2.2.2.2. Verificación DRC

La verificación DRC (*Design Rule Check*) es crucial para garantizar que el diseño del circuito pueda fabricarse de acuerdo con las especificaciones del fabricante de la tecnología. Esto implica verificar aspectos como la distancia entre capas de metal, el ancho y área adecuados, entre otras reglas, que aseguren la viabilidad de la fabricación.

Estas normas deben considerarse durante el desarrollo del *layout* y se detallan en la documentación de la tecnología^[24]. La herramienta DRC permite realizar esta verificación de forma automática^[51]. Una vez que se cumplen las reglas de diseño, se procede a la verificación LVS, que garantiza la equivalencia eléctrica entre el circuito del *layout* y el esquemático^[52].

No es necesario completar el diseño antes de ejecutar una verificación DRC; estas comprobaciones se realizan de manera iterativa para garantizar que se cumplan todas las reglas a medida que se diseña. Del mismo modo, incluso si la verificación LVS no es válida, se puede verificar el circuito paso a paso. Por lo tanto, estas comprobaciones se realizan de forma iterativa.

La verificación DRC no sólo verifica la capacidad del chip de ser fabricado en términos de geometría, sino que también tiene en cuenta la comprobación de antenas. Esta verificación asegura que se tomen medidas para prevenir el llamado "efecto antena"^[17].

El efecto antena (formalmente conocido como daño del óxido de puerta inducido por plasma) ocurre cuando una pista de metal está muy cerca del polisilicio de la puerta de un transistor. Durante el proceso de fabricación, la pista puede cargarse con una tensión más alta de lo normal, lo

que puede dañar el transistor. Para prevenir este efecto, se implementan estrategias de protección como, por ejemplo, introducir saltos entre capas de metal a través de vías para una misma pista. Estas medidas aseguran que, durante el proceso de fabricación, no se produzcan diferencias de potencial que puedan dañar el transistor.

Al correr en la terminal de *Magic* el comando “*drc fin*”, se debe observar una salida como la mostrada en la Figura 38.

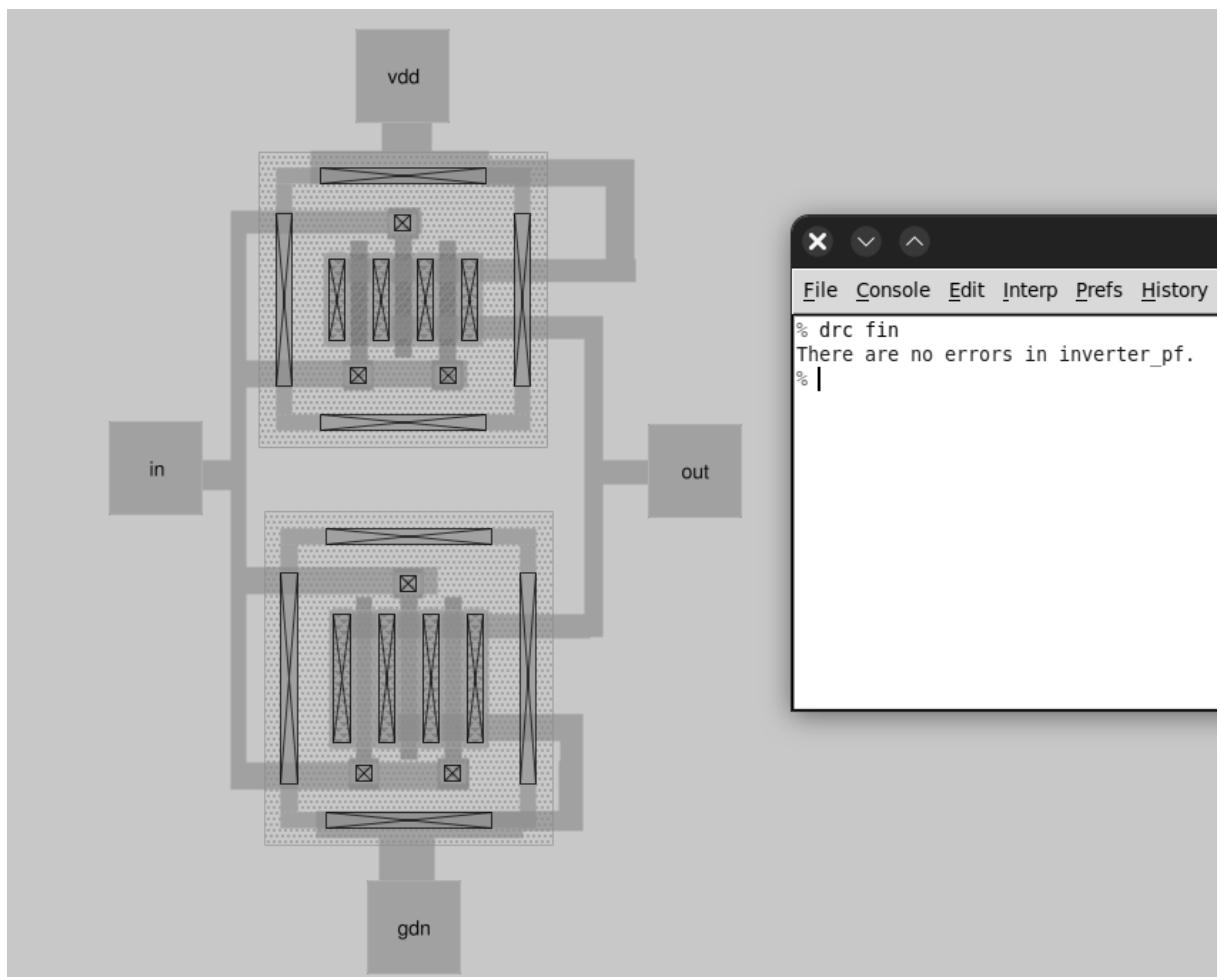


Figura 38 Salida del comando ‘drc fin’ (Fuente: Propia)

5.2.2.2.3. Análisis Post Layout - extracción de capacidades parásitas

Los comandos para la terminal de *Magic* son:

extract all:

Este comando indica a *Magic* que realice la extracción de todos los dispositivos y conexiones presentes en el diseño.

ext2spice hierarchy on:

Este comando habilita la opción de mantener la jerarquía en el *netlist* generado.

ext2spice scale off:

Este comando desactiva el escalado automático del *netlist* generado.

ext2spice cthresh 0:

Este comando establece el umbral de capacitancia a cero, lo que significa que se incluirán todas las capacidades en el *netlist* generado.

ext2spice -d -o postlayout.spice -f ngspice:

Este comando realiza la conversión del *netlist* extraído al formato Spice, específicamente en formato compatible con Ngspice, una herramienta de simulación de circuitos. El parámetro *-d* indica que se desea una salida detallada, *-o postlayout.spice* especifica el nombre del archivo de salida, y *-f ngspice* indica el formato de salida para Ngspice.

El archivo *postlayout.spice* (el cual se encuentra en el mismo directorio del *Inverter*) generado a partir del diseño del *Inverter* en la tecnología *sky130A* proporciona una descripción detallada de la estructura eléctrica del circuito. Este archivo contiene la información necesaria para simular el comportamiento del *Inverter* en un entorno Spice. Cada línea del archivo representa un componente del circuito, como transistores y capacitancias parásitas, junto con las conexiones entre ellos.

Las líneas que comienzan con **C** describen las capacitancias parásitas entre los nodos del circuito, las cuales se generan durante el proceso de diseño y son cruciales para comprender el comportamiento eléctrico del circuito.

Al analizar el archivo *postlayout.spice*, se puede obtener una comprensión más profunda de cómo interactúan los componentes del *Inverter* y cómo afectan su rendimiento eléctrico. Una celda en el archivo *postlayout.spice* se puede observar como en la Figura 39.

```

.subckt inverter in gnd Vdd out
XXM2 in gnd out gnd in in gnd out sky130_fd_pr_nfet_01v8_VME5ZM
Xsky130_fd_pr_pfet_01v8_MTKDLE_0 Vdd out in out in Vdd Vdd in gnd
sky130_fd_pr_pfet_01v8_MTKDLE
C0 out Vdd 0.249855f
C1 gnd Vdd 0.028178f
C2 in out 0.328467f
C3 in gnd 0.395959f
C4 gnd out 0.197259f
C5 in Vdd 0.346742f
C6 Vdd 0 2.277895f
C7 gnd 0 0.731533f
C8 out 0 0.883589f
C9 in 0 1.790535f
.ends

```

Figura 39 Cell postlayout.spice (Fuente: Propia)

5.2.2.3. Análisis LVS con Netgen

Netgen es una herramienta utilizada para comparar *Netlists*, un proceso conocido como *LVS*, por sus siglas en inglés que significa "Layout vs. Schematic" (Diseño vs. Esquemático). Este paso es crucial en el flujo de diseño de circuitos integrados, ya que garantiza que la geometría del diseño coincida con el circuito esperado.

Circuitos muy pequeños pueden omitir este paso al confirmar el funcionamiento a través de extracción y simulación. Por otro lado, los circuitos digitales muy grandes suelen ser generados por herramientas a partir de descripciones de alto nivel, utilizando compiladores que aseguran la geometría de diseño correcta. La necesidad más notable de *LVS* se encuentra en circuitos analógicos grandes o de señal mixta que no se pueden simular en un tiempo razonable. Incluso para circuitos pequeños, *LVS* se puede realizar mucho más rápido que la simulación y proporciona retroalimentación que facilita encontrar errores, en comparación con la misma. En la bibliografía se encuentran instrucciones acerca de cómo instalar y configurar la herramienta para llevar a cabo el proyecto^[4].

En resumen, se crea un directorio dedicado para *Netgen*, donde se almacena un archivo *.tcl* con las configuraciones necesarias para la verificación.

El comando utilizado para correr la verificación es:

```
ln -s /usr/local/share/pdk/sky130A/libs.tech/netgen/sky130A_setup.tcl setup.tcl
```

5.2.2.3.1. Layout vs. Schematic del Inverter

Para iniciar la verificación *LVS* del *Inverter* utilizando la herramienta *Magic*, primero se extraen los archivos necesarios con los siguientes comandos:

```
extract do local  
extract all  
ext2spice lvs  
ext2spice
```

Finalmente, para llevar a cabo la verificación *LVS* del *Inverter*, se puede emplear el siguiente comando en la terminal, estando situado en el directorio de Netgen configurado previamente:

```
netgen -batch lvs "../mag/inverter_MAG.spice inverter" "../xschem/inverter_Xschem.spice inverter"
```

Este comando activará el proceso de verificación *LVS* utilizando los archivos *SPICE* del *Inverter* obtenidos de las herramientas *Magic* y *Xschem*, respectivamente. Es importante aquí asegurarse de que los nombres de los archivos y las ubicaciones sean correctos de acuerdo con la configuración establecida.

Una salida exitosa se verá como la siguiente Figura 40:

*Circuit was modified by parallel/series device merging.
New circuit summary:*

Contents of circuit 1: Circuit: 'inverter'

Circuit inverter contains 2 device instances.

Class: sky130_fd_pr_nfet_01v8 instances: 1

Class: sky130_fd_pr_pfet_01v8 instances: 1

Circuit contains 4 nets.

Contents of circuit 2: Circuit: 'inverter'

Circuit inverter contains 2 device instances.

Class: sky130_fd_pr_nfet_01v8 instances: 1

Class: sky130_fd_pr_pfet_01v8 instances: 1

Circuit contains 4 nets.

Circuit 1 contains 2 devices, Circuit 2 contains 2 devices.

Circuit 1 contains 4 nets, Circuit 2 contains 4 nets.

Final result:

Circuits match uniquely.

Logging to file "comp.out" disabled

LVS Done.

Figura 40 Verificación LVS Inverter (Fuente: Propia)

5.2.3. Mixer

Para llevar a cabo el *Mixer* se inicia con los siguientes directorios, tal y como se observa en la Figura 41:

```
Proyecto_Mixer
├── mag
│   └── Mixer.mag
│       └── Mixer.spice
├── netgen
│   └── setup.tcl -> /usr/local/share/pdk/sky130A/libs.tech/netgen/sky130A_setup.tcl
└── xschem
    ├── Mixer.sch
    └── Mixer.spice
        └── xschemrc
```

Figura 41 Directorios Mixer (Fuente: Propia)

5.2.3.1. Mixer con Xschem

En esta etapa, se opta por un resumen de los pasos previos, considerando la exhaustiva explicación del *Inverter*. Ahora se avanza hacia la presentación detallada del desarrollo del *Mixer* en *Xschem*, ofreciendo una visión más amplia y completa del proceso, como en la Figura 42.

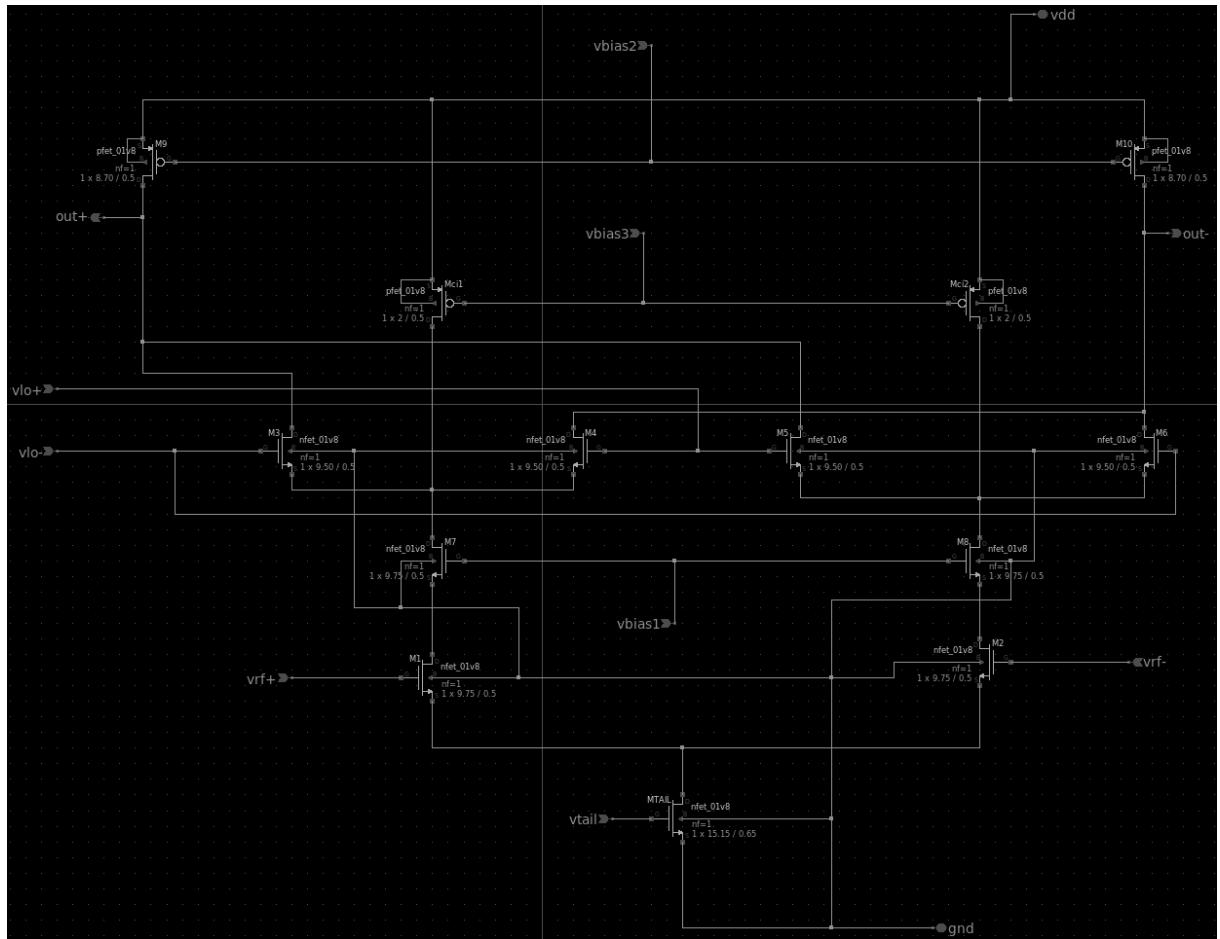


Figura 42 Mixer con Xschem (Fuente: Propia)

5.2.3.2. Mixer con Magic

Es crucial obtener de manera análoga al procedimiento seguido para el *Inverter*, el archivo *Mixer.spice* desde *Xschem*, con el fin de abrirlo luego desde la herramienta *Magic*. Este paso resulta fundamental para mantener la coherencia en el proceso de diseño y garantizar una transición fluida entre las distintas etapas del proyecto. Recordando que archivo .spice es un tipo de archivo de texto que contiene una descripción de un circuito electrónico en un formato que puede ser interpretado por simuladores SPICE (Simulation Program with Integrated Circuit Emphasis). Estos simuladores se utilizan para analizar y prever el comportamiento de circuitos electrónicos. Importar archivos SPICE en Magic es un paso crucial en el flujo de diseño de circuitos integrados que permite verificar, simular y optimizar el diseño antes de la fabricación, asegurando así que el

producto final funcione según las especificaciones, en este caso se importa el archivo .spice del mixer y en la Figura 43 se observan las distintas celdas, las cuales ya fueron creadas y son de uso público.

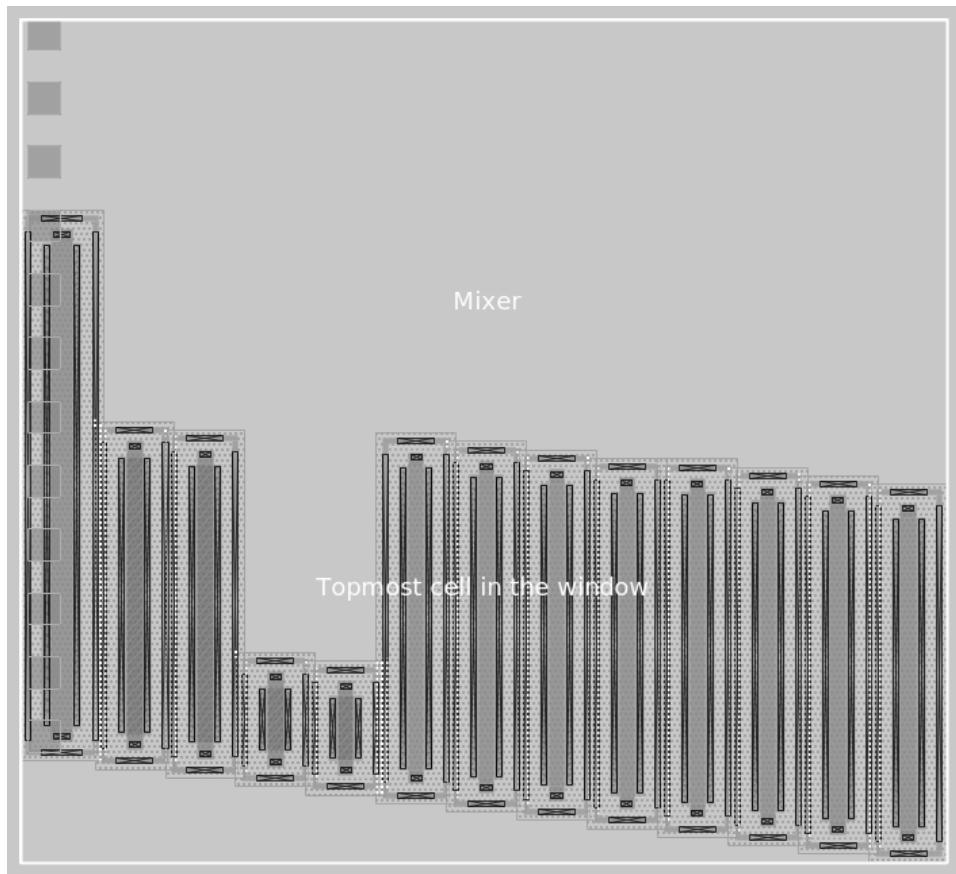


Figura 43 Mixer.spice importado en Magic (Fuente: Propia)

Después de realizar las conexiones manualmente, resulta una configuración como la que se observa en la Figura 44.

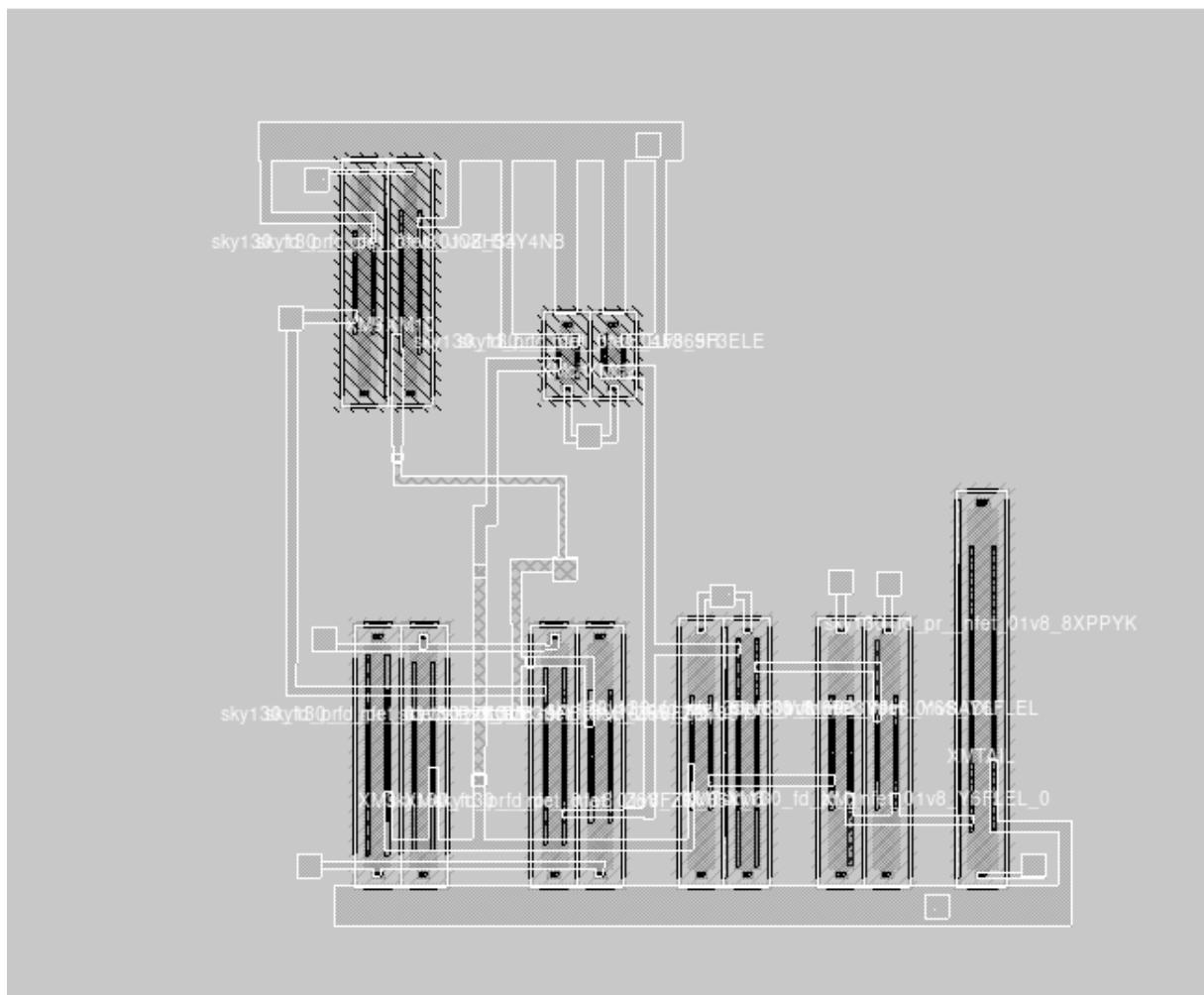


Figura 44 Observación de conexiones con Magic (Fuente: Propia)

En la Figura 45 se observan las conexiones finales entre las diferentes celdas o dispositivos y sus respectivos puertos.

