

UNIVERSIDAD DE COSTA RICA

Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0424 - Laboratorio de Circuitos Digitales

Reporte de Proyecto Final:

Diseño e implementación en FPGA de un sistema de comunicación I²C y generación de barrido de tensión para control digital en la plataforma Nexys A7

Profesor:

MSc. Marco Villalta Fallas

Estudiantes:

Gabriel Siles Chaves - C17530

Jorge Loría Chaves - C04406

Grupo 01

I ciclo de 2025

Índice

1. Introducción	2
2. Objetivos	2
2.1. Objetivo General	2
2.2. Objetivos específicos	2
3. Alcances	2
3.1. Desarrollo del Sistema I ² C	2
3.2. Desarrollo de Barrido de Tensión	2
3.3. Metodología de Implementación	3
3.4. Interfaz de Expansión para Futuras Integraciones	3
4. Marco Teórico	3
4.1. Líneas de Comunicación: SDA y SCL	3
4.1.1. Modo de Operación: Velocidades del Bus	4
4.2. Concepto de Maestro y Esclavo	4
4.2.1. Manejo de Dispositivos en un Bus I ² C	4
4.3. Protocolo de Comunicación I ² C	5
4.3.1. Formato de los Datos: START, STOP, ACK, NACK	5
4.4. Transmisión de Bytes	5
4.5. Arbitraje y Sincronización en un Sistema Multi-Maestro	6
4.6. Beneficios para Diseñadores y Aplicaciones del I ² C	6
4.6.1. Beneficios para Diseñadores	7
4.6.2. Ámbitos de Aplicación	7
4.7. Limitaciones y Desafíos del I ² C	8
4.7.1. Velocidades de Transferencia Limitadas	8
5. Desarrollo	8
6. Análisis de Resultados	18
7. Conclusiones y Recomendación	19
Referencias	20

1. Introducción

El presente proyecto se diseña e implementa un sistema en FPGA capaz de habilitar la comunicación I²C y generar un barrido de tensión mediante lógica programada en hardware, utilizando como plataforma la tarjeta Nexys A7. Esta implementación sirve como base funcional para, en etapas posteriores, integrarse con un driver de lente líquido y desarrollar un sistema de control completo. El enfoque se centra en la infraestructura digital necesaria para establecer un canal de comunicación confiable con periféricos compatibles con I²C, así como en la generación controlada de señales de salida que puedan adaptarse a diversas aplicaciones. El uso de FPGA permite aprovechar ventajas como la alta velocidad de procesamiento, el paralelismo y la capacidad de personalizar el hardware según los requerimientos del sistema final. Esta etapa inicial constituye un paso clave hacia el desarrollo de un controlador embebido orientado a aplicaciones ópticas, biomédicas o industriales. El código desarrollada para la implementación del proyecto se encuentra en el siguiente repositorio: <https://git.ucr.ac.cr/GABRIEL.SILES/ie0424-gj>.

2. Objetivos

2.1. Objetivo General

Diseñar e implementar un sistema en FPGA que habilite la comunicación I²C mediante pines físicos y permita generar un barrido controlado de tensión como parte de una lógica programada en hardware.

2.2. Objetivos específicos

- Implementar en FPGA un módulo maestro de comunicación I²C, capaz de generar señales por los pines correspondientes (SDA y SCL) cumpliendo con los requisitos del protocolo.
- Diseñar un sistema de barrido de tensión en hardware, que entregue valores variables por pines de salida según una secuencia determinada o programable dentro los valores operables de la FPGA.
- Establecer una arquitectura base que permita, en futuras etapas, integrar controladores externos en C o RISC-V, utilizando la comunicación I²C ya habilitada para enviar datos o comandos.

3. Alcances

3.1. Desarrollo del Sistema I²C

El sistema I²C implementado permitirá una comunicación en modo maestro-esclavo. Su funcionamiento será verificado mediante pruebas internas, utilizando módulos esclavos conectados directamente en la FPGA y no dispositivos externos.

3.2. Desarrollo de Barrido de Tensión

Se incluirá un sistema de barrido de tensión controlado mediante una máquina de estados en la FPGA, capaz de generar señales de salida con valores variables según una secuencia lógica.

3.3. Metodología de Implementación

El diseño y la simulación se llevarán a cabo utilizando el entorno Vivado y el lenguaje de descripción de hardware Verilog. El proyecto incluirá etapas de simulación y pruebas físicas en la FPGA para validar el correcto funcionamiento de los módulos desarrollados.

3.4. Interfaz de Expansión para Futuras Integraciones

Se establecerá una arquitectura base que permitirá, en futuras versiones, integrar controladores externos (como microcontroladores u otros sistemas) utilizando la comunicación I²C, lo cual facilitará la expansión del sistema sin modificar su estructura interna.

4. Marco Teórico

El bus I²C (Inter-Integrated Circuit) es un protocolo de comunicación bidireccional y serie desarrollado por Philips Semiconductors (actualmente NXP Semiconductors) en 1982. Este protocolo permite la transferencia de datos entre dispositivos electrónicos utilizando solo dos líneas: la línea de datos serial (SDA) y la línea de reloj serial (SCL). I²C se ha convertido en un estándar ampliamente utilizado en aplicaciones de comunicación en sistemas embebidos debido a su simplicidad y flexibilidad. (1)

El principal atractivo del bus I²C radica en su capacidad para permitir la comunicación entre múltiples dispositivos utilizando un número reducido de pines, lo que facilita el diseño y la implementación de circuitos electrónicos compactos y de bajo costo. A pesar de su simplicidad, I²C soporta comunicaciones en sistemas complejos y con múltiples dispositivos conectados al mismo bus, funcionando bajo una arquitectura maestro-esclavo. (2)

I²C ha sido adoptado en una amplia variedad de aplicaciones, desde controladores de pantalla, sensores, dispositivos de almacenamiento, hasta microcontroladores y sistemas de comunicación de datos. Además, a lo largo de los años, se han introducido mejoras en el protocolo, como modos de alta velocidad y mejoras en la gestión de los dispositivos, lo que ha permitido ampliar su uso en sistemas más avanzados. (2)

Este protocolo no solo se ha limitado a aplicaciones de consumo, sino que también ha sido adaptado para ser utilizado en otras arquitecturas de comunicación como el *System Management Bus* (SMBus), el *Power Management Bus* (PMBus), y el *Intelligent Platform Management Interface* (IPMI), donde se amplían las capacidades del I²C para cubrir necesidades específicas de gestión y control. (3)

En el contexto de este proyecto, el objetivo general es diseñar e implementar un sistema en FPGA que habilite la comunicación I²C mediante pines físicos y permita la generación de un barrido controlado de tensión, todo ello dentro de una lógica programada en hardware. Esto implicará la creación de un módulo maestro de comunicación I²C, capaz de generar señales en los pines correspondientes (SDA y SCL) cumpliendo con los requisitos del protocolo.

