Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Diseño de un módulo periférico de protocolo I2C para microcontroladores RISC-V

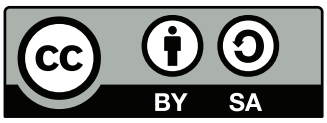
Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Presenta:

Sebastián Porras Villarreal

Cartago, Costa Rica

29 de junio de 2025



INSTITUTO TECNOLÓGICO DE COSTA RICA
ESCUELA DE INGENIERÍA ELECTRÓNICA
PROYECTO DE GRADUACIÓN
ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado Diseño de un módulo periférico de protocolo I2C para microcontroladores RISC-V, realizado por el señor Sebastián Porras Villarreal, y hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Dr. Ing. Alfonso Chacón Rodríguez
Profesor lector

Dr. Ing. Ronny García Ramírez
Profesor lector

Roberto C.

Dr. Ing. Roberto Carlos Molina Robles
Profesor asesor

Cartago, 12 de junio de 2025

Yo, Sebastián Porras Villarreal, declaro que el presente proyecto de graduación ha sido realizado en su totalidad por mi persona, aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos que he utilizado material bibliográfico, he procedido a indicar las fuentes mediante citas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Sebastián Porras Villarreal
Cédula 5-0437-0655
Cartago, 29 de junio de 2025

Resumen

En este reporte se presenta los resultados obtenidos sobre el diseño, implementación y validación de un módulo periférico de protocolo I2C para microcontroladores RISC-V, desarrollado en SystemVerilog y sintetizado con un PDK de 65 nm de TSMC. El flujo de diseño contempló la cración de dos FSM, una para el maestro y otra para el esclavo, verificadas inicialmente en testbenches independientes y posteriormente integradas al bus de datos de SIWA para demostrar operaciones de lectura y escritura correctas. Para la implementación física, se emplearon las herramientas VCS, Verdi en la fase de verificación RTL, y Fusion Compiler para la generación de layout seguido de análisis DRC y evaluación de potencia, área y timings.

Palabras clave: I2C, RISC-V, Periférico, SystemVerilog, FSM, Maestro, Esclavo, Testbench, Layout, análisis y Verificación.

Abstract

This report presents the results obtained from the design, implementation, and validation of an I²C-protocol peripheral module for RISC-V microcontrollers, developed in SystemVerilog and synthesized using a TSMC 65 nm PDK. The design flow included creating two finite-state machines—one for the master and one for the slave—which were first verified in independent testbenches and then integrated into the SIWA data bus to demonstrate correct read and write operations. For the physical implementation, VCS and Verdi were used during RTL verification, and Fusion Compiler handled layout generation followed by DRC analysis and evaluation of power, area, and timing.

Keywords: I2C, RISC-V, Periférico, SystemVerilog, FSM, Maestro, Esclavo, Testbench, Layout, análisis y Verificación.

A mis padres, Óscar y Carolina, por siempre apoyarme; mis abuelos Virgilio, Gradely, Jerónimo (QEPD) y Dilcia gracias, a mis hermanos Josesteban y Luis Roberto. Así también a demás familia y amigos que siempre me apoyaron y creyeron en mi, en especial a mi amigo Andrés y a su madre Sileny (QEPD). Gracias por todo.

Agradecimientos

Primeramente agradecer a mi familia, ya que sin el apoyo de ellos esto no podría ser posible. Gracias por creer en mí y apoyarme desde un inicio de esta etapa de mi vida. Gracias por todo Óscar, Carolina y mis hermanos por su amor incondicional, así como a mis abuelos Virgilio y Gradely por ser siempre ejemplos a seguir.

A grandes amigos y compañeros que conocí en la universidad, con los que compartí esta etapa de mi vida, Brandon, Diego, Marco, Kendall Rivera, Tobías, Luis, Beto, Santiago, Andrés y Fabián, gracias por todo, la universidad no hubiera sido la misma sin ustedes.

En especial agradecer a mi amigo Andrés y a su madre Sileny (QEPD) por abrirme las puertas de su casa cuando más lo ocupaba siendo un desconocido para ellos, gracias.

Al profesor Roberto Molina, por permitirme realizar este proyecto, y por su ayuda brindada a lo largo del trabajo.

A mis jefes del trabajo en Intel, Ruddy y Javier por brindarme mi primera oportunidad laboral como estudiante y como ingeniero en electrónica, gracias por confiar en mi.

A otras grandes amistades como Esteban, Kendall Garita, Dylan, Julián, Christopher, Reyshel y Jurimar, gracias por estar presentes siempre.

Finalmente agradecer a todos los amigos, compañeros y personas que conocí durante toda esta estapa de mi vida que, de una u otra forma, me ayudaron para que lograra conseguir este gran logro en mi vida.

Índice general

Índice de figuras VIII

1. Introducción 1
- 1.1. Entorno . . . 1
- 1.2. Síntesis del Problema . . . 2
- 1.3. Enfoque de la solución . . . 3
- 1.4. Objetivos . . . 3
  - 1.4.1. Meta . . . 3
  - 1.4.2. Objetivo General . . . 3
  - 1.4.3. Objetivos específicos . . . 3
- 1.5. Estructura del documento . . . 3

2. Marco teórico 5
- 2.1. Antecedentes . . . 5
- 2.2. Flujo VLSI . . . 5
- 2.3. I2C: . . . 7
  - 2.3.1. Señales SDA y SCL . . . 8
  - 2.3.2. Condición de inicio y de parada . . . 8
  - 2.3.3. Formato de Bytes . . . 8
  - 2.3.4. Dirección y bit R/W . . . 8
  - 2.3.5. Velocidades de bus . . . 9
- 2.4. Dispositivos Periféricos . . . 9
- 2.5. Herramientas de Hardware . . . 10
  - 2.5.1. VCS . . . 10
  - 2.5.2. Verdi . . . 10
  - 2.5.3. Fusion Compiler . . . 10

3. Propuesta de Diseño 12
- 3.1. RTL I2C . . . 12
  - 3.1.1. Master . . . 14
  - 3.1.2. Slave . . . 16
- 3.2. Integración del módulo en el bus de SIWA . . . 19

3.3. Layout 20

4. Resultados y Análisis **21**
4.1. RTL 21
4.2. Layout 25

5. Conclusiones y Recomendaciones **36**
5.1. Conclusiones 36
5.2. Recomendaciones 36

Bibliografía **38**

Índice de figuras

2.1. Flujo de Diseño VLSI. Elaboración propia 7
2.2. START and STOP conditions Extraído de [2] 8
2.3. Transferencia de datos del bus I2C [2] 9
2.4. Transferencia de datos del bus I2C [2] 9

3.1. Diagrama de bloques para etapa de procesamiento de señal. Elaboración propia. 13
3.2. Diagrama de Estados Maestro. Elaboración propia. 16
3.3. Diagrama de Estados Esclavo. Elaboración propia. 19

4.1. Waveform solo módulo maestro modo escritura. Elaboración propia. 22
4.2. Waveform maestro escribiendo en el esclavo. Elaboración propia. 23
4.3. Waveform maestro en modo read con el bus incorporado. Elaboración propia. . . 24
4.4. Waveform Empaquetado de dato maestro en modo read con el bus incorporado. Elaboración propia. 24
4.5. Waveform maestro en modo write con el bus incorporado. Elaboración propia. . 25
4.6. Floor Planning. Elaboración propia. 26
4.7. Power Rings and fences clock. Elaboración propia. 27
4.8. Power Plan. Elaboración propia. 27
4.9. Colocación de Celdas. Elaboración propia. 28
4.10. Layout Final máster. Elaboración propia. 29
4.11. DRC Clean. Elaboración propia. 29
4.12. Análisis de Área. Elaboración propia. 30
4.13. Análisis de Potencia TT. Elaboración propia. 30
4.14. Análisis de Potencia SS. Elaboración propia. 31
4.15. Análisis de Hold. Elaboración propia. 32
4.16. Análisis de Setup. Elaboración propia. 33
4.17. QoR parte 1. Elaboración propia. 34
4.18. QoR parte 2. Elaboración propia. 35
4.19. QoR parte 3. Elaboración propia. 35

Capítulo 1

Introducción

1.1. Entorno

En la Escuela de Ingeniería en Electrónica del Instituto Tecnológico de Costa Rica se encuentra el DCiLAB, el Laboratorio de Diseño de Circuitos Integrados. Este laboratorio, dirigido por el Dr. Roberto Molina, es un espacio donde profesores y estudiantes colaboran en el desarrollo de proyectos enfocados en el diseño, la verificación funcional y las pruebas de circuitos integrados.

Anteriormente, en colaboración con el Departamento de Ingeniería Eléctrica de la Universidad Católica de Uruguay, se desarrolló un proyecto llamado SIWA. Este consiste en un microcontrolador diseñado para aplicaciones médicas implantables y dispositivos de baja potencia. Está basado en una arquitectura RISC-V RV32I y fue fabricado utilizando tecnología CMOS de 180 nm [1].

El SIWA incluye dos puertos de comunicación en su diseño, que emplean los protocolos SPI y UART para garantizar la transferencia de datos y la comunicación efectiva. Además, todos los periféricos están implementados como dispositivos mapeados a memoria (MMIO), lo que permite al CPU interactuar con cada uno de ellos escribiendo en su dirección de memoria específica. [3].

La implementación de puertos de comunicación con diversos protocolos tiene el potencial de adaptar el proyecto a nuevas aplicaciones, permitiendo la integración con dispositivos que no utilicen los protocolos SPI y UART.

El protocolo de comunicación I2C es ampliamente utilizado en aplicaciones de microcontroladores debido a su simplicidad y eficiencia en la transmisión de datos entre dispositivos. Sin embargo, al igual que otros protocolos de comunicación como UART y SPI, el diseño de un módulo para cada uno de estos protocolos requiere espacio y aumenta el costo de producción, tanto en el diseño RTL como en la implementación VLSI.

Contar con un conjunto diverso de protocolos de comunicación es esencial para aumentar la compatibilidad con una mayor cantidad de dispositivos, lo que aporta flexibilidad al proyecto SIWA. La elección de un protocolo de comunicación adecuado depende de las características

específicas del proyecto, ya que diferentes protocolos ofrecen ventajas y desventajas dependiendo del tipo de aplicación. Algunos de los factores a considerar incluyen:

- Velocidad de Comunicación: Para una comunicación más rápida utilizamos SPI, para más flexibilidad UART y el I2C es el más lento de los 3, pero se puede configurar de una manera más demandante.
- Diseño de circuito: El I2C se maneja mejor en términos de espacio con múltiples dispositivos, SPI en diseños grandes, y UART para simplicidad y versatilidad.
- Distancia y ambiente de comunicación: El UART es robusto sobre largas distancias, mientras que el I2C es mejor en distancias cortas.
- Requerimientos de Duplex: SPI y UART tiene capacidades de full duplex, mientras que I2C está limitado a half duplex.

[4].

A partir de los aspectos mencionados, podemos observar que existe una clara diferencia entre los protocolos de comunicación, y que cada uno tiene ventajas dependiendo del contexto y los requerimientos específicos del proyecto. En la industria, existen dispositivos que solo son compatibles con un único protocolo de comunicación. Esto significa que, al no contar con un puerto I2C, por ejemplo, la comunicación con esos dispositivos se vuelve mucho más difícil o incluso inviable.

En un proyecto previo, se logró conectar una pantalla OLED que utiliza el protocolo de comunicación I2C al microcontrolador SIWA. Este logro se alcanzó mediante la emulación del protocolo I2C a través de software, utilizando los puertos GPIO del SIWA para simular el comportamiento del protocolo. [5].

Actualmente, el Laboratorio de Diseño de Circuitos Integrados no dispone de módulos I2C listos para ser utilizados en el proyecto SIWA ni en otros proyectos. Debido a esta limitación, todas las comunicaciones con dispositivos que operan con este protocolo deben realizarse mediante una emulación de hardware, como se hizo con la pantalla OLED. Esto impide realizar pruebas físicas de la comunicación, ya que faltan los bloques RTL correspondientes para implementar el protocolo de manera directa.

1.2. Síntesis del Problema

El Laboratorio de Diseño de Circuitos Integrados enfrenta la limitación de no contar con módulos I2C listos para ser utilizados en proyectos como el SIWA y otros desarrollos. Esto obliga a emular el protocolo I2C a través de software utilizando los puertos GPIO, lo que dificulta realizar pruebas físicas de la comunicación. La ausencia de bloques RTL para la implementación directa de I2C limita la capacidad de verificar y validar el funcionamiento de dispositivos que dependen de este protocolo, lo que retrasa el avance de los proyectos y reduce la flexibilidad en la integración de nuevos dispositivos.

1.3. Enfoque de la solución

La solución se lleva acabo apartir de una investigación del protocolo de comunicación I2C, una vez investigado el funcionamiento del protocolo de comunicación se procede a la creación de un módulo RTL utilizando el lenguaje HDL SystemVerilog, posteriormente se verifica con una testbench el funcionamiento de este módulo.

Posteriormente se realiza las pruebas del módulo master diseñado con el bus de datos del microcontrolador SIWA, utilizando también un módulo slave creado para realizar las respectivas pruebas de lectura y escritura. Por último, se procede a generar el layout del módulo master, generando reporte de DRC, análisis de potencia, área y timings.

1.4. Objetivos

1.4.1. Meta

Incorporar el modulo I2C en SIWA y validar con un dispositivo I2C con SIWA, o manipular un dispositivo a traves del I2C.

1.4.2. Objetivo General

Diseñar un módulo I2C que sea compatible con el bus de SIWA, utilizando un PDK de 65 nm de TSMC.

1.4.3. Objetivos específicos

- Diseñar el RTL en SystemVerilog un módulo I/O compatible con SIWA que opere el protocolo I2C.
- Sintetizar el módulo I2C diseñado para construir su implementación física, de acuerdo al flujo de diseño en VLSI.
- Realizar el análisis de tiempo, potencia y área del módulo implementado, para determinar el perfil del bloque RTL.

1.5. Estructura del documento

En el capítulo 2 se ofrece un resumen teórico de los fundamentos necesarios para el desarrollo de este proyecto. En el capítulo 3 se detalla el desarrollo a seguir para la implementación del módulo I2C, se presentan diagramas de bloques y de estados para entender mejor el funcionamiento del módulo. En el capítulo 4 se presentan los resultados

obtenidos durante todo el desarrollo del proyecto y un análisis de cada resultado obtenido. Finalmente en el capítulo 5 se presentan las conclusiones obtenidas de los resultados y recomendaciones a considerar para futuros proyectos relacionados a esta tesis o temas similares.