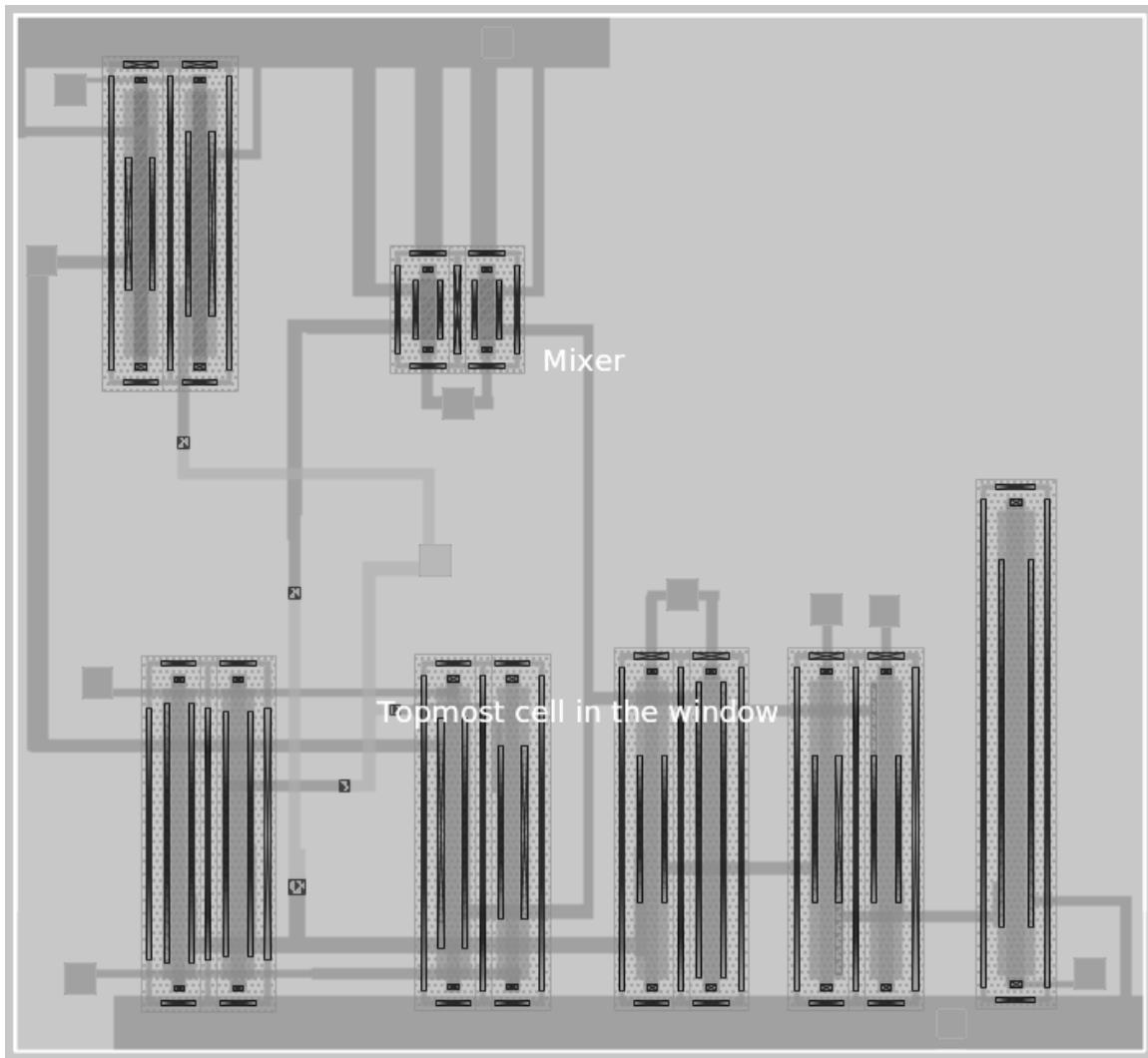


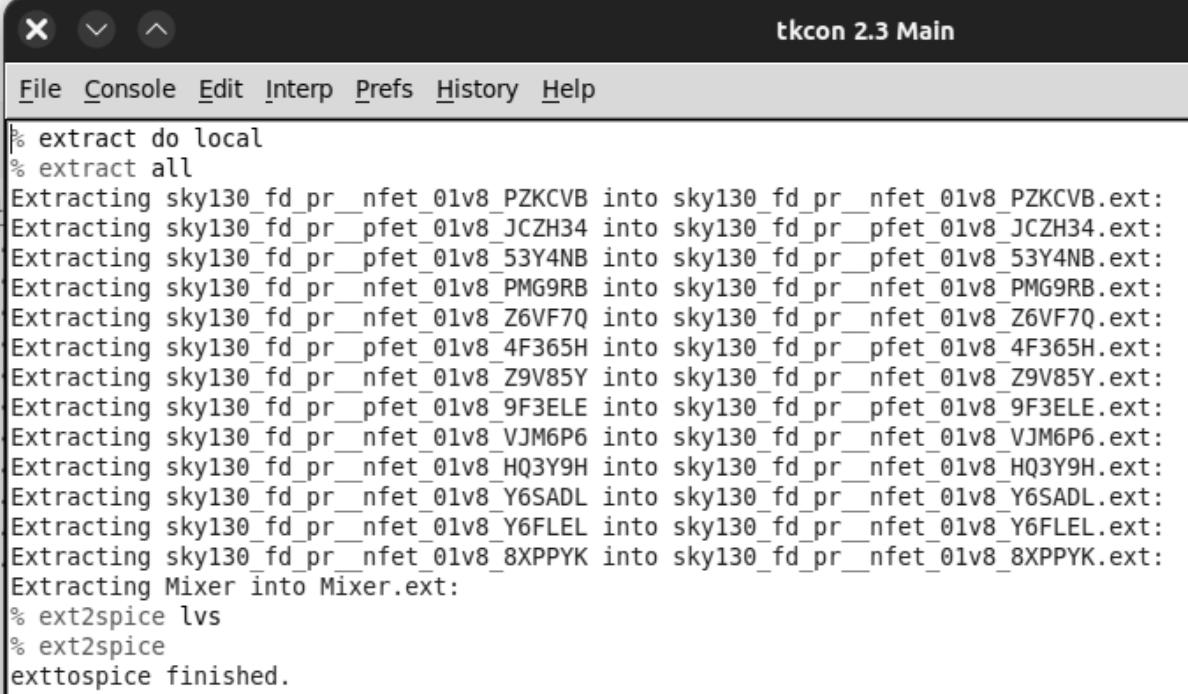
Figura 45 Fin de conexiones con Magic (Fuente: Propia)

5.2.3.2.1. Extracción de archivos .ext .spice .mag del Mixer

Al igual que con el *Inverter*, se procede a la extracción de archivos, utilizando los mismos comandos.

```
extract do local  
extract all  
ext2spice lvs  
ext2spice
```

En caso de éxito en la extracción, la terminal de *Magic* debería mostrar una salida similar a la que se observa en la Figura 46.



```
% extract do local
% extract all
Extracting sky130_fd_pr_nfet_01v8_PZKCVB into sky130_fd_pr_nfet_01v8_PZKCVB.ext:
Extracting sky130_fd_pr_pfet_01v8_JCZH34 into sky130_fd_pr_pfet_01v8_JCZH34.ext:
Extracting sky130_fd_pr_pfet_01v8_53Y4NB into sky130_fd_pr_pfet_01v8_53Y4NB.ext:
Extracting sky130_fd_pr_nfet_01v8_PMG9RB into sky130_fd_pr_nfet_01v8_PMG9RB.ext:
Extracting sky130_fd_pr_nfet_01v8_Z6VF7Q into sky130_fd_pr_nfet_01v8_Z6VF7Q.ext:
Extracting sky130_fd_pr_pfet_01v8_4F365H into sky130_fd_pr_pfet_01v8_4F365H.ext:
Extracting sky130_fd_pr_nfet_01v8_Z9V85Y into sky130_fd_pr_nfet_01v8_Z9V85Y.ext:
Extracting sky130_fd_pr_pfet_01v8_9F3ELE into sky130_fd_pr_pfet_01v8_9F3ELE.ext:
Extracting sky130_fd_pr_nfet_01v8_VJM6P6 into sky130_fd_pr_nfet_01v8_VJM6P6.ext:
Extracting sky130_fd_pr_nfet_01v8_HQ3Y9H into sky130_fd_pr_nfet_01v8_HQ3Y9H.ext:
Extracting sky130_fd_pr_nfet_01v8_Y6SADL into sky130_fd_pr_nfet_01v8_Y6SADL.ext:
Extracting sky130_fd_pr_nfet_01v8_Y6FLEL into sky130_fd_pr_nfet_01v8_Y6FLEL.ext:
Extracting sky130_fd_pr_nfet_01v8_8XPPYK into sky130_fd_pr_nfet_01v8_8XPPYK.ext:
Extracting Mixer into Mixer.ext:
% ext2spice lvs
% ext2spice
exttospice finished.
```

Figura 46 Salida de la consola de *Magic* luego de los comandos de extracción (Fuente: Propia)

5.2.3.2.2. Verificación DRC

```

tkcon 2.3 Main
File Console Edit Interp Prefs History Help
Extracting sky130_fd_pr_nfet_01v8_Z6VF7Q into sky130_fd_pr_nfet_01v8_Z6VF7Q.ext:
Extracting sky130_fd_pr_pfet_01v8_4F365H into sky130_fd_pr_pfet_01v8_4F365H.ext:
Extracting sky130_fd_pr_nfet_01v8_Z9V85Y into sky130_fd_pr_nfet_01v8_Z9V85Y.ext:
Extracting sky130_fd_pr_pfet_01v8_9F3ELE into sky130_fd_pr_pfet_01v8_9F3ELE.ext:
Extracting sky130_fd_pr_nfet_01v8_VJM6P6 into sky130_fd_pr_nfet_01v8_VJM6P6.ext:
Extracting sky130_fd_pr_nfet_01v8_HQ3Y9H into sky130_fd_pr_nfet_01v8_HQ3Y9H.ext:
Extracting sky130_fd_pr_nfet_01v8_Y6SADL into sky130_fd_pr_nfet_01v8_Y6SADL.ext:
Extracting sky130_fd_pr_nfet_01v8_Y6FLEL into sky130_fd_pr_nfet_01v8_Y6FLEL.ext:
Extracting sky130_fd_pr_nfet_01v8_8XPPYK into sky130_fd_pr_nfet_01v8_8XPPYK.ext:
Extracting Mixer into Mixer.ext:
% ext2spice lvs
% ext2spice
exttospice finished.
% drc fin
There are no errors in Mixer.
%

```

Figura 47 Salida de la consola de Magic luego del comando drc (Fuente: Propia)

En la Figura 47 se observa que el *Mixer* presenta las dimensiones correctas para operar con la tecnología asociada a este proyecto, específicamente Skywater 130nm. Cabe mencionar que tanto el diseño analógico como el digital se llevan a cabo utilizando esta tecnología. En el caso del diseño digital, esta verificación se realiza de manera automática mediante OpenLane.

5.2.3.2.3. Análisis Post Layout extracción de capacidades parásitas

A continuación, repitiendo los pasos realizados para la creación *Inverter* y utilizando los mismos comandos, se tiene:

```

extract all
ext2spice hierarchy on
ext2spice scale off
ext2spice cthresh 0
ext2spice -d -o postlayout.spice -f ngspice

```



The screenshot shows a terminal window titled "tkcon 2.3 Main". The menu bar includes "File", "Console", "Edit", "Interp", "Prefs", "History", and "Help". The main text area displays the following command-line session:

```
exttospice finished.  
% drc fin  
There are no errors in Mixer.  
% extract all  
Extracting sky130_fd_pr_nfet_01v8_PZKCVB into sky130_fd_pr_nfet_01v8_PZKCVB.ext:  
Extracting sky130_fd_pr_pfet_01v8_JCZH34 into sky130_fd_pr_pfet_01v8_JCZH34.ext:  
Extracting sky130_fd_pr_pfet_01v8_53Y4NB into sky130_fd_pr_pfet_01v8_53Y4NB.ext:  
Extracting sky130_fd_pr_nfet_01v8_PMG9RB into sky130_fd_pr_nfet_01v8_PMG9RB.ext:  
Extracting sky130_fd_pr_nfet_01v8_Z6VF7Q into sky130_fd_pr_nfet_01v8_Z6VF7Q.ext:  
Extracting sky130_fd_pr_pfet_01v8_4F365H into sky130_fd_pr_pfet_01v8_4F365H.ext:  
Extracting sky130_fd_pr_nfet_01v8_Z9V85Y into sky130_fd_pr_nfet_01v8_Z9V85Y.ext:  
Extracting sky130_fd_pr_pfet_01v8_9F3ELE into sky130_fd_pr_pfet_01v8_9F3ELE.ext:  
Extracting sky130_fd_pr_nfet_01v8_VJM6P6 into sky130_fd_pr_nfet_01v8_VJM6P6.ext:  
Extracting sky130_fd_pr_nfet_01v8_HQ3Y9H into sky130_fd_pr_nfet_01v8_HQ3Y9H.ext:  
Extracting sky130_fd_pr_nfet_01v8_Y6SADL into sky130_fd_pr_nfet_01v8_Y6SADL.ext:  
Extracting sky130_fd_pr_nfet_01v8_Y6FEL into sky130_fd_pr_nfet_01v8_Y6FEL.ext:  
Extracting sky130_fd_pr_nfet_01v8_8XPPYK into sky130_fd_pr_nfet_01v8_8XPPYK.ext:  
Extracting Mixer into Mixer.ext:  
% ext2spice hierarchy on  
% ext2spice scale off  
% ext2spice cthresh 0  
% ext2spice -d -o postlayout.spice -f ngspice  
exttospice finished.  
%
```

Figura 48 Fin de extracciones con Magic (Fuente: Propia)

En el fragmento proporcionado a continuación Figura 49 se muestra la primera celda del diseño Post Layout, mientras que el Post Layout completo se detalla en el Anexo B:

```

* NGSPICE file created from Mixer.ext - technology: sky130A

.subckt sky130_fd_pr_nfet_01v8 Y6SADL a_n210_n1149# a_n50_n1063# a_50_n975# a_n108_n975#
X0 a_50_n975# a_n50_n1063# a_n108_n975# a_n210_n1149# sky130_fd_pr_nfet_01v8 ad=2.8275
pd=20.08 as=2.8275 ps=20.08 w=9.75 l=0.5
**devattr s=113100,4016 d=113100,4016
C0 a_50_n975# a_n50_n1063# 0.062988f
C1 a_n108_n975# a_n50_n1063# 0.064897f
C2 a_50_n975# a_n108_n975# 0.336849f
C3 a_50_n975# a_n210_n1149# 0.819905f
C4 a_n108_n975# a_n210_n1149# 0.611795f
C5 a_n50_n1063# a_n210_n1149# 0.48776f
.ends

Logging to file "comp.out" disabled
LVS Done.

```

Figura 49 Fragmento Post Layout (Fuente: Propia)

En la Figura 49 se presenta el archivo postlayout.spice, el cual registra las capacitancias parásitas. Este archivo es parte fundamental del proceso de fabricación del chip, ya que se busca minimizar dichas capacitancias.

5.2.3.2.4. Layout vs. Schematic del Mixer

Desde el directorio de *netgen* y se ejecuta el siguiente comando:

```
netgen -batch lvs "../mag/Mixer.spice Mixer" "../xschem/Mixer.spice Mixer"
```

La siguiente Figura 50 es una síntesis de la salida de consola del proceso de verificación *LVS*, mientras que la salida completa se encuentra detallada en el Anexo C:

```

Contents of circuit 1: Circuit: 'Mixer'
Circuit Mixer contains 13 device instances.
Class: sky130_fd_pr_nfet_01v8 instances: 9
Class: sky130_fd_pr_pfet_01v8 instances: 4
Circuit contains 17 nets.
Contents of circuit 2: Circuit: 'Mixer'
Circuit Mixer contains 13 device instances.
Class: sky130_fd_pr_nfet_01v8 instances: 9
Class: sky130_fd_pr_pfet_01v8 instances: 4
Circuit contains 17 nets.

Circuit 1 contains 13 devices, Circuit 2 contains 13 devices.
Circuit 1 contains 17 nets, Circuit 2 contains 17 nets.

Final result:
Circuits match uniquely.

.

Logging to file "comp.out" disabled
LVS Done.

```

Figura 50 Verificación LVS (Fuente: Propia)

La salida del comando indicó que ambos archivos, tanto el esquemático como el layout, fueron leídos correctamente, por eso se lee la línea “*Circuits match uniquely.*”. A pesar de algunas advertencias sobre subcircuitos no definidos, se crearon definiciones de celdas de marcador de posición para estos subcircuitos.

Posteriormente, se realizó la comparación entre los circuitos *Mixer* del esquemático y del layout. El informe de comparación reveló que ambos circuitos contenían la misma cantidad de dispositivos y redes, lo que indica una coincidencia en el diseño.

El resultado final de la verificación *LVS* mostró que los circuitos coincidían de manera única, lo que sugiere que no había discrepancias significativas entre el diseño del esquemático y el diseño del layout para el circuito *Mixer*.

Por lo tanto, se concluye que el diseño del circuito *Mixer* en el esquemático y el layout son consistentes y concuerdan entre sí^[11].

5.2.4. Schmitt Trigger

En esta sección se implementa un circuito *Schmitt Trigger*^[41] que permite contar la frecuencia de una señal sinusoidal Figura 51. En este proyecto, el mezclador (*Mixer*) toma dos señales y produce una señal de salida que contiene componentes de las frecuencias combinadas (suma y diferencia de las frecuencias de entrada). Luego, esta señal mezclada pasa por un demodulador que limpia la señal, eliminando componentes no deseados. La señal limpia resultante entra de nuevo al chip a través de los pines del *Schmitt Trigger*, el cual genera un pulso cuando el voltaje alcanza un cierto nivel y ese pulso ingresa a un contador de pulsos digital, luego al procesador *PicoRV32* el cual, finalmente, envía la cuenta por *UART* fuera del chip.

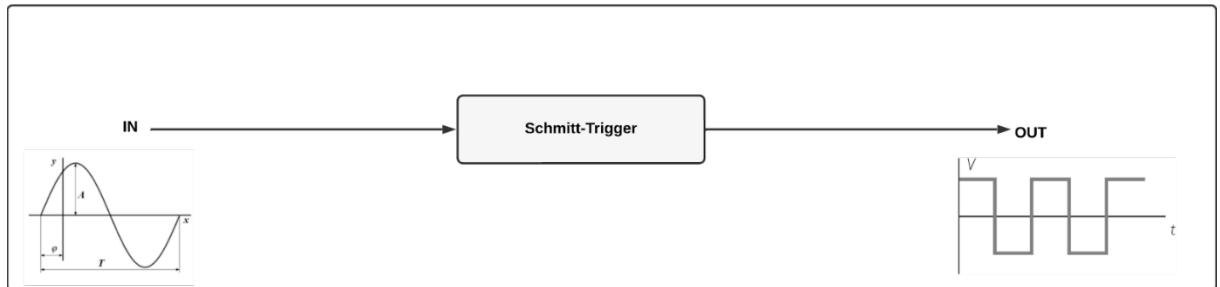


Figura 51 Descripción funcionalidad del Schmitt Trigger (Fuente: Propia)

5.2.4.1. Schmitt Trigger con Xschem

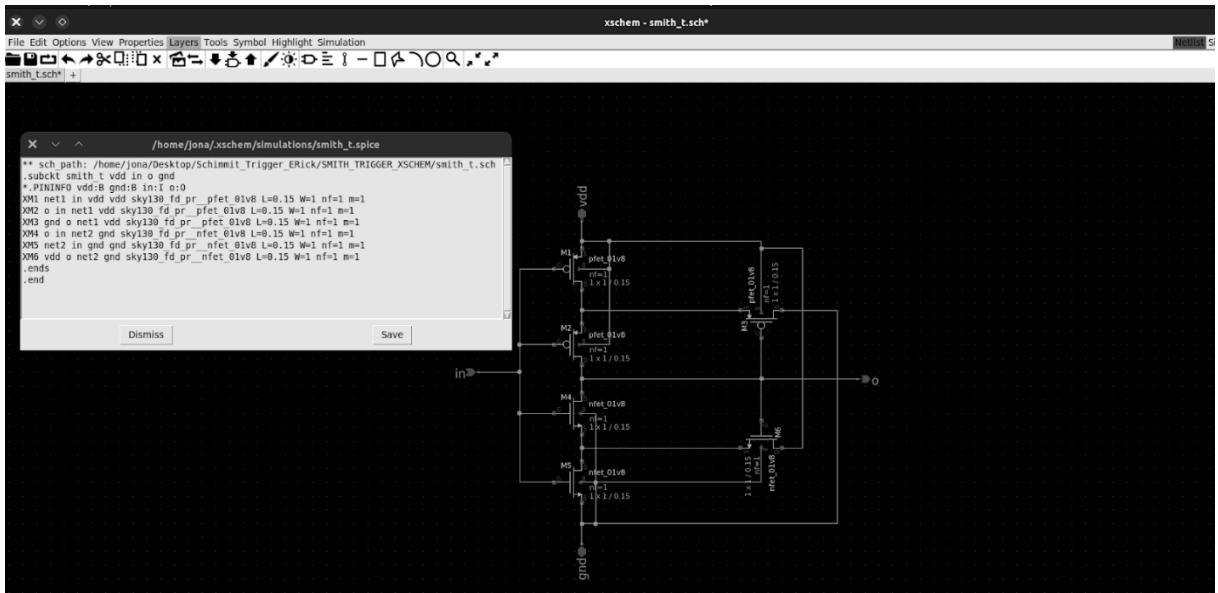


Figura 52 Schmitt Trigger en Xschem (Fuente: Propia)

En la Figura 52 se muestra el desarrollo del Schmitt Trigger en Xschem como parte del flujo de trabajo para el desarrollo de dispositivos analógicos. La extracción del netlist se realiza correctamente.

5.2.4.2. Schmitt Trigger con LTSPICE

En este caso, se utiliza el programa LTSpice, como se muestra en la Figura 53 para simular la salida del Schmitt trigger. En la Figura 54, se puede observar que la señal sinusoidal es la fuente de la cual se obtendrán los pulsos. La señal cuadrada es el resultado buscado.

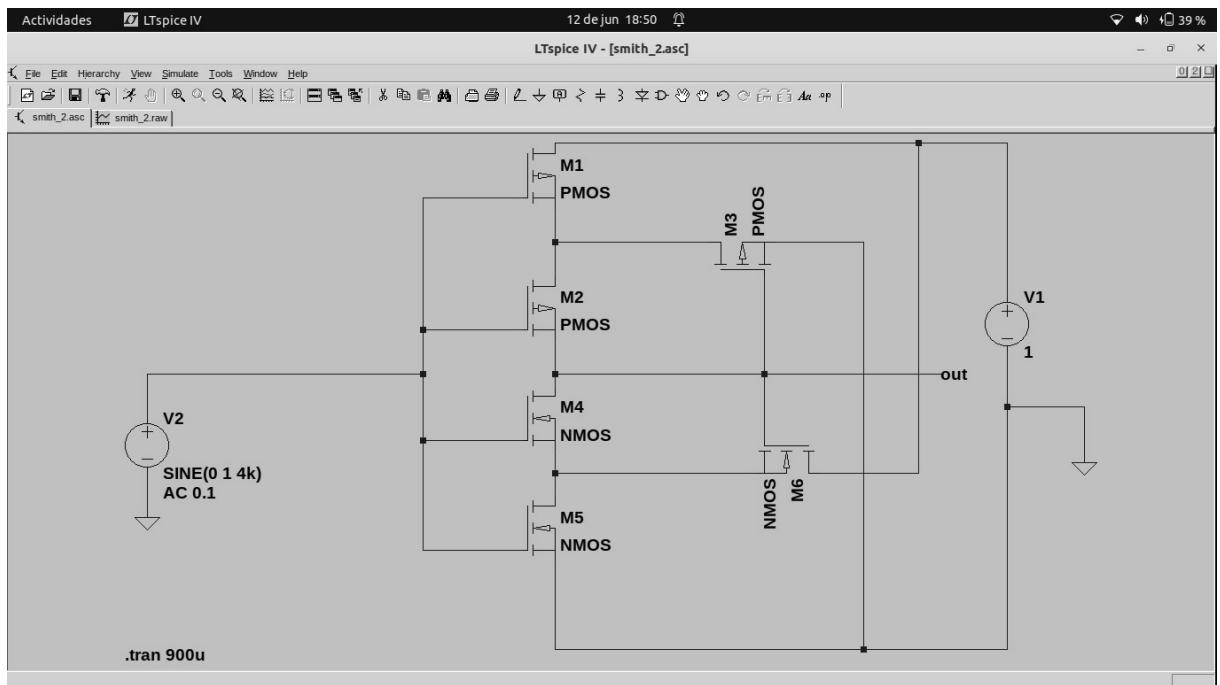


Figura 53 Schmitt Trigger en LTSPICE (Fuente: Propia)