4.1. Líneas de Comunicación: SDA y SCL

El bus I²C se compone de dos líneas principales:

- **SDA (Serial Data Line):** Esta línea se utiliza para la transmisión y recepción de datos. Es una línea bidireccional, lo que significa que puede enviar y recibir información entre el maestro

y los dispositivos esclavos.

- **SCL (Serial Clock Line):** Esta línea es utilizada para la señal de reloj, generada por el maestro, para sincronizar las transferencias de datos en el bus. Todos los dispositivos conectados al bus deben seguir esta señal de reloj para asegurar que los datos sean leídos o escritos de manera correcta.

Ambas líneas, SDA y SCL, están conectadas a una fuente de alimentación a través de resistencias de pull-up, lo que significa que las líneas normalmente se mantienen en un nivel alto (HIGH) y solo cambian a un nivel bajo (LOW) cuando un dispositivo en el bus realiza una acción.(3)

4.1.1. Modo de Operación: Velocidades del Bus

El bus I²C opera en diferentes modos de velocidad, lo que permite adaptarse a las necesidades de comunicación de distintos dispositivos. Las velocidades más comunes son las siguientes:(3)

- **Modo Estándar:** Opera a una velocidad de hasta 100 kbit/s. Este modo es suficiente para muchas aplicaciones simples de comunicación, donde las velocidades de transmisión no son un factor crítico.
- **Modo Rápido (Fast Mode):** Este modo permite una velocidad de hasta 400 kbit/s, proporcionando un rendimiento más rápido en aplicaciones que requieren mayor capacidad de transferencia de datos.
- **Fast-mode Plus (Fm+):** Este modo incrementa aún más la velocidad de transferencia, alcanzando hasta 1 Mbit/s. Se utiliza cuando se requieren transferencias de datos aún más rápidas.
- **Modo Alta Velocidad (High Speed Mode):** El modo de alta velocidad permite velocidades de hasta 3.4 Mbit/s, lo cual es útil para aplicaciones que requieren un rendimiento extremadamente rápido.

El bus I²C es compatible hacia atrás, lo que significa que los dispositivos pueden operar a diferentes velocidades, siempre que se utilicen modos compatibles con los dispositivos presentes en el sistema.

4.2. Concepto de Maestro y Esclavo

En una comunicación I²C, siempre existe al menos un dispositivo maestro y uno o más dispositivos esclavos. El maestro es el único dispositivo capaz de generar la señal de reloj (SCL) y de iniciar las transferencias de datos en el bus. Los esclavos, por su parte, son dispositivos que responden a las solicitudes del maestro. Un dispositivo esclavo no puede iniciar la comunicación, sino que debe esperar a que el maestro lo dirija.

Cada dispositivo esclavo en el bus tiene una dirección única que le permite ser identificado por el maestro. El maestro utiliza estas direcciones para dirigirse específicamente a cada dispositivo en el bus durante las transferencias de datos.(3)

4.2.1. Manejo de Dispositivos en un Bus I²C

Los dispositivos conectados al bus I²C pueden ser tanto transmisores como receptores de datos. Durante una transferencia de datos, el maestro inicia la comunicación y controla el flujo de información, mientras que los esclavos responden en función de las instrucciones que reciben. Los dispositivos

esclavos pueden estar configurados para recibir o transmitir datos según el tipo de operación solicitada por el maestro.

El protocolo I²C también incluye mecanismos de control de flujo como la señal de **ACK** (Acknowledge) y **NACK** (Not Acknowledge), que garantizan que cada byte de datos enviado se ha recibido correctamente antes de continuar con la transferencia.(2)

Este tipo de comunicación hace posible que múltiples dispositivos compartan el mismo bus, lo que reduce la complejidad del sistema y la cantidad de pines necesarios para la conexión, haciendo que el bus I²C sea ideal para sistemas con muchos dispositivos periféricos conectados.

4.3. Protocolo de Comunicación I²C

El protocolo I²C define el formato y la secuencia de datos que se transfieren entre los dispositivos conectados al bus. A continuación, se describe cómo se lleva a cabo la transmisión de datos dentro del protocolo I²C, incluyendo el formato de los datos, la transmisión de bytes y los mecanismos de arbitraje y sincronización en un sistema multi-maestro.

4.3.1. Formato de los Datos: START, STOP, ACK, NACK

En I²C, cada transferencia de datos comienza y termina con condiciones especiales que garantizan la correcta comunicación entre los dispositivos. Las condiciones más importantes son:(3)

- **START (S)**: La transferencia de datos en el bus I²C comienza con una condición de inicio. Esta condición se genera cuando la línea de datos (**SDA**) cambia de nivel alto a bajo mientras la línea de reloj (**SCL**) se mantiene alta. La condición START es generada por el maestro para iniciar la comunicación con un dispositivo esclavo.
- **STOP (P)**: Una transferencia de datos se termina con una condición de parada, la cual ocurre cuando la línea **SDA** cambia de nivel bajo a alto mientras la línea **SCL** está alta. Esta señal indica al bus que la transferencia ha finalizado y que el bus está disponible para nuevas comunicaciones.
- **ACK (Acknowledge)**: Después de cada byte de datos transferido, el receptor (ya sea maestro o esclavo) debe enviar una señal de acknowledge (**ACK**) para confirmar que ha recibido el byte correctamente. Esto se realiza bajando la línea **SDA** durante el noveno ciclo del reloj.
- **NACK (Not Acknowledge)**: Si el receptor no puede recibir más datos o si ha ocurrido un error, enviará una señal de not acknowledge (**NACK**). Esto se indica manteniendo la línea **SDA** en nivel alto durante el noveno ciclo de reloj.

4.4. Transmisión de Bytes

Los datos en el bus I²C se transfieren en bloques de 8 bits, conocidos como bytes. La transmisión de un byte sigue el siguiente formato:(3)

- El maestro (o esclavo en algunos casos) coloca los 8 bits de datos en la línea **SDA**.
- El bit más significativo (**MSB**) se transmite primero.
- Después de la transmisión del byte, el receptor debe enviar un **ACK** o **NACK**.

- Si el receptor envía un ACK, el maestro o esclavo puede continuar con la siguiente transferencia de byte. Si se envía un NACK, la transferencia de datos se detiene.

Una vez que el byte es transferido, la línea **SCL** se alterna, generando un pulso de reloj para cada bit. Durante el ciclo alto de **SCL**, los dispositivos leen los datos en **SDA**, mientras que durante el ciclo bajo, el maestro coloca los bits de datos en **SDA**.