Capítulo 2

Marco teórico

En este capítulo se presentan los conceptos necesarios para el entendimiento de la propuesta de diseño y soluciones presentes en este documento. Se inicia explicando un flujo VLSI completo desde las especificaciones hasta la fabricación del circuito integrado, seguido de las bases teóricas del protocolo I2C, con la explicación de algunas señales de importancia, terminando con una breve explicación de dispositivos periféricos y de las herramientas de hardware utilizadas en este proyecto.

2.1. Antecedentes

En el año 2021 profesores del DCiLAB en conjunto del departamento de ingeniería eléctrica de la Universidad Católica del Uruguay desarrollaron el proyecto denominado *Low-level algorithm for a software-emulated I2C I/O module in general purpose RISC-V based microcontrollers*. En este proyecto lograron la comunicación por medio del protocolo I2C realizando una emulación del protocolo por medio de software y los puertos GPIO de SIWA, obteniendo una comunicación entre una pantalla OLED y el microcontrolador. La máquina de estados implementada fue de un total de 13 estados, en el que cuentan con estados como, configuraciones iniciales, cargar variables en registros, configuración e inicialización del microcontrolador, esperar el interruptor del timer, checkear la transacción I2C, reset del dispositivo I2C, inicio de la transacción, limpiar la línea SCL, setear bit de data o ACK en la línea SDA, setear la línea SCL, mantener y esperar que se complete un ciclo SCL, parar la transacción, y la rutina final de configuración [5].

2.2. Flujo VLSI

El flujo de diseño VLSI (Very Large Scale Integration) es un proceso complejo que abarca todas las etapas, desde la concepción inicial hasta la producción de dispositivos electrónicos.

- **Especificación y arquitectura:** En esta etapa se definen los requisitos para el circuito integrado o para el SoC (System on a Chip), incluida la funcionalidad, rendimiento, consumo de energía y restricciones de área, definir las especificaciones permiten crear una representación de alto nivel del sistema al mismo tiempo que considera la viabilidad económica de varios componentes.
- **Entrada de diseño:** Sienta las bases para los chips VLSI, ya que aqui se deciden el concepto general y la estructura del IC. Implica la creación de una representación de diseño de alto nivel utilizando lenguaje de descripción de hardware (HDL), como verilog, systemverilog o VHDL.
- **Verificación funcional:** Implica verificar que el diseño de alto nivel cumple con las especificaciones simulándolo mediante un simulador de hardware. Se utilizan varias técnicas para verificar los diseños. Estos incluyen verificación formal, simulación funcional y análisis de tiempo.
- **Síntesis:** El diseño de alto nivel se traduce en una netlist a nivel de puerta, una colección de puertas lógicas y flips-flops que implementan el diseño. Las herramientas de síntesis conectan la funcionalidad descrita en el HDL a un conjunto de celdas estándar.
- **Optimización del diseño:** La lista de redes de nivel de puerta está optimizada para varias restricciones de diseño, como el consumo de energía, el tiempo y el área. El mapeo, la optimización de área, la corección de reglas de diseño y la optimización de retrasos son algunos tipos comunes de procesos de optimización de diseño que se utilizan en esta etapa.
- **Diseño físico:** Este paso implica la colocación de las puertas y el enrutamiento de las interconexiones para cumplir con las restricciones de tiempo y área. Aquí, el diagrama de circuito se convierte en un diseño geométrico. Se trata de un diseño que describe los componentes lógicos, como los diodos y los transistores, y las interconexiones entre ellos.
- **Verificación de reglas de diseño (DRC):** El diseño físico se compara con un conjunto de reglas de diseño para garantizar que se pueda fabricar. Cada proceso de semiconductores tiene sus propias reglas para garantizar que el proceso de fabricación no resulte en fallas en los chips, algunas de las reglas más comunes son: ancho mínimo, espaciado mínimo, avance metálico ancho, espaciado al final de la línea y área mínima.
- **Verificación del diseño:** El diseño se verifica mediante simulaciones para garantizar que cumple con las especificaciones. Este paso ayuda a minimizar los riesgos asociados con los errores de diseño.
- **Cinta adhesiva:** Una vez que se verifica el diseño físico, el diseño final se envía a las instalaciones de fabricación. En este paso, se fabrican circuitos integrados físicos con la ayuda de las especificaciones de diseño.

- **Pruebas:** Una vez fabricado el IC o SoC se prueba para garantizar que cumple con las especificaciones y que no hay defectos de fabricación.

  [6]

En la Figura 2.1 podemos observar el flujo de diseño descrito anteriormente, sin embargo, es importante tener en cuenta que este flujo es reiterativo y no siempre es un proceso lineal.

Figura 2.1. Flujo de Diseño VLSI. Elaboración propia

Es esencial que los flujos de diseño se adapten a las nuevas tecnologías emergentes y que los diseñadores se mantengan al día con los avances.

2.3. I2C:

En este capítulo encontramos las bases teóricas para el desarrollo del proyecto, estos fundamentos son referentes al manual de usuario UM10204 de NXP Semiconductors, anteriormente Philips Semiconductors, los cuales son los desarrolladores del protocolo I2C.

2.3.1. Señales SDA y SCL

El protocolo se basa en dos cables seriales que llevan la información entre los dispositivos conectados al bus de datos, SDA (serial data) y SCL (serial clock). Ambas señales son líneas bidireccionales [2]

2.3.2. Condición de inicio y de parada

Todas las transacciones inician con una condición de START (S) y terminan con la condición de STOP (P), una transición de alto a bajo en la línea SDA mientras que SCL está en alto define una condición de START, mientras que una transición de bajo a alto en la linea SDA mientras que SCL está en alto define una condición de STOP [2].

En la Figura 2.2 podemos observar el comportamiento de las condiciones de STOP y START con las líneas SDA y SCL.

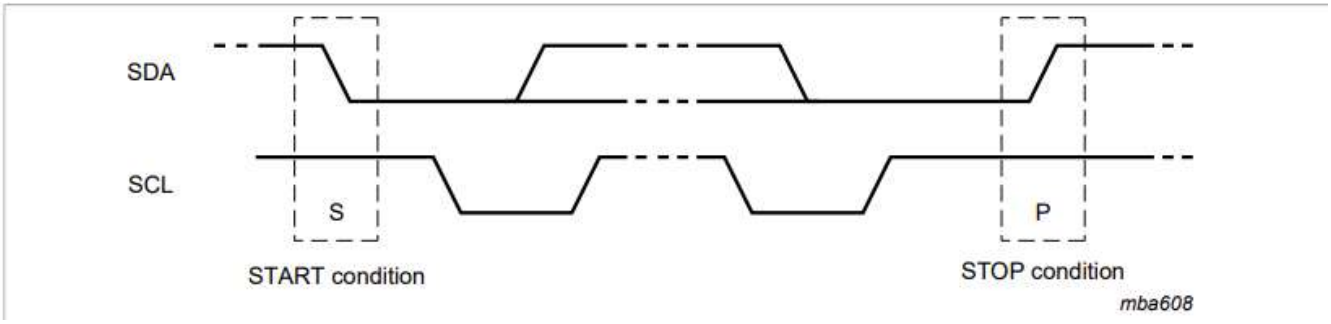


Figura 2.2. START and STOP conditions Extraído de [2]

2.3.3. Formato de Bytes

Todos los byte puestos en la línea SDA tienen que ser de una longitud de 8 bits, el número de bytes que pueden ser transferidos por transferencia no está restringido, cada byte debe de ser seguido de un bit Acknowledge, los datos son transfreridos con el MSB de primero, en el caso que el objetivo no recibe o transmite un byte completo de datos puede forzar la línea de reloj SCL en bajo para forzar al controlador en un estado de espera, la transferencia de datos continua cuando el objetivo esté listo con otro byte de datos y habilita de nuevo la línea SCL [2]. En la Figura 2.3 podemos observar la transferencia de datos en el bus de I2C.

2.3.4. Dirección y bit R/W

Después de una condición de START (S), una dirección del objetivo se envía, esta dirección tiene un largo de 8 bits, seguidos por un bit de escritura o lectura, un cero indica una transmisión (escritura) y un uno indica que se está pidiendo datos (lectura), en la Figura 2.4 podemos observar una trama completa de datos.

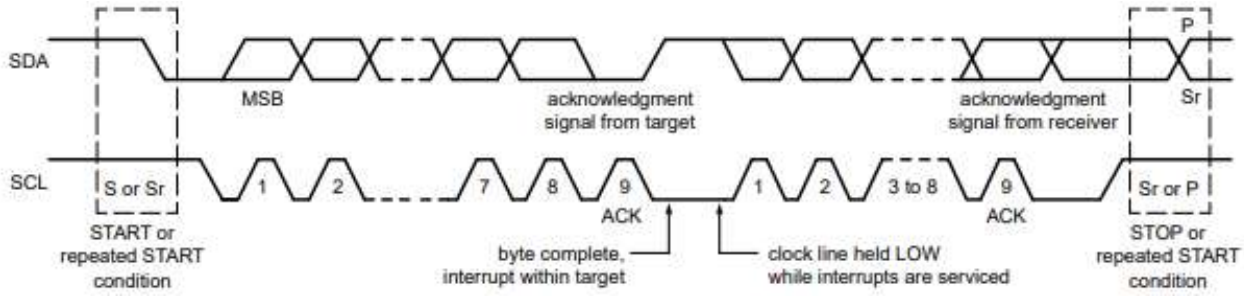


Figura 2.3. Transferencia de datos del bus I2C [2]

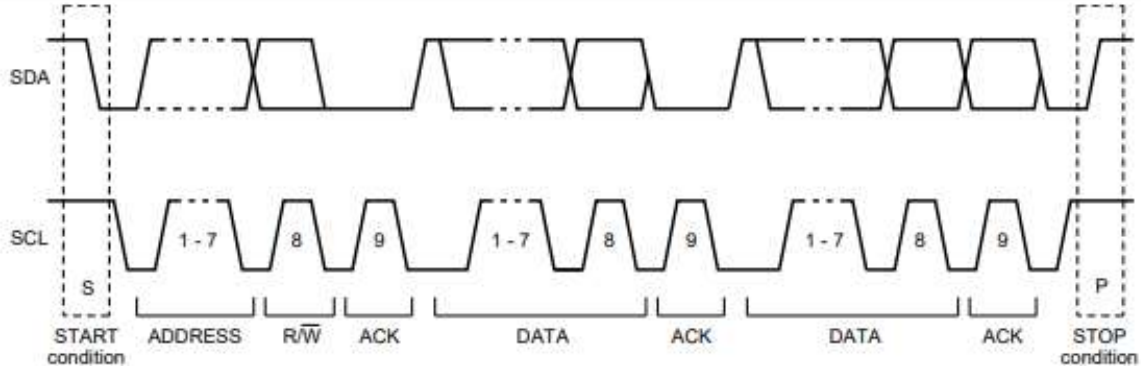


Figura 2.4. Transferencia de datos del bus I2C [2]

2.3.5. Velocidades de bus

Originalmente el bus I2C estaba limitado a 100 kbits/s, con el tiempo se agregaron algunas especificaciones y ahora tiene 5 categorías de velocidad de operación, para un bus bidireccional tenemos el standard-mode (Sm) que va hasta los 100 kbit/s, fast-mode (Fm) hasta los 400 kbit/s, Fast-mode Plus (Fm+) hasta 1 Mbit/s, High-speed mode (Hs-mode) a 3.4 Mbit/s y para busses unidireccionales tenemos el Ultra Fast-mode (UFm) que va hasta una velocidad de 5 Mbit/s [2].

2.4. Dispositivos Periféricos

Dependiendo de la aplicación prevista, un microcontrolador puede contener varios componentes auxiliares, como interfaces de entrada/salida (E/S) que incluyen temporizadores, contadores, convertidores de señal analógica a digital (ADC) y digital a analógica (DAC) y protocolos de comunicación (UART, SPI, I2C). Los auxiliares también pueden incluir componentes como pantallas LCD, puertos de conectividad Ethernet o interfaces para este tipo de módulos [18].

Algunos de los periféricos más comunes que se pueden encontrar en los microcontroladores actuales son: GPIO, Timers, USART, I2C, RTC, Watchdog, ADC, DAC y DMA [17].

2.5. Herramientas de Hardware

Para la validación del módulo diseñado en SystemVerilog, se utilizarán herramientas de Synopsys:

2.5.1. VCS

VCS es la principal herramienta de verificación utilizada por la mayoría de las principales compañías de semiconductores del mundo. VCS ofrece los motores de simulación y de resolución de restricciones de más alto rendimiento del sector. VCS incluye características innovadoras para lograr un mayor rendimiento y habilitar flujos de verificación shift-left desde las primeras fases del ciclo de diseño, de modo que se detecten más errores de forma temprana. Entre estas innovaciones destacan la Verificación de Restricciones de Diseño de Synopsys (SDC Verification), la Optimización Inteligente de Cobertura (Intelligent Coverage Optimization, ICO), la Optimización Dinámica de Rendimiento (Dynamic Performance Optimization, DPO) y la Carga Dinámica de Pruebas (Dynamic Test Loading, DTL), que amplían significativamente el alcance de la solución [19].

2.5.2. Verdi

La plataforma de gestión de depuración y verificación Verdi es una solución integral diseñada para optimizar la gestión de entrada, depuración y verificación de diseños. Gracias a sus robustas capacidades y a su conexión la base de datos de señales (FSDB) más popular, Verdi le permite planificar, ejecutar y determinar la cobertura de sus regresiones de simulación. Además, Verdi ofrece capacidades de depuración de primer nivel que le brindan información sobre todos los flujos de diseño y verificación. Verdi incluye potente tecnología de IA para automatizar los pasos de depuración más complejos y tediosos, y navegar fácilmente por entornos de diseño diversos y complejos [20].

2.5.3. Fusion Compiler

Fusion Compiler cuenta con una arquitectura única de RTL a GDSII que permite a los clientes replantear el potencial de sus diseños y alcanzar rápidamente un alto grado de diferenciación, Ofrece, de manera nativa, niveles superiores de potencia, rendimiento y área, además de tiempos de respuesta líderes en la industria. La combinación de Fusion Compiler con Synopsys DSO.ai impulsa aún mayor productividad, entregando resultados en un plazo que antes solo se podía imaginar. Entre los principales beneficios de FC tenemos que cuenta con motores unificados de RTL a GDSII que desbloquean los mejores resultados en rendimiento, consumo de potencia y área, arquitectura de modelo de datos integrada que proporciona capacidad, escalabilidad y productividad sin igual y un análisis de timing, de

signoff, extracción parasítica y evaluación de potencia integrados, eliminando iteraciones adicionales de diseño [21].