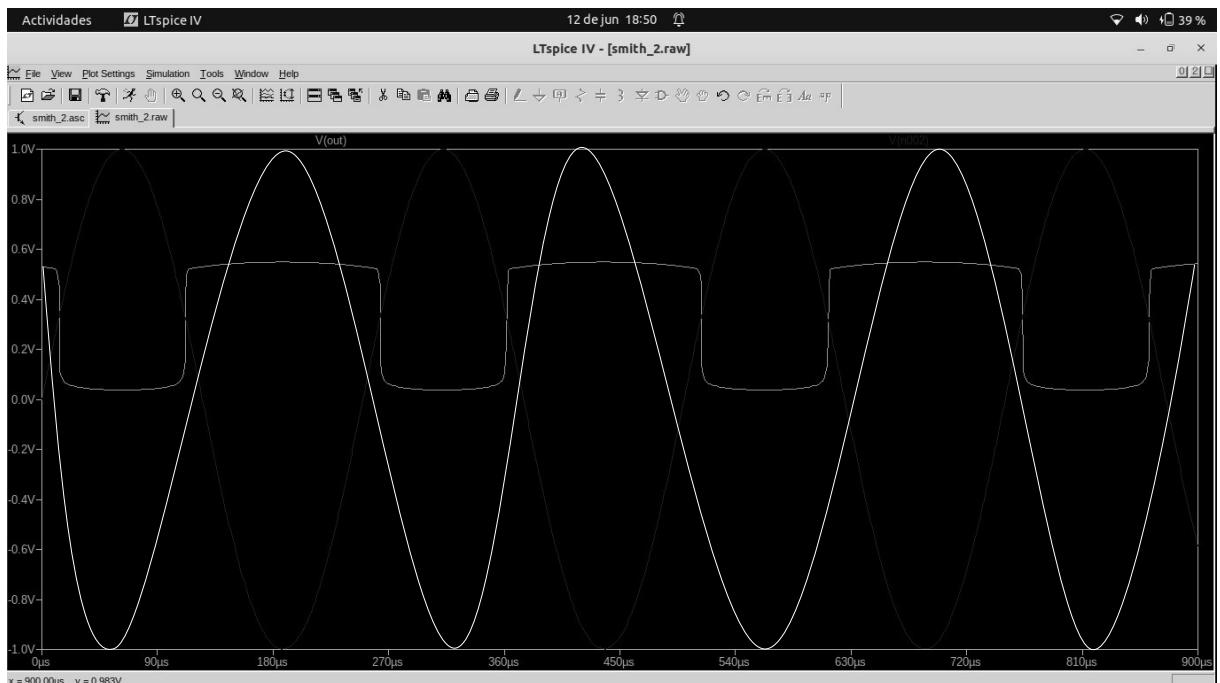


Figura 54 Schmitt Trigger Simulación en LTSPICE (Fuente: Propia)

5.2.4.3. Schmitt Trigger con Magic

Una vez que se extrajo el archivo *.spice* desde *Xschem* se procede a la realización de las uniones en *Magic*, como se observa en la figura Figura 55

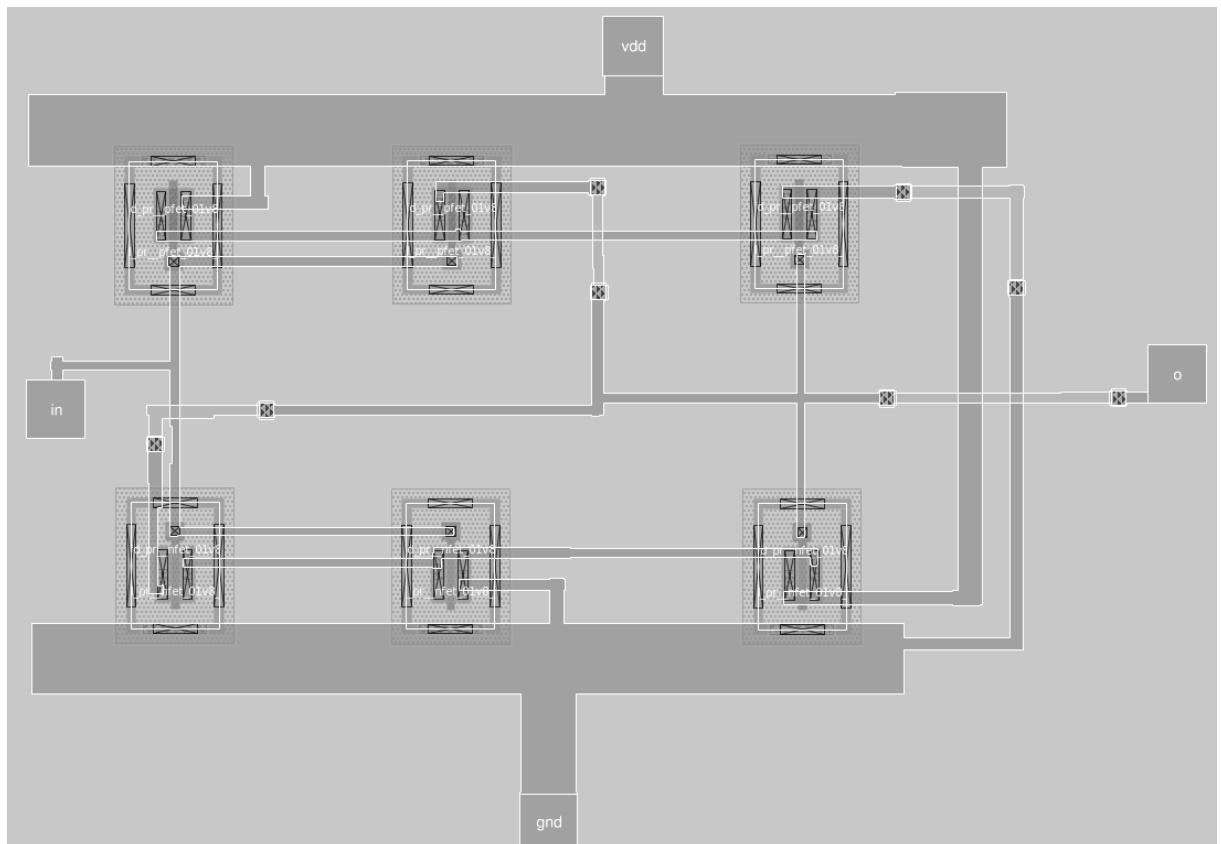


Figura 55 Schmitt Trigger en *Magic* (Fuente: Propia)

Luego de ser diseñado el circuito, se extrajeron todos los archivos necesarios para luego integrarlos al chip. La forma de llevar a cabo este proceso, así como cualquier otro circuito, sigue los mismos pasos mencionados en la subsección 5.2.2.

Capítulo 6: *Efabless Platform*

6.1. Introducción

Efabless se destaca como una plataforma innovadora que impulsa la creatividad y la colaboración en las industrias de semiconductores y sistemas de hardware. La empresa proporciona acceso tanto a IP existente como a la posibilidad de solicitar nuevos diseños a una comunidad de expertos, ofreciendo así soluciones flexibles a las empresas de chips. Esta estrategia permite abordar necesidades específicas y desafíos en el desarrollo de productos, al mismo tiempo que fomenta la diversidad y la innovación en el diseño de hardware. La capacidad de convertir las visiones de productos en realidades comercializables mediante la colaboración demuestra el potencial transformador de la plataforma *Efabless* en el campo de la tecnología integrada^[6].

Efabless ofrece repositorios de código abierto que contienen las características necesarias para el diseño de chips. En particular, nos centraremos en examinar el repositorio de Caravan en su versión analógica para la implementación del chip. Esto se ilustra en la Figura 56.

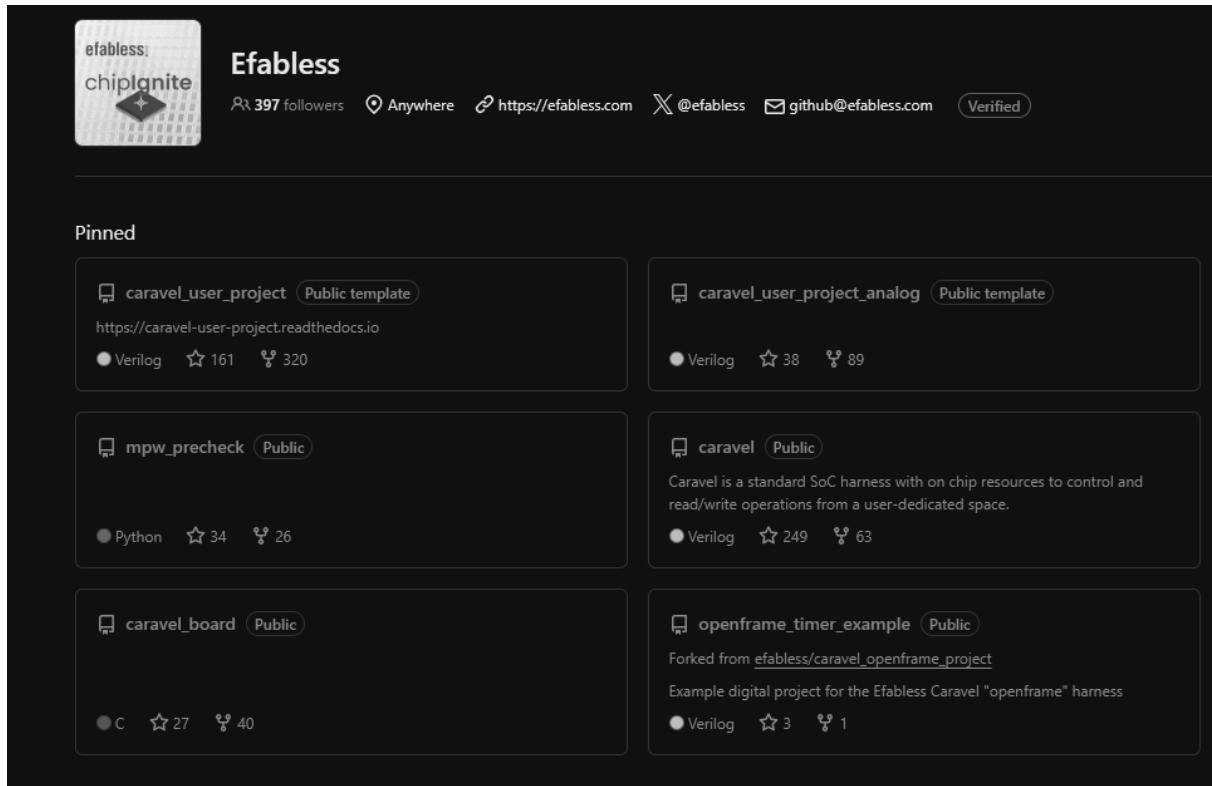


Figura 56 Repositorio de Efabless (Fuente: Propia)

Existen dos repositorios principales públicos disponibles: el primero es "*caravel_user_project*", reconocido como el repositorio más popular de Efabless, comúnmente denominado "*Caravel*" o "*Caravel Digital*"^[2]. El segundo es "*caravel_user_project_analog*", también conocido como "*Caravan*" o "*Caravel Analogico*"^[9]. Este último repositorio contiene todas las herramientas necesarias para ensamblar el chip.

Herramientas Utilizadas en el Repositorio de Caravel:

- *Magic*
 - Se utiliza para crear y editar el layout físico del IC, y para realizar verificaciones de diseño (DRC - Design Rule Check) y extracción de parámetros (LVS - Layout Versus Schematic).
- *Xschem*
 - Permite la creación y edición de esquemáticos de circuitos, que luego se pueden exportar a formatos *SPICE* para simulación.
- *Netgen*
 - Se utiliza para realizar comparaciones de LVS entre el esquemático y el layout para asegurar que ambos coincidan.
- *OpenLane*

- o Facilita el flujo de diseño desde la descripción RTL hasta el GDSII, incluyendo síntesis, colocación, enrutamiento y verificación.
- *Yosys*
 - o Convierte la descripción RTL (Register Transfer Level) en una netlist de puertas lógicas, la cual puede ser utilizada en el flujo de diseño.
- *OpenRoad*
 - o Proporciona capacidades avanzadas de colocación y enrutamiento para optimizar el diseño físico del circuito.
- *KLayout*
 - o Permite la visualización y edición manual de layouts GDSII, así como la ejecución de scripts para automatizar tareas de verificación y manipulación de layouts.
- *GHDL*
 - o Facilita la simulación y verificación de descripciones en VHDL para asegurar que el comportamiento del diseño RTL es el esperado.
- *Verilator*
 - o Permite la simulación de modelos Verilog a alta velocidad, útil para verificar el comportamiento funcional del diseño.
- *Cocotb*
 - o Se utiliza para escribir y ejecutar tests de simulación, integrando fácilmente con otros simuladores como Verilator

La función principal de estos repositorios es proporcionar acceso libre a los usuarios. Por lo tanto, al clonarlos, es importante tener en cuenta que una vez que el proyecto esté completo y se haya integrado en la plataforma de *Efabless*, será visible para toda la comunidad de Efabless. Este requisito es fundamental al utilizar estas tecnologías.

Posteriormente, se procede a crear el repositorio utilizando uno de estos dos templates disponibles.

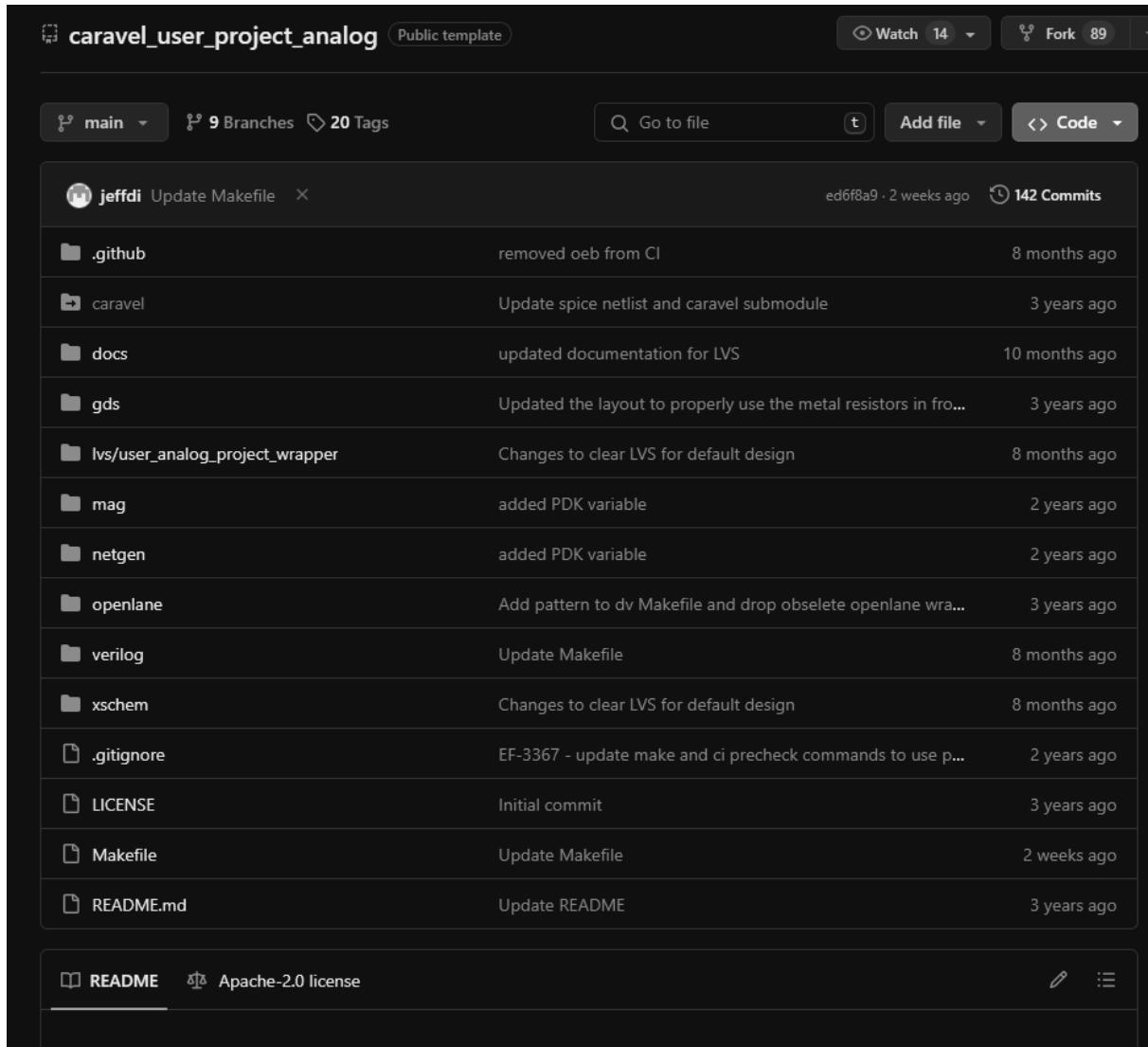


Figura 57 Repositorio *caravel_user_project_analog* (Fuente: Propia)

Para este propósito, se accede al repositorio "*caravel_user_project_analog*" y se examinan los diferentes directorios que forman parte del proceso de diseño del chip. Dentro de estos directorios, se encuentran submódulos que deben instalarse para poder utilizar el repositorio de manera local. Para utilizar los submódulos, se ejecuta el comando "*make setup*". En este proyecto, se realizaron algunas modificaciones debido a que la versión de *Caravan* está desactualizada en comparación con *Caravel*. Para resolver esta discrepancia, se empleó el *Makefile* del repositorio de Caravel en *Caravan*.

Por defecto, al ejecutar "*make setup*", se instalan los siguientes submódulos:

- "*check_dependencies*": Verifica las dependencias necesarias para la versión actual de Caravel.

- "*install*": Instala el submódulo de Caravel, así como "mcw" y "cocotb" (aunque este último no se utiliza en este caso, se mantiene instalado para futuras versiones).
- "*setup-timing-scriptsprecheck*".

Además de estas herramientas predeterminadas, Efabless también ofrece otro submódulo llamado "caravel_lite", que está configurado por defecto en el Makefile del repositorio. Sin embargo, si se necesita utilizar la versión completa, se debe exportar la variable "CARAVEL_LITE=0", lo que descargará la versión completa (en la terminal de Linux realizar *export CARAVEL_LITE=0*). Para este desarrollo, se requiere la versión completa, ya que "caravel_lite" no contiene todos los archivos necesarios para Caravan. De esta manera, en este proyecto se realizó una actualización de varios componentes de Caravan a partir de Caravel.

Una vez completado este proceso, el repositorio estará disponible localmente como se muestra en la Figura 58.

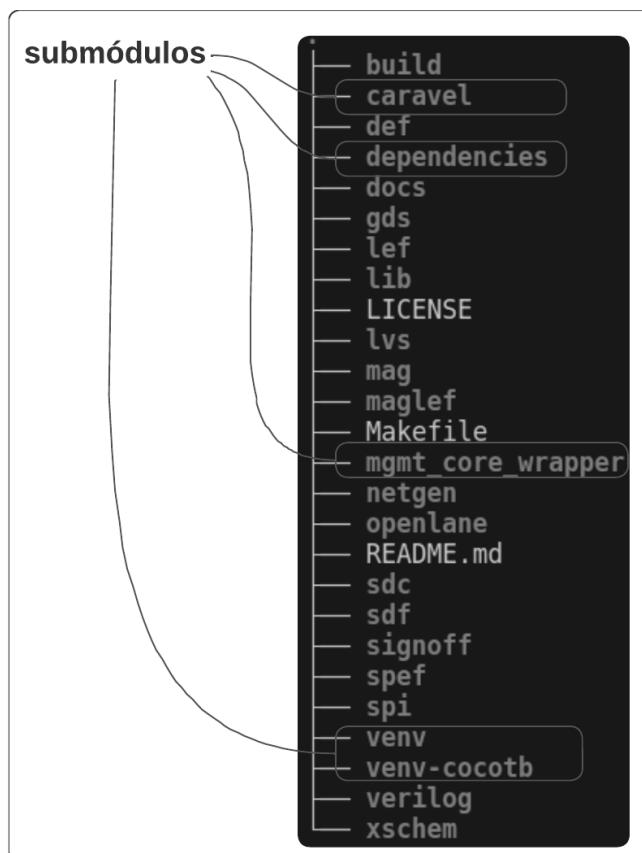


Figura 58 make setup (Fuente: Propia)

A continuación, se procede a reemplazar el directorio "./verilog" de Caravan por el de Caravel. En cada uno de estos directorios se han colocado los códigos para las pruebas. Una vez realizado esto, el directorio ./verilog en *Caravan* se ve como la Figura 59.

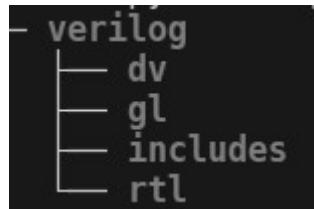


Figura 59 Directorio verilog en repositorio Caravan (Fuente: Propia)

6.2. Integración del circuito digital

Una vez clonado el repositorio de *caravel_user_project_analog*, se accede al directorio correspondiente, donde se encuentran varios archivos. El siguiente paso implica la navegación al directorio de Verilog, donde se hallan los archivos que se muestran en la Figura 60.

```
jona@pat ~ [v] 3.10.12 > ~/.../test_mixer/verilog > └── main └── tree -L 2
.
└── dv
    ├── cocotb
    ├── local-install.md
    ├── Makefile
    ├── README.md
    ├── setup-cocotb.py
    ├── test_mixer
    ├── test_uart
    └── wb_prueba
.
└── gl
    ├── caravan_core.v
    ├── caravel_core.v
    ├── gpio_defaults_block_0402.v
    ├── gpio_defaults_block_0403.v
    ├── gpio_defaults_block_0801.v
    ├── gpio_defaults_block_1803.v
    ├── gpio_defaults_block_1808.v
    ├── test_mixer.nl.v
    ├── test_mixer.v
    ├── user_analog_project_wrapper.nl.v
    ├── user_analog_project_wrapper.v
    ├── user_analog_proj_example.nl.v
    ├── user_analog_proj_example.v
    ├── user_id_programming.v
    └── user_project_wrapper.v
.
└── includes
    ├── includes.gl.caravel_user_project
    ├── includes.gl+sdf.caravel_user_project
    └── includes.rtl.user_analog_project_wrapper
.
└── rtl
    ├── caravel_core.v
    ├── defines.v
    ├── test_mixer_copy.v
    ├── test_mixer.v
    ├── uprj_netlists.v
    ├── user_analog_project_wrapper_copy.v
    └── user_analog_project_wrapper.v
    └── userDefines.v
```

Figura 60 caravel_user_project_analog (Fuente: Propia)

Dentro del directorio de *Verilog*, se encuentran los archivos relevantes para este proyecto. Es importante revisar cuidadosamente estos archivos para comprender cómo se estructura el código y qué funciones realizan en el contexto del proyecto en curso.

En el directorio “/dv” se encuentra el archivo *Makefile* principal, luego, un subdirectorio para cada una de las pruebas que luego se cargarán dentro del *Management SoC Wrapper* (Risc-V) a modo de simulación.

El directorio “/rtl” contiene los archivos *Verilog* los cuales son submódulos que se integrarán en el *user_analog_project_wrapper.v*. El top del proyecto es *user_analog_project_wrapper.v* desde este archivo debemos integrar todos los submódulos digitales.

El directorio “/gl” incluye todas las *netlists* elaboradas por *OpenLane*, quien carga los archivos necesarios para el proyecto, de manera automática.

La Figura 61 ilustra la composición del Caravel Harness Chip.

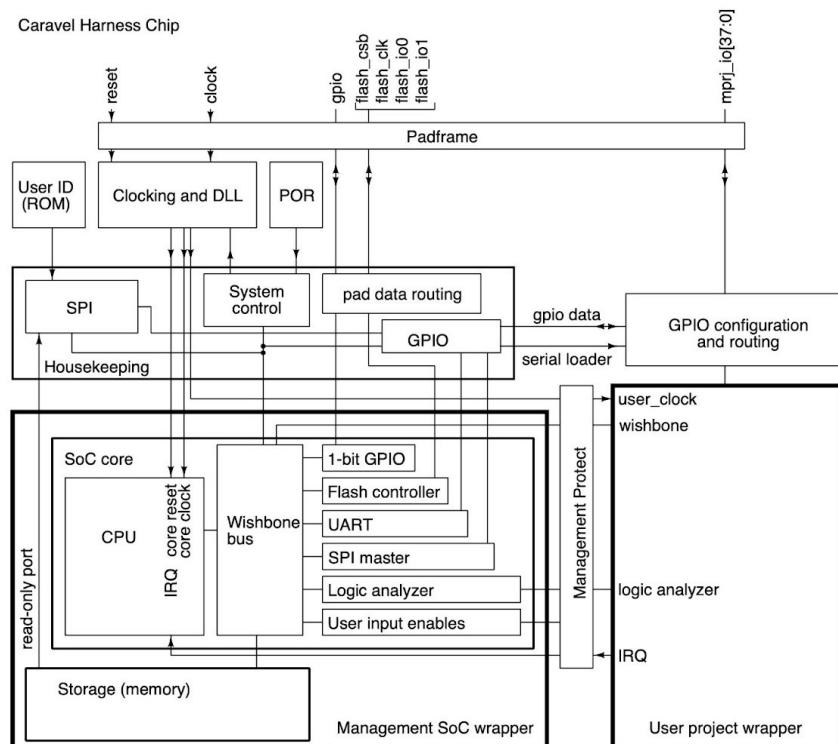


Figura 61 Caravel Harness Chip (Fuente: Caravel [2])

Desde el *Management SoC wrapper* se pueden testear la parte de *user_analog_project_wrapper* (en la Figura 61 aparece como “User project wrapper”). Cuando el proyecto es puramente digital se utiliza *OpenLane* para hacer el *hardening* en el archivo *test_mixer.v*, en este caso y convertirlo en el GDSII.