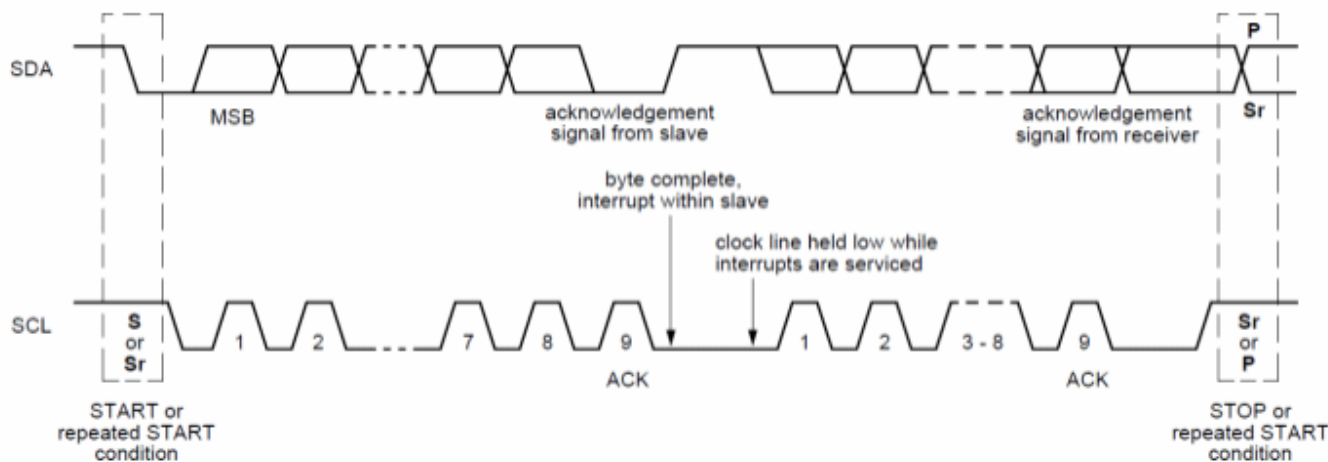


Figura 1: Transferencia de datos. Imagen recuperada de: N. Semiconductors (2)

4.5. Arbitraje y Sincronización en un Sistema Multi-Maestro

En sistemas donde existen múltiples dispositivos maestros, es necesario un mecanismo de arbitraje y sincronización para asegurar que solo un maestro controle el bus en un momento dado. Si más de un maestro intenta generar un **START** al mismo tiempo, se produce una situación de arbitraje.

El arbitraje en **I²C** se realiza bit a bit. Si un maestro intenta enviar un bit y detecta que otro maestro está enviando un bit diferente, el maestro que haya enviado el bit incorrecto se retira de la transmisión y se convierte en un esclavo. De esta manera, se evita la colisión de datos y se asegura que solo un maestro siga controlando el bus.(3)

Además, la sincronización entre los dispositivos en un sistema multi-maestro se realiza mediante la conexión de las líneas de reloj (**SCL**) de todos los maestros. Si un maestro genera un reloj más rápido que otro, el maestro más lento puede extender el ciclo de reloj, lo que garantiza que todos los dispositivos se mantengan sincronizados.(3)

4.6. Beneficios para Diseñadores y Aplicaciones del I²C

El bus **I²C** ha sido ampliamente adoptado en la industria electrónica debido a su eficiencia y simplicidad. Fue diseñado como una solución de bajo costo y fácil implementación para la comunicación entre circuitos integrados (ICs) en sistemas embebidos. Gracias a su arquitectura ligera y versátil, **I²C** ofrece múltiples ventajas a los diseñadores de hardware y firmware, convirtiéndose en un estándar de facto en numerosos dispositivos comerciales e industriales.(4)

4.6.1. Beneficios para Diseñadores

Entre las principales ventajas del bus I²C para el diseño de sistemas electrónicos se destacan las siguientes:(5)

- **Simplicidad del hardware:** Solo requiere dos líneas de conexión (SDA y SCL) para establecer comunicación entre múltiples dispositivos, lo que reduce significativamente la complejidad del diseño de placas y el número de pines necesarios en el microcontrolador.
- **Escalabilidad:** I²C permite conectar múltiples dispositivos esclavos a un mismo bus, lo que facilita la expansión de funcionalidades sin necesidad de rediseñar la arquitectura base del sistema.
- **Compatibilidad universal:** Existe una amplia variedad de dispositivos compatibles con I²C disponibles en el mercado, incluyendo sensores, memorias, ADC/DAC, y controladores de pantalla, lo que facilita la integración con diferentes tecnologías.
- **Interfaz integrada en microcontroladores:** La mayoría de los microcontroladores modernos incluyen módulos I²C en hardware, lo cual permite una implementación directa sin requerir codificación de bajo nivel para emulación.
- **Soporte para comunicación multi-maestro:** Aunque no es común en todos los diseños, el protocolo I²C incluye mecanismos de arbitraje y sincronización para permitir múltiples maestros en el mismo bus sin colisiones de datos.

4.6.2. Ámbitos de Aplicación

Gracias a sus características, el bus I²C es utilizado en una gran variedad de aplicaciones dentro de la electrónica digital. Algunos ejemplos incluyen:(3)

- **Sensores:** Sensores de temperatura, acelerómetros, giroscopios, magnetómetros, entre otros, utilizan I²C para transmitir datos de medición a un microcontrolador.
- **Memorias EEPROM:** Dispositivos de memoria no volátil usan I²C para almacenar configuraciones o datos persistentes de manera sencilla.
- **Microcontroladores y SoCs:** El I²C se utiliza como canal de comunicación entre procesadores y periféricos internos o externos, como convertidores A/D y D/A.
- **Controladores de pantalla:** Control de brillo, contraste y color en pantallas LCD y OLED, así como configuración de resoluciones, se maneja frecuentemente mediante I²C.
- **Relojes en tiempo real (RTC):** Muchos RTCs utilizan I²C para entregar información horaria al sistema principal.
- **Configuración de hardware en servidores y telecomunicaciones:** A través de buses como IPMI o SMBus, basados en I²C, se realiza monitoreo de temperatura, voltajes y velocidad de ventiladores.

4.7. Limitaciones y Desafíos del I²C

4.7.1. Velocidades de Transferencia Limitadas

Una de las principales limitaciones de I²C es la velocidad de transferencia. Aunque I²C es adecuado para muchas aplicaciones de baja velocidad, no puede competir con otros protocolos de comunicación como SPI en términos de velocidad.⁽⁶⁾

- El bus I²C soporta velocidades de hasta 3.4 Mbit/s en su modo de alta velocidad (Hs), lo que es adecuado para la mayoría de aplicaciones de sistemas embebidos, pero insuficiente para aplicaciones que requieren una gran cantidad de transferencia de datos en tiempo real.
- La velocidad de transferencia también puede verse afectada por la cantidad de dispositivos conectados al bus, ya que una mayor carga de capacitancia puede ralentizar la comunicación.
- En comparación con otros buses como SPI, que puede operar a velocidades mucho mayores (hasta 10 Mbit/s o más), I²C es mucho más lento y puede no ser adecuado para aplicaciones de alto rendimiento.

5. Desarrollo

1. Fase de investigación:

Inicialmente, se realizó una extensa búsqueda de documentación del protocolo I²C, para tener claro su correcto funcionamiento y no omitir nada a la hora de su implementación. Se usó como base la información y código realizado para el curso de Circuitos Digitales 2.