Capítulo 3

Propuesta de Diseño

Este capítulo describe a detalle la propuesta de diseño del módulo maestro, así también como un módulo esclavo para verificar la funcionalidad básica del maestro I2C, utilizando las herramientas de Synopsys para su validación. Se explicará a detalle cada uno de los diagramas propuestos para el diseño.

El flujo de diseño de este proyecto va apartir de las especificaciones del módulo I2C para dar entrada al diseño RTL utilizando el lenguaje HDL SystemVerilog, una vez comprobado la funcionalidad del módulo RTL se procede a la integración de este bloque en el bus de datos del microcontrolador SIWA, se comprueba la funcionalidad de nuevo del módulo, pero esta vez incorporado con el bus de datos, para dar paso al diseño del layout del bloque, pasando por pasos de verificación como lo son las reglas de diseño (DRC) y generando análisis de tiempos, potencia y área con las herramientas respectivas.

Para las pruebas a realizar del RTL se procedió de la siguiente manera, primero se diseñó un módulo master, y se realiza una testbench únicamente con este módulo maestro, una vez comprobado la funcionalidad básica del maestro se procede a diseñar e incorporar a las pruebas un módulo esclavo, y una vez se vuelva a verificar las funciones básicas entre ambos módulos se procede a integrar el bus de datos del microcontrolador SIWA, este bus de datos solo se usará en las pruebas para recibir o enviar datos desde y hacia el maestro.

3.1. RTL I2C

En la Figura 3.1 se puede observar el diagrama de bloques general de la solución a realizar. En este diagrama podemos observar el funcionamiento básico del módulo maestro a diseñar, en el que se va a basar el diagrama de estados y la máquina de estados a implementar.

Al principio el módulo debe de entrar en un estado de IDLE, esperando la señal de start, una vez reciba esta señal el módulo sabe que tiene que comunicarse con un esclavo, y este lo hara enviando un dato igual al número de dirección del esclavo con el que desea comunicarse, una vez se envía la dirección del esclavo, procede a enviarse un bit que define si se desea escribir o leer, estos dos casos son similares, se busca enviar o recibir un byte completo, una

vez termina esta transacción procede a parar y volver otra vez a IDLE esperando otra señal de start para volver a repetir el mismo proceso.

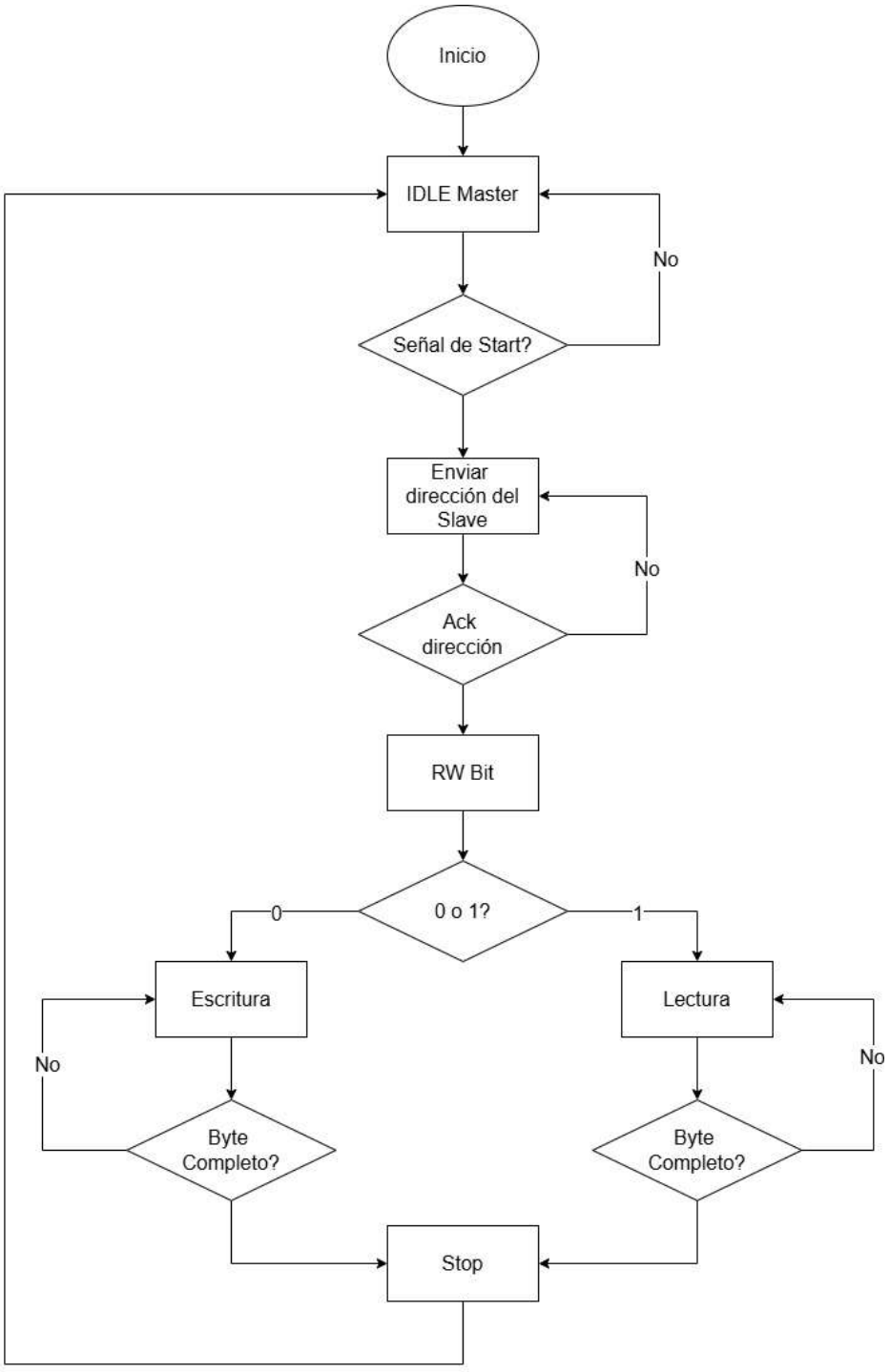


Figura 3.1. Diagrama de bloques para etapa de procesamiento de señal. Elaboración propia.

3.1.1. Master

Señales de entrada, salida e internas del módulo maestro:

- CLK_FREQ_HZ: (Parámetro) Frecuencia del reloj del sistema. Se usa para generar la señal SCL mediante un divisor.
- I2C_FREQ_HZ: (Parámetro) Frecuencia deseada en el bus I2C.
- clk: (Entrada) Reloj síncrono del módulo. Toda la lógica secuencial se actualiza en el flanco positivo de clk.
- rst_n: (Entrada) Reset asíncrono activo en bajo.
- start: (Entrada) Pulso de un ciclo de clk que indica al maestro que debe iniciar una nueva transacción I2C.
- addr: (Entrada) Dirección de 7 bits del esclavo I2C con el que nos queremos comunicar.
- rw: (Entrada) Bit que indica modo de operación.
- wdata: (Entrada) Byte de datos que el maestro desea escribir en el esclavo, es válido solo si rw es igual a 0.
- rdata: (Salida) Byte leído desde el esclavo, se alimenta desde un registro de desplazamiento shift_reg.
- done: (Salida) Pulso de un ciclo de clk que se genera en el momento en que la transacción I2C ha finalizado.
- sda: (Entrada/Salida) Línea de datos bidireccional, implementada como un pin inout tri para poder conducir 0 (arrastrar la línea a bajo) o bien dejarla en alta impedancia para que el esclavo o la resistencia pull-up la lleven a 1.
- scl: (salida) Línea de reloj I2C, se genera internamente a partir de clk mediante un divisor.
- div_cnt: (Señal interna) Contador que controla el periódo de SCL.
- scl_int: (Señal interna) Reloj I2C interno, antes de asignarlo a la salida scl, cambia de valor cada DIV ciclos.
- sda_out_en: (Señal interna) Controla si el maestro conduce SDA o la deja en alta impedancia.
- sda_out_val: (Señal interna) Valor que el maestro pone en SDA cuando sda_out_en es igual a 1.
- shift_reg: (Señal interna) Registro de 8 bits donde el maestro acumula los datos recibidos durante READ_BYTE.

- rdata: (Señal interna) Salida asignada a shift_reg, refleja el byte leído del esclavo.
- bit_cnt: (Señal interna) Cuenta cuántos bits faltan por transmitir o recibir. Evita desbordes con "guard".
- state: (Señal interna) Contiene el estado actual de la FSM.
- next: (Señal interna) Señal combinacional que define el siguiente estado según la lógica de la FSM.

En la Figura 3.2 se observa un diagrama de estados el cual corresponde al utilizado para el diseño del módulo maestro en SystemVerilog.

La máquina de estados del maestro I²C arranca en `IDLE`, donde tanto `SDA` como `SCL` permanecen en alta impedancia y el bus está inactivo. El maestro no realiza ninguna acción mientras la señal `start` permanezca en 0; sólo cuando esa señal pasa a 1 se interpreta que debe iniciarse una transmisión. En ese momento se carga el contador de bits con el valor 7 y se transita al estado `START`.

En `START` el maestro genera la condición de arranque: primero fuerza `SDA = 0` mientras `SCL = 1`, obteniendo la transición "1→0 en `SDA` con `SCL` en 1" que define el START. Después espera a que `SCL` caiga a 0, garantizando que el reloj esté en reposo antes de enviar datos. Cuando ese pulso de reloj ha descendido por completo, la FSM avanza a `SEND_ADDR`.

En `SEND_ADDR` el maestro transmite, uno tras otro, los siete bits de dirección y el bit de lectura/escritura. Cada vez que detecta un pulso alto efectivo de `SCL`, coloca en `SDA` el siguiente bit de `{addr[6:0], rw}` y decrementa el contador. Mientras queden bits por enviar, permanece en `SEND_ADDR`. Cuando el contador llega a 0 exactamente durante un flanco alto de reloj, significa que el octavo bit se ha enviado y la FSM pasa a `ADDR_ACK`.

En `ADDR_ACK`, el maestro suelta `SDA` para que el esclavo coloque un ACK o un NACK. En el siguiente flanco bajo de `SCL` el maestro evalúa la respuesta: si es ACK y el bit de R/W indicaba escritura (`rw = 0`), pasa a `WRITE_BYTE`; si es ACK y el bit de R/W indicaba lectura (`rw = 1`), pasa a `READ_BYTE`. Si recibe un NACK, va directamente a `STOP`.

En `WRITE_BYTE` el maestro envía el byte de datos al esclavo. Cada flanco alto de `SCL` hace que coloque el siguiente bit de `data_in` en `SDA` y que el contador baje uno. Cuando el contador llega a 0 en un flanco alto de reloj, significa que el octavo bit de datos ya se envió; en ese mismo instante la FSM avanza a `WRITE_ACK`.

En `WRITE_ACK`, el maestro libera `SDA` para recibir el ACK del esclavo. En el primer flanco bajo de `SCL`, el esclavo conduce `SDA = 0` si confirma recepción o `SDA = 1` si hubo error. Tras captar esa respuesta, la FSM va a `STOP`.

Si la operación era lectura, la FSM entra en `READ_BYTE`. Allí, en cada flanco alto de `SCL`, el maestro lee el bit que el esclavo ha colocado en `SDA` y decrementa el contador. Cuando el contador llega a 0 en un flanco alto, se ha leído el octavo bit y la FSM pasa a `READ_ACK`.

En `READ_ACK` el maestro conduce `SDA = 1` en el primer flanco bajo de `SCL` para enviar un NACK que indica al esclavo el fin de lectura. A continuación la FSM avanza a `STOP`.

En `STOP`, el maestro primero asegura que `SDA = 0` mientras `SCL = 1`, luego, en ese mismo pulso alto de reloj, sube `SDA = 1`, generando la transición "0→1 en `SDA` con `SCL` = 1" que cierra la comunicación. En el flanco bajo siguiente, levanta la señal `done` para indicar que la transacción ha concluido, y regresa a `IDLE`.

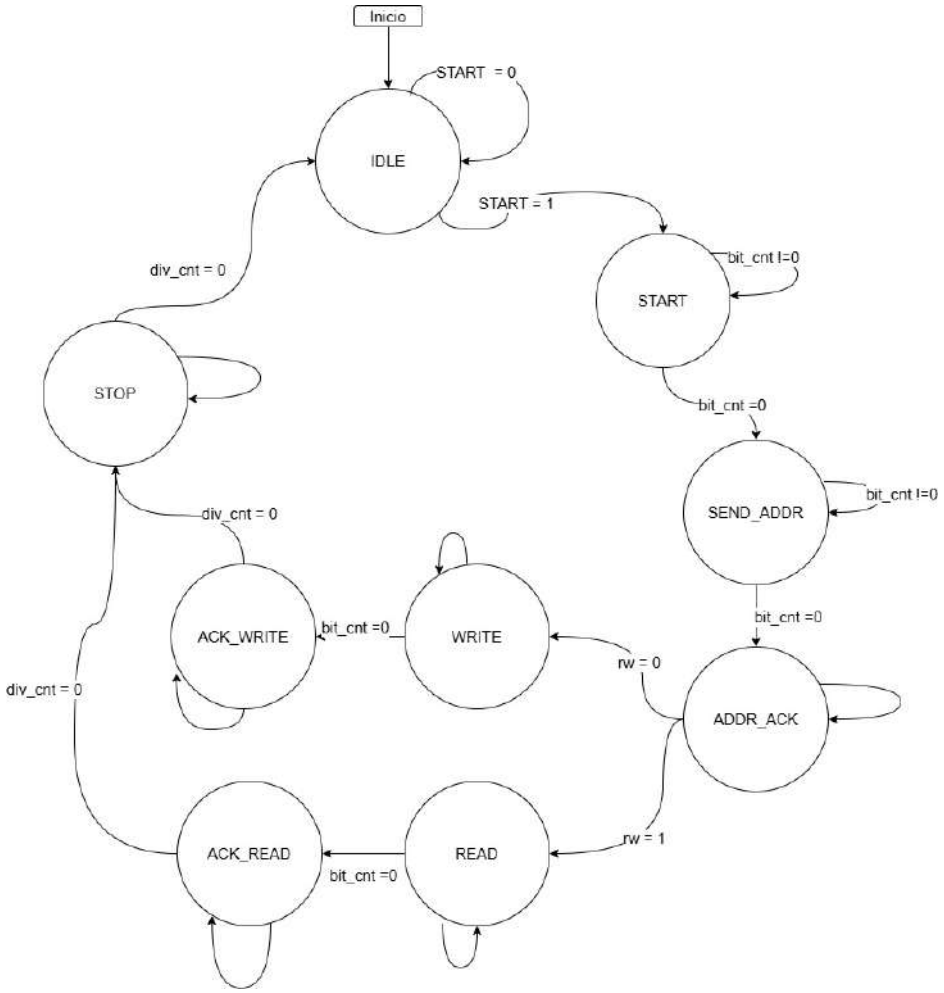


Figura 3.2. Diagrama de Estados Maestro. Elaboración propia.