6.2.1. OpenLane

OpenLane^[10] es una plataforma innovadora de implementación de silicio que soporta herramientas de código abierto como *Yosys*, *OpenROAD*, *Magic*, *KLayout*, junto con otras utilidades de código abierto y propietarias. Desde el año 2020, *OpenLane* ha sido utilizado en cada *Open MPW* y en el servicio de *shuttle* de *chipIgnite*.

El flujo de trabajo de *OpenLane* puede ser descrito por la Figura 62.

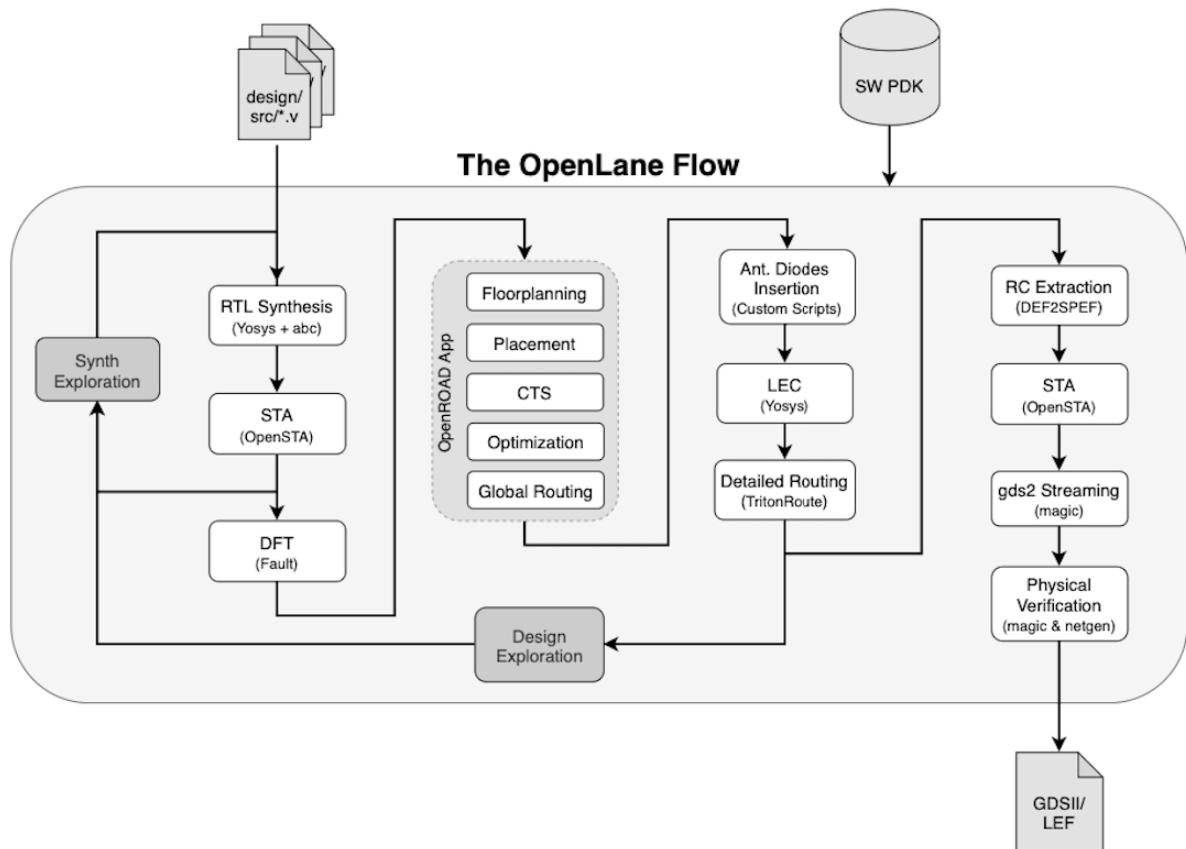


Figura 62 Flow OpenLane.v (Fuente: OpenLane [21])

Antes de realizar el *harness* con *OpenLane* se deben configurar los pines que se conectarán entre el *wrapper* y la macro(*test_mixer*). Para ello, se debe configurar el archivo que se encuentra en *verilog/rtl/userDefines.v*, que al ser configurado resulta como se puede visualizar en la Figura 63.

```

55 `define USER_CONFIG_GPIO_5_INIT    `GPIO_MODE_USER_STD_INPUT_NOPULL
56 `define USER_CONFIG_GPIO_6_INIT    `GPIO_MODE_USER_STD_INPUT_NOPULL
57 `define USER_CONFIG_GPIO_7_INIT    `GPIO_MODE_USER_STD_INPUT_NOPULL
58 `define USER_CONFIG_GPIO_8_INIT    `GPIO_MODE_MGMT_STD_INPUT_NOPULL
59 `define USER_CONFIG_GPIO_9_INIT    `GPIO_MODE_MGMT_STD_INPUT_NOPULL
60 `define USER_CONFIG_GPIO_10_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
61 `define USER_CONFIG_GPIO_11_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
62 `define USER_CONFIG_GPIO_12_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
63 `define USER_CONFIG_GPIO_13_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
64
65 // Configurations of GPIO 14 to 24 are used on caravel but not caravan.
66 `define USER_CONFIG_GPIO_14_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
67 `define USER_CONFIG_GPIO_15_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
68 `define USER_CONFIG_GPIO_16_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
69 `define USER_CONFIG_GPIO_17_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
70 `define USER_CONFIG_GPIO_18_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
71 `define USER_CONFIG_GPIO_19_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
72 `define USER_CONFIG_GPIO_20_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
73 `define USER_CONFIG_GPIO_21_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
74 `define USER_CONFIG_GPIO_22_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
75 `define USER_CONFIG_GPIO_23_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
76 `define USER_CONFIG_GPIO_24_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
77
78 `define USER_CONFIG_GPIO_25_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
79 `define USER_CONFIG_GPIO_26_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
80 `define USER_CONFIG_GPIO_27_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
81 `define USER_CONFIG_GPIO_28_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
82 `define USER_CONFIG_GPIO_29_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
83 `define USER_CONFIG_GPIO_30_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
84 `define USER_CONFIG_GPIO_31_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
85 `define USER_CONFIG_GPIO_32_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
86 `define USER_CONFIG_GPIO_33_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
87 `define USER_CONFIG_GPIO_34_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
88 `define USER_CONFIG_GPIO_35_INIT   `GPIO_MODE_USER_STD_OUTPUT
89 `define USER_CONFIG_GPIO_36_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
90 `define USER_CONFIG_GPIO_37_INIT   `GPIO_MODE_MGMT_STD_INPUT_NOPULL
91

```

Figura 63 Resumen de userDefines.v (Fuente: Propia)

Luego, para integrar el proyecto digital realizado anteriormente, llamado *test_mixer* (después de las verificaciones pertinentes y observaciones en *gtkwave*), se procede a endurecerlo a realizar el *hardening*.

Para llevar a cabo el *hardening*, en el repositorio del proyecto se encuentra el directorio *OpenLane*, como se observa en la Figura 64.

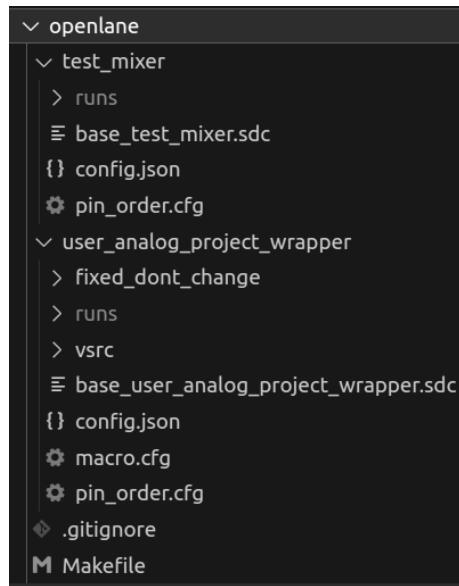


Figura 64 Directorio OpenLane Caravan (Fuente: Propia)

Luego se accede al directorio *test_mixer*, donde se encuentran las configuraciones de la macro (se llama *macro* a cada submódulo del *wrapper*) que se va a realizar el *hardening*.

Allí se deben modificar los archivos:

- *base_test_mixer.sdc*
Esta configuración especifica las restricciones de diseño.
- *pin_order.cfg*
En este archivo se pueden observar los pines que se definieron para esta macro. Por ejemplo, se encuentra el pin del *clock*, los pines de entrada y salida a la macro, y la ubicación de los mismos, ya sea #S #N #E #WR, Figura 65.
- *config.json*
Archivo principal para la ejecución de *OpenLane*. Este archivo contiene el nombre de la macro, archivos Verilog, definición del *clock*, líneas de tensión, tamaño de la macro, entre otras configuraciones.

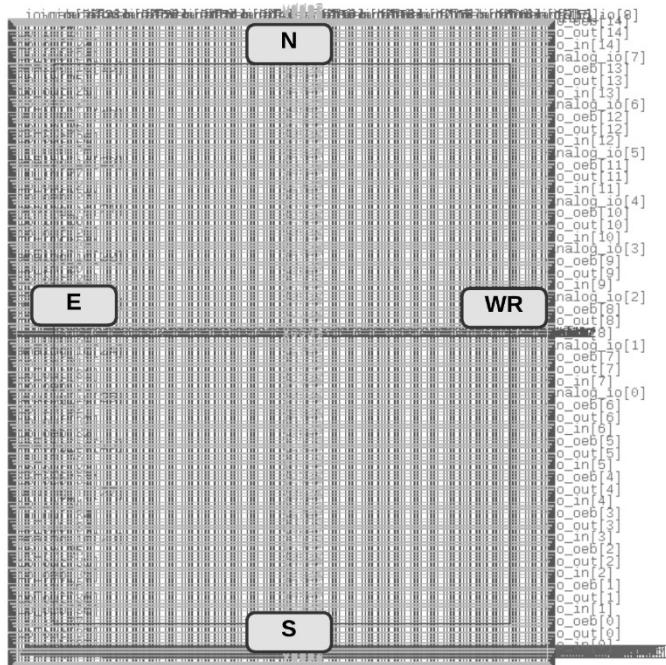


Figura 65 Ubicación pines en user_project_wrapper_empty.gds (Fuente: Propia)

Luego se utiliza el comando `make test_mixer` para que comience a correr el flujo de diseño del chip *OpenLane*.

OpenLane, al igual que las aplicaciones nombradas anteriormente, corren dentro del contenedor de Efabless en Docker. Una vez finalizada la ejecución del comando, y si todo resultó exitoso, se obtiene una salida de consola como se muestra en la Figura 66.

```
[INFO]: Created metrics report at '../home/jona/Desktop/Proyecto_Final/caravan/openlane/test_mixer/runs/24_06_07_16_37/reports/metrics.csv'.
[WARNING]: There are max fanout violations in the design at the typical corner. Please refer to '../home/jona/Desktop/Proyecto_Final/caravan/openlane/test_mixer/runs/24_06_07_16_37/reports/signoff/33-rcx_sta.checks.rpt'.
[INFO]: There are no hold violations in the design at the typical corner.
[INFO]: There are no setup violations in the design at the typical corner.
[SUCCESS]: Flow complete.
```

Figura 66 Salida luego del comando make test_mixer (Fuente: Propia)

Luego de correr el comando se puede observar la creación de un nuevo directorio llamado *runs*.

Dentro del directorio *runs* se encuentran todos las salidas proporcionadas por *OpenLane*, dos de las cuales consisten en los archivos *.gds* y *.lef*, que se encuentran en el path

`openlane/test_mixer/runs/(date)/results/final/gds/test_mixer.gds`

y

`openlane/test_mixer/runs/(date)/results/final/lef/test_mixer.lef.`

El archivo `.gds` se observa en la Figura 67, utilizando la herramienta *KLayout*^[14].

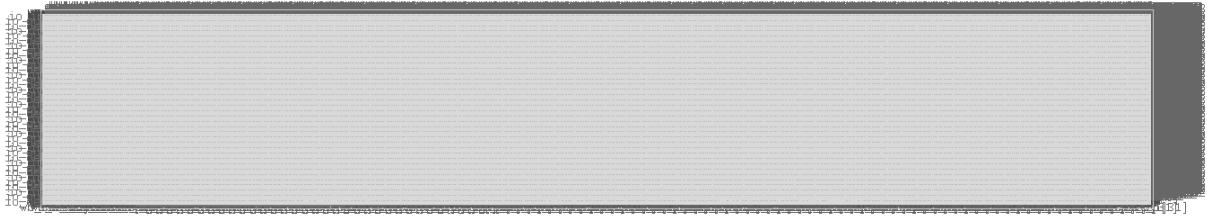


Figura 67 Archivo `.gds` de la macro `test_mixer` con *KLayout* (Fuente: Propia)

```
└─ openlane
   └─ test_mixer
      └─ runs
         └─ 24_05_27_16_39
            ├ logs
            ├ reports
            └─ results
               ├ cts
               └─ final
                  ├ def
                  ├ gds
                  └─ lef
                     └─ test_mixer.lef
                        ├ lib
                        ├ mag
                        ├ maglef
                        ├ sdc
                        ├ sdf
                        ├ spef
                        ├ spi
                        └ verilog
```

Figura 68 Archivo `test_mixer.lef` (Fuente: Propia)

En *OpenLane*, la salida `.lef` (Liberty Exchange Format) en la Figura 68, contiene información sobre el diseño físico del circuito integrado, incluyendo detalles como las ubicaciones de los

componentes, el enrutamiento de las pistas, las dimensiones de las celdas estándar y otros parámetros importantes para la fabricación del chip. Este archivo .lef es utilizado por otras herramientas de diseño de circuitos integrados para realizar tareas como el diseño de placas de circuito impreso, la simulación de la disposición física del chip y la verificación del diseño. En resumen, el archivo *.lef* generado por *OpenLane* proporciona información esencial sobre la geometría y la topología del diseño físico del circuito integrado, lo que facilita su integración con otras herramientas de diseño y fabricación como lo son *Magic*, *Klayout* entre otras.

Los archivos también se colocan en los directorios de *Caravan* para que se utilicen mediante *OpenLane* con la ejecución del comando *make user_analog_project_wrapper*. Este comando emplea la macro anterior y la integra en el *wrapper*.

Se requiere una configuración adicional para este proceso, que consiste en posicionar la macro en el "user_analog_project_wrapper" dentro del archivo *macro.cfg*^[16].

En este caso, *macro.cfg* contiene la siguiente línea:

```
test_mixer 60 15 N
```

Después de ejecutar el comando, la configuración de los pines ya está establecida para el *wrapper* y se obtiene el archivo GDS, tal como se visualiza en la Figura 69.

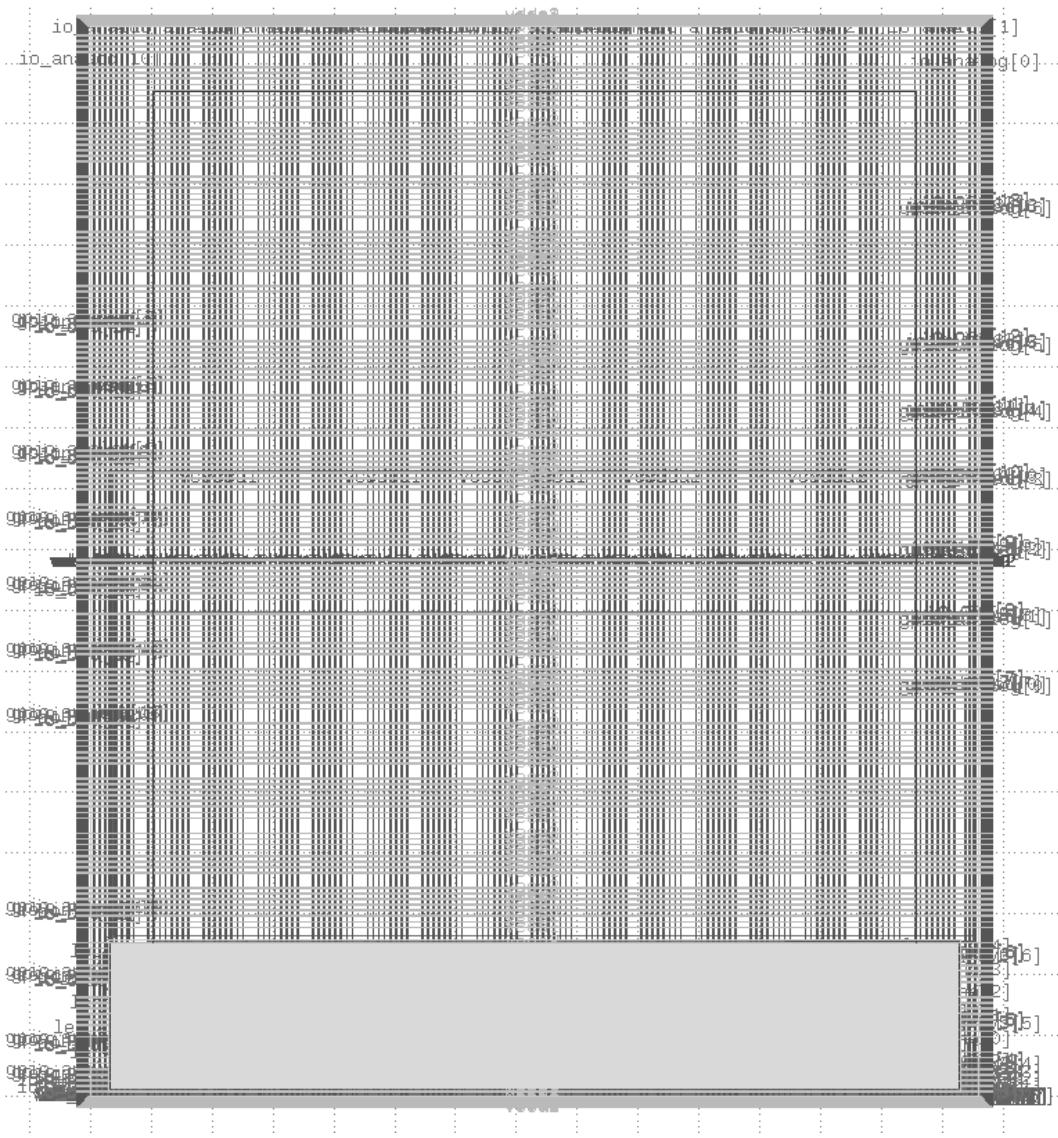


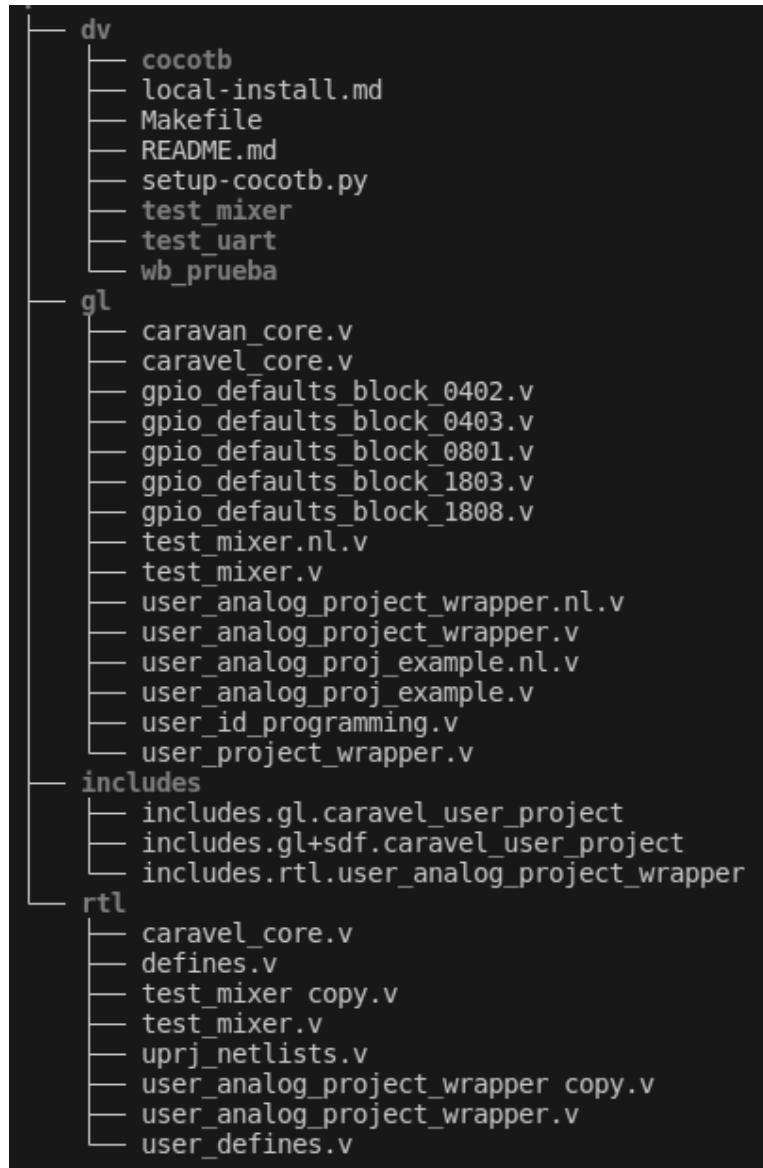
Figura 69 Archivo .gds de la macro *user_analog_project_wrapper* con Klayout (Fuente: Propia)

Si todos los pasos se ejecutaron correctamente se obtiene el GDSII de la Figura 69. En la próxima sección se analizará cómo agregar el *Mixer* al *wrapper* (*user_analog_project_wrapper*).

6.3. Integración del circuito analógico

El proyecto se caracteriza por enfocarse en el diseño de una sección de alta frecuencia analógica con una porción digital mínima, lo cual justifica la utilización de *Caravan*, en vez de *Caravel*,

además *Caravan* contiene protección *ESD* en algunos de sus pines (Tabla 7 - Anexo A). A continuación, se profundiza en esta característica particular. En la Figura 70, se observan los directorios y subdirectorios necesarios para la creación del área digital del chip.



```

.
├── dv
│   ├── cocotb
│   │   ├── local-install.md
│   │   ├── Makefile
│   │   ├── README.md
│   │   ├── setup-cocotb.py
│   │   ├── test_mixer
│   │   ├── test_uart
│   │   └── wb_prueba
├── gl
│   ├── caravan_core.v
│   ├── caravel_core.v
│   ├── gpio_defaults_block_0402.v
│   ├── gpio_defaults_block_0403.v
│   ├── gpio_defaults_block_0801.v
│   ├── gpio_defaults_block_1803.v
│   ├── gpio_defaults_block_1808.v
│   ├── test_mixer.nl.v
│   ├── test_mixer.v
│   ├── user_analog_project_wrapper.nl.v
│   ├── user_analog_project_wrapper.v
│   ├── user_analog_proj_example.nl.v
│   ├── user_analog_proj_example.v
│   ├── user_id_programming.v
│   └── user_project_wrapper.v
├── includes
│   ├── includes.gl.caravel_user_project
│   ├── includes.gl+sdf.caravel_user_project
│   └── includes.rtl.user_analog_project_wrapper
└── rtl
    ├── caravel_core.v
    ├── defines.v
    ├── test_mixer_copy.v
    ├── test_mixer.v
    ├── uprj_netlists.v
    ├── user_analog_project_wrapper_copy.v
    └── user_analog_project_wrapper.v
        └── userDefines.v

```

Figura 70 Path (Fuente: Propia)

Dentro del repositorio *Caravan* se puede observar *user_analog_project_wrapper.empty*, lo cual es la representación *GDSII* de la *user_project_wrapper* en la Figura 71, es decir, no contiene macros o submódulos en el *wrapper*.