2. Open Core:

Como punto de partida para la implementación del módulo I²C en la FPGA, se utilizaron diversos recursos disponibles en OpenCores. Los siguientes proyectos proporcionaron códigos y módulos preexistentes que sirvieron como base para desarrollar los módulos de comunicación I²C:

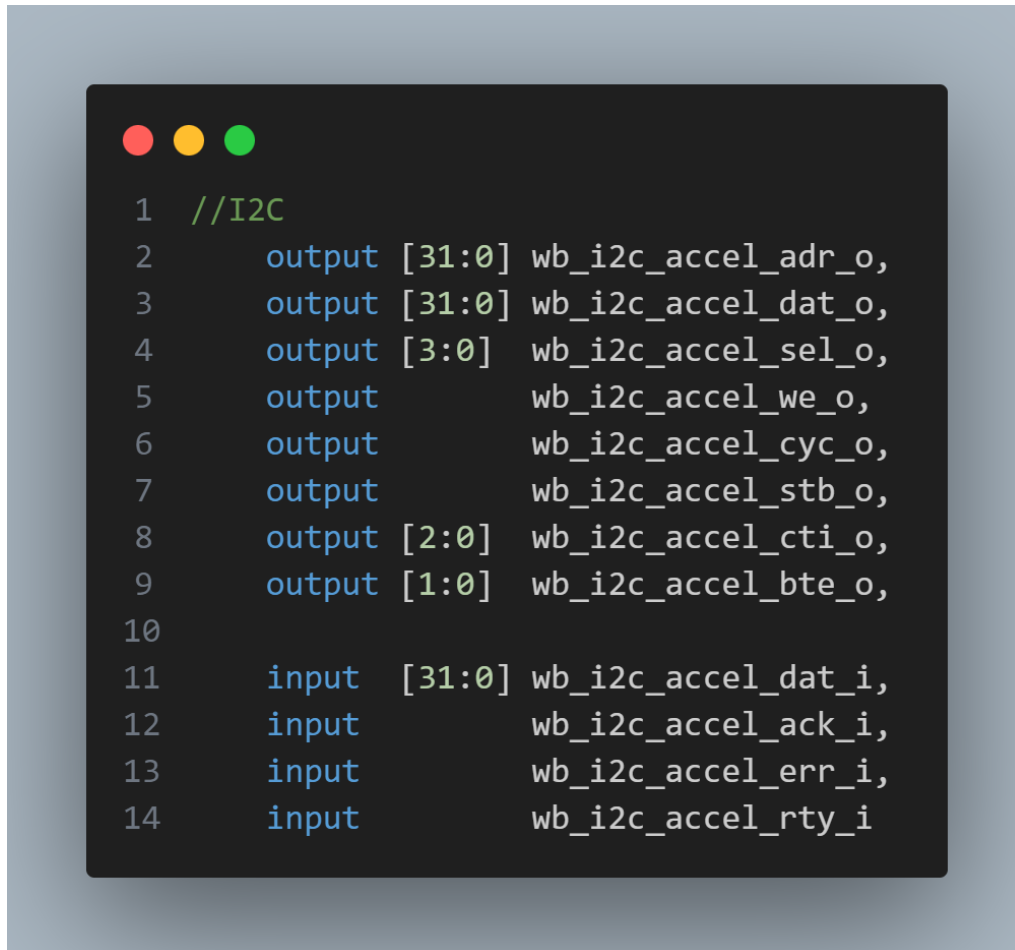
- I2C Slave Core ⁽⁷⁾
- I2C Core ⁽⁸⁾
- I2C Master/Slave Core ⁽⁹⁾
- I2C Master/Slave Core (versión mejorada) ⁽¹⁰⁾
- I2C to Wishbone Interface ⁽¹¹⁾
- I2C Parallel Master ⁽¹²⁾
- IICMB (I2C Message Block) ⁽¹³⁾

Se revisaron cada uno de estos módulos disponibles, observando su arquitectura y funcionalidad. Finalmente, se seleccionó el módulo I2C Slave Core como punto de partida para la implementación del proyecto.

3. Inicio de la implementación:

Master:

Se inicio con la implementación del master, para ello en vivado modificamos el archivo de `wb_intercon.v` y se agregaron las entradas y salidas que iba a tener el protocolo I^2C :

A screenshot of a code editor window with a dark background and light-colored text. The code is Verilog, defining I2C master signals. It includes output signals for address, data, select, write enable, clock, stop, control, and busy, and input signals for data, acknowledge, error, and ready. The code is numbered from 1 to 14.

```
1 //I2C
2     output [31:0] wb_i2c_accel_adr_o,
3     output [31:0] wb_i2c_accel_dat_o,
4     output [3:0]  wb_i2c_accel_sel_o,
5     output        wb_i2c_accel_we_o,
6     output        wb_i2c_accel_cyc_o,
7     output        wb_i2c_accel_stb_o,
8     output [2:0]  wb_i2c_accel_cti_o,
9     output [1:0]  wb_i2c_accel_bte_o,
10
11     input  [31:0] wb_i2c_accel_dat_i,
12     input        wb_i2c_accel_ack_i,
13     input        wb_i2c_accel_err_i,
14     input        wb_i2c_accel_rty_i
```

Figura 2: Definición de entradas y salidas del wishbone

Seguidamente, se modificó el archivo de `intercon.vh` y se definieron los wires a utilizar y luego se realizó la instancia del con el wishbone. Además se agregan las señales al mux de wishbone:

```

1 wire [31:0] wb_m2s_i2c_accel_adr; // Dirección para I2C
2 wire [31:0] wb_m2s_i2c_accel_dat; // Datos de entrada
3 wire [3:0]  wb_m2s_i2c_accel_sel; // Selección de bytes
4 wire      wb_m2s_i2c_accel_we;   // Señal de escritura
5 wire      wb_m2s_i2c_accel_cyc;  // Señal de ciclo
6 wire      wb_m2s_i2c_accel_stb;  // Señal de strobe
7 wire [2:0] wb_m2s_i2c_accel_cti; // Tipo de ciclo
8 wire [1:0] wb_m2s_i2c_accel_bte; // Tipo de transferencia
9
10 wire [31:0] wb_s2m_i2c_accel_dat; // Datos leídos desde I2C
11 wire      wb_s2m_i2c_accel_ack;  // Acknowledge de I2C
12 wire      wb_s2m_i2c_accel_err;  // Error de I2C
13 wire      wb_s2m_i2c_accel_rty;  // Retry de I2C