3.1.2. Slave

Señales de entrada salida e internas del módulo slave:

- SLAVE_ADDR: Parámetro de 7 bits, corresponde a la dirección fija del esclavo en el bus I2C.
- INIT_BYTE: Parámetro de 8 bits que corresponden al Byte que el esclavo enviará en caso de que el maestro inicie una operación de lectura. Se guarda en data_buf.
- clk: (Entrada) Reloj síncrono interno del módulo. Toda la FSM se actualiza en el flanco positivo de clk.

- rst_n: (Entrada) Reset asíncrono activo en bajo, cuando rst_n es igual a 0, el módulo inicializa todas las señales internas a valores por defecto.
- sda: (Entrada/Salida) Línea de datos I2C bidireccional. Se declara como inout tri, porque el esclavo puede leer de ella o conducirla a nivel bajo (open-drain).
- scl: (Entrada) Línea de reloj I2C, el esclavo detecta sus flancos para muestrear o escribir bits.
- sync_sda: (Señal interna) Copia sincronizada al reloj clk de la línea SDA Física.
- sync_scl: (Señal interna):Copia sincronizada al reloj clk de la línea SCL física.
- prev_sda:(Señal interna) Valor anterior de sync_sda, usado para detecar flancos.
- prev_scl: (Señal interna) Valor anterior de sync_scl, usado para detectar flancos.
- start_cond: (Señal interna) Se pone a 1 cuando SDA cae mientras que SCL permanece en 1.
- bit_cnt: (Señal interna) Contador de bits durante las fases de recepción y envío de un byte.
- addr_reg: (Señal interna) Almacena los 7 bits de dirección enviados por el maestro.
- rw_flag: (Señal interna) Indica si el maestro quiere leer o escribir tras la dirección.
- data_buf: (Señal interna) Buffer temporal que va armando el byte que entra desde el maestro.
- shift_reg: (Señal interna) Registro de desplazamiento donde se almacena el byte completo de llegada, para su uso posterior.
- sda_oe: (Señal interna) Salida habilitada para SDA, cuando está en 1, el esclavo gobierna la línea SDA, si es 0 la deja libre.
- sda_o: (Señal interna) Nivel lógico que el esclavo coloca en la línea SDA cuando sda_oe es 1.

La máquina de estados del esclavo I²C se encuentra en la Figura 3.3 y comienza en `ST_IDLE`, con `SDA` en alta impedancia mientras monitorea `SDA` y `SCL`. Mientras `SDA` permanezca en 1 y `SCL` también en 1, no ocurre START y el esclavo continúa en reposo. Cuando el maestro baja `SDA` de 1 a 0 con `SCL` en 1, el esclavo detecta START, carga `bit_cnt = 7` y pasa a `ST_ADDR`.

En `ST_ADDR`, el esclavo lee ocho bits (siete de dirección, uno de R/W) muestreando `SDA` en cada flanco alto de `SCL`. Con cada pulso alto, captura el valor de `SDA` en un buffer de byte y decrementa `bit_cnt`. Cuando el contador llega a 0 en un flanco alto, significa que se han

recibido los ocho bits; entonces extrae los siete primeros como dirección y asigna el octavo a `rw_flag`, reinicia `bit_cnt = 7` y transita a `ST_ADDR_ACK`.

En `ST_ADDR_ACK`, el esclavo conduce `SDA = 0` (ACK) o `SDA = 1` (NACK) en el siguiente flanco bajo de `SCL`, comparando la dirección recibida con `SLAVE_ADDR`. Si coincide y `rw_flag = 0` (escritura), entra en `ST_WRITE`; si coincide y `rw_flag = 1` (lectura), entra en `ST_READ`. En caso de dirección incorrecta, emite NACK y va a `ST_FIN`.

En `ST_WRITE`, el esclavo recibe un byte del maestro. En cada flanco alto de `SCL`, captura el bit de `SDA` y decrementa `bit_cnt`. Cuando `bit_cnt` llega a 0 en un flanco alto, se han recibido ocho bits; en ese mismo flanco el esclavo arma un registro interno con esos bits y avanza a `ST_WRITE_ACK`.

En `ST_WRITE_ACK`, el esclavo conduce `SDA = 0` (ACK) en el primer flanco bajo de `SCL` para confirmar la recepción del byte, y luego va a `ST_FIN`.

Si la operación era lectura, el esclavo entra en `ST_READ`. Allí, en cada flanco bajo de `SCL`, coloca en `SDA` el siguiente bit del dato a enviar; en cada pulso alto decrementa `bit_cnt`. Cuando el contador llega a 0 tras un pulso alto, ha completado el byte y avanza a `ST_READ_ACK`.

En `ST_READ_ACK`, el esclavo conduce `SDA = 1` (NACK) en el siguiente flanco bajo de `SCL` para indicar fin de lectura, y luego va a `ST_FIN`.

En `ST_FIN`, el esclavo libera `SDA` (alta impedancia) y espera que el maestro genere la condición de STOP: `SDA = 1` y `SCL = 1` simultáneamente. Cuando detecta ambas líneas en 1, retorna a `ST_IDLE`, quedando listo para una nueva transacción.

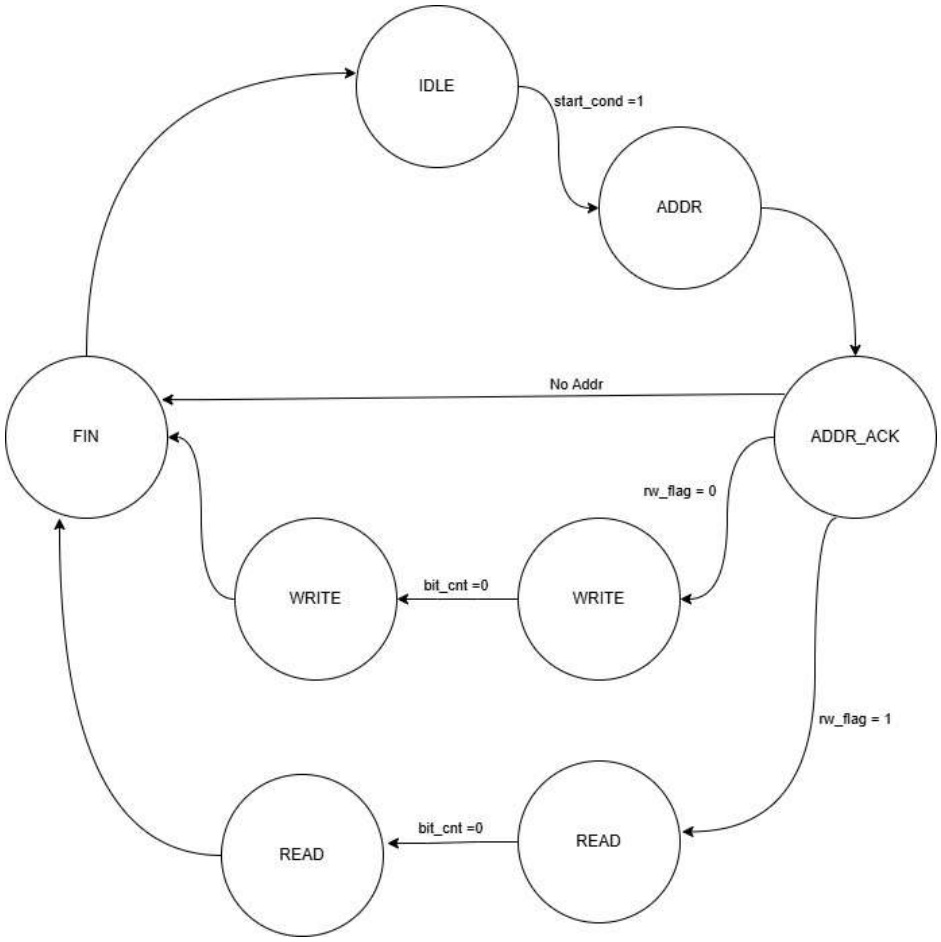


Figura 3.3. Diagrama de Estados Esclavo. Elaboración propia.

3.2. Integración del módulo en el bus de SIWA

Se diseña un módulo TOP que funciona como conector entre el módulo maestro con el bus de SIWA. Este módulo podra empaquetar datos para ser enviados al bus SIWA o también podra desempaquetar datos para poder ser enviados por medio del protocolo de comunicación.

Señales de entrada, salida e internas del módulo de integración:

- CLK_FREQ_HZ: (Parámetro) Frecuencia del reloj de sistema en HZ.
- I2C_FREQ_Hz: (Parámetro) Velocidad deseada del bus I2C en Hz.
- clk: (Entrada) Señal de reloj principal del wrapper.
- rst_n: (Entrada) Reset asíncrono activo en bajo.
- pdng_mbc: (Entrada) Pulso que indica "paquete pendiente.^{en} la interfaz SIWA.
- D_pop_mbc: (Entrada) Datos a leer del bus SIWA.

- pop_mbc: (Salida) Señal de pop que el wrapper debe generar para leer un paquete del bus SIWA.
- push_mbc: (Salida) Señal de push que el wrapper genera para escribir un paquete al SIWA.
- D_push_mbc: (Salida) Datos que el wrapper empuja al bus SIWA en respuesta a la operación I2C.
- sda: (Entrada y Salida) Línea bidireccional de datos I2C.
- scl: (Salida) Línea de reloj I2C.

3.3. Layout

En la generación del layout del módulo se procedió a utilizar el tutorial de Celda Básica Fusion Compiler, el cual nos guía a través de un flujo de diseño completo utilizando Fusion Compiler y la tecnología TSMC para la implementación del diseño, el proceso comienza con la creación de la biblioteca y la lectura del RTL. A continuación, se sigue con el flujo de compilación, seguido de la planificación y la colocación del diseño. Después, se procede a la síntesis del árbol de reloj y al enrutamiento. Finalmente, se realiza la verificación final para asegurarse de que el diseño cumpla con todas las especificaciones técnicas necesarias [22].

Capítulo 4

Resultados y Análisis

4.1. RTL

En esta sección se discutirán y analizaran los resultados obtenidos del diseño propuesto en las secciones 3.2, 3.3 y 3.2. Donde se mostraran los waveforms obtenidos para cada uno de los módulos respectivos.

En la Figura 4.1 observamos el waveform obtenido por medio de una testbench básica y únicamente el módulo master a implementar, esta figura tiene como función comprobar que el módulo maestro puede ser capaz de enviar datos por medio de la línea SDA, para posteriormente ser escritos en el esclavo. Como se observa en la Figura 4.1 tenemos algunas señales como clk, rst_n, rw, state, wdata, addr, bit_cnt, scl y sda.

El registro de la señal addr es únicamente para que el maestro sepa que dirección tiene que mandar por medio de las líneas bidireccionales para lograr la comunicación, en este caso estamos intentando contactar a un módulo esclavo con una dirección fija de 42 en hexadecimal, bit_cnt nos ayuda a poder contar los bits que han sido enviados por cada ciclo de SCL, rst_n, como se menciona en las secciones 3.1 es activo en bajo, por lo que debe de permanecer siempre en alto para que no se reinicie el proceso, en la señal de state, tenemos los estados en los que va transicionando la máquina de estados mientras va realizando la escritura, por otro lado en el registro wdata, es el dato que queremos mandar por medio del protocolo, en este caso estamos intentando escribir un número en hexadecimal el cual es A5.

En los rectángulos 1 y 2 podemos observar como se envian datos por medio de las líneas bidireccionales, por ejemplo en el rectaángulo número 1, podemos leer la secuencia de bits **1000 0100** el cual corresponde en hexadecimal al número 84, en esta secuencia va incluída la dirección del esclavo al que se quiere conectar de los bits 7 al 1 **1000010**, el cual corresponde al número 42 en hexadecimal, y en el bit 0 se encuentra la condición rw, en este caso como estamos simulando un proceso de escritura es 0. Por otro lado en el rectángulo 2, tenemos el dato a escribir en la línea SDA en esta parte encontramos la secuencia **10100101** que corresponde al número en hexadecimal A5.

Figura 4.1. Waveform solo módulo maestro modo escritura. Elaboración propia.

Una vez comprobado que el módulo maestro pueda realizar transacciones de manera correcta, procedemos a incorporar el módulo esclavo en las pruebas, en la Figura 4.2 se observa un waveform obtenido de la simulación del maestro con el esclavo en modo de escritura, a diferencia del waveform de la Figura 4.1 en este podemos observar dos señales referentes a los estados, en el que la señal state de 4 bits es para la FSM del maestro y la señal state de 3 bits es para la FSM del esclavo, en los rectángulo 1 y 2 de la Figura 4.2 podemos observar las mismas secuencias de bits en sda que en la señal sda de la Figura 4.1, sin embargo la diferencia es que aqui podemos observar con más claridad los valores en hexadecimal de la dirección del esclavo, en la señal addr_reg se observa como cambia de 0 a 42 en el estado que corresponde al envío de la dirección, mientras que el dato final en byte_buf en este estado corresponde al 84 de la dirección más el bit de rw, una vez verificado la dirección y el modo de operación la máquina de estados del esclavo reinicia este registro de byte_buf para poder almacenar el byte a escribir bit por bit como se observa que su valor final es de A5.