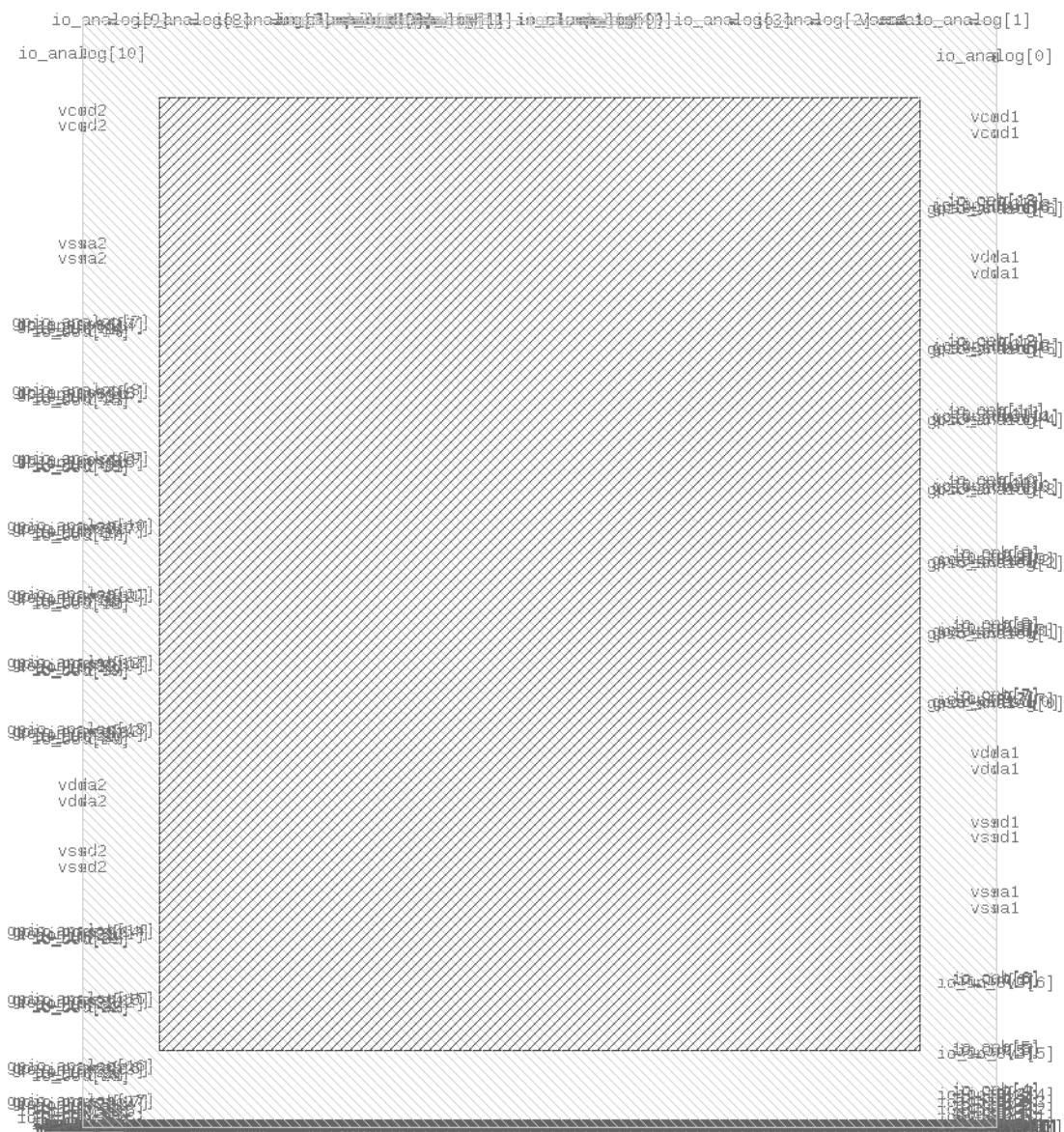


Figura 71 user analog project wrapper empty.gds (Fuente: Propia)

El wrapper de Caravan difiere del de Caravel, lo que demanda una atención especial al trabajar con señales analógicas. Esto se debe a que las señales analógicas de alta frecuencia son susceptibles a la contaminación por señales digitales o el ruido interno del chip. Para mitigar este riesgo, es recomendable utilizar cables más anchos y evitar las corrientes parásitas. Además, es beneficioso emplear condensadores de desacoplamiento para mejorar la integridad de la señal. Esta práctica es

especialmente útil dada la amplitud de la región de *user_analog_project_wrapper*, la cual es suficiente para la mayoría de los circuitos enviados a través del *harness* (región aprovechada).

Los circuitos analógicos se diseñan en *Magic* y se someten a pruebas en *SPICE*. Posteriormente, se realiza la verificación *LVS* (Layout vs. Schematic) con *Netgen* como se vio anteriormente. Para un circuito analógico, el diseño se lleva a cabo en *Magic* y luego se integra en el *user_analog_project_wrapper_empty.mag*. Es importante tener en cuenta que el nombre del archivo top del proyecto siempre debe ser *user_analog_project_wrapper*, ya sea para archivos de tipo *.lef*, *.gds*, *.v* u otras especificaciones pertinentes. En este caso diseñamos en *Magic* el proyecto que se tiene como objetivo y luego una vez que las pruebas de *lvs* son correctas se procede a unirlas en el *user_analog_project_wrapper* está unión se realiza manualmente ya que *OpenLane* no puede realizarlo ya que no se puede desarrollar un *Verilog* para este caso debido a que el *Verilog* describe circuitos digitales y no analógicos.

En su forma estándar, *Verilog* está orientado hacia el diseño y simulación de circuitos digitales, pero su flexibilidad permite a los diseñadores utilizarlo para describir sistemas que incluyen tanto componentes digitales como analógicos. Sin embargo, es importante destacar que *Verilog* no es tan robusto ni especializado para el diseño de circuitos analógicos como herramientas específicas como *SPICE*.

En este proyecto se quiere hacer uso de las dos posibilidades antes mencionadas por un lado se tiene una parte digital que se materializa (es decir crear las archivos de especificaciones del circuitos como lo son el *.gds .lef*), utilizando *OpenLane* y luego se integra el archivo *.mag* generado desde *Magic* (*Mixer.mag*), esto se realiza de esta manera ya que se quiere utilizar el procesador incorporado en el chip el *Risc-V (picorv32)* como se observa en la Figura 72 a modo de descriptivo:

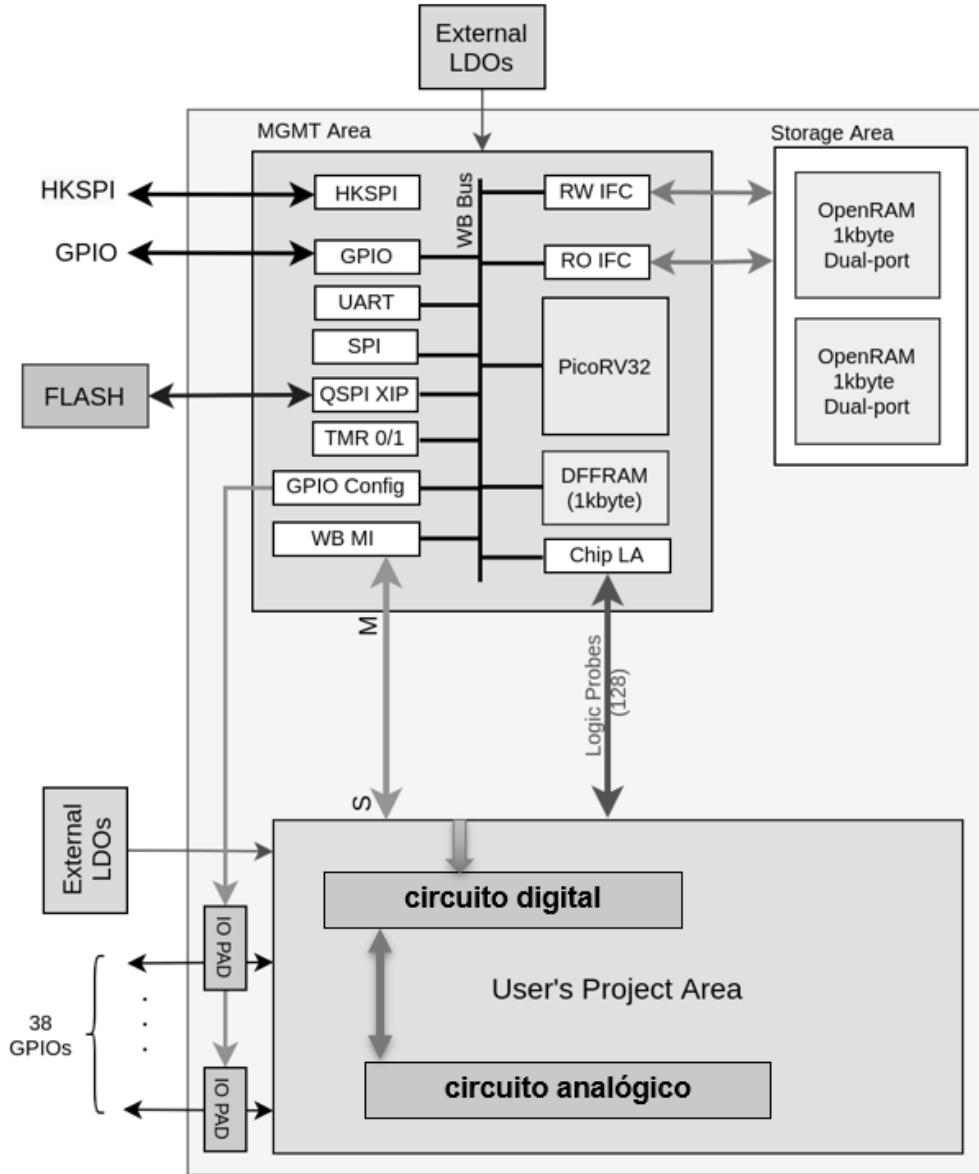


Figura 72 Descripción del proyecto (Fuente: Efabless [6])

Se quiere hacer uso de este procesador ya que viene incorporado en el chip, mediante la conexión al exterior `hkspi`^[43] se puede cargar el `.hexa` que se ejecutará en el *Chip*.

El proyecto consta de dos partes, como se mencionó anteriormente: una parte puramente analógica, donde se desarrolla un proyecto de *Mixer* y *el Schmitt Trigger*, y la parte digital, que monitorea las respuestas del *Mixer* luego de salir del Chip y pasar por un demodulador externo. La Figura 73 ilustra el objetivo del proyecto.

El SPI de "housekeeping"(`hkspi`) es un dispositivo SPI que puede ser accedido desde un host remoto a través de una interfaz serial estándar de 4 pines. La implementación del SPI es en modo 0, con nuevos datos en SDI capturados en el flanco ascendente de SCK, y datos de salida presentados

en el flanco descendente de SCK (para ser muestrados en el siguiente flanco ascendente de SCK). Los pines SPI se comparten con el GPIO del área de usuario.

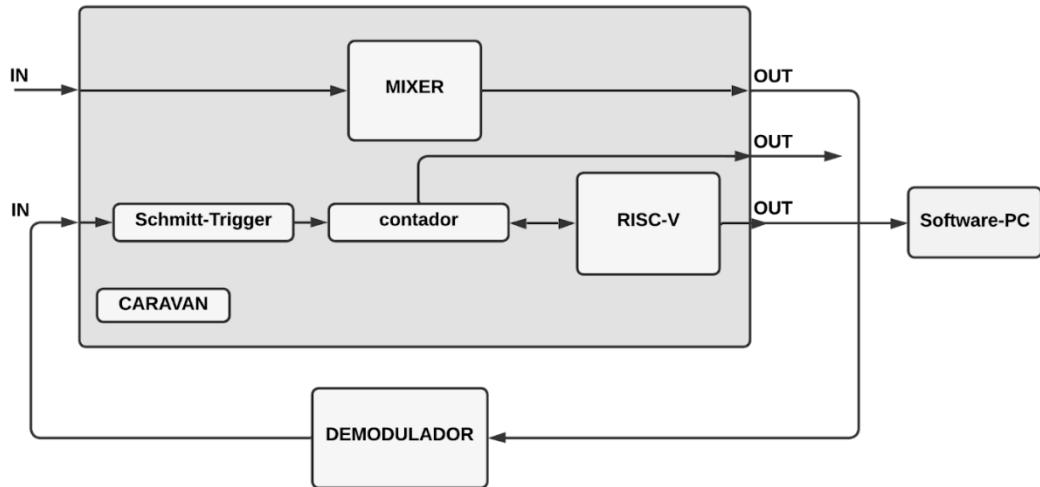


Figura 73 Figura abstracta del objetivo final (Fuente: Propia)

Se puede integrar un procesador personalizado como submódulos al *user_project_wrapper* evitando usar el procesador provisto por *Efabless*.

En la Figura 74 se observa la integración del *Mixer* en *user_analog_project_wrapper.mag*, luego la unión a la parte digital se realiza manualmente.

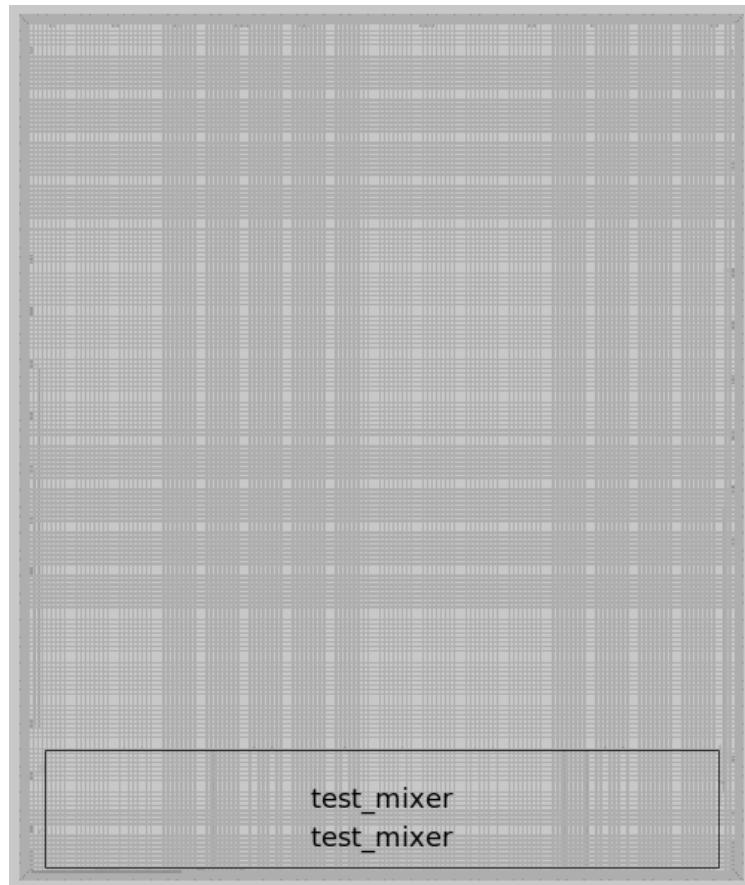


Figura 74 Macro *test_mixer* en *user_analog_project_wrapper* (Fuente: Propia)

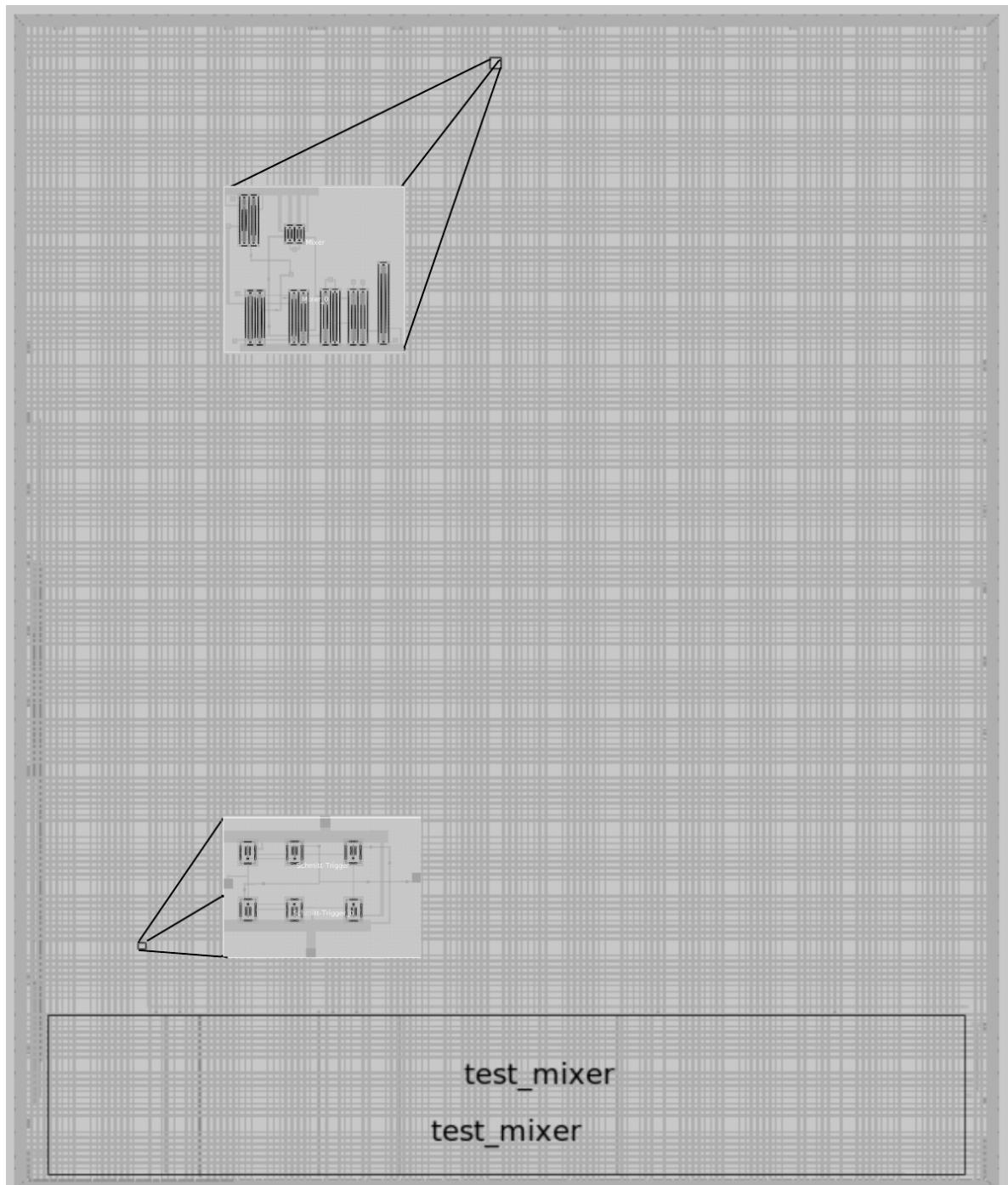


Figura 75 Mixer, Schmitt Trigger y contador digital (*test_mixer*) (Fuente: Propia)

Las conexiones realizadas manualmente fueron a los pines que se observan en la Tabla 5, para el *Mixer*.

<i>vlo+</i>	<i>io_analog[4]</i>
<i>vcc</i>	<i>vccd1</i>
<i>gnd</i>	<i>vssd1</i>
<i>vlo-</i>	<i>io_analog[5]</i>
<i>vbias1</i>	<i>vccd2</i>
<i>vbias2</i>	<i>vccd2</i>
<i>vbias3</i>	<i>vccd2</i>
<i>out+ (vout voltaje de frecuencia intermedia)</i>	<i>io_analog[7]</i>
<i>out-</i>	<i>io_analog[3]</i>
<i>vrf+ (voltaje de radiofrecuencia)</i>	<i>io_analog[1]</i>
<i>vrf-</i>	<i>io_analog[2]</i>
<i>vtail</i>	<i>vccd2</i>

Tabla 5 Conexiones del Mixer (Fuente: Propia)

Las conexiones del *Schmitt Trigger* se realizaron a los pines de la Tabla 6.

<i>in</i>	<i>gpio_analog[14]</i>
<i>vcc</i>	<i>io_in_3v3[21]</i>
<i>gnd</i>	<i>io_out[21]</i>
<i>out</i>	<i>io_in[7]</i>

Tabla 6 Conexiones del Schmitt Trigger (Fuente: Propia)

Al aplicar el comando "*gds write mixer.gds*", se obtiene el archivo GDS, que incluye la parte digital deseada como se muestra en la Figura 76.

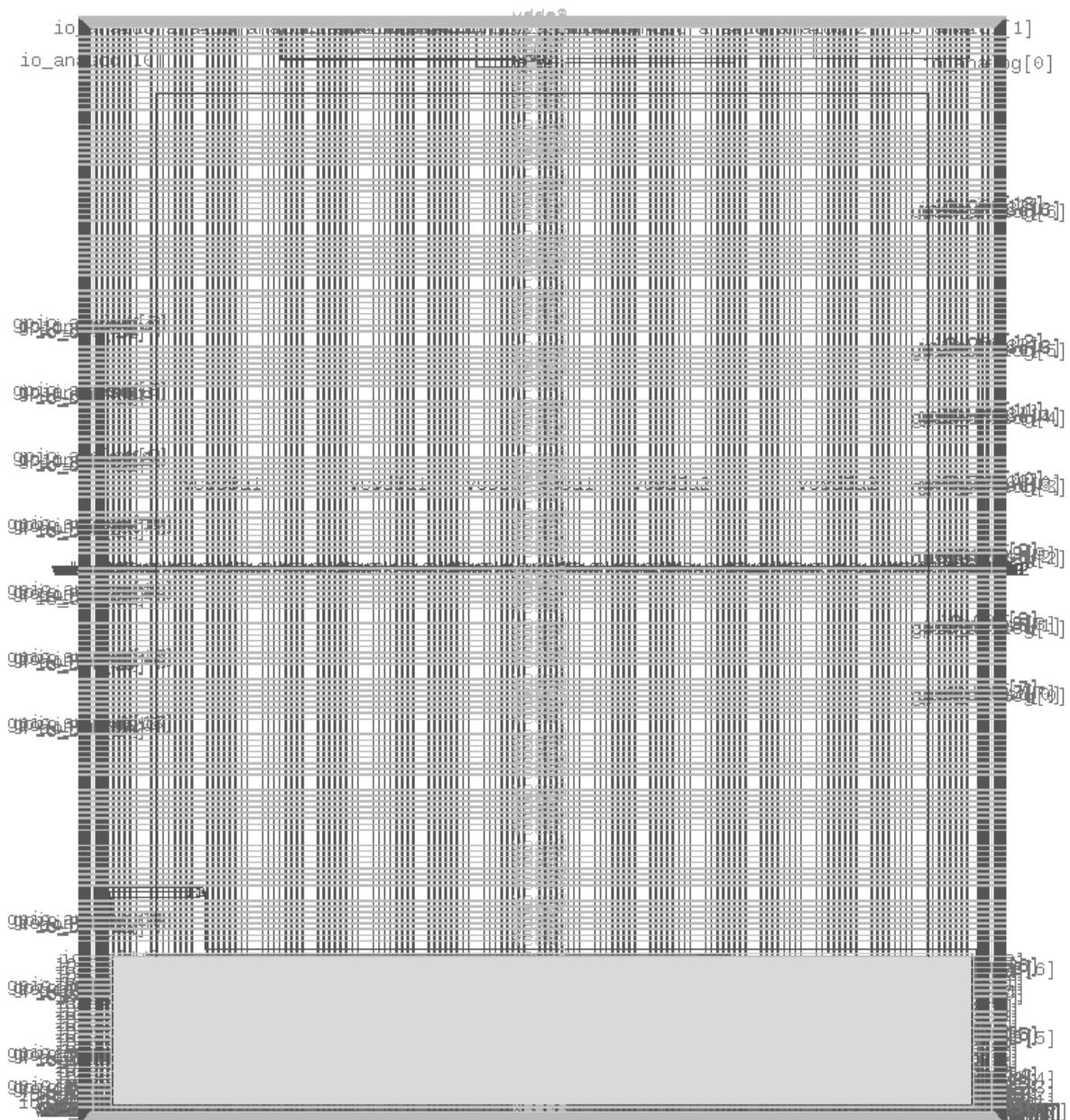


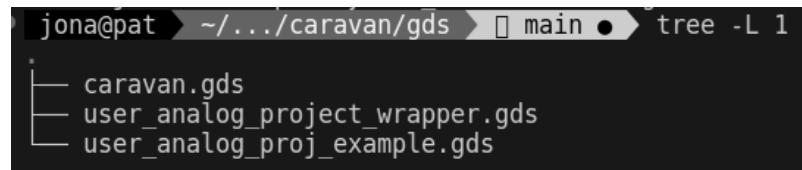
Figura 76 Mixer, test_mixer, user_analog_project_wrapper Klayout (Fuente: Propia)

Luego se aplican los siguientes comandos:

caravel:

```
env USER_ID=0000000F make set_user_id  
make gpio_defaults
```

```
ulimit -n 2024
make ship
caravan:
env USER_ID=0000000F make set_user_id
make gpio_defaults
ulimit -n 2024
make truck
```



```
jona@pat ~/.../caravan/gds main tree -L 1
.
├── caravan.gds
└── user_analog_project_wrapper.gds
    └── user_analog_proj_example.gds
```

Figura 77 Directorio *Caravan/gds* con *caravan.gds* (Fuente: Propia)

Después de ejecutar el comando se puede observar el archivo *caravan.gds* en */caravan/gds* y, al abrirlo con *KLayout*, se observa como en la Figura 78.