```

Figura 3: Definición de variables

```

1 // Conexión para el I2C
2 .wb_i2c_accel_adr_o (wb_m2s_i2c_accel_adr), // Dirección del I2C
3 .wb_i2c_accel_dat_o (wb_m2s_i2c_accel_dat), // Datos a escribir
4 .wb_i2c_accel_sel_o (wb_m2s_i2c_accel_sel), // Selección de bytes
5 .wb_i2c_accel_we_o (wb_m2s_i2c_accel_we),  // Señal de escritura
6 .wb_i2c_accel_cyc_o (wb_m2s_i2c_accel_cyc), // Señal de ciclo
7 .wb_i2c_accel_stb_o (wb_m2s_i2c_accel_stb), // Señal de strobe
8 .wb_i2c_accel_cti_o (wb_m2s_i2c_accel_cti), // Tipo de ciclo
9 .wb_i2c_accel_bte_o (wb_m2s_i2c_accel_bte), // Tipo de transferencia
10 .wb_i2c_accel_dat_i (wb_s2m_i2c_accel_dat), // Datos leídos desde I2C
11 .wb_i2c_accel_ack_i (wb_s2m_i2c_accel_ack), // Acknowledge de I2C
12 .wb_i2c_accel_err_i (wb_s2m_i2c_accel_err), // Error de I2C
13 .wb_i2c_accel_rty_i (wb_s2m_i2c_accel_rty) // Retry de I2C

```

Figura 4: Instancia con el wishbone

```

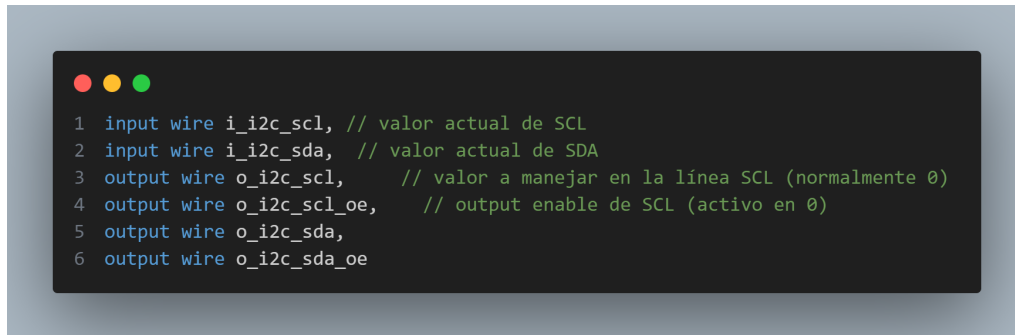
1 wb_mux
2   #(.num_slaves (12),
3     .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100,
4                  32'h00001200, 32'h00001240, 32'h00001280, 32'h000012c0,
5                  32'h00001400, 32'h00001800, 32'h00002000, 32'h00002800}),
6     .MATCH_MASK ({32'hffffff00, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0,
7                  32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0,
8                  32'hffffffc0, 32'hffffffc0, 32'hfffff000, 32'hfffff000}))
9   wb_mux_io (
10    .wbs_adr_o ({..., wb_i2c_accel_adr_o}),

```

```
11 .wbs_dat_o ({..., wb_i2c_accel_dat_o}),
12 .wbs_sel_o ({..., wb_i2c_accel_sel_o}),
13 .wbs_we_o  ({..., wb_i2c_accel_we_o}),
14 .wbs_cyc_o ({..., wb_i2c_accel_cyc_o}),
15 .wbs_stb_o ({..., wb_i2c_accel_stb_o}),
16 .wbs_cti_o ({..., wb_i2c_accel_cti_o}),
17 .wbs_bte_o ({..., wb_i2c_accel_bte_o}),
18 .wbs_dat_i ({..., wb_i2c_accel_dat_i}),
19 .wbs_ack_i ({..., wb_i2c_accel_ack_i}),
20 .wbs_err_i ({..., wb_i2c_accel_err_i}),
21 .wbs_rty_i ({..., wb_i2c_accel_rty_i})
22 );
```

Listing 1: Instancia del módulo wb_mux con 12 esclavos

Se prosiguió con el módulo de swervolf_core.v donde se definen las entradas y salidas con las que se va a contar en el protocolo. Además, se definieron las señales del master, se expande la señal de salida y se instancia el módulo I^2C



```
1 input wire i_i2c_scl, // valor actual de SCL
2 input wire i_i2c_sda, // valor actual de SDA
3 output wire o_i2c_scl, // valor a manejar en la línea SCL (normalmente 0)
4 output wire o_i2c_scl_oe, // output enable de SCL (activo en 0)
5 output wire o_i2c_sda,
6 output wire o_i2c_sda_oe
```

Figura 5: Definición de entradas y salidas

```

1 i2c_master_top i2c_inst (
2     // Wishbone slave interface
3     .wb_clk_i  (clk), // Reloj
4     .wb_rst_i  (wb_rst), // Reset
5     .arst_i    (1'b0), // No se usa
6     .wb_adr_i  (wb_m2s_i2c_accel_adr[2:0]), // Dirección
7     .wb_dat_i  (wb_m2s_i2c_accel_dat[7:0]), // Datos a escribir
8     .wb_we_i   (wb_m2s_i2c_accel_we), // Señal de escritura
9     .wb_cyc_i  (wb_m2s_i2c_accel_cyc), // Señal de ciclo válido
10    .wb_stb_i   (wb_m2s_i2c_accel_stb), // Señal de strobe
11    .wb_dat_o   (i2c_rdt), // Datos leídos desde I2C
12    .wb_ack_o   (wb_s2m_i2c_accel_ack), // Acknowledgment
13    .wb_inta_o  (i2c_irq), // Interrupción de I2C
14
15    // Pads I2C (líneas físicas)
16    .scl_pad_i  (i_i2c_scl),
17    .scl_pad_o  (o_i2c_scl),
18    .scl_padoen_o (o_i2c_scl_oe),
19    .sda_pad_i  (i_i2c_sda),
20    .sda_pad_o  (o_i2c_sda),
21    .sda_padoen_o (o_i2c_sda_oe)
22 ); // SDA de retorno

```

Figura 6: Instancia del protocolo I^2C

```

1 // Señales para el máster I2C
2 wire [7:0] i2c_rdt; // Datos leídos por el I2C
3 wire i2c_irq;      // Interrupción de I2C
4 // Expandimos la salida de 8 bits a 32 bits
5 assign wb_s2m_i2c_accel_dat = {24'd0, i2c_rdt};
6
7 // Que no se usan en el modulo master de I2C
8 assign wb_s2m_i2c_accel_err = 1'b0;
9 assign wb_s2m_i2c_accel_rty = 1'b0;

```

Figura 7: Señales del master y expansión de la salida

Se sigue con el archivo `rvfpganexys.sv` en donde debemos definir los cables, las entradas y salidas que cuenta el protocolo. Luego, se debe instanciar en el `swervolf` y se hacen dos asigna-

ciones (`i2c_scl_i` y `i2c_sda_i`) simplemente capturan el valor presente en las líneas físicas, permitiendo que la lógica interna lea el estado actual del bus.

```

1  // I2C
2  inout wire      io_i2c_sda,    // SDA del master (output)
3  inout wire      io_i2c_scl,    // SCL del master (output)
4  output reg [7:0] AN,
5  output reg      CA, CB, CC, CD, CE, CF, CG,
6  output wire      o_accel_cs_n,
7  output wire      o_accel_mosi,
8  input wire       i_accel_miso,
9  output wire      accel_sclk);
10
11 wire [15:0]      gpio_out;
12
13 // I2C wires
14 wire i2c_scl_i;
15 wire i2c_sda_i;
16 wire i2c_scl_o;
17 wire i2c_scl_oe;
18 wire i2c_sda_o;
19 wire i2c_sda_oe;