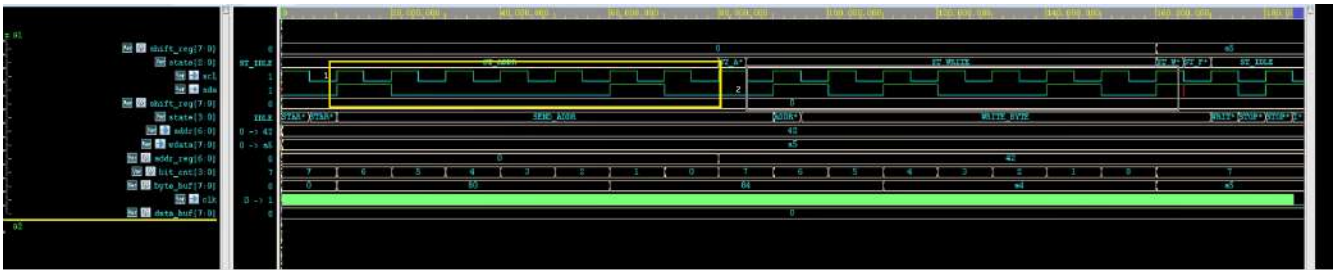


Figura 4.2. Waveform maestro escribiendo en el esclavo. Elaboración propia.

En la Figura 4.3 observamos señales similares a las Figuras 4.2 y 4.1, sin embargo, en esta figura tenemos ya el bus SIWA incorporado a las pruebas, la imagen corresponde a un waveform de una prueba de lectura de un dato almacenado en el esclavo, por lo que lo primero es mandar desde el bus un paquete que contenga únicamente los datos de dirección y el bit de rw, como podemos observar en la señal pkt_reg en el que observamos desde el comienzo el número en hexadecimal de **30802**, que corresponde a la secuencia de bits **0011 0000 1000 0000 0010**, donde los bits 0 y 1 corresponden a bits de start y de reset, los bits desde el 3 al 10, corresponden al dato que se desea escribir, en este caso como es un proceso de lectura corresponde a la secuencia **0000 0000**, seguido de los bits de dirección que van desde el bit 11 al bit 17 con la secuencia binaria **0100 001** que corresponde al igual que en otras pruebas al número 42 en hexadecimal, y seguido de esta secuencia tenemos el bit de rw, como estamos iniciando una secuencia de read tenemos un 1 en el bit 18 del paquete.

Se comprueba la secuencia en el rectángulo número 1 que sea la adecuada para la dirección del esclavo y el bit de rw correspondientes y como se observa en las señales de estados el esclavo entra en un modo de read, que este modo le indica que tiene que tomar el control de la línea SDA para enviar datos hacía el maestro, en el rectángulo 2, observamos la secuencia binaria **0101 1010**, que corresponde al valor que deseamos leer, valor que corresponde en hexadecimal a 5A. Al igual que en la señal dbg_rdata, observamos como va cambiando el registro de lectura mediante van llegando los bits en sda, empezando con un 1, 2, 5, b, 16, 2d y 5a para poder comprobar que se leyó de forma correcta. Una vez las FSM se encuentren en IDLE y no exista ninguna transferencia de datos entre el maestro y el esclavo se procede a empaquetar el dato leído con el módulo wrapper.

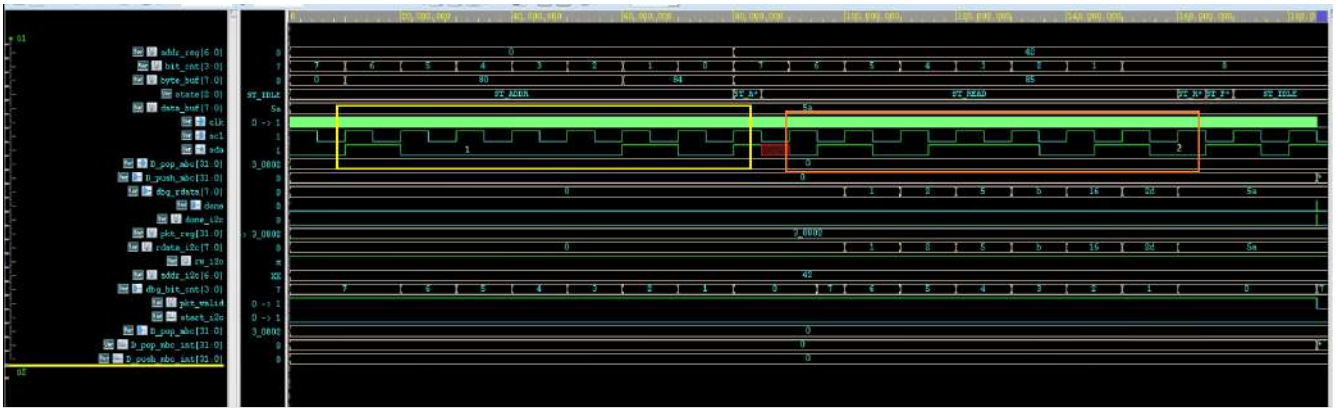


Figura 4.3. Waveform maestro en modo read con el bus incorporado. Elaboración propia.

En la Figura 4.4 tenemos la parte final del waveform de la Figura 4.3, en el que observamos el paquete que el maestro va a entregar al bus de datos, en la señal D_push_mbc observamos como cambia de 0 a 30969 en hexadecimal, valor que corresponde a la secuencia en binario **0011 0000 1001 0110 1001**, donde tenemos el mismo orden de los bits de dirección, rw, pero en lugar del dato de escritura tenemos el dato que leímos desde el esclavo, desde los bits 2 al 9 **0101 1010** que corresponde al valor en hexadecimal de 5A.

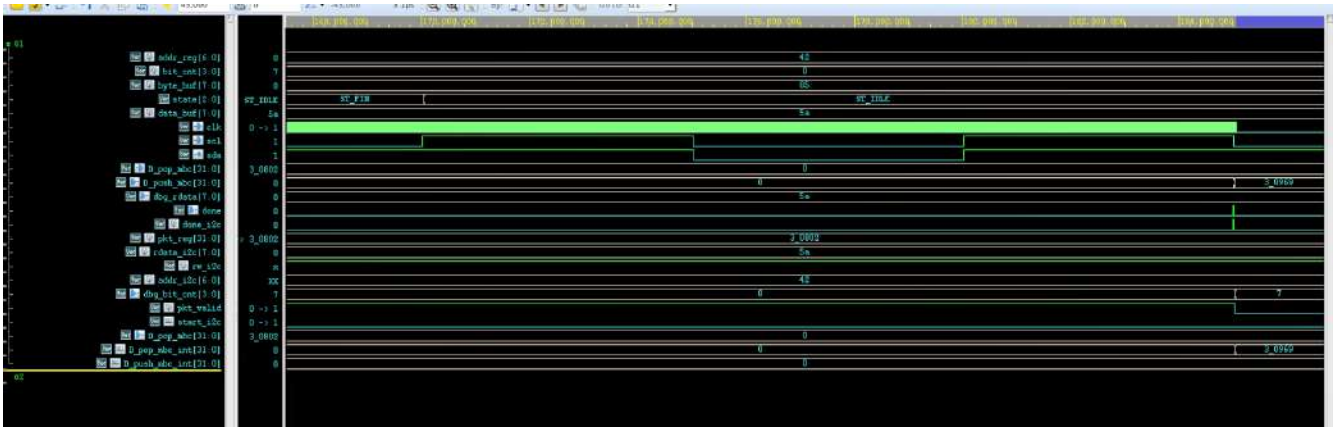


Figura 4.4. Waveform Empaquetado de dato maestro en modo read con el bus incorporado. Elaboración propia.

Al igual que en la Figura 4.3 en la Figura 4.5 tenemos el waveform del módulo master conectado al slave con el bus de datos incorporado, pero en esta ocasión está en modo de escritura. En el registro pkt_reg tenemos el valor en hexadecimal de 10a96, que corresponde

en binario a **0001 0000 1010 1001 0110**, que sigue la misma estructura explicada anteriormente.

Las señales son las mismas, en el rectángulo número 1 tenemos el envío de la dirección correspondiente, pero esta vez con el bit de rw en 0, ya que estamos escribiendo un byte en el esclavo.

En el rectángulo 2 obtenemos la secuencia de bits **1010 0101** el cual corresponde a A5 en hexadecimal, y es almacenado posteriormente en el registro shift_reg, una vez se terminen de enviar todos los bits respectivos. Se observa que el byte enviado corresponde correctamente a los bits que se deseaban escribir enviados desde el bus de datos.

Figura 4.5. Waveform maestro en modo write con el bus incorporado. Elaboración propia.

4.2. Layout

En la Figura 4.6 podemos observar la planificación de como se van a colocar los bloques lógicos y de memoria dentro del área del chip.

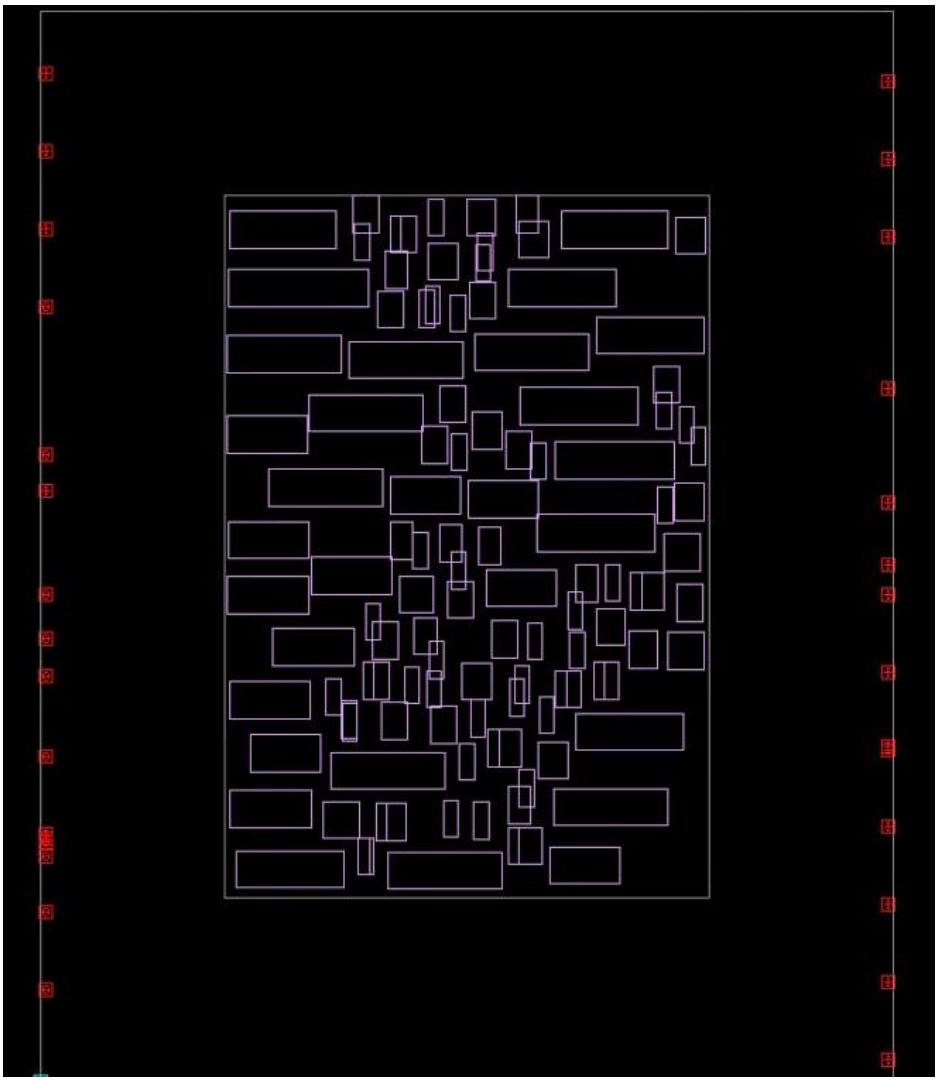


Figura 4.6. Floor Planning. Elaboración propia.

Se puede observar la colocación de los anillos de alimentación y la colocación para el reloj que va a alimentar el módulo en la Figura 4.7.

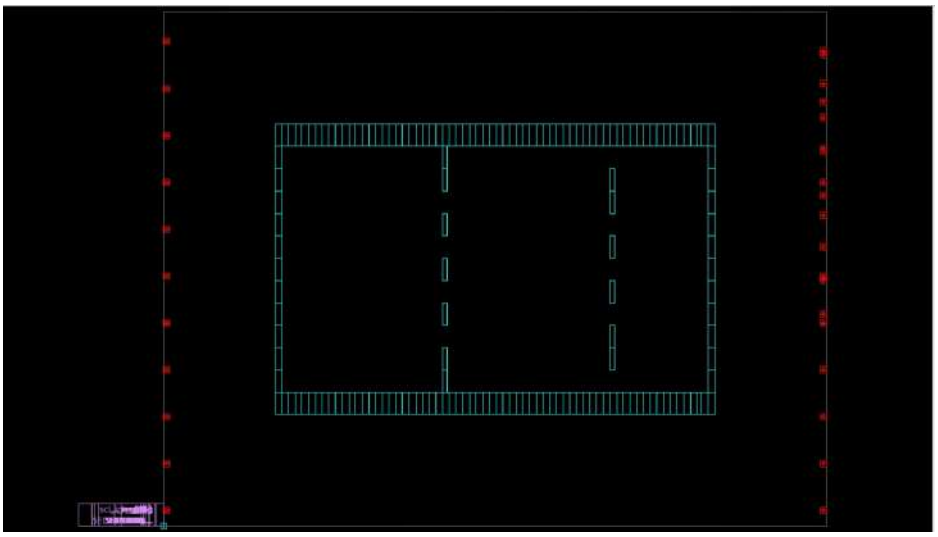


Figura 4.7. Power Rings and fences clock. Elaboración propia.

En la Figura 4.8 se muestra la inicialización del power-plan y del grid de placement. Todos los rieles de VDD/VSS están definidos, los well-ties colocados, y las placements rows demarcadas, sin embargo, aún no hay celdas ni rutas de señal, porque corresponde al paso intermedio que fija la red eléctrica y la retícula para la posterior colocación de lógica.

