

Diseño e implementación en FPGA de un sistema de comunicación I<sup>2</sup>C  
y generación de barrido de tensión para control digital en la plataforma Nexys A7  
Presentación Final Proyecto Laboratorio de Circuitos Digitales

Gabriel Siles Chaves - C17530  
Jorge Loría Chaves - C04406



02 de julio del 2025

## Objetivo General

Diseñar e implementar un sistema en FPGA que habilite la comunicación I<sup>2</sup>C mediante pines físicos y permita generar un barrido controlado de tensión como parte de una lógica programada en hardware.

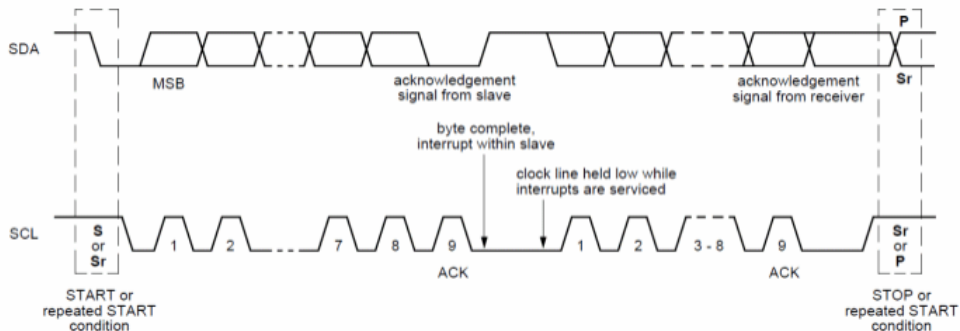
## Objetivos Específicos

- Módulo maestro I<sup>2</sup>C en FPGA que genere señales por SDA y SCL cumpliendo el protocolo.
- Barrido de tensión programable desde la FPGA mediante salidas variables.
- Arquitectura base para integración futura con controladores externos (C o RISC-V) vía I<sup>2</sup>C.

## Contexto y Motivación

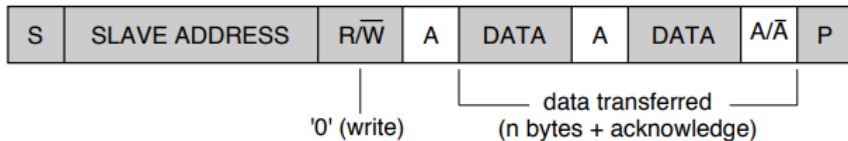
- Se eligió I<sup>2</sup>C por ser un protocolo utilizado en la industria actual.
- La compatibilidad de I<sup>2</sup>C bajo uso de pines y compatibilidad con sensores, microcontroladores y periféricos comunes.


## Trama típica del protocolo I<sup>2</sup>C




*Cada byte transmitido está sincronizado con el reloj SCL y confirmado con un bit de ACK.*

## Escritura I<sup>2</sup>C



 from master to slave

 from slave to master

A = acknowledge (SDA LOW)

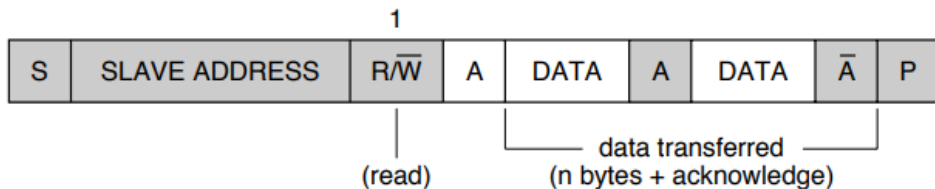
$\bar{A}$  = not acknowledge (SDA HIGH)

S = START condition

P = STOP condition

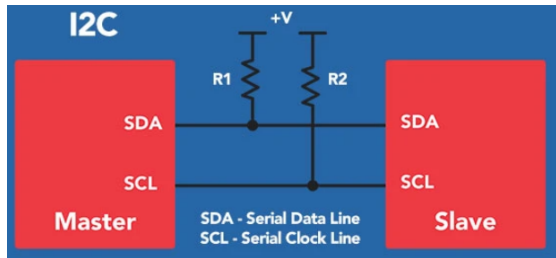
*El maestro envía dirección + datos, el esclavo responde con ACK tras cada byte.*