```

Figura 8: Definición de wires,entradas y salidas

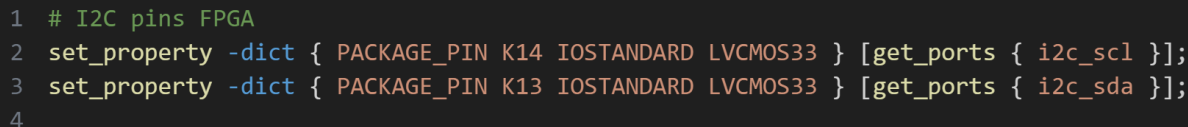
```

1  .i_i2c_scl      (i2c_scl_i),
2  .i_i2c_sda      (i2c_sda_i),
3  .o_i2c_scl      (i2c_scl_o),
4  .o_i2c_scl_oe   (i2c_scl_oe),
5  .o_i2c_sda      (i2c_sda_o),
6  .o_i2c_sda_oe   (i2c_sda_oe));
7
8
9  assign io_i2c_scl      = i2c_scl_oe ? 1'bz : i2c_scl_o;
10 assign io_i2c_sda      = i2c_sda_oe ? 1'bz : i2c_sda_o;
11
12 assign i2c_scl_i = io_i2c_scl;
13 assign i2c_sda_i = io_i2c_sda;

```

Figura 9: Instancia y assigns de las señales SDA y SCL

Ahora, en el archivo `rvfpganexys.xdc` se debe hacer la conexión de los pines que se van a utilizar en la comunicación I^2C :



```

1 # I2C pins FPGA
2 set_property -dict { PACKAGE_PIN K14 IOSTANDARD LVCMOS33 } [get_ports { i2c_scl }];
3 set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { i2c_sda }];
4

```

Figura 10: Pines I^2C

Se continua ahora, definieron los siguientes módulos:

■ **i2c_master_top.v:**

El módulo `i2c_master_top` implementa un controlador maestro I^2C totalmente funcional. Permite la comunicación entre un sistema basado en FPGA.

Este módulo actúa como la unidad principal de control del protocolo I^2C y expone una interfaz programable mediante registros internos, los cuales pueden ser accedidos a través del bus Wishbone. Además, controla directamente las señales físicas del protocolo I^2C (líneas SDA y SCL) mediante lógica triestado.

Internamente, se comunica con un submódulo llamado `i2c_master_byte_ctrl`, encargado del manejo a nivel de bits: generación de condiciones START y STOP, transmisión y recepción de datos, y manejo del reconocimiento (ACK/NACK).

Las funcionalidades principales del módulo son:

- Generación de condiciones **START** y **STOP** según el protocolo I^2C .
- Envío y recepción de datos en bloques de 8 bits.
- Gestión del bit de reconocimiento (ACK/NACK).
- Detección de condiciones de **bus ocupado** y **pérdida de arbitraje** en configuraciones multimaster.
- Generación de interrupciones al finalizar una operación o ante errores.
- Soporte para configuración mediante registros accesibles por Wishbone.
- Control triestado de las líneas SDA y SCL.

■ **wb_master_model:**

Es un modelo funcional de un maestro Wishbone utilizado principalmente para fines de prueba (testbench). Simula el comportamiento de un maestro en un bus Wishbone y permite realizar operaciones de lectura, escritura y comparación sobre periféricos esclavos conectados al bus.

Este módulo no representa hardware sintetizable, sino que sirve como generador de transacciones para validar el funcionamiento de módulos esclavos bajo el protocolo Wishbone. Se utiliza típicamente en ambientes de simulación para automatizar pruebas de comunicación.

■ **i2c_master_bit_ctrl.v:**

Es responsable de generar las transiciones eléctricas correctas sobre las líneas SCL y SDA del bus I²C, implementando el protocolo a nivel de bit. Recibe comandos desde un controlador superior (como `i2c_master_byte_ctrl`) y traduce cada uno de ellos en secuencias de estados que controlan el valor y el momento de activación de las líneas.

Este módulo es esencial para implementar la lógica de bajo nivel del bus I²C, incluyendo el inicio de condiciones **START** y **STOP**, así como las operaciones de lectura y escritura de bits, respetando los requisitos de temporización definidos por la especificación del protocolo.

Este módulo se conecta directamente con el entorno físico (pads SDA/SCL) y garantiza que las operaciones se ejecuten con el tiempo adecuado, ajustado por el valor de `clk_cnt`, de acuerdo al modo de operación (estándar o rápido).

■ **2c_master_byte_ctrl.v:**

Implementa el control de nivel de byte para el protocolo I²C. Se encuentra entre el módulo de alto nivel (`i2c_master_top`) y el módulo de nivel de bit (`i2c_master_bit_ctrl`), y su función principal es traducir comandos de lectura, escritura, inicio y parada en secuencias de bits utilizando una máquina de estados finita.

Este módulo gestiona un registro de desplazamiento de 8 bits para enviar y recibir datos serie, y mantiene el seguimiento del estado de la transacción actual mediante un contador y una máquina de estados interna. También controla el envío del bit de reconocimiento (ACK/NACK) y la validación de finalización de cada operación mediante la señal `cmd_ack`.

La máquina de estados de este módulo comprende seis estados principales: **IDLE**, **START**, **WRITE**, **READ**, **ACK** y **STOP**, cada uno ejecutando una función específica dentro del ciclo de comunicación. En conjunto con el controlador de bits, permite cumplir con el protocolo I²C en tiempo y forma adecuados.

■ **i2c_master_defines.v:**

Contiene definiciones de constantes, utilizadas en los módulos del controlador maestro I²C. Estas macros permiten una codificación más legible, reutilizable y fácil de mantener dentro del diseño en Verilog.

Una de las secciones más importantes de este archivo define los comandos que la máquina de estados del controlador de bits reconoce y ejecuta. Estos comandos son utilizados por el módulo `i2c_master_byte_ctrl` para indicar al módulo `i2c_master_bit_ctrl` qué operación realizar sobre el bus.