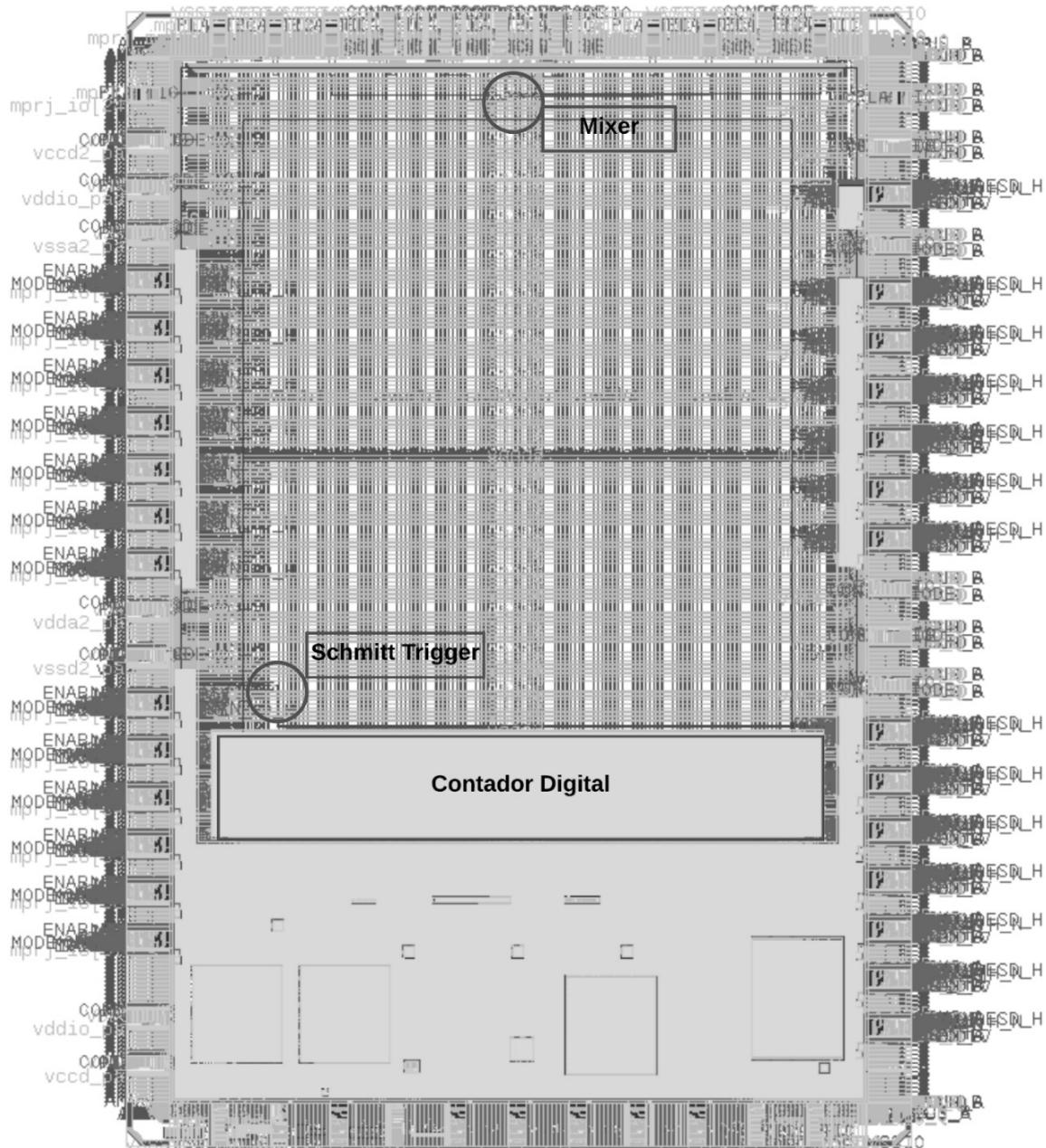


Figura 78 Chip Caravan con Mixer, Schmitt Trigger y parte Digital (Fuente: Propia)

Se adjuntó una imagen que representa el diseño .gds del chip. Este archivo es esencial, ya que será cargado en la plataforma de *Efabless* para su evaluación, marcando así el inicio del proceso para el próximo lanzamiento. Esta etapa es crucial para avanzar en el desarrollo del proyecto.

La Figura 79 ilustra la representación física del chip.

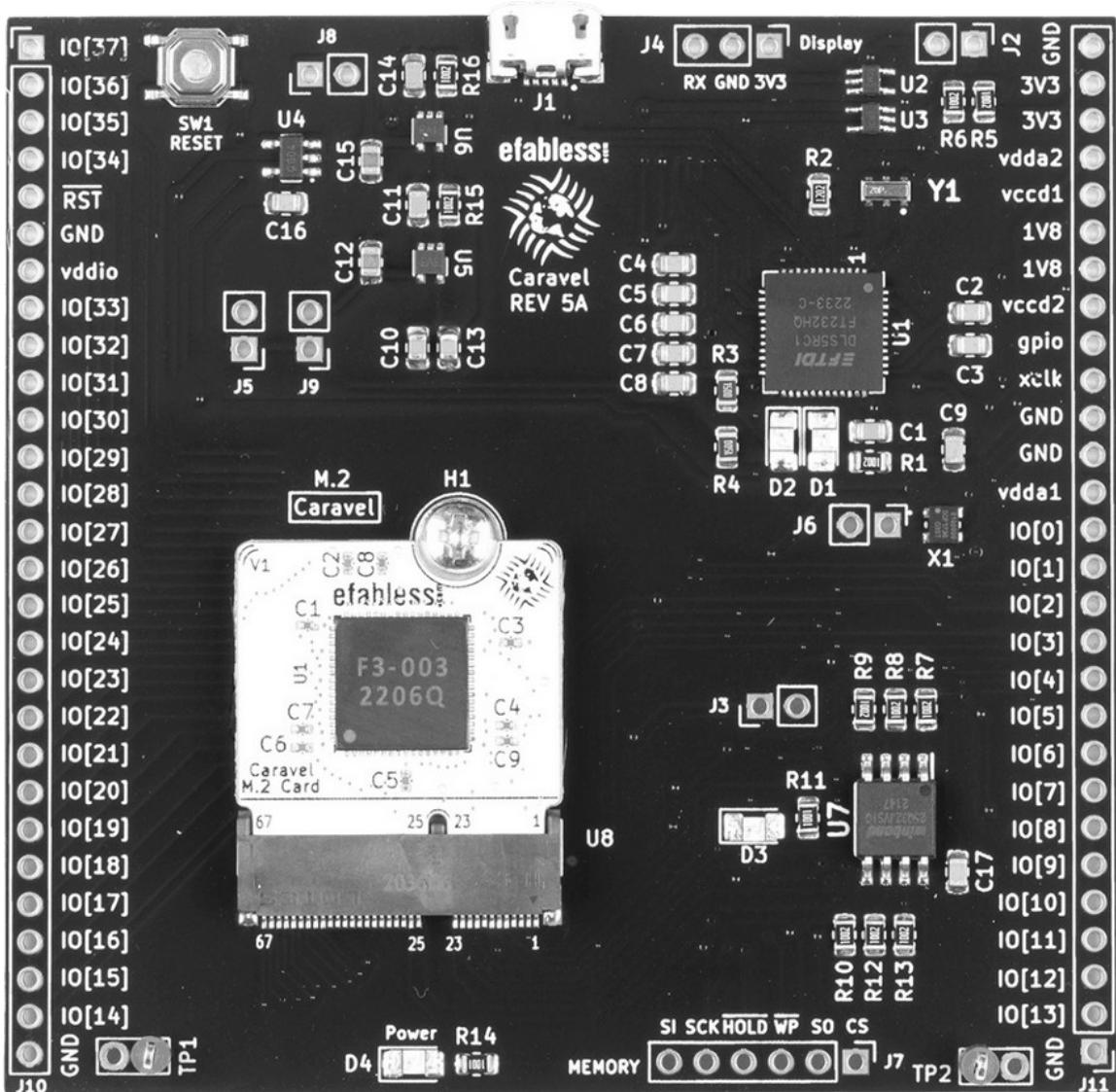


Figura 79 Chip Final (Fuente: Efabless [6])

Una vez finalizado el proyecto se debe esperar al próximo lanzamiento y coordinar con los patrocinadores para conocer la fecha del mismo, para lo cual *Efabless* utiliza *Slack* como canal o vía de comunicación.

El proyecto no sólo fue documentado por escrito, sino también mediante videos, los cuales se encuentran en la sección 8.

PARTE III

RESULTADOS Y CONCLUSIONES

Resultados

Como resultado de este trabajo se pudo obtener la imagen .gds del chip completo, esto se logró en base al trabajo desarrollado en el repositorio del proyecto. El mismo está dedicado al desarrollo en Caravan, es decir, con un gran componente analógico. También se logró testear, desde el Risc-V mediante la compilación del código en C y luego, con Icarus Verilog, la parte digital en Verilog. Despues se añade esa parte al wrapper utilizando OpenLane. La parte analógica se testeó a través de LTSpice y, a continuación, se diseñó en Xschem y finalmente se verifica en Magic. Ademas la parte analógica se añade al wrapper manualmente utilizando Magic. Posteriormente, se observan las imágenes que forman parte del proceso llevado a cabo para el trabajo.

Parte digital con gtkwave:

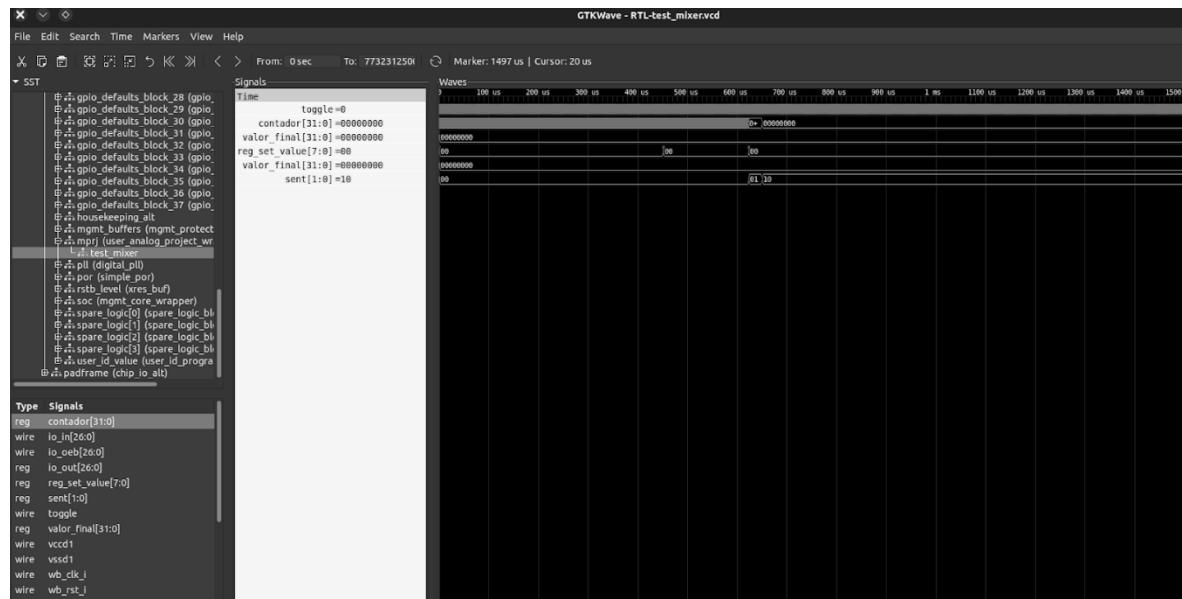


Figura 80 Salida del chip caravan observada en GTKWave (Fuente: Propia)

Schmitt Trigger en LTSpice:

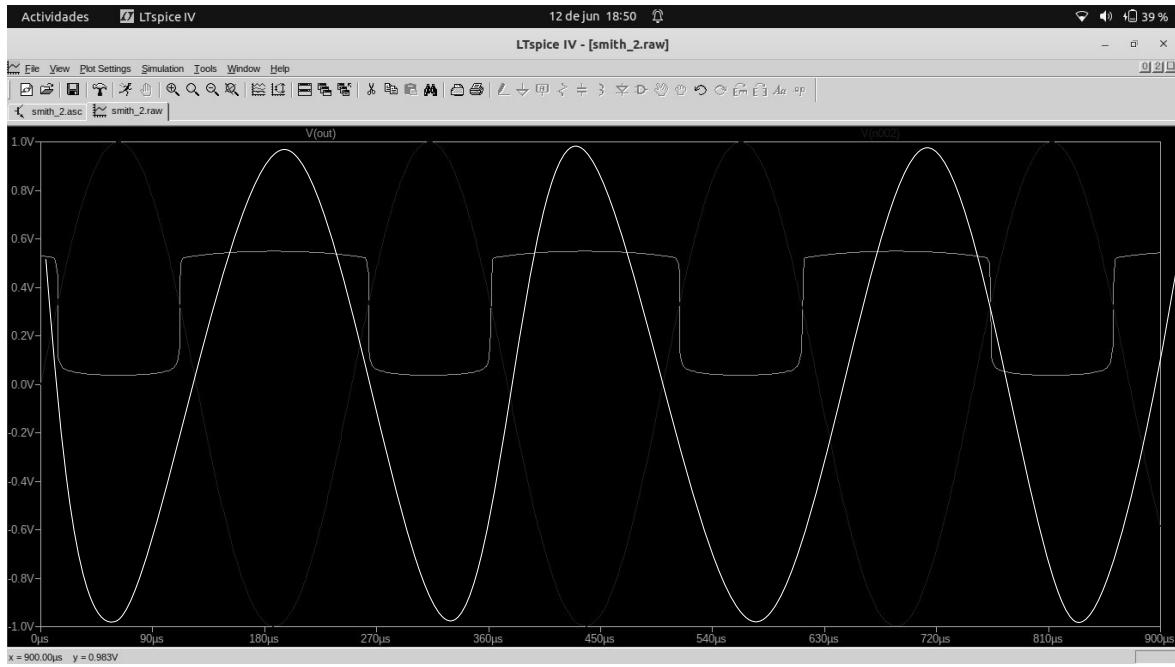


Figura 81 Schmitt Trigger Simulación en LTSPICE (Fuente: Propia)

Mixer en LTSpice:

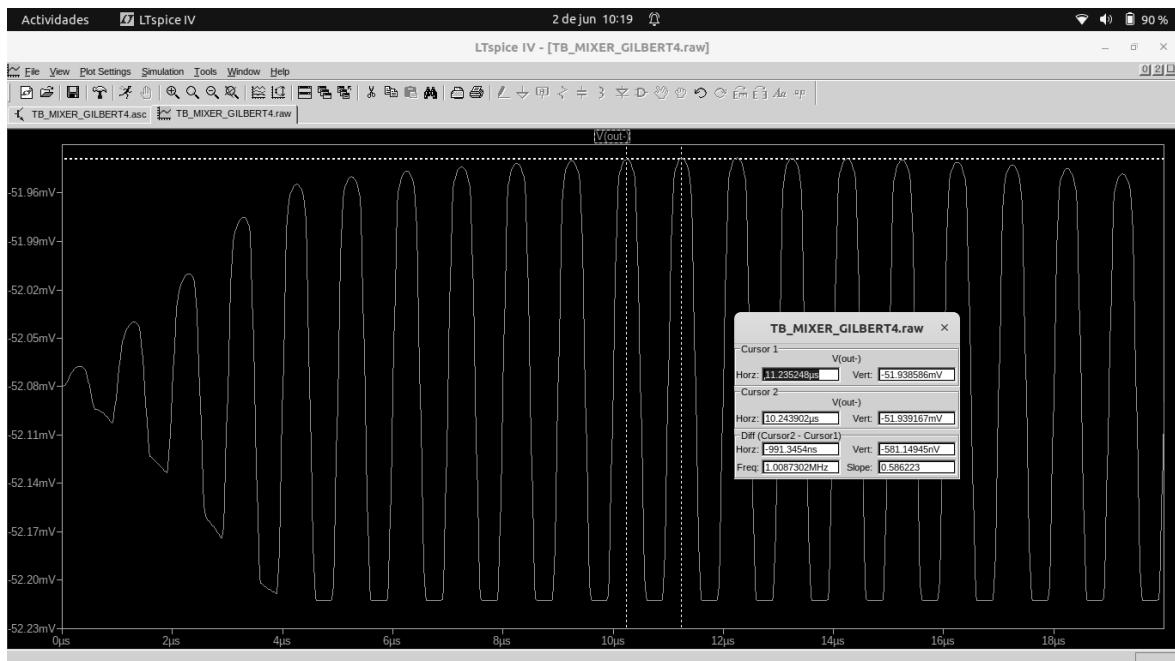


Figura 82 Simulacion en LTSpice del Mixer (Fuente: Propia)

En las imágenes anteriores se puede visualizar:

- En la primera imagen Figura 80 se visualiza la salida de la simulación realizada con archivos verilogs y un archivo en hexa el cual se añadió al modulo del Risc-V para obtener la respuesta del SoC.
- En la segunda imagen Figura 81 se puede observar la salida del Schmitt Trigger, en la cual se visualiza cómo una señal sinuosoidal se transforma en una señal de pulsos cuadrada.
- En la tercera imagen Figura 82 se observa la señal de salida del Mixer o mezclador de señales.

Conclusiones

Efabless ha desempeñado un papel fundamental al proporcionar herramientas de código abierto a la comunidad de diseñadores y entusiastas. Sin embargo, esta misma comunidad ha contribuido significativamente al identificar errores o incompatibilidades y ofrecer soporte. Por lo tanto, al diseñar un chip propio, es crucial prestar atención a posibles incompatibilidades con versiones futuras de PDKs, *Magic*, Caravel, Caravan, *Xschem*, Python y, especialmente, del diseño del *harness* de Caravel.

En este proyecto se presenta un tutorial detallado sobre cómo llevar a cabo un proyecto de sistema en un chip (SoC) utilizando herramientas de código abierto. Este enfoque permite a los estudiantes acceder al diseño de productos de alta calidad, siguiendo pasos similares a los utilizados en la industria del software. Los dispositivos SoC están experimentando un crecimiento significativo debido al aumento en la demanda de chips para aplicaciones específicas. Para el futuro de este proyecto, se espera que las nuevas generaciones de estudiantes puedan aprovechar el repositorio del laboratorio y participar activamente en la comunidad de *Efabless* a través de medios como Slack. Aunque este proyecto inicial enfrenta desafíos, como las funcionalidades de preverificación y la verificación de *gl*, que no se mencionaron, creemos que este inicio servirá de inspiración para las generaciones futuras. La comunidad de *Efabless* es altamente activa, lo que facilita un progreso dinámico con el apoyo de sus creadores.

Se han realizado modificaciones significativas al *harness* de Caravel. Estas modificaciones incluyen un cambio de interés en el procesador RISC-V, pasando de usar Vix a Pico, lo que proporciona mayores prestaciones en comparación con la versión anterior. Al respecto, consideramos hacer especial hincapié en todos los cambios llevados a cabo para adecuar los repositorios disponibles a la idea de este proyecto para poder completar la realización del mismo. Es por ello, que un resultado valioso tras concretar este trabajo es la documentación originada, ya que sienta las bases para continuar y expandir este tipo de ideas.

También es importante destacar que no se pudo cargar el diseño en la plataforma. Aunque hemos utilizado el repositorio de la plataforma para preparar el chip, es necesario ponerlo en cola para el próximo envío espacial. En este caso particular, esto no se realizó, pero se espera que las próximas personas que continúen con este proyecto puedan completar este paso y participar en los desafíos para obtener el chip.

Además, es relevante recordar que este proyecto se ejecutará en una computadora de alto rendimiento, accesible de forma remota para los estudiantes. Esto facilitará el uso ágil del entorno y minimizará problemas de recursos en diferentes computadoras, dado que el consumo aumenta exponencialmente a medida que el proyecto crece.

Es importante tener en cuenta la versatilidad del proyecto de Efabless, con una parte de la comunidad que brinda soporte a repositorios específicos, como el que contiene procesadores y arquitecturas diferentes. Para acceder a la información correspondiente, la comunidad de Efabless cuenta con diversos canales en Slack. Estos canales ofrecen un espacio para discusiones, preguntas y colaboraciones entre los miembros de la comunidad, lo cual también impulsa el desarrollo de proyectos como éste.

Trabajos futuros

Para futuros trabajos, se propone el desarrollo de un dispositivo más amplio utilizando las herramientas y conocimientos adquiridos en este proyecto. El objetivo es facilitar un acceso más rápido a las herramientas de diseño y verificación de chips.

Ya hay estudiantes utilizando este repositorio como base para sus proyectos integradores, combinando diseño analógico y diseño digital. Este trabajo proporciona un buen punto de partida en esta área de investigación.

Además, se puede utilizar cocotb para obtener simulaciones más rápidas, y adaptar las APIs propuestas por Caravel para Caravan.

El avance continuo de las herramientas de código abierto, debido a la interacción constante entre los creadores y los usuarios, permitirá que los proyectos futuros tengan objetivos cada vez más ambiciosos.

Videos relacionados

- Videos referidos al Capítulo 3:
 - Inverter
https://drive.google.com/drive/folders/1fOGG06ml8irXXV_mhZMJbo1x_B_JE7K7
 - Mixer
<https://drive.google.com/drive/folders/1qyeWviYYsk8TQQNxTU5xLzVvSiOHEkJy>
- Videos referidos al Capítulo 4:
 - https://drive.google.com/drive/folders/15A_eIpTgSyXSwGXf-NXc0JcmE1ULQTM5
 - <https://efabless.com/getting-started>
 - <https://efabless.com/demo-design>
 - https://www.youtube.com/@efabless_channel
 - <https://www.youtube.com/watch?v=zJhnmilXGPo&list=PLZuGFJzpFksB57YCxIQ50DPkvNFMpfCXd>
 - https://www.youtube.com/watch?v=MNuoyz_MM-c&t=237s
 - <https://www.youtube.com/watch?v=RPppaGdjbj0>

Referencias

- [1] Caravel. (n.d.). Caravel - Read the docs.io. Efabless Caravel “harness” SoC — Caravel Harness documentation. Accedido el 15/12/2023. <https://caravel-harness.readthedocs.io/en/latest/>
- [2] Caravel. (n.d.). Caravel - Read the Docs.io. Caravel Management SoC - Litex — Caravel Management SoC documentation. Accedido el 15/12/2023. <https://caravel-mgmt-soc-litex.readthedocs.io/en/latest/>
- [3] Edwards, T. (n.d.). RTimothyEdwards/magic: Magic VLSI Layout Tool. GitHub. Accedido el 20/12/2023.
<https://github.com/RTimothyEdwards/magic>
- [4] Edwards, T. (n.d.). RTimothyEdwards/netgen: Netgen complete LVS tool for comparing SPICE or verilog netlists. GitHub. Accedido el 20/12/2023.
<https://github.com/RTimothyEdwards/netgen>
- [5] Efabless. (n.d.). Docker Hub. Docker Hub - Efabless. Accedido el 4/1/2024
<https://hub.docker.com/u/efabless>
- [6] Efabless. (n.d.). Efabless. Efabless. Accedido el 4/1/2024. <https://efabless.com/>
- [7] Efabless. (n.d.). Efabless - Caravel User Project. Efabless - Caravel User Project. Accedido el 16/12/2023.
https://github.com/efabless/caravel_user_project
- [8] Efabless. (n.d.). Efabless - Caravel User Project. GitHub Efabless. Accedido el 16/12/2023.
https://github.com/efabless/caravel_user_project/tree/main/verilog/dv/la_test1
- [9] Efabless. (n.d.). Efabless - Caravel User Project Analog. Efabless - Caravel User Project Analog. Accedido el 16/12/2023. https://github.com/efabless/caravel_user_project_analog
- [10] Efabless. (n.d.). OpenLane The Open-Source Infrastructure Platform for Silicon Development. efabless.com. Accedido el 10/1/2024. <https://efabless.com/openlane>
- [11] GitHub. (n.d.). GNU Toolchain for RISC-V, including GCC. GitHub. Accedido el 10/12/2023.
<https://github.com/riscv-collab/riscv-gnu-toolchain>
- [12] Institute for Integrated Circuits, Johannes Kepler University Linz. (n.d.). Magic VLSI Cheatsheet. GitHub. Accedido el 5/3/2024.
https://github.com/iic-jku/osic-multitool/blob/main/magic-cheatsheet/magic_cheatsheet.pdf
- [13] Instituto Tecnológico de Costa Rica. (n.d.). Instituto Tecnológico de Costa Rica. Repositorio TEC. Accedido el 13/12/2023. <https://repositoriotec.tec.ac.cr/bitstream/handle/2238/7066/diseno-mezclador-frecuencias-topologia-gilbert.pdf?sequence=1&isAllowed=y>
- [14] Klayout. (n.d.). Klayout - intro. Klayout. Accedido el 14/1/2024
<https://www.klayout.de/intro.html>
- [15] LCSR-2024. (n.d.). Test Mixer. Test Mixer. Accedido el 3/5/2024. https://github.com/LCSR-2024/test_mixer
- [16] LCSR-2024. (n.d.). Test Mixer. GitHub. Accedido el 3/5/2024.
https://github.com/LCSR-2024/test_mixer/tree/main/openlane/user_analog_project_wrapper
- [17] MB, D., & Hegde, A. (2024, March 21). What Is The Antenna Effect in VLSI? ChipEdge. Accedido el 10/5/2024.
<https://chipedge.com/what-is-the-antenna-effect-in-vlsi/>
- [18] Mukherjee, A. (n.d.). PHYSICAL VERIFICATION USING Sky130. GitHub. Accedido el 15/2/2024.
<https://github.com/Avnish21/VSD-Physical-Verification-Using-Sky130/blob/main/README.md>
- [19] Open Circuit Design. (2017, April 25). Netgen 1.5. Open Circuit Design. Accedido el 15/2/2024.
<http://opencircuitdesign.com/netgen/>
- [20] Open Circuit Design. (2024, March 10). Magic VLSI Layout Tool Version 8. Open Circuit Design. Accedido el 15/2/2024. <http://opencircuitdesign.com/magic/>
- [21] OpenLane. (n.d.). OpenLane - ReadTheDocs.io. OpenLane. Accedido el 20/12/2023.
<https://openlane.readthedocs.io/en/latest/index.html>