## Lectura I<sup>2</sup>C



*El maestro solicita datos y el esclavo responde, cada byte puede ser confirmado con ACK o finalizado con NACK.*

## Comparación visual: I<sup>2</sup>C vs SPI



**I<sup>2</sup>C: 2 líneas, SDA y SCL**

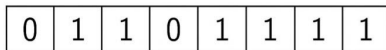


**SPI: más líneas, comunicación full-duplex**

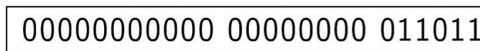


## Recepción I<sup>2</sup>C en FPGA (8 → 32 bits)

Entrada serie desde i<sup>2</sup>C (8 bits)



Padding (relleno con ceros)



Registro de 32 bits para uso interno

*Los datos I<sup>2</sup>C se amplían a 32 bits con ceros para alinearse al bus interno.*

# Módulos I<sup>2</sup>C

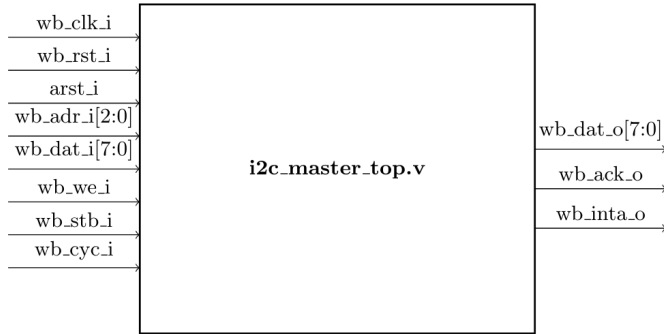
## Revisión de módulos en OpenCores:

Se evaluaron diseños disponibles para FPGA, entre ellos:

- I2C Slave Core
- I2C Core
- I2C Master/Slave Core
- I2C Master/Slave Core (mejorado)
- I2C to Wishbone Interface
- I2C Parallel Master
- IICMB (I2C Message Block)

*Se seleccionó **I2C Slave Core** como base para el diseño del esclavo I<sup>2</sup>C.*

## Diagrama I<sup>2</sup>C Primario (*Master*)



*Módulo a implementar i2c\_master\_top.v*

# Controladores Byte/Bit I<sup>2</sup>C Primario (*Master*)

## Controlador de Byte

- Coordina la transmisión de datos byte por byte usando una máquina de estados.
- Envía comandos como START, STOP, WRITE y READ al controlador de bit.

## Controlador de Bit

- Manipula directamente las líneas SDA y SCL para realizar las operaciones físicas.
- Ejecuta cada comando a nivel de bits (lectura, escritura, inicio, parada).

*Ambos trabajan en conjunto para implementar el protocolo I<sup>2</sup>C en hardware.*

# Implementación I<sup>2</sup>C Primario (Master)

```
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
5 // Instanciamos el módulo I2C
6 i2c_master_top i2c_inst (
7     // Wishbone slave interface
8     .wb_clk_i (clk), // Reloj
9     .wb_rst_i (wb_rst), // Reset
10    .arst_i (1'b0), // No se usa
11    .wb_adr_i (wb_m2s_i2c_accel_adr[2:0]), // Dirección
12    .wb_dat_i (wb_m2s_i2c_accel_dat[7:0]), // Datos a escribir
13    .wb_we_i (wb_m2s_i2c_accel_we), // Señal de escritura
14    .wb_cyc_i (wb_m2s_i2c_accel_cyc), // Señal de ciclo válido
15    .wb_stb_i (wb_m2s_i2c_accel_stb), // Señal de strobe
16    .wb_dat_o (i2c_rdt), // Datos leídos desde I2C
17    .wb_ack_o (wb_s2m_i2c_accel_ack), // Acknowledgment
18    .wb_inta_o (i2c_irq), // Interrupción de I2C
19
20    // Pads I2C (líneas físicas)