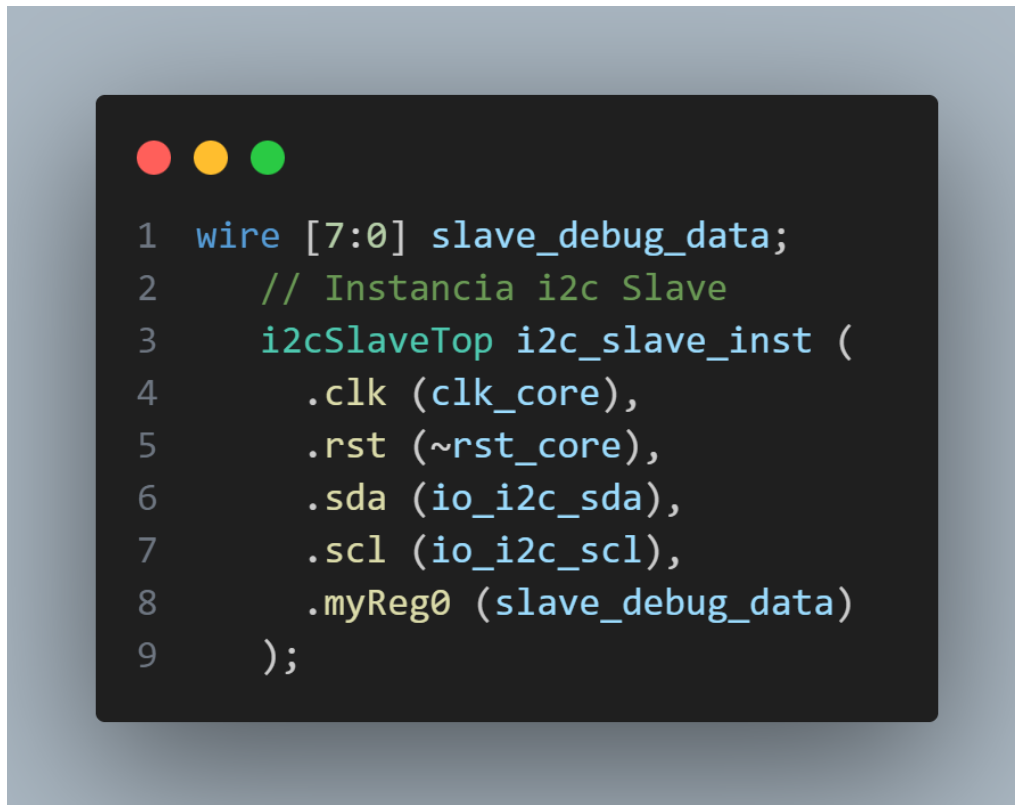
Comandos definidos:

- `'I2C_CMD_NOP` 4'b0000
Comando nulo. No realiza ninguna acción; mantiene al controlador en estado de espera.
- `'I2C_CMD_START` 4'b0001
Genera una condición **START** en el bus I²C.
- `'I2C_CMD_STOP` 4'b0010
Genera una condición **STOP** para finalizar una transacción.
- `'I2C_CMD_WRITE` 4'b0100
Inicia una operación de escritura (envío de bit por la línea SDA).

- 'I2C_CMD_READ 4'b1000
Inicia una operación de lectura (lectura de bit desde SDA).

Slave:

Inicialmente se modifico el archivo rvfpganexys.sv para definir un wire y realizar la instanciación del top module del slave:



```
1 wire [7:0] slave_debug_data;
2 // Instancia i2c Slave
3 i2cSlaveTop i2c_slave_inst (
4     .clk (clk_core),
5     .rst (~rst_core),
6     .sda (io_i2c_sda),
7     .scl (io_i2c_scl),
8     .myReg0 (slave_debug_data)
9 );
```

Figura 11: Modificación rvfpganexys.sv

Luego, se implementaron los siguientes módulos del slave para lograr su correcto funcionamiento

■ i2cSlaveTop.v:

Su función es encapsular la lógica del módulo i2cSlave, proporcionando una interfaz simple de conexión con un sistema externo. Este módulo expone una interfaz básica que incluye las señales de reloj, reinicio, y las líneas I²C (SDA y SCL). Internamente, instancia el módulo i2cSlave y configura varios registros que pueden ser leídos por un maestro I²C.

Interfaz externa:

- clk: señal de reloj del sistema.
- rst: señal de reinicio activo alto.
- sda: línea bidireccional de datos I²C.
- scl: línea de reloj I²C.
- myReg0: salida de 8 bits que representa un registro interno accesible desde el maestro.

■ i2cSlave.v:

Su función es recibir y responder a comandos provenientes de un maestro I²C, controlando las líneas SDA (datos) y SCL (reloj), y gestionando el acceso a registros internos de 8 bits. Este módulo actúa como núcleo del esclavo.

El valor de salida en la línea SDA se controla mediante lógica de triestado, utilizando la señal `sdaOut`. Además, se aplica una lógica de sincronización para el reset y una detección precisa de eventos críticos del protocolo, garantizando el cumplimiento temporal del estándar I²C.

■ registerInterface.v:

Implementa un bloque de registros direccionables utilizados en un sistema esclavo I²C. Su propósito es permitir que un maestro I²C pueda leer y escribir datos en registros internos mediante una dirección enviada a través del bus.

El módulo está diseñado para operar bajo control del módulo `serialInterface`, el cual se encarga de interpretar la dirección, el dato y las señales de escritura a partir de la comunicación serie sobre las líneas SDA/SCL.

Comportamiento funcional:

- **Lectura:** En cada flanco de subida del reloj, el módulo asigna a `dataOut` el valor del registro especificado por la dirección `addr`. Si la dirección no está en el rango válido, se retorna `0x00`.
- **Escritura:** Si la señal `writeEn` está activa, y la dirección corresponde a uno de los registros `myReg0-myReg3`, se actualiza el contenido del registro con el valor de `dataIn`.

■ serialInterface.v:

Implementa la lógica de control serie del protocolo I²C en modo esclavo. Su función principal es interpretar las señales del bus I²C (SCL y SDA), identificar el tipo de transacción (lectura o escritura), gestionar la comunicación bit a bit, y transferir los datos entre el maestro y los registros internos del esclavo.

Comportamiento funcional:

- Implementa una máquina de estados codificada en 4 bits que cubre los flujos de lectura y escritura de acuerdo al protocolo I²C.
- Gestiona un contador de bits para la transmisión y recepción de bytes (8 bits por operación).
- Identifica direcciones y comandos enviados por el maestro y los categoriza como lectura o escritura.
- Transfiere datos al maestro durante una lectura y los almacena en registros durante una escritura.
- Responde con señales ACK/NAK según la validez de la dirección recibida o el estado de la operación.

■ serialInterface.v:

Se implementa la lógica de control serie del protocolo I²C en modo esclavo. Su función principal es interpretar las señales del bus I²C (SCL y SDA), identificar el tipo de transacción (lectura o escritura), gestionar la comunicación bit a bit, y transferir los datos entre el maestro y los registros internos del esclavo.

Este módulo actúa como un puente entre la línea serie y la interfaz de registros, decodificando la dirección del registro, los datos recibidos y generando señales de control como escritura y lectura. Internamente se basa en una máquina de estados finita que cumple con los requisitos temporales del protocolo I²C.