- [22] RISC-V. (n.d.). RISC-V. RISC-V.org. Accedido el 17/12/2023. <https://riscv.org/>
- [23] SkyWater. (n.d.). SkyWater - Read the Docs.io. Welcome to SkyWater SKY130 PDK's documentation! — SkyWater SKY130 PDK 0.0.0-369-g7198cf6 documentation. Accedido el 5/3/2024.
<https://skywater-pdk.readthedocs.io/en/main/index.html>
- [24] SkyWater Technology Foundry. (n.d.). Biblioteca y archivos PDK 130 nm. GitHub - SkyWater-pdk. Accedido el 7/2/2024. <https://github.com/google/skywater-pdk>
- [25] Source Forge. (n.d.). GTKWave. GTKWave. Accedido el 8/2/2024.
<https://gtkwave.sourceforge.net/>
- [26] Source Forge. (n.d.). Xschem Sky130 Integration. Tutorial Xschem Sky130. Accedido el 7/2/2024.
https://xschem.sourceforge.io/stefan/xschem_man/tutorial_xschem_sky130.html
- [27] Source Forge. (n.d.). Xschem - Source Forge.io. Xschem - Source Forge.io. Accedido el 3/1/2024.
<https://xschem.sourceforge.io/stefan/index.html>
- [28] Source Forge. (n.d.). XSCHM TUTORIAL. XSchem. Accedido el 4/2/2024.
https://xschem.sourceforge.io/stefan/xschem_man/install_xschem.html
- [29] Texas Instruments. (n.d.). ElectrostaticDischarge (ESD). Texas Instruments - ElectrostaticDischarge(ESD). Accedido el 5/4/2024.
https://www.ti.com/lit/an/ssya010a/ssya010a.pdf?ts=1717462507391&ref_url=https%253A%252F%252Fwww.ti.com%252Fsite%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US
- [30] Venn, M. (n.d.). Terminology. Zero to ASIC Course. Accedido el 8/1/2024.
<https://www.zerotoasiccourse.com/terminology/>
- [31] Wikipedia. (n.d.). Cable harness. Wikipedia. Accedido el 5/4/2024.
https://en.wikipedia.org/wiki/Cable_harness
- [32] Wikipedia. (n.d.). Gilbert cell. Wikipedia. Accedido el 5/4/2024.
https://en.wikipedia.org/wiki/Gilbert_cell
- [33] Wikipedia. (n.d.). Logic analyzer. Wikipedia. Accedido el 5/4/2024.
https://en.wikipedia.org/wiki/Logic_analyzer
- [34] Wikipedia. (n.d.). Process Design Kit. Wikipedia - Process Design Kit. Accedido el 5/4/2024.
https://en.wikipedia.org/wiki/Process_design_kit
- [35] Wikipedia. (n.d.). Register-transfer level. Wikipedia. Accedido el 5/4/2024.
https://en.wikipedia.org/wiki/Register-transfer_level
- [36] Wikipedia. (n.d.). System on a Chip. Wikipedia. Accedido el 5/4/2024.
https://en.wikipedia.org/wiki/System_on_a_chip
- [37] Wikipedia. (n.d.). Verilog. Wikipedia. Accedido el 5/4/2024.
<https://es.wikipedia.org/wiki/Verilog>
- [38] Williams, S. (n.d.). Icarus Verilog — Icarus Verilog documentation. GitHub Pages. Accedido el 5/4/2024.
<https://steveicarus.github.io/iverilog/>
- [39] YosisHQ. (n.d.). YosysHQ/picorv32: PicoRV32 - A Size-Optimized RISC-V CPU. GitHub. Accedido el 20/12/2023. <https://github.com/YosysHQ/picorv32>
- [40] Institute for Integrated Circuits, Johannes Kepler University Linz. (n.d.). IIC-OSIC-TOOLS. GitHub. Accedido el 3/4/2024. <https://github.com/iic-jku/IIC-OSIC-TOOLS>
- [41] Wikipedia. (n.d.). Schmitt trigger. Wikipedia. Accedido el 11/4/2024.
https://en.wikipedia.org/wiki/Schmitt_trigger
- [42] Efabless. (n.d.). Make Your Own Chips for Free. efabless.com. Accedido el 25/2/2024.
https://efabless.com/open_shuttle_program
- [43] Efabless - Housekeeping-SPI. Efabless Caravel "harness" SoC — CIIC Harness documentation. Read the Docs. Accedido el 26/2/2024.
<https://caravel-docs.readthedocs.io/en/wavedrom-regis/>
- [44] Wikipedia. (n.d.). Troubleshooting. Wikipedia. Accedido el 6/4/2024.
<https://en.wikipedia.org/wiki/Troubleshooting>
- [45] Efabless. (n.d.). Tiny Tapeout. Tiny Tapeout: Quicker, easier and cheaper to make your own chip! Accedido el 6/4/2024. <https://www.tinytapeout.com/>

- [46] Primer seminario sobre diseño de circuitos integrados con software libre Accedido el 17/1/2024. <https://www.youtube.com/watch?v=OgAELqhwvPI&t=606s>
- [47] Efabless. (n.d.). *Housekeeping SPI — Caravel Harness documentation*. Caravel Harness. Accedido el 12/3/2024.
<https://caravel-harness.readthedocs.io/en/latest/housekeeping-spi.html#housekeeping-spi-modes>
- [48] Venn, M. (n.d.). *0 to ASIC demo*. Google Drive. Accedido el 15/1/2024.
<https://docs.google.com/presentation/d/14npvuiGxsS3C-yo2vOsbZpNKGehwTtIYRFN6k3ZVR4M/edit#slide=id.p>
- [49] Analog Devices. (n.d.). *LTspice Information Center*. Analog Devices. Accedido el 20/12/2023.
<https://www.analog.com/en/resources/design-tools-and-calculators/ltpice-simulator.html>
- [50] Wikipedia. (n.d.). *Desmodulación*. Wikipedia. Accedido el 20/12/2023.
<https://es.wikipedia.org/wiki/Desmodulaci%C3%B3n>
- [51] Open Circuit Design. (2020, March 7). *Magic-8.3 Command Reference*. Magic-8.3 Command Reference. Accedido el 7/3/2024. <http://opencircuitdesign.com/magic/commandref/drc.html>
- [52] Wikipedia. (n.d.). *Layout Versus Schematic*. Wikipedia. Accedido el 8/3/2024.
https://en.wikipedia.org/wiki/Layout_Versus_Schematic
- [53] Wikipedia. (n.d.). *Test bench*. Wikipedia. Accedido el 12/3/2024.
https://en.wikipedia.org/wiki/Test_bench
- [54] Wikipedia. (n.d.). *Wafer (electronics)*. Wikipedia. Accedido el 7/4/2024.
[https://en.wikipedia.org/wiki/Wafer_\(electronics\)](https://en.wikipedia.org/wiki/Wafer_(electronics))
- [55] Efabless Caravel GitHub. Accedido el 27/12/2023
<https://github.com/efabless/caravel/tree/main/docs>

Anexos

Anexo A

Caravel vs. Caravan

Caravel

El proyecto de usuario del chip Caravel puede utilizar los pines GPIO como señales analógicas, y este es el método preferido, ya que los pines GPIO tienen protección ESD.

Las restricciones en el uso de pines GPIO para analógico son las siguientes:

1. El rango de voltaje de la señal analógica debe estar entre **VSSIO** y **VDDIO**. En la placa de demostración enviada con Caravel, **VDDIO** se configurará en **3,3V** de un regulador de voltaje externo. Sin embargo, **VDDIO** puede ser cualquier donde está en el rango de **1,8 V a 5,5 V**.
2. El rango de frecuencia de los pads **GPIO** es de **0 a 60MHz**.
3. Las señales analógicas deben conectarse a los pines **analog_io** del contenedor del proyecto del usuario. Este pin se conecta al pad a través de una resistencia de 120 ohmios, para protección ESD. Sin embargo, se recomienda colocar un diodo cerca del terminal en el circuito del proyecto del usuario para cualquier señal de entrada que de otra manera no esté conectada a la difusión, para protección ESD adicional. Esta resistencia debe incluirse en las simulaciones a nivel de sistema.
4. Cuando se conecta una señal analógica a un pad GPIO, los buffers de entrada y salida del pad **GPIO** deben desactivarse, configurando la configuración GPIO en **GPIO_MODE_USER_STD_ANALOG** (ver `defs.h`). Idealmente, los buffers deberían desactivarse de forma predeterminada al encender el chip, lo que se hace aplicando la misma configuración en el archivo `userDefines.v`. Esto garantiza que los buffers digitales nunca se activarán para esos GPIO.
5. Las señales analógicas no pueden usar GPIO de 0 a 6 o GPIO 36 y 37. Esto evita que las señales críticas como los pines de modo de depuración, housekeeping SPI, and flash QSPI flash no puedan funcionar debido a la presencia de una señal analógica constante. en la pad. Por lo tanto, hay hasta 28 pines GPIO que se pueden utilizar para señalización analógica. Tenga en cuenta que los nombres de las señales **analog_io[27:0]** se desplazan en relación con los nombres de los pads **GPIO (mprj_io)**. Entonces **analog_io[0]** se conecta a **mprj_io[7]**, y así sucesivamente hasta **analog_io[27]** que se conecta a **mprj_io[34]**.

Caravan

En el caso de que se necesite un pin que requiera voltajes superiores a **5,5 V**, inferiores a **0,0 V**, tenga una frecuencia superior a **60 MHz** o no pueda tolerar la resistencia de la serie de 120 ohmios, entonces el chip Caravan proporciona **11 pads** que son conexiones directas. desde el core hasta el pad. Estos pads reemplazan a los pads **mprj_io[14]** a **mprj_io[24]** y se extienden por la parte superior del **padframe**.

WARNING:

Los pads analógicos NO brindan protección ESD, porque el uso de los pads es abierto y los requisitos son diferentes para la protección de, por ejemplo, alto voltaje, voltaje negativo y frecuencia muy alta.

Todos los pads distintos de los 11 que tienen conexiones directas desde el proyecto del usuario al pad son los mismos pads que se usan en Caravel, por lo que hay hasta 17 pines GPIO que se pueden usar para señalización analógica bajo las mismas restricciones que se indicaron anteriormente para Caravel. Estos pines reciben un nombre diferente en Caravan, que es **user_gpio_analog[17:0]**.

Debido a que los circuitos analógicos a menudo funcionan a 3,3 V, los circuitos digitales para controlar dichos circuitos deben utilizar la biblioteca de celdas estándar digital **HVL** para compatibilidad con 3,3 V. Estos circuitos se pueden conectar directamente a las entradas de E/S si se utilizan los pines **io_in_3v3[26:0]**. Estos pines son copias de las entradas digitales del pin GPIO en el dominio de 3,3 V. Sin embargo, tenga en cuenta que no hay una salida GPIO correspondiente en el dominio de 3,3 V. Las salidas de 3,3 V deben convertirse en nivel al dominio de 1,8 V utilizando, por ejemplo, la celda **sky130_fd_sc_hvl_lsbuhv2lv_1**, antes de conectarse a **io_out** o **io_oeb**.

La correspondencia completa entre los pines mprj_io y las conexiones internas se muestra a continuación, copiada de **caravel_user_project_analog/verilog/rtl/user_analog_proj_example.v**

Caravan signal connections to I/O pins:

I/O pin, user_project_digital_connection, user_project_analog_connection, optional_power clamp_connection
descargar electrostática (esd)

I/O pin	user project digital connection	user project analog connection	optional power clamp connection
mpkj_io[37]	io_in/out/oeb/in_3v3[26]	---	---
mpkj_io[36]	io_in/out/oeb/in_3v3[25]	---	---
mpkj_io[35]	io_in/out/oeb/in_3v3[24]	gpio_analog/noesd[17]	---
mpkj_io[34]	io_in/out/oeb/in_3v3[23]	gpio_analog/noesd[16]	---
mpkj_io[33]	io_in/out/oeb/in_3v3[22]	gpio_analog/noesd[15]	---
mpkj_io[32]	io_in/out/oeb/in_3v3[21]	gpio_analog/noesd[14]	---
mpkj_io[31]	io_in/out/oeb/in_3v3[20]	gpio_analog/noesd[13]	---
mpkj_io[30]	io_in/out/oeb/in_3v3[19]	gpio_analog/noesd[12]	---

l			
mpkj_io[29] l	io_in/out/oeb/in_3v3[18]	gpio_analog/noesd[11]	---
mpkj_io[28] l	io_in/out/oeb/in_3v3[17]	gpio_analog/noesd[10]	---
mpkj_io[27] l	io_in/out/oeb/in_3v3[16]	gpio_analog/noesd[9]	---
mpkj_io[26] l	io_in/out/oeb/in_3v3[15]	gpio_analog/noesd[8]	---
mpkj_io[25] l	io_in/out/oeb/in_3v3[14]	gpio_analog/noesd[7]	---
mpkj_io[24] l	---	user_analog[10]	---
mpkj_io[23] l	---	user_analog[9]	---
mpkj_io[22] l	---	user_analog[8]	---
mpkj_io[21] l	---	user_analog[7]	---
mpkj_io[20] l	---	user_analog[6]	clamp_high/low[2]
mpkj_io[19] l	---	user_analog[5]	clamp_high/low[1]
mpkj_io[18] l	---	user_analog[4]	clamp_high/low[0]
mpkj_io[17] l	---	user_analog[3]	---
mpkj_io[16] l	---	user_analog[2]	---
mpkj_io[15] l	---	user_analog[1]	---
mpkj_io[14] l	---	user_analog[0]	---
mpkj_io[13] l	io_in/out/oeb/in_3v3[13]	gpio_analog/noesd[6]	---
mpkj_io[12] l	io_in/out/oeb/in_3v3[12]	gpio_analog/noesd[5]	---
mpkj_io[11] l	io_in/out/oeb/in_3v3[11]	gpio_analog/noesd[4]	---

mprj_io[10]	io_in/out/oeb/in_3v3[10]	gpio_analog/noesd[3]	---
mprj_io[9]	io_in/out/oeb/in_3v3[9]	gpio_analog/noesd[2]	---
mprj_io[8]	io_in/out/oeb/in_3v3[8]	gpio_analog/noesd[1]	---
mprj_io[7]	io_in/out/oeb/in_3v3[7]	gpio_analog/noesd[0]	---
mprj_io[6]	io_in/out/oeb/in_3v3[6]	---	---
mprj_io[5]	io_in/out/oeb/in_3v3[5]	---	---
mprj_io[4]	io_in/out/oeb/in_3v3[4]	---	---
mprj_io[3]	io_in/out/oeb/in_3v3[3]	---	---
mprj_io[2]	io_in/out/oeb/in_3v3[2]	---	---
mprj_io[1]	io_in/out/oeb/in_3v3[1]	---	---
mprj_io[0]	io_in/out/oeb/in_3v3[0]	---	---

Tabla 7 Pines Caravan (Fuente: Repositorio [55])

Tres de las once conexiones analógicas directas del chip Caravel van a pads que tienen **clamps** de voltaje debajo. Una clamp de voltaje es un circuito que protege contra eventos de ESD al detectar un rápido aumento de voltaje en una plataforma de suministro de energía y habilitar un interruptor que cortocircuita el suministro de energía a una tierra cercana, reduciendo el pico de voltaje del evento y desviando la corriente a través de un camino cercano a los pads y lejos de circuitos sensibles. Cada **clamp** tiene una conexión positiva (**clamp_high**) y una conexión negativa (**clamp_low**). Ninguno de estos pines está conectado de forma predeterminada. El pin **clamp_high** debe conectarse a una fuente de alimentación, preferiblemente la que está conectada al pad directamente encima de ella. El pin **clamp_low** debe conectarse a un retorno a tierra. Debido a la naturaleza del **user_project_wrapper** como un módulo directo, la ruta de derivación actual será mucho más larga que la ruta corta ideal. Asegúrese de que este camino sea lo más ancho posible.

El circuito de **clamp** es un tipo de **clamp** de alto voltaje diseñado para funcionar con una fuente de alimentación igual a **VDDIO**, o nominalmente **3,3 V** para la placa de demostración (y de lo contrario, dentro del rango de **1,8 V** a **5,5 V**). Debido a que el rango de voltaje de **E/S** incluye **1,8 V**, este **clamp** funcionará a **1,8 V**. Sin embargo, proporciona la mejor protección ESD para suministros de **3,3 V**. No debe utilizarse con ningún suministro superior a **VDDIO**.

Debido a la gran cantidad de circuitos (los clamp) directamente debajo del pad, los tres pads con las clamps no están diseñados para uso a alta velocidad. Estos pads se utilizan mejor para entradas de fuente de alimentación adicionales para el chip analógico. Las tres pads que contienen clamps también están diseñadas para proporcionar la mayor cantidad de corriente, hasta **265 mA** para cada pad (ver más abajo). La conexión de pin en el usuario. El límite del **wrapper** del proyecto consta de dos puertos, de 25 um de ancho, cada uno de los cuales comprende una pila de metales 3, 4 y 5. Para obtener la máxima corriente a través del pad sin crear problemas de electromigración, conéctese a los tres metales en ambos puertos. Se espera que el enrutamiento del suministro de energía en Caravan se realice manualmente. Permite menos de 1,5 mA por micrón de ancho en metal3 y metal4 para satisfacer las reglas de electromigración, y menos de 2,3 mA por micrón en metal5. La corriente máxima por panel de alimentación dedicado es

$$((25\mu m * 2) * (1.5 + 1.5 + 2.3)) = 265mA$$

Debido a la forma el circuito **wrapper** se "coloca" en el chip el harness Caravel o Caravan, se debe realizar una verificación de continuidad en el momento de la cinta para garantizar que los pins del **wrapper** se conecten correctamente a las ubicaciones correspondientes en el harness. Esto requiere que cada pin del diseño esté en una red única. Debido a este requisito, los pins en el **user_wrapper** no pueden estar en cortocircuito; de lo contrario, solo uno de los pins en cortocircuito puede representarse como un puerto de subcircuito en la lista de red SPICE extraída. Debido a que el cortocircuito entre pines es un caso de uso probable, especialmente en diseños analógicos, el procedimiento recomendado al conectar pins entre sí es colocar una "resistencia metálica" delante de la conexión del pin en todas las conexiones que no sean la principal. La mayoría de las conexiones de los pins son de metal3, por lo que se prefiere una resistencia de metal3. La "resistencia metálica" en el diseño es una capa de identificación, no una capa de máscara, por lo que el metal debe ser continuo hasta la conexión, con la capa identificadora de resistencia abarcando todo el ancho de la conexión. Por ejemplo, es posible que el usuario desee unir **VDDA1** y **VDDA2** para duplicar la capacidad actual de la fuente de alimentación del dominio de **3,3 V**. El usuario debe enrutar un bus de alimentación y conectarlo directamente al pin **VDDA1**. Luego, se puede hacer una ruta al pin **VDDA2**, pero debe pasar a través de una resistencia metal3 antes de realizar la conexión al pin. Cualquier resistencia de este tipo debe representarse como un dispositivo en un dibujo esquemático para que el diseño pase LVS.

Asignación de fuentes de alimentación

Como se mencionó anteriormente, las fuentes de alimentación se pueden conectar entre sí si es necesario. Estas son las fuentes de alimentación disponibles:

- VCCD1/VSSD1 : User domain 1, 1.8V power
- VCCD2/VSSD2 : User domain 2, 1.8V power
- VDDA1/VSSA1 : User domain 1, 3.3V power
- VDDA2/VSSA2 : User domain 2, 3.3V power
- VDDIO/VSSIO : Management domain, 3.3V power supply to padframe
- VCCD/VSSD : Management domain, 1.8V power supply to padframe and SoC

Todas las conexiones de pad que son pins de chip están en el dominio VDDIO. Todas las conexiones de los pads de bajo voltaje al núcleo del chip están en el dominio VCCD, y los únicos pins de alto voltaje (io_in_3v3; ver arriba) conectados al contenedor del proyecto del usuario están en el dominio VDDIO. Cualquiera de las fuentes de alimentación del usuario que se encuentren en el mismo dominio de energía se pueden conectar entre sí para proporcionar capacidad de corriente adicional. Por lo tanto, **VCCD1** y **VCCD2** pueden conectarse entre sí (además de conectarse entre sí **VSSD1** y **VSSD2**); y **VDDA1** y **VDDA2** pueden conectarse juntos (junto con **VSSA1** y **VSSA2**). El proyecto del usuario no tiene acceso directo a los dominios de energía del área de administración, incluidos los suministros que controlan las E/S del padframe.

Configuración de encendido del pin GPIO

El diseño Caravel para MPW-two incluye una nueva característica que permite al diseñador del área de proyecto del usuario especificar cómo serán configurados los pins GPIO al encender.

Para MPW-one, todos los pins GPIO del área de usuario (mprj_io[0] a mprj_io[37]) tenían un configuración fija al encender con acceso de administración y una entrada función en mprj_io[37:6]. mprj_io[5:1] pertenece al SPI de **housekeeping** y están configurados para uso SPI; mprj_io[0] es para depurar el sistema pero fue sin usar en MPW-one. El propósito de esta configuración es mantener la corriente que genera el chip en las salidas hasta después del encendido. Es responsabilidad del administrador del programa flash SoC configurar los pins GPIO para la función que necesita el proyecto del usuario.

Hubo dos problemas con esta configuración:

(1) La configuración del GPIO depende completamente del SoC de administración y (2) es posible que un proyecto de usuario se diseñe de manera que el proyecto de usuario intente comenzar a comunicarse con el mundo exterior antes de que el SoC de administración haya configurado el GPIO, y puede terminar en un estado estancado antes de que pueda configurarse.

Para hacer el sistema más flexible, el nuevo diseño permite la configuración de los GPIOs al encenderse se configurarán de forma personalizada. La configuración es descrita en el archivo "**userDefines.v**". Un conjunto predeterminado de definiciones correspondientes a la configuración original de MPW-one se proporciona con el repositorio de caravel en el archivo **verilog/rtl/userDefines.v**.

El archivo "**userDefines.v**" contiene un conjunto de definiciones Verilog en la forma:

```
'definir USER_CONFIG_GPIO_<n>_INIT <valor>
```

donde <n> es el índice **GPIO**; por ejemplo, **USER_CONFIG_GPIO_5** corresponde a fijar **mprj_io[5]**. El <valor> predeterminado es un valor de 13 bits que es el bit configuración de la configuración GPIO. Debido a que el valor del bit sin procesar es una forma inconveniente, se han añadido una serie de definiciones Verilog adicionales realizadas en la parte superior del archivo. Estas definiciones tienen los mismos nombres que los del fichero "**defs.h**" incluido en los programas de gestión **SoC** en C.

Estos son los valores que probablemente sean de interés para el proyecto del usuario. diseñador, y son los siguientes:

GPIO_MODE_MGMT_STD_INPUT_NOPULL (13'h0403):

El SoC de gestión tiene acceso al pin GPIO.

El pin es una entrada (salida deshabilitada) no tiene pull-up y pull-down.

GPIO_MODE_MGMT_STD_INPUT_PULLUP (13'h0c03):

El SoC de gestión tiene acceso al pin GPIO.

El pin es una entrada (salida deshabilitada) y tiene un pull-up de 5kOhm.

GPIO_MODE_MGMT_STD_OUTPUT (13'h1809):

El SoC de gestión tiene acceso al pin GPIO.

El pin es una salida (entrada deshabilitada).

GPIO_MODE_MGMT_STD_BIDIRECTIONAL (13'h1801):

El SoC de gestión tiene acceso al pin GPIO.

El pin es una salida o una entrada, según el estado del pin de habilitación de salida.

Sólo los pines GPIO 0 (depuración), 1 (limpieza) SPI SDO), 35 (SPI maestro SDO), 36 (flash IO2) y 37 (flash IO3) se pueden configurar como bidireccionales y la función bidireccional solo lo utiliza la función del sistema asociada (depuración, mantenimiento) SPI o maestro SPI).

GPIO_MODE_MGMT_STD_ANALOG (13'h000b):

El SoC de gestión tiene acceso al pin GPIO.

Todos los buffers digitales (entrada y salida) están desactivados. No hay diferencia efectiva entre el control de usuario o de gestión en este caso. Sólo los proyectos de usuario pueden suministrar señales analógicas al GPIO pads.

GPIO_MODE_USER_STD_INPUT_NOPULL (13'h0402):

El proyecto de usuario tiene acceso al pin GPIO.

El pin es una entrada (salida deshabilitada) y no tiene pull-up ni pull-down.

GPIO_MODE_USER_STD_INPUT_PULLDOWN (13'h0802):

El proyecto de usuario tiene acceso al pin GPIO.

El pin es una entrada (salida deshabilitada) y tiene un menú desplegable de 5kOhm.

GPIO_MODE_USER_STD_INPUT_PULLUP (13'h0c02):

El proyecto de usuario tiene acceso al pin GPIO.

El pin es una entrada (salida deshabilitada) y tiene un pull-up de 5kOhm.

GPIO_MODE_USER_STD_OUTPUT (13'h1808):

El proyecto de usuario tiene acceso al pin GPIO.

El pin es una salida (entrada deshabilitada).

GPIO_MODE_USER_STD_BIDIRECTIONAL (13'h1800):

El proyecto de usuario tiene acceso al pin GPIO.

El pin es bidireccional. La entrada siempre está habilitada y la salida está habilitado si el pin OEB (barra de habilitación de salida) correspondiente está reducido por el proyecto del usuario.

GPIO_MODE_USER_STD_OUT_MONITORED (13'h1802):

El proyecto de usuario tiene acceso al pin GPIO.

El pin es bidireccional (ver modo bidireccional, arriba).

El valor del pin se copia al SoC de gestión para propósitos de monitoreo de señal (es decir, el pin simultáneamente actúa como el modo **MGMT_STD_INPUT_NOPULL** como se ve desde la SoC de gestión).

GPIO_MODE_USER_STD_ANALOG (13'h000a):

El proyecto de usuario tiene acceso al pin GPIO.

Tanto el buffer de entrada como el de salida están deshabilitados. Si el proyecto del usuario se conecta a una señal analógica a este pad, aparecerá sin búfer en el pad.

Los índices GPIO 0 a 5 no están representados en este archivo, porque el diseño de Caravel requiere que la función de depuración y la función SPI de mantenimiento sea accesible durante el encendido inicial y mientras se mantiene el SoC de administración en reinicio. Esto permite que el **housekeeping** acceda al reinicio completo del chip y los modos de programación de paso, de modo que la placa de demostración no pueda ser "bloqueado" accidentalmente escribiendo un programa que impida el sistema funcione y evita que el SPI de limpieza o las funciones de depuración de ser

accedido. Si desea que el proyecto de usuario se ejecute sin configuración desde el programa de gestión de SoC, deberá evitar el uso de los pines GPIO 0 a 5. Si necesita utilizar los pines 0 a 5, tendrán que ser configurados por el programa de gestión SoC.

La configuración predeterminada para todos los pines GPIO es "**GPIO_MODE_MGMT_STD_INPUT_NOPULL**", correspondiente a un pad que está bajo el control del SoC de gestión y está configurado como entrada, con el búfer de salida deshabilitado.

Para establecer diferentes valores predeterminados, copie el archivo "**userDefines.v**" al espacio de usuario del proyecto y colóquelo en el directorio **verilog/rtl/**. Entonces se debe cambiar la definición de cada uno de los pines GPIO para corresponder al GPIO configuración que el proyecto necesita al inicio.

La configuración en "**userDefines.v**" es suficiente para verilog full-chip simulación. Los cambios reales en el diseño se realizan en el momento de la grabación, cuando se ensambla el chip Caravel. El contenido de "**userDefines.v**" es utilizado para programar vía el diseño de bloque predeterminado de GPIO. En el diseño final y GDS se refleja esta definición de configuración.

Anexo B

postlayout.spice

```
* NGSPICE file created from Mixer.ext - technology: sky130A

.subckt sky130_fd_pr_nfet_01v8_Y6SADL a_n210_n1149# a_n50_n1063#
a_50_n975# a_n108_n975#
X0 a_50_n975# a_n50_n1063# a_n108_n975# a_n210_n1149#
sky130_fd_pr_nfet_01v8 ad=2.8275 pd=20.08 as=2.8275 ps=20.08 w=9.75 l=0.5
**devattr s=113100,4016 d=113100,4016
C0 a_50_n975# a_n50_n1063# 0.062988f
C1 a_n108_n975# a_n50_n1063# 0.064897f
C2 a_50_n975# a_n108_n975# 0.336849f
C3 a_50_n975# a_n210_n1149# 0.819905f
C4 a_n108_n975# a_n210_n1149# 0.611795f
C5 a_n50_n1063# a_n210_n1149# 0.48776f
.ends

.subckt sky130_fd_pr_nfet_01v8_Y6FLEL a_n210_n1149# a_n50_n1063#
a_50_n975# a_n108_n975#
X0 a_50_n975# a_n50_n1063# a_n108_n975# a_n210_n1149#
sky130_fd_pr_nfet_01v8 ad=2.8275 pd=20.08 as=2.8275 ps=20.08 w=9.75 l=0.5
**devattr s=113100,4016 d=113100,4016
C0 a_50_n975# a_n50_n1063# 0.064897f
C1 a_n108_n975# a_n50_n1063# 0.062988f
C2 a_50_n975# a_n108_n975# 0.336849f
C3 a_50_n975# a_n210_n1149# 0.611795f
C4 a_n108_n975# a_n210_n1149# 0.821788f
C5 a_n50_n1063# a_n210_n1149# 0.48776f
.ends

.subckt sky130_fd_pr_nfet_01v8_PZKCVB a_n50_n1038# a_50_n950#
a_n108_n950# a_n210_n1124#
X0 a_50_n950# a_n50_n1038# a_n108_n950# a_n210_n1124#
sky130_fd_pr_nfet_01v8 ad=2.755 pd=19.58 as=2.755 ps=19.58 w=9.5 l=0.5
**devattr s=110200,3916 d=110200,3916
C0 a_50_n950# a_n50_n1038# 0.063286f
C1 a_n108_n950# a_n50_n1038# 0.109746f
C2 a_50_n950# a_n108_n950# 0.578324f
C3 a_50_n950# a_n210_n1124# 0.783383f
C4 a_n108_n950# a_n210_n1124# 1.06096f
C5 a_n50_n1038# a_n210_n1124# 0.512003f
.ends

.subckt sky130_fd_pr_nfet_01v8_PMG9RB a_n50_n1038# a_50_n950#
a_n108_n950# a_n210_n1124#
X0 a_50_n950# a_n50_n1038# a_n108_n950# a_n210_n1124#
sky130_fd_pr_nfet_01v8 ad=2.755 pd=19.58 as=2.755 ps=19.58 w=9.5 l=0.5
**devattr s=110200,3916 d=110200,3916
C0 a_50_n950# a_n50_n1038# 0.034295f
```

```

C1 a_n108_n950# a_n50_n1038# 0.05164f
C2 a_50_n950# a_n108_n950# 0.418488f
C3 a_50_n950# a_n210_n1124# 0.656996f
C4 a_n108_n950# a_n210_n1124# 0.757556f
C5 a_n50_n1038# a_n210_n1124# 0.518551f
.ends

.subckt sky130_fd_pr_nfet_01v8_8XPPYK a_n123_n1515# a_n65_n1570#
a_65_n1515# a_n225_n1689#
X0 a_65_n1515# a_n65_n1570# a_n123_n1515# a_n225_n1689#
sky130_fd_pr_nfet_01v8 ad=4.3935 pd=30.88 as=4.3935 ps=30.88 w=15.15
l=0.65
**devattr s=175740,6176 d=175740,6176
C0 a_65_n1515# a_n65_n1570# 0.192222f
C1 a_n123_n1515# a_n65_n1570# 0.192222f
C2 a_65_n1515# a_n123_n1515# 0.909199f
C3 a_65_n1515# a_n225_n1689# 1.32867f
C4 a_n123_n1515# a_n225_n1689# 1.32867f
C5 a_n65_n1570# a_n225_n1689# 0.570216f
.ends

.subckt sky130_fd_pr_nfet_01v8_Z6VF7Q a_n50_n1038# a_50_n950#
a_n108_n950# a_n210_n1124#
X0 a_50_n950# a_n50_n1038# a_n108_n950# a_n210_n1124#
sky130_fd_pr_nfet_01v8 ad=2.755 pd=19.58 as=2.755 ps=19.58 w=9.5 l=0.5
**devattr s=110200,3916 d=110200,3916
C0 a_50_n950# a_n50_n1038# 0.074932f
C1 a_n108_n950# a_n50_n1038# 0.077286f
C2 a_50_n950# a_n108_n950# 0.576568f
C3 a_50_n950# a_n210_n1124# 0.750751f
C4 a_n108_n950# a_n210_n1124# 0.764718f
C5 a_n50_n1038# a_n210_n1124# 0.48776f
.ends

.subckt sky130_fd_pr_nfet_01v8_Z9V85Y a_n50_n1038# a_50_n950#
a_n108_n950# a_n210_n1124#
X0 a_50_n950# a_n50_n1038# a_n108_n950# a_n210_n1124#
sky130_fd_pr_nfet_01v8 ad=2.755 pd=19.58 as=2.755 ps=19.58 w=9.5 l=0.5
**devattr s=110200,3916 d=110200,3916
C0 a_50_n950# a_n50_n1038# 0.074932f
C1 a_n108_n950# a_n50_n1038# 0.05164f
C2 a_50_n950# a_n108_n950# 0.420437f
C3 a_50_n950# a_n210_n1124# 0.685981f
C4 a_n108_n950# a_n210_n1124# 0.547951f
C5 a_n50_n1038# a_n210_n1124# 0.48776f
.ends

.subckt sky130_fd_pr_nfet_01v8_VJM6P6 a_n210_n1149# a_n50_n1063#
a_50_n975# a_n108_n975#
X0 a_50_n975# a_n50_n1063# a_n108_n975# a_n210_n1149#
sky130_fd_pr_nfet_01v8 ad=2.8275 pd=20.08 as=2.8275 ps=20.08 w=9.75 l=0.5
**devattr s=113100,4016 d=113100,4016
C0 a_50_n975# a_n50_n1063# 0.052879f
C1 a_n108_n975# a_n50_n1063# 0.052879f

```

```

C2 a_50_n975# a_n108_n975# 0.387039f
C3 a_50_n975# a_n210_n1149# 0.499622f
C4 a_n108_n975# a_n210_n1149# 0.499622f
C5 a_n50_n1063# a_n210_n1149# 0.48776f
.ends

.subckt sky130_fd_pr_nfet_01v8_HQ3Y9H a_n210_n1149# a_n50_n1063#
a_50_n975# a_n108_n975#
X0 a_50_n975# a_n50_n1063# a_n108_n975# a_n210_n1149#
sky130_fd_pr_nfet_01v8 ad=2.8275 pd=20.08 as=2.8275 ps=20.08 w=9.75 l=0.5
**devattr s=113100,4016 d=113100,4016
C0 a_50_n975# a_n50_n1063# 0.065024f
C1 a_n108_n975# a_n50_n1063# 0.065024f
C2 a_50_n975# a_n108_n975# 0.601683f
C3 a_50_n975# a_n210_n1149# 0.868072f
C4 a_n108_n975# a_n210_n1149# 0.868072f
C5 a_n50_n1063# a_n210_n1149# 0.48776f
.ends

.subckt sky130_fd_pr_pfet_01v8_JCZH34 a_n50_n967# a_n108_n870#
w_n246_n1089# a_50_n870#
+ VSUBS
X0 a_50_n870# a_n50_n967# a_n108_n870# w_n246_n1089#
sky130_fd_pr_pfet_01v8 ad=2.523 pd=17.98 as=2.523 ps=17.98 w=8.7 l=0.5
**devattr s=100920,3596 d=100920,3596
C0 a_n108_n870# w_n246_n1089# 0.286555f
C1 a_50_n870# w_n246_n1089# 0.286555f
C2 a_n50_n967# w_n246_n1089# 0.315361f
C3 a_50_n870# a_n108_n870# 0.393711f
C4 a_n108_n870# a_n50_n967# 0.05833f
C5 a_50_n870# a_n50_n967# 0.05833f
C6 a_50_n870# VSUBS 0.199473f
C7 a_n108_n870# VSUBS 0.199473f
C8 a_n50_n967# VSUBS 0.187452f
C9 w_n246_n1089# VSUBS 4.63286f
.ends

.subckt sky130_fd_pr_pfet_01v8_9F3ELE a_n50_n297# a_50_n200#
a_n108_n200# w_n246_n419#
+ VSUBS
X0 a_50_n200# a_n50_n297# a_n108_n200# w_n246_n419#
sky130_fd_pr_pfet_01v8 ad=0.58 pd=4.58 as=0.58 ps=4.58 w=2 l=0.5
**devattr s=23200,916 d=23200,916
C0 a_n108_n200# w_n246_n419# 0.14569f
C1 a_50_n200# w_n246_n419# 0.147195f
C2 a_n50_n297# w_n246_n419# 0.315117f
C3 a_50_n200# a_n108_n200# 0.094372f
C4 a_n108_n200# a_n50_n297# 0.017016f
C5 a_50_n200# a_n50_n297# 0.017016f
C6 a_50_n200# VSUBS 0.084165f
C7 a_n108_n200# VSUBS 0.0856f
C8 a_n50_n297# VSUBS 0.16872f
C9 w_n246_n419# VSUBS 1.8552f
.ends

```

```

.subckt sky130_fd_pr_pfet_01v8_4F365H a_n50_n297# a_50_n200#
a_n108_n200# w_n246_n419#
+ VSUBS
X0 a_50_n200# a_n50_n297# a_n108_n200# w_n246_n419#
sky130_fd_pr_pfet_01v8 ad=0.58 pd=4.58 as=0.58 ps=4.58 w=2 l=0.5
**devattr s=23200,916 d=23200,916
C0 a_n108_n200# w_n246_n419# 0.140472f
C1 a_50_n200# w_n246_n419# 0.162983f
C2 a_n50_n297# w_n246_n419# 0.315117f
C3 a_50_n200# a_n108_n200# 0.133283f
C4 a_n108_n200# a_n50_n297# 0.017016f
C5 a_50_n200# a_n50_n297# 0.028662f
C6 a_50_n200# VSUBS 0.11652f
C7 a_n108_n200# VSUBS 0.067516f
C8 a_n50_n297# VSUBS 0.16872f
C9 w_n246_n419# VSUBS 1.85116f
.ends

.subckt sky130_fd_pr_pfet_01v8_53Y4NB a_n50_n967# a_n108_n870#
w_n246_n1089# a_50_n870#
+ VSUBS
X0 a_50_n870# a_n50_n967# a_n108_n870# w_n246_n1089#
sky130_fd_pr_pfet_01v8 ad=2.523 pd=17.98 as=2.523 ps=17.98 w=8.7 l=0.5
**devattr s=100920,3596 d=100920,3596
C0 a_n108_n870# w_n246_n1089# 0.394773f
C1 a_50_n870# w_n246_n1089# 0.394773f
C2 a_n50_n967# w_n246_n1089# 0.310821f
C3 a_50_n870# a_n108_n870# 0.546531f
C4 a_n108_n870# a_n50_n967# 0.07964f
C5 a_50_n870# a_n50_n967# 0.07964f
C6 a_50_n870# VSUBS 0.277547f
C7 a_n108_n870# VSUBS 0.277547f
C8 a_n50_n967# VSUBS 0.18394f
C9 w_n246_n1089# VSUBS 4.57436f
.ends

.subckt Mixer vdd gnd vlo- out- vlo+ vbias2 vbias3 vrf+ vrf- vtail vbias1 out+
XXM1 gnd vrf+ m1_5180_850# m1_4048_1162#
sky130_fd_pr_nfet_01v8_Y6SADL
XXM2 gnd vrf- m1_5180_850# m1_4436_2196#
sky130_fd_pr_nfet_01v8_Y6FLEL
XXM3 vlo- m1_962_656# out+ gnd sky130_fd_pr_nfet_01v8_PZKCVB
XXM4 vlo+ m1_962_656# out- gnd sky130_fd_pr_nfet_01v8_PMG9RB
XXMTAIL m1_5180_850# vtail gnd gnd sky130_fd_pr_nfet_01v8_8XPPYK
XXM5 vlo+ m1_2744_894# out+ gnd sky130_fd_pr_nfet_01v8_Z6VF7Q
XXM6 vlo- m1_2744_894# out- gnd sky130_fd_pr_nfet_01v8_Z9V85Y
XXM7 gnd vbias1 m1_4048_1162# m1_962_656#
sky130_fd_pr_nfet_01v8_VJM6P6
XXM8 gnd vbias1 m1_4436_2196# m1_2744_894#
sky130_fd_pr_nfet_01v8_HQ3Y9H
XXM9 vbias2 out+ vdd vdd gnd sky130_fd_pr_pfet_01v8_JCZH34
XXMc2 vbias3 vdd m1_2744_894# vdd gnd sky130_fd_pr_pfet_01v8_9F3ELE
XXMc1 vbias3 vdd m1_962_656# vdd gnd sky130_fd_pr_pfet_01v8_4F365H

```

```

XXM10 vbias2 out- vdd vdd gnd sky130_fd_pr_pfet_01v8_53Y4NB
C0 vbias1 vrf+ 0.04072f
C1 m1_4436_2196# vrf+ 0.090483f
C2 vrf- m1_4436_2196# 0.006232f
C3 m1_4048_1162# m1_2744_894# 0.169109f
C4 m1_4048_1162# m1_5180_850# 0.001308f
C5 vbias1 m1_2744_894# 0.174568f
C6 vdd vbias2 0.653058f
C7 vbias1 m1_5180_850# 0.003178f
C8 m1_2744_894# out- 0.351103f
C9 m1_4436_2196# m1_2744_894# 3.71e-20
C10 vrf- vrf+ 0.092405f
C11 m1_2744_894# vlo+ 0.008106f
C12 m1_4436_2196# m1_5180_850# 0.125253f
C13 out- vdd 0.381582f
C14 m1_4436_2196# vtail 1.8e-19
C15 m1_962_656# m1_4048_1162# 0.02315f
C16 m1_962_656# vbias2 0.006527f
C17 vlo- m1_4048_1162# 1.88e-19
C18 m1_962_656# vbias1 0.041703f
C19 out+ vbias2 0.014007f
C20 vlo- vbias1 0.005311f
C21 m1_962_656# out- 0.764945f
C22 m1_962_656# m1_4436_2196# 0.002144f
C23 m1_962_656# vlo+ 0.251472f
C24 out+ out- 0.47663f
C25 m1_2744_894# vrf+ 6.91e-20
C26 out+ vlo+ 0.36689f
C27 vlo- out- 0.047432f
C28 m1_5180_850# vrf+ 0.002f
C29 vrf- m1_5180_850# 0.12077f
C30 vlo- m1_4436_2196# 7.73e-20
C31 vlo- vlo+ 0.254583f
C32 vtail vrf+ 6.55e-20
C33 vrf- vtail 0.003585f
C34 vbias3 vbias2 2.57e-20
C35 vbias3 out- 0.006336f
C36 m1_962_656# vrf+ 7.65e-19
C37 m1_2744_894# m1_5180_850# 8.59e-19
C38 m1_2744_894# vdd 0.185584f
C39 vtail m1_5180_850# 0.005493f
C40 m1_962_656# m1_2744_894# 0.444609f
C41 m1_962_656# m1_5180_850# 0.011578f
C42 out+ m1_2744_894# 0.001419f
C43 m1_962_656# vdd 0.171392f
C44 vlo- m1_2744_894# 0.057464f
C45 out+ vdd 0.139936f
C46 vbias3 m1_2744_894# 0.109264f
C47 out+ m1_962_656# 0.382396f
C48 vbias3 vdd 0.237773f
C49 vlo- m1_962_656# 0.711116f
C50 vlo- out+ 0.104122f
C51 vbias1 m1_4048_1162# 0.066573f
C52 m1_962_656# vbias3 0.025101f
C53 m1_4048_1162# m1_4436_2196# 0.190177f

```

```
C54 out- vbias2 0.08841f
C55 vbias1 out- 1.9e-21
C56 vbias1 vlo+ 6.89e-21
C57 vlo+ out- 0.408377f
C58 m1_4048_1162# vrf+ 0.00113f
C59 vbias3 gnd 0.523372f
C60 vdd gnd 14.571484f
C61 vbias2 gnd 0.365475f
C62 m1_4436_2196# gnd 1.780702f
C63 m1_4048_1162# gnd 1.191069f
C64 vbias1 gnd 1.369267f
C65 m1_2744_894# gnd 3.669666f
C66 vtail gnd 1.028648f
C67 out- gnd 3.201187f
C68 vlo+ gnd 1.790222f
C69 m1_962_656# gnd 4.271135f
C70 out+ gnd 4.04321f
C71 vlo- gnd 2.191175f
C72 vrf- gnd 0.655737f
C73 m1_5180_850# gnd 2.748746f
C74 vrf+ gnd 0.658237f
.ends
```

Anexo C

```
Netgen 1.5.272 compiled on Do 28. Mär 11:12:42 CET 2024
Warning: netgen command 'format' use fully-qualified name '::netgen::format'
Warning: netgen command 'global' use fully-qualified name '::netgen::global'
Reading netlist file ../mag/Mixer.spice
Call to undefined subcircuit sky130_fd_pr_nfet_01v8
Creating placeholder cell definition.
Call to undefined subcircuit sky130_fd_pr_pfet_01v8
Creating placeholder cell definition.
Reading netlist file ../xschem/Mixer.spice
Call to undefined subcircuit sky130_fd_pr_nfet_01v8
Creating placeholder cell definition.
Call to undefined subcircuit sky130_fd_pr_pfet_01v8
Creating placeholder cell definition.
```

Reading setup file setup.tcl

```
Model sky130_fd_pr_nfet_01v8 pin 1 == 3
No property mult found for device sky130_fd_pr_nfet_01v8
No property sa found for device sky130_fd_pr_nfet_01v8
No property sb found for device sky130_fd_pr_nfet_01v8
No property sd found for device sky130_fd_pr_nfet_01v8
No property nf found for device sky130_fd_pr_nfet_01v8
No property nrd found for device sky130_fd_pr_nfet_01v8
No property nrs found for device sky130_fd_pr_nfet_01v8
No property area found for device sky130_fd_pr_nfet_01v8
No property perim found for device sky130_fd_pr_nfet_01v8
No property topography found for device sky130_fd_pr_nfet_01v8
Model sky130_fd_pr_nfet_01v8 pin 1 == 3
No property as found for device sky130_fd_pr_nfet_01v8
No property ad found for device sky130_fd_pr_nfet_01v8
No property ps found for device sky130_fd_pr_nfet_01v8
No property pd found for device sky130_fd_pr_nfet_01v8
No property mult found for device sky130_fd_pr_nfet_01v8
No property sa found for device sky130_fd_pr_nfet_01v8
No property sb found for device sky130_fd_pr_nfet_01v8
No property sd found for device sky130_fd_pr_nfet_01v8
No property nrd found for device sky130_fd_pr_nfet_01v8
No property nrs found for device sky130_fd_pr_nfet_01v8
No property area found for device sky130_fd_pr_nfet_01v8
No property perim found for device sky130_fd_pr_nfet_01v8
No property topography found for device sky130_fd_pr_nfet_01v8
Model sky130_fd_pr_pfet_01v8 pin 1 == 3
No property mult found for device sky130_fd_pr_pfet_01v8
No property sa found for device sky130_fd_pr_pfet_01v8
No property sb found for device sky130_fd_pr_pfet_01v8
No property sd found for device sky130_fd_pr_pfet_01v8
No property nf found for device sky130_fd_pr_pfet_01v8
No property nrd found for device sky130_fd_pr_pfet_01v8
No property nrs found for device sky130_fd_pr_pfet_01v8
```

```
No property area found for device sky130_fd_pr_pfet_01v8
No property perim found for device sky130_fd_pr_pfet_01v8
No property topography found for device sky130_fd_pr_pfet_01v8
Model sky130_fd_pr_pfet_01v8 pin 1 == 3
No property as found for device sky130_fd_pr_pfet_01v8
No property ad found for device sky130_fd_pr_pfet_01v8
No property ps found for device sky130_fd_pr_pfet_01v8
No property pd found for device sky130_fd_pr_pfet_01v8
No property mult found for device sky130_fd_pr_pfet_01v8
No property sa found for device sky130_fd_pr_pfet_01v8
No property sb found for device sky130_fd_pr_pfet_01v8
No property sd found for device sky130_fd_pr_pfet_01v8
No property nrd found for device sky130_fd_pr_pfet_01v8
No property nrs found for device sky130_fd_pr_pfet_01v8
No property area found for device sky130_fd_pr_pfet_01v8
No property perim found for device sky130_fd_pr_pfet_01v8
No property topography found for device sky130_fd_pr_pfet_01v8
Comparison output logged to file comp.out
Logging to file "comp.out" enabled
Circuit sky130_fd_pr_nfet_01v8 contains no devices.
Circuit sky130_fd_pr_pfet_01v8 contains no devices.
```

Contents of circuit 1: Circuit: 'Mixer'

Circuit Mixer contains 13 device instances.

Class: sky130_fd_pr_nfet_01v8 instances: 9

Class: sky130_fd_pr_pfet_01v8 instances: 4

Circuit contains 17 nets.

Contents of circuit 2: Circuit: 'Mixer'

Circuit Mixer contains 13 device instances.

Class: sky130_fd_pr_nfet_01v8 instances: 9

Class: sky130_fd_pr_pfet_01v8 instances: 4

Circuit contains 17 nets.

Circuit 1 contains 13 devices, Circuit 2 contains 13 devices.

Circuit 1 contains 17 nets, Circuit 2 contains 17 nets.

Final result:

Circuits match uniquely.

.

Logging to file "comp.out" disabled

LVS Done.