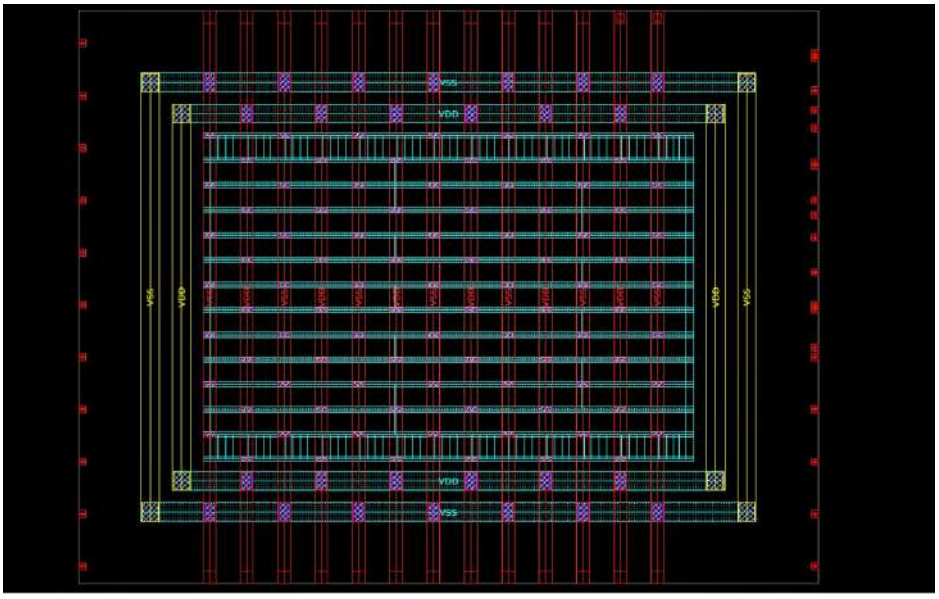


Figura 4.8. Power Plan. Elaboración propia.

En la vista de la Figura 4.9 se muestra cómo el sintetizador/placement ha colocado las celdas estándar en las filas de core, sobre la malla de alimentación ya establecida. Aún no hay rutas de señal dibujadas, pero si se han fijado las posiciones exactas de cada instancia, alineada con railes de VDD/VSS correspondientes.

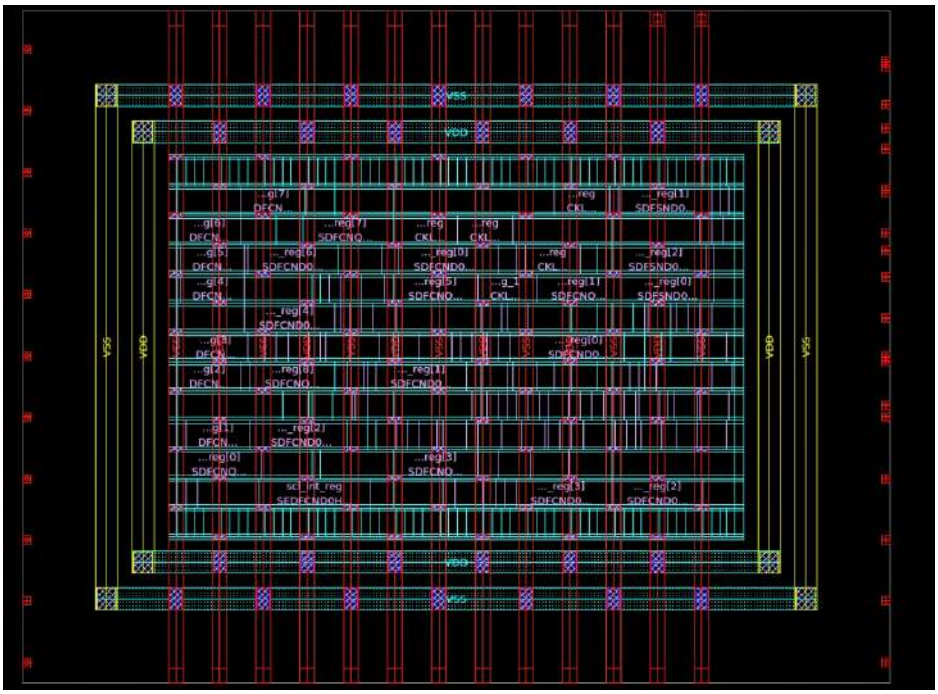


Figura 4.9. Colocación de Celdas. Elaboración propia.

El layout completo del módulo maestro se puede observar en la Figura 4.10, en esta vista se observa todas las celdas estándar en su posición y alimentadas, las rutas de señal que conecta cada pin de cada celda, así como los pads de I/O, atravesando varios capas de metal, se pueden apreciar los railes de VDD/VSS y los well-ties en su posición definitiva.

Los siguientes análisis de DRC, timing post-route se hará sobre este diagrama. dado que aquí ya están definidos completamente los ruteos y la red de alimentación del chip.

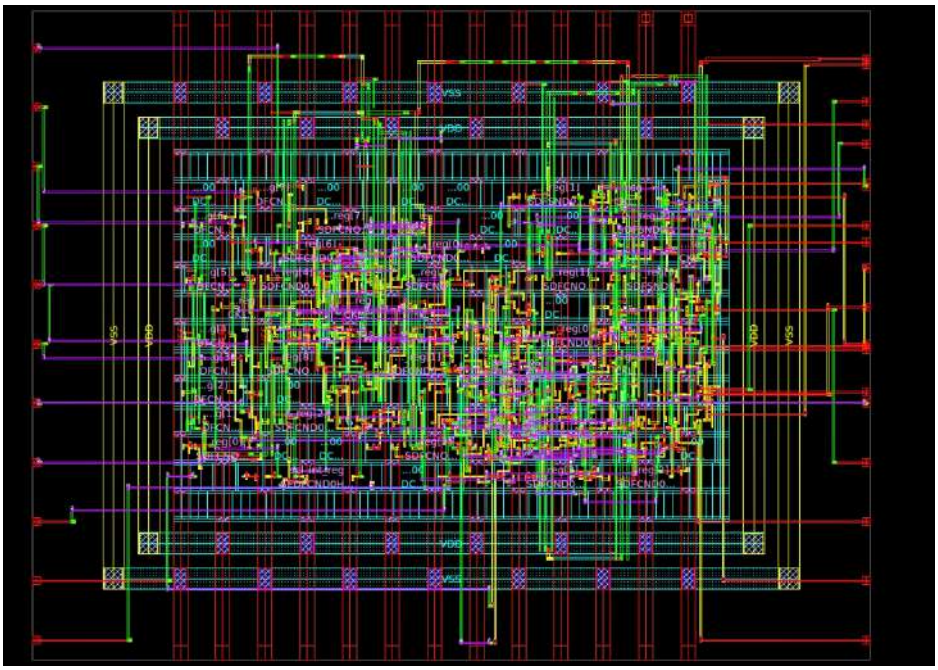


Figura 4.10. Layout Final máster. Elaboración propia.

La Figura 4.11 observamos el reporte de DRC límpio, lo que nos indica que el diseño físico del módulo maestro no contiene violaciones de reglas, por lo que está listo para ser entragado como GDS.

Figura 4.11. DRC Clean. Elaboración propia.

En la Figura 4.12 observamos el informe de área, que muestra que el diseño utiliza 142 celdas estándar: 112 combinacionales que ocupan 242.80 um², 30 secuenciales con 322 um²

y 15 buffers/inversores que suman 18.80 um², sin macros externas. En total, todas las celdas consumen 564.80 um² de silicio. Este desglose indica que más del 57 % del área corresponde a lógica con memoria, mientras que el resto es lógica pura o buffers de reloj.

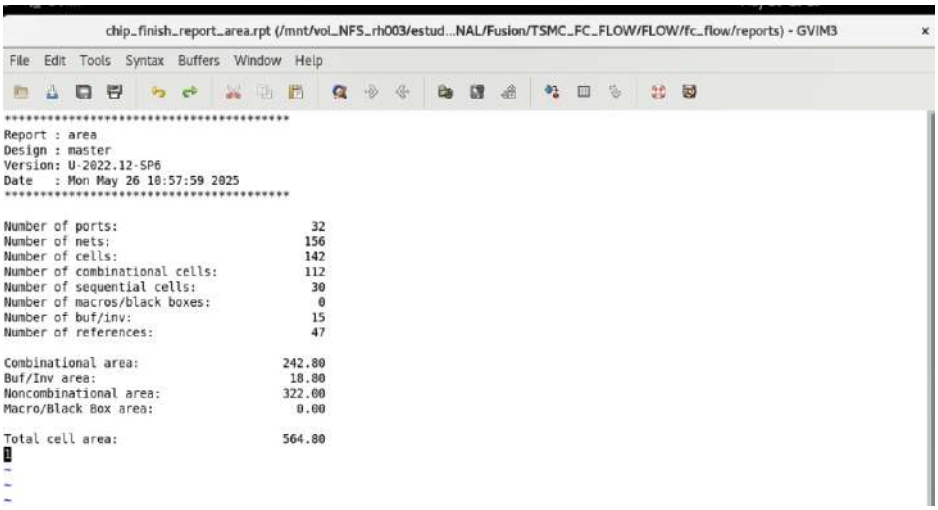


Figura 4.12. Análisis de Área. Elaboración propia.

En las Figuras 4.13 y 4.14 tenemos los informes de potencia generados para el diseño del módulo en tecnología 65 nm, en condiciones típicas el consumo dinámico total es de 39 uW, de los cuales el 70 % se debe al consumo interno de las celdas y el 30 % restante a la conmutación de las redes, la red de distribución de reloj acapara el 69 % de ese consumo dinámico, pues dispara todos los flip-flops cada ciclo. Los registros aportan otro 19 % y la lógica combinacional un 11 %. Las fugas estáticas suman 12 uW, lo que significa que la corriente de leakage supone aproximadamente un 23 % del consumo total.

Por otro lado en el SS, el consumo dinámico cae a 24 uW, manteniendo la misma proporción interna/combinacional. La red de reloj aumenta su peso relativo hasta el 86 % del dinámico, ya que en esta esquina lenta las puertas combinacionales cambian muy despacio. El leakage se reduce drásticamente a 0.13 uW, casi despreciable, porque en corner slow los transistores filtran muy poca corriente.

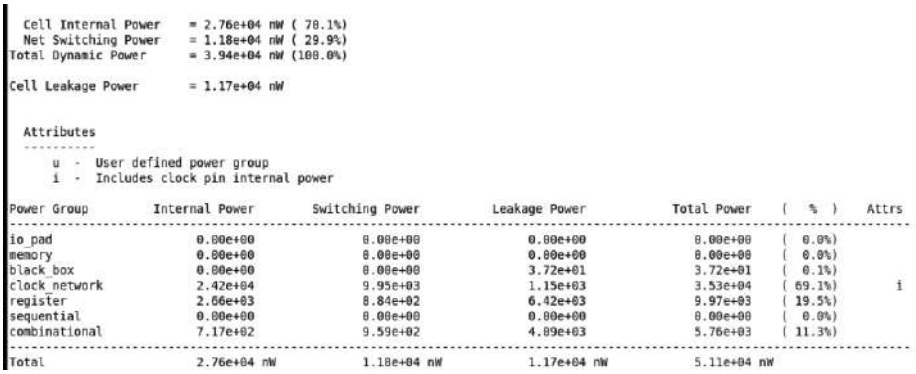
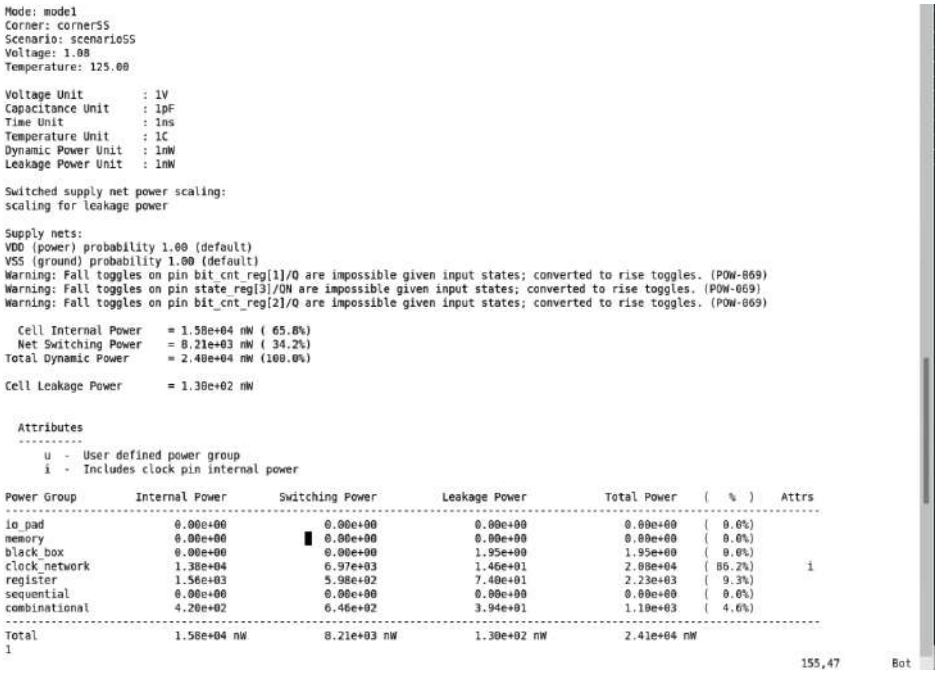


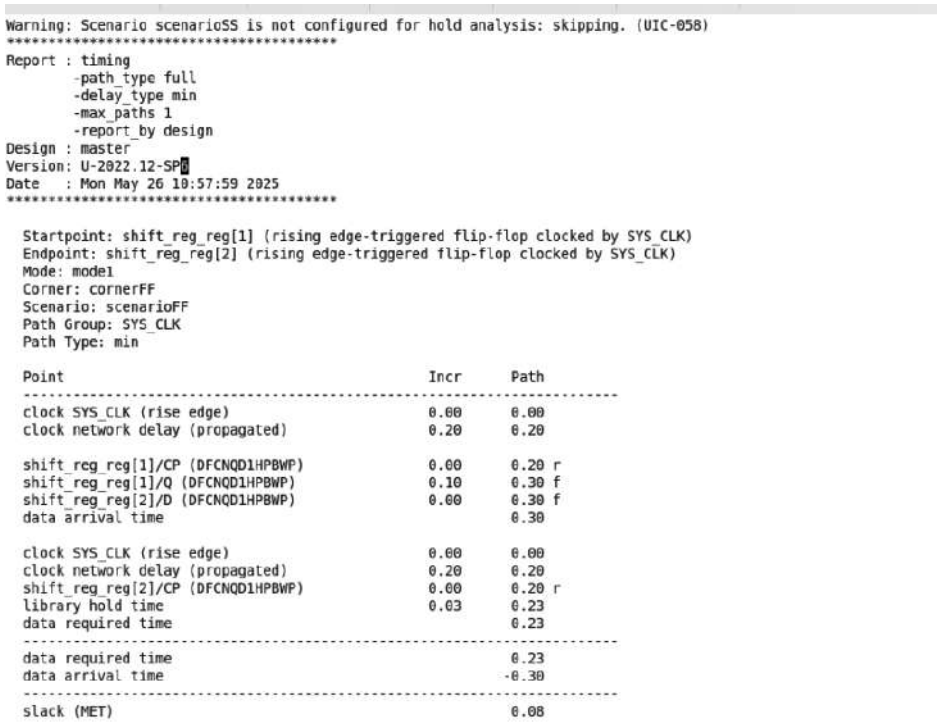
Figura 4.13. Análisis de Potencia TT. Elaboración propia.



```
Mode: mode1
Corner: cornerSS
Scenario: scenarioSS
Voltage: 1.08
Temperature: 125.00

Voltage Unit          : 1V
Capacitance Unit      : 1pF
Time Unit             : 1ns
Temperature Unit      : 1C
Dynamic Power Unit    : 1mW
Leakage Power Unit    : 1nW

Switched supply net power scaling:
scaling for leakage power

Supply nets:
VDD (power) probability 1.00 (default)
VSS (ground) probability 1.00 (default)
Warning: Fall toggles on pin bit_cnt_reg[1]/Q are impossible given input states; converted to rise toggles. (POW-869)
Warning: Fall toggles on pin state_reg[3]/ON are impossible given input states; converted to rise toggles. (POW-869)
Warning: Fall toggles on pin bit_cnt_reg[2]/Q are impossible given input states; converted to rise toggles. (POW-869)

  Cell Internal Power    = 1.58e+04 mW ( 65.8%)
  Net Switching Power    = 8.21e+03 mW ( 34.2%)
Total Dynamic Power      = 2.40e+04 mW (100.0%)

Cell Leakage Power       = 1.30e+02 nW


  Attributes
  ----------
      u  -  User defined power group
      i  -  Includes clock pin internal power

Power Group         Internal Power       Switching Power         Leakage Power           Total Power    (   %  )    Attrs
--------------------------------------------------------------------------------------------------------------------------
io_pad                 0.00e+00             0.00e+00              0.00e+00              0.00e+00    (  0.0%)
memory                 0.00e+00             0.00e+00              0.00e+00              0.00e+00    (  0.0%)
black_box              0.00e+00             0.00e+00              1.95e+00              1.95e+00    (  0.0%)
clock_network          1.38e+04             6.97e+03              1.46e+01              2.08e+04    ( 86.2%)       i
register               1.56e+03             5.98e+02              7.40e+01              2.23e+03    (  9.3%)
sequential             0.00e+00             0.00e+00              0.00e+00              0.00e+00    (  0.0%)
combinational          4.20e+02             6.46e+02              3.94e+01              1.10e+03    (  4.6%)
--------------------------------------------------------------------------------------------------------------------------
Total                  1.58e+04 nW          8.21e+03 nW           1.30e+02 nW           2.41e+04 nW
1
                                                                                                  155,47        Bot
```

Figura 4.14. Análisis de Potencia SS. Elaboración propia.

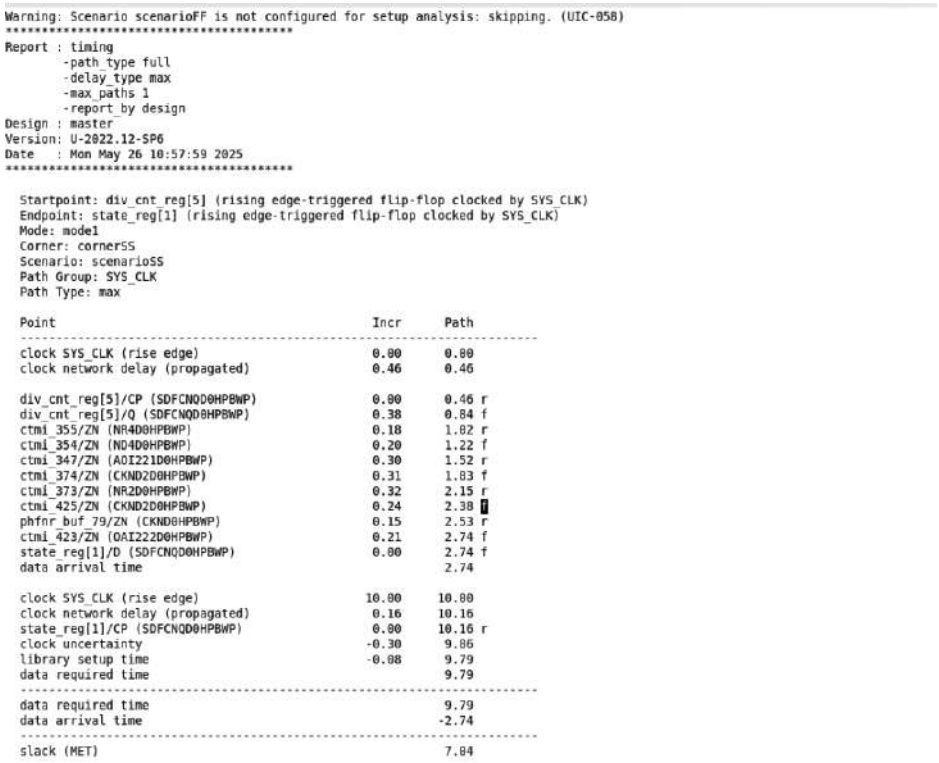
El reporte de hold de la Figura 4.15 muestra que, en la esquina rápida, la ruta más breve desde la salida Q de shift_reg_reg[1] hasta la entrada D de shift_reg_reg[2] tarda 0.30 ns en propagarse (0.20 ns de retraso de la red de reloj + 0.10 ns de salida del flop1). El flop de destino exige retener el valor antiguo durante 0.23 ns (0.20 ns de clock network + 0.03 ns de hold interno). Como el dato nuevo no puede llegar antes de 0.30 ns, existe un slack positivo de 0.08 ns, lo que garantiza que no se produzca una violación de hold. En otras palabras, el flop2 retiene el valor anterior lo suficiente para evitar que el dato siguiente sea capturado prematuramente.



```
Warning: Scenario scenarioSS is not configured for hold analysis: skipping. (UIC-058)
****************************************
Report : timing
        -path_type full
        -delay_type min
        -max_paths 1
        -report_by design
Design : master
Version: U-2022.12-SP1
Date   : Mon May 26 10:57:59 2025
****************************************

  Startpoint: shift_reg_reg[1] (rising edge-triggered flip-flop clocked by SYS_CLK)
  Endpoint: shift_reg_reg[2] (rising edge-triggered flip-flop clocked by SYS_CLK)
  Mode: mode1
  Corner: cornerFF
  Scenario: scenarioFF
  Path Group: SYS_CLK
  Path Type: min

  Point                                          Incr       Path
  -----------------------------------------------------------------------------
  clock SYS_CLK (rise edge)                      0.00       0.00
  clock network delay (propagated)               0.20       0.20

  shift_reg_reg[1]/CP (DFCNQD1HPBWP)             0.00       0.20 r
  shift_reg_reg[1]/Q (DFCNQD1HPBWP)              0.10       0.30 f
  shift_reg_reg[2]/D (DFCNQD1HPBWP)              0.00       0.30 f
  data arrival time                                         0.30

  clock SYS_CLK (rise edge)                      0.00       0.00
  clock network delay (propagated)               0.20       0.20
  shift_reg_reg[2]/CP (DFCNQD1HPBWP)             0.00       0.20 r
  library hold time                              0.03       0.23
  data required time                                        0.23
  -----------------------------------------------------------------------------
  data required time                                        0.23
  data arrival time                                        -0.30
  -----------------------------------------------------------------------------
  slack (MET)                                               0.08
```

Figura 4.15. Análisis de Hold. Elaboración propia.

El análisis de setup para la esquina lenta (SS) y 100 MHz se muestra en la Figura 4.15, muestra que la ruta más crítica desde la salida Q de div_cnt_reg[5] hasta la entrada D de state_reg[1], suma 2,74 ns de retraso de datos. el tiempo disponible para captura en state_reg[1], tras contar el retraso de la red de reloj (0.16 ns), la incertidumbre de reloj (-0.30 ns) y el tiempo de setup interno (-0.08 ns), es 9.79 ns. Por tanto queda un slack de 7.04 ns, confirmando que la ruta cumple el periódo de 10 ns sin violaciones de setup.

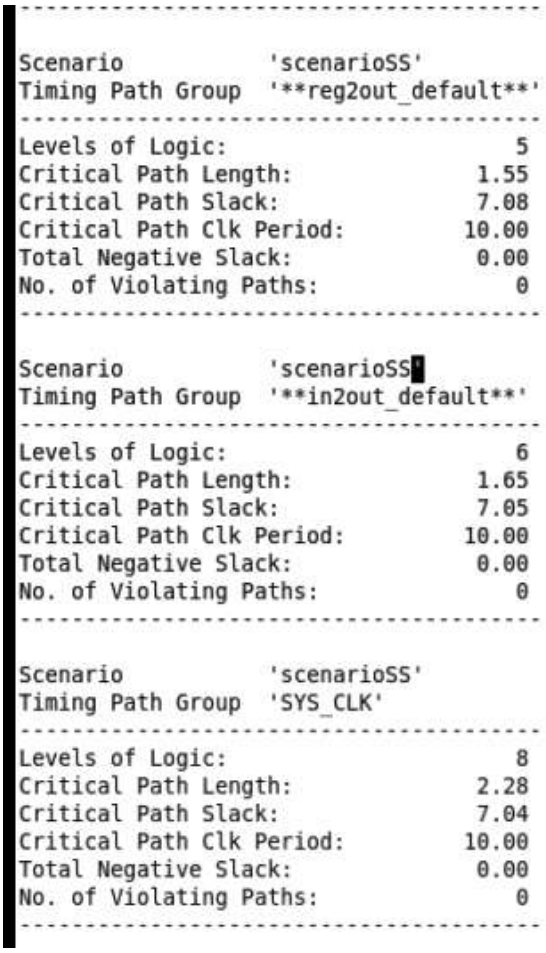


```
Warning: Scenario scenarioFF is not configured for setup analysis: skipping. (UIC-058)
****************************************
Report : timing
        -path_type full
        -delay_type max
        -max_paths 1
        -report_by design
Design : master
Version: U-2022.12-SP6
Date   : Mon May 26 10:57:59 2025
****************************************

  Startpoint: div_cnt_reg[5] (rising edge-triggered flip-flop clocked by SYS_CLK)
  Endpoint: state_reg[1] (rising edge-triggered flip-flop clocked by SYS_CLK)
  Mode: mode1
  Corner: cornerSS
  Scenario: scenarioSS
  Path Group: SYS_CLK
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock SYS_CLK (rise edge)                0.00       0.00
  clock network delay (propagated)         0.46       0.46

  div_cnt_reg[5]/CP (SDFCNQD0HPBWP)        0.00       0.46 r
  div_cnt_reg[5]/Q (SDFCNQD0HPBWP)         0.38       0.84 f
  ctmi_355/ZN (NR4D0HPBWP)                 0.18       1.02 r
  ctmi_354/ZN (ND4D0HPBWP)                 0.20       1.22 f
  ctmi_347/ZN (AOI221D0HPBWP)              0.30       1.52 r
  ctmi_374/ZN (CKND2D0HPBWP)               0.31       1.83 f
  ctmi_373/ZN (NR2D0HPBWP)                 0.32       2.15 r
  ctmi_425/ZN (CKND2D0HPBWP)               0.24       2.38 f
  phfnr_buf_79/ZN (CKND8HPBWP)             0.15       2.53 r
  ctmi_423/ZN (OAI222D0HPBWP)              0.21       2.74 f
  state_reg[1]/D (SDFCNQD0HPBWP)           0.00       2.74 f
  data arrival time                                   2.74

  clock SYS_CLK (rise edge)               10.00      10.00
  clock network delay (propagated)         0.16      10.16
  state_reg[1]/CP (SDFCNQD0HPBWP)          0.00      10.16 r
  clock uncertainty                       -0.30       9.86
  library setup time                      -0.08       9.79
  data required time                                  9.79
  -----------------------------------------------------------
  data required time                                  9.79
  data arrival time                                  -2.74
  -----------------------------------------------------------
  slack (MET)                                        7.04
```

Figura 4.16. Análisis de Setup. Elaboración propia.

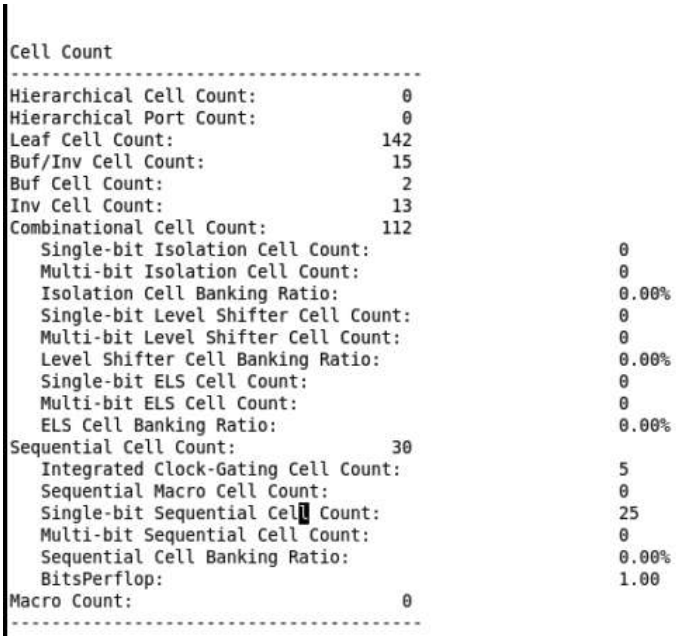
En las Figuras 4.17, 4.18 y 4.19 tenemos el reporte de las pruebas de calidad de los resultados en 65 nm, con respecto al Timing, en corner SS (1.08 V, 125 grados C) cumple con una frecuencia de 100 MHz sin slack negativo. las rutas críticas tardan entre 1.55 ns y 2.28 ns, dejando más de 7 ns de holgura.

Con el conteo de celdas, tenemos 142 instancias totales, la suma neta de celdas ocupan 564,80 um², con espacio para routing y power rails, el core completo abarca 948 um². La longitud total de pistas es 2 679.54 um, y no hay violaciones de diseño (156 nets limpias). En conjunto, el diseño cumple timing, usa área moderada y no infrige reglas de manufactura.



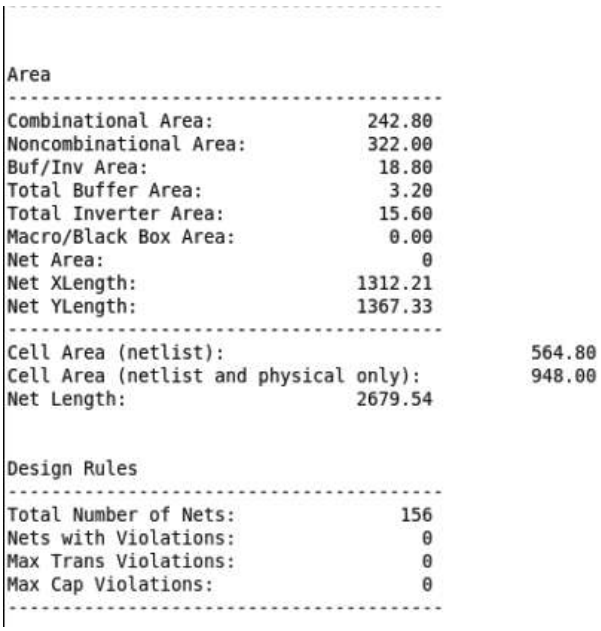
```
----------------------------------------

Scenario           'scenarioSS'
Timing Path Group  '**reg2out_default**'
----------------------------------------
Levels of Logic:                     5
Critical Path Length:             1.55
Critical Path Slack:              7.08
Critical Path Clk Period:        10.00
Total Negative Slack:             0.00
No. of Violating Paths:              0
----------------------------------------

Scenario           'scenarioSS'
Timing Path Group  '**in2out_default**'
----------------------------------------
Levels of Logic:                     6
Critical Path Length:             1.65
Critical Path Slack:              7.05
Critical Path Clk Period:        10.00
Total Negative Slack:             0.00
No. of Violating Paths:              0
----------------------------------------

Scenario           'scenarioSS'
Timing Path Group  'SYS_CLK'
----------------------------------------
Levels of Logic:                     8
Critical Path Length:             2.28
Critical Path Slack:              7.04
Critical Path Clk Period:        10.00
Total Negative Slack:             0.00
No. of Violating Paths:              0
----------------------------------------
```

Figura 4.17. QoR parte 1. Elaboración propia.



```
Cell Count
---------------------------------------
Hierarchical Cell Count:              0
Hierarchical Port Count:              0
Leaf Cell Count:                    142
Buf/Inv Cell Count:                  15
Buf Cell Count:                       2
Inv Cell Count:                      13
Combinational Cell Count:           112
   Single-bit Isolation Cell Count:                    0
   Multi-bit Isolation Cell Count:                     0
   Isolation Cell Banking Ratio:                       0.00%
   Single-bit Level Shifter Cell Count:                0
   Multi-bit Level Shifter Cell Count:                 0
   Level Shifter Cell Banking Ratio:                   0.00%
   Single-bit ELS Cell Count:                          0
   Multi-bit ELS Cell Count:                           0
   ELS Cell Banking Ratio:                             0.00%
Sequential Cell Count:               30
   Integrated Clock-Gating Cell Count:                 5
   Sequential Macro Cell Count:                        0
   Single-bit Sequential Cell Count:                   25
   Multi-bit Sequential Cell Count:                    0
   Sequential Cell Banking Ratio:                      0.00%
   BitsPerflop:                                        1.00
Macro Count:                         0
---------------------------------------
```

Figura 4.18. QoR parte 2. Elaboración propia.



```
---------------------------------------


Area
---------------------------------------
Combinational Area:              242.80
Noncombinational Area:           322.00
Buf/Inv Area:                     18.80
Total Buffer Area:                 3.20
Total Inverter Area:              15.60
Macro/Black Box Area:              0.00
Net Area:                             0
Net XLength:                     1312.21
Net YLength:                     1367.33
---------------------------------------
Cell Area (netlist):                           564.80
Cell Area (netlist and physical only):         948.00
Net Length:                      2679.54


Design Rules
---------------------------------------
Total Number of Nets:             156
Nets with Violations:               0
Max Trans Violations:               0
Max Cap Violations:                 0
---------------------------------------
```

Figura 4.19. QoR parte 3. Elaboración propia.

Capítulo 5

Conclusiones y Recomendaciones

5.1. Conclusiones

A lo largo del proyecto se logró diseñar, implementar y validar exitosamente un controlador maestro I2C compatible con el bus SIWA en tecnología TSMC 65 nm. En la etapa RTL se comprobó la FSM del maestro y del esclavo, luego la integración con el wrapper demostró transacciones correctas de lectura y escritura. Durante todo el flujo VLSI (síntesis, floorplan, placement, CTS y routing) no se registraron violaciones de DRC, y los análisis de timing arrojaron slacks positivos superiores a 7 ns sobre el periodo de 10 ns, garantizando cumplimiento de setup y hold en todos los corners.

El informe de área reveló que las 142 celdas estándar ocupan 564.8 um^2 (948 um^2 incluyendo whitespace y power rails), lo que deja margen suficiente para el routing. En cuanto a potencia, el consumo dinámico nominal de 39 uW y el leakage de 12 uW (23 % del total) confirman que la red de clock es la mayor fuente de disipación (aproximadamente 70 % del consumo dinámico). En el corner slow el dinámico cae a 24 uW y el leakage a 0.13 uW, mostrando robustez bajo condiciones extremas.

5.2. Recomendaciones

En futuras implementaciones, se recomienda optimizar el clock tree, incorporando clock-gating en secciones inactivas y revisar el balance de buffer para reducir el consumo dinámico, que supera el 60 % del total, estudiar técnicas de power-gating o multi-Vt en bloques poco activos para disminuir el 23 % de consumo estático en condiciones nominales.

Con respecto a la FSM maestro se recomienda extender la FSM para soportar transferencias multi-byte y manejo de errores con NACKs repetidos, fortaleciendo la robustez frente a condiciones adversas en el bus, aparte de realizar pruebas que integren más módulo del microcontrolador SIWA. Por otro lado, se recomienda realizar pruebas del módulo utlizando por completo los bits del bus de datos SIWA.

Por último se recomienda prepararse para generar el GDSII final, acompañandolo de una extracción parasítica y un último análisis de IR-drop para asegurar la integridad de la red de alimentación antes de la fabricación.

Bibliografía

[1] R. Garcia-Ramirez, A. Chacon-Rodriguez, R. Castro-Rodriguez, A. Arnaud, M. Miguez, J. Gak, R. Molina-Robles, G. Madrigal-Boza, M. Oviedo-Hernandez, E. Solera-Bolanos, D. Salazar- Sibaja, D. Sanchez- Jimenez, M. Fonseca-Rodriguez, J. Arrieta-Solorzano, R. Rimolo-Donadio (2021). "Siwa: a RISC-V RV321 based Micro-Controller for Implantable Medical Applications", IEEE Xplore

[2] NXP Semiconductors (2021) ÜM10204", En línea: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[3] R. Garcia-Ramirez, A. Chacon-Rodriguez, R. Castro-Rodriguez, A. Arnaud, M. Miguez, J. Gak, R. Molina-Robles, G. Madrigal-Boza, M. Oviedo-Hernandez, E. Solera-Bolanos, D. Salazar- Sibaja, D. Sanchez- Jimenez, M. Fonseca-Rodriguez, J. Arrieta-Solorzano, R. Rimolo-Donadio (2020), "Siwa: A custom RISC-V based system on chip (SoC) for low power medical applications"Microelectronics 98

[4] N. Samba (2024), Ünderstanding and Selecting in 2024: I2C, SPI, UART Explained", parlezvoustech, En línea: = https://www.parlezvoustech.com/en/comparaison-protocoles-communication-i2c-spi-uart/: :text=Circuit

[5] Roberto Molina-Robles, Ronny Garcia-Ramirez, Alfonso Chacon-Rodriguez, Renato Rimolo-Donadio, Alfredo Arnaud (2021), "Low-level algorithm for a software-emulated I2C I/O module in general purpose RISC-V based microcontrollers", IEEE URUCON

[6] Aseem garg, "What is VLSI Design Flow? An overview", En línea: = https://trainings.internshala.com/blog/what-is-vlsi-design-flow/

[7] Embedded Wala, "Protocol I2C", En línea: = https://embeddedwala.com/Blogs/DigitalCommunication/G
Started-with-I2C:-A-Beginners-Guide

[8] Yida, Ï2C Communication - All abour I2C with Diagrams", En línea: = https://www.seeedstudio.com/blog/2019/09/26/i2c-communication-interface-and-protocol-with-diagrams/

[9] Scott Campbell, "Basics of The I2C communication Protocol", En línea: = https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/google$_v$ignette

[10] nextpcb, "SPI vs I2C vs Uart", en línea: = https://www.nextpcb.com/blog/spi-i2c-uart

[11] Circuit Cove, .Exploring SystemVerilogs Key Features", En línea: = https://circuitcove.com/introduction-to-systemverilog-key-features/: :text=Exploring

[12] Miguel Angel Freire Rubio, Ïntroduccion al lenguaje VHDL", En línea: = https://www.cartagena99.com/recursos/electronica/apuntes/Manual

[13] Synopsys, Çustom Compiler", En línea: = https://www.synopsys.com/content/dam/synopsys/impleme compiler-ds.pdf

[14] Chris Edwards, "Siemens pushes DRC to the left", En línea: = https://www.techdesignforums.com/blog/2022/07/13/siemens-calibre-drc-realtime/

[15] Siemens, Çalibre Physical verification", En línea: = https://eda.sw.siemens.com/en-US/ic/calibre-design/physical-verification/

[16] Cadence, "Virtuoso Layout Suite", En línea: = https://www.cadence.com/en$_U$S/home/tools/custom–ic – analog – rf – design/layout – design/virtuoso – layout – suite.html

[17] Marius Monton Macian "Microcontroladres: Periféricos.En línea: https://openaccess.uoc.edu/server/api/core/bitstreams/4843b3bb-91eb-494e-8dca-28b31d2b0ba6/content

[18] Josh Scheinder, Ian Smalley (2024), "¿Qué es un microcontrolador?", En línea: https://www.ibm.com/mx-es/think/topics/microcontroller

[19] Synopsys, "VCS: Functional Verification Solution", En línea: https://www.synopsys.com/verification/simulation/vcs.html

[20] Synopsys, "Verdi", En línea: https://www-synopsys-com.translate.goog/verification/debug/verdi.html?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc

[21] Synopsys, "Fusion Compiler", En línea: https://www.synopsys.com/implementation-and-signoff/physical-implementation/fusion-compiler.html

[22] Randall Mariño Oviedo, Tiler Ureña Vargas (2024), Çelda Básica Fusion Compiler"

Anexo A: Módulo Master

Se muestra el encabezado del módulo maestro diseñado en SystemVerilog, módulo completo lo pueden encontrar en la siguiente área:

/mnt/vol_NFS_rh003/estudiantes/sporrasv/I2C/Ver_2_pnx/TEST_with_SIWA_BUS/Funcionando/BUS_READ_FINAL/Master_READ_FUNCIONANDO.sv

```
‘timescale 1ns/1ps
module i2c_master #(
    parameter CLK_FREQ_HZ = 100_000_000,   // frecuencia del reloj de
   sistema
    parameter I2C_FREQ_HZ = 100_000       // velocidad objetivo del bus
)(
    input  logic        clk,
    input  logic        rst_n,
    input  logic        start,       // pulso para iniciar transacci n
    input  logic [6:0]  addr,        // direcci n de 7 bits
    input  logic        rw,          // 0 = write, 1 = read
    input  logic [7:0]  wdata,       // dato a escribir
    output logic [7:0]  rdata,       // dato le do
    output logic        done,        // pulso fin de operaci n
    inout  tri          sda,
    output logic        scl
);
```

Listing 5.1: Encabezado del módulo `i2c_master`

Anexo B: Módulo Slave

A continuación se muestra el encabezado del módulo esclavo diseñado en SystemVerilog, módulo completo lo pueden encontrar en la siguiente área:

/mnt/vol_NFS_rh003/estudiantes/sporrasv/I2C/Ver_2_pnx/TEST_with_SIWA_BUS/Funcionando/BUS_READ_FINAL/slave_read_FUNCIONANDO.sv

```
`timescale 1ns/1ps

//
    ------------------------------------------------------------------------------

// i2c_slave_real2.sv
//  Slave I C 7-bit addr + R/W + 8-bit data,
//  FSM  nica  en clk, detecci n de flancos SCL + debug bit-a-bit.
//
    ------------------------------------------------------------------------------

module i2c_slave_real2 #(
    parameter logic [6:0] SLAVE_ADDR = 7'h42,
    parameter logic [7:0] INIT_BYTE  = 8'h00
)(
    input  logic      clk,
    input  logic      rst_n,
    inout  tri        sda,
    input  logic      scl
);
```

Listing 5.2: Encabezado del módulo `i2c_slave_real2`

Anexo C: Módulo wrapper

muestra el encabezado del módulo wrapper diseñado en SystemVerilog, módulo completo lo pueden encontrar en la siguiente área:

/mnt/vol_NFS_rh003/estudiantes/sporrasv/I2C/Ver_2_pnx/TEST_with_SIWA_BUS/Funcionando/BUS_READ_FINAL/master_wrapper.sv

```
`timescale 1ns/1ps
//------------------------------------------------------------
// Wrapper SIWA     i2c_master
//------------------------------------------------------------
module i2c_master_if #(
    parameter int CLK_FREQ_HZ = 100_000_000,
    parameter int I2C_FREQ_HZ = 100_000
)(
    input  logic        clk,
    input  logic        rst_n,

    // Bus SIWA
    input  logic        pdndg_mbc,
    input  logic [31:0] D_pop_mbc,
    output logic        pop_mbc,
    output logic        push_mbc,
    output logic [31:0] D_push_mbc,

    // L neas f sicas I C
    inout  tri          sda,
    output logic        scl,

    // Debug
    output logic        done,
    output logic [3:0]  dbg_bit_cnt,
    output logic [7:0]  dbg_wdata,
    output logic [7:0]  dbg_rdata
);
```

Listing 5.3: Encabezado del módulo `i2c_master_if`

Anexo D: Testbench Final y archivos Fusion Compiler

Para las la testbench final utilizada se encuentra en el área:
/mnt/vol_NFS_rh003/estudiantes/sporrasv/I2C/Ver_2_pnx/TEST_with_SIWA_BUS/Funcionando/BUS_READ_FINAL/READ.sv

Para los archivos y resultados obtenidos utilizando Fusion Compiler se puede acceder al área:
/mnt/vol_NFS_rh003/estudiantes/sporrasv/I2C/Ver_2_pnx/TEST_with_SIWA_BUS/Funcionando/BUS_READ_FINAL/fusion