■ **timescale.v:**

Define la escala de tiempo utilizada en las simulaciones del proyecto mediante la directiva de compilación `\timescale` de Verilog. Su función es establecer la unidad de tiempo y la precisión de redondeo para todos los módulos que lo incluyan.

Contenido del archivo:

```
'timescale 1ns / 1ps
```

4. Desarrollo de pruebas:

El programa está diseñado con el propósito es enviar datos hacia un dispositivo esclavo I²C, así como reflejar esos datos localmente mediante LEDs conectados a un GPIO.

Estructura general del programa:

- Habilita todos los pines del GPIO como salidas para controlar los LEDs.
- Define una función `i2c_write_byte()` que envía un byte a un registro específico del esclavo utilizando el protocolo I²C en tres pasos: START + dirección, escritura de registro, y escritura de dato con STOP.
- Dentro del `main()`, se ejecuta un ciclo infinito que:
 - Envía un valor incremental al registro 0 del esclavo I²C.
 - Muestra ese mismo valor en los LEDs de la FPGA.
 - Aplica una demora para que los cambios sean visibles.

Con este tester se puede validar tanto la transmisión de datos hacia un esclavo I²C, como la sincronización visual del sistema a través de los LEDs.

6. Análisis de Resultados

La implementación del protocolo I²C dentro del entorno SweRVolf en la FPGA Nexys A7 requirió instanciar los módulos open-source tanto para el master como para el slave. El módulo master, *i2c_master_top.v*, fue integrado como un periférico dentro del bus Wishbone, utilizando señales de control estándar como `cyc`, `stb`, `we`, `ack` y registros mapeados en memoria (`PRER`, `CTR`, `TXR`, `RXR`, `CR`, `SR`) para permitir el acceso desde software. ES importante que este se realizó la instancia de los wires de wishbone y los PADS que son las líneas físicas. El esclavo, por su parte, se mantuvo como un módulo independiente conectado a las físicas *SDA* y *SCL*, sin integración directa al Wishbone. Esto permitió simular un escenario maestro-esclavo similar al de un microcontrolador externo (por ejemplo, un Arduino), facilitando su posterior reemplazo o expansión.

Para habilitar la comunicación, se configuró el *prescaler* del maestro ajustando los registros `PRER_HI` y `PRER_LO` con base en la frecuencia del reloj de la FPGA (50 MHz), permitiendo generar una señal de reloj `SCL` cercana a 100 kHz, adecuada para operaciones en modo estándar del I²C. Se observó que el envío de datos desde el maestro hacia el esclavo funciona correcta sin generar errores, ya que el sistema funcionaba con un solo maestro y un solo esclavo.

La implementación fue validada desde software en C mediante el uso de registros mapeados en memoria que permitieron la escritura de datos hacia el bus I²C. Para la prueba de la implementación del I²C se realizó una de las practicas implementadas en los laboratorios de desplegar en los LEDS una cuenta que itera y aumenta en 1 pero que se realiza por medio de la comunicación I²C. Además utilizando los switches como entrada digital, se tradujo el valor binario leídos en la FPGA a una transmisión por I²C, la cual fue leída por el esclavo y desplegada a través de LEDs. Este flujo de datos confirmó que el maestro generaba correctamente las condiciones `START`, `ADDRESS`, `WRITE` y `STOP`, y que el esclavo respondía mediante la línea `ACK`. Este código es de importancia ya que se buscaría por medió de este reconocimiento de los switches poder transmitir por las líneas `SDA` y `SCL` para implementar un PWM que genere el barrido de tensión.

- Para la demostración de la verificación del módulo de comunicación I²C se puede observar en el siguiente enlace: https://youtu.be/8HaMGcXndaw?si=t7KNim-9M_66yxvI.
- Para la demostración del barrido por medio de lectura de switches usando el módulo de comunicación I²C se puede observar en el siguiente enlace: <https://youtu.be/NeU4jiCdq8U?si=gGIy-31HtS2wrbn9>.

En cuanto al diseño estructural, la inclusión del maestro en el `swervolf_core.v` y su conexión mediante el `wb_intercon.v` exigieron declarar todas las señales necesarias, tanto internas como aquellas de control triestado (`sda_padoen_o`, `scl_padoen_o`). La correcta configuración de los pines físicos en el archivo XDC, particularmente las señales `io_i2c_sda` e `io_i2c_scl`, fue clave para el funcionamiento del sistema, así como la lógica en `rvfpganexys.v` que resolvía las condiciones de dirección de bus.

7. Conclusiones y Recomendación

Se pudo validar la viabilidad de implementar un sistema master-slave I²C en una FPGA, haciendo uso de módulos obtenidos y adaptados de OpenCores y complementándolos con lógica de control personalizada. Se cumplió el objetivo de establecer una comunicación confiable entre módulos internos utilizando el protocolo I²C, así como el de simular un barrido de voltaje controlado mediante los switches disponibles en la plataforma.

Uno de los principales logros fue la integración completa del bus I²C dentro de la arquitectura Wishbone, permitiendo su acceso desde software en C de forma transparente. Además, se obtuvo un sistema funcionalmente ampliable, en el que se podrían agregar nuevos esclavos (por ejemplo, un Arduino externo) sin necesidad de rediseñar la infraestructura. El diseño modular y asegura que la implementación pueda evolucionar a sistemas más complejos a futuro.

Para futuros desarrollos, se recomienda refinar la lógica de control para soportar múltiples registros en el esclavo, permitiendo una mayor cantidad de configuraciones. También sería valioso implementar mecanismos de verificación más automatizados, como *testbenches* que validen diferentes condiciones de bus y respuestas esperadas. Además, se sugiere integrar un convertidor digital-analógico real, como el MCP4725, a través del bus I²C para generar una salida de voltaje física, validando así el barrido de tensión más allá de la simulación.

Referencias

1. J. Hull, "Take this bus everywhere - the ubiquitous i2c serial bus." <https://www.nxp.com/docs/en/presentation/FTF-SMI-N1919.pdf>, 2016. Public Use.
2. J. Wu, "A basic guide to i²c," 2025.
3. N. Semiconductors, "I2c-bus specification and user manual," 2007. Accessed: 2025-06-25.
4. A. Newswire, "I2c bus market 2024 to 2034: Global growth driven by expanding application in consumer electronics and automotive sectors," 2024.
5. N. Monitor, "Embedded pc: Advantages and disadvantages," 2025.
6. Solectroshop, "Diferencias de los protocolos de comunicación: Uart vs i²c vs spi," 2025. Accedido: 2025-06-25.
7. OpenCores, "I2c slave core," 2024.
8. OpenCores, "I2c core," 2024.
9. OpenCores, "I2c master/slave core," 2024.
10. OpenCores, "I2c master/slave core (version improved)," 2024.
11. OpenCores, "I2c to wishbone interface," 2024.
12. OpenCores, "I2c parallel master," 2024.
13. OpenCores, "Iicmb (i2c message block)," 2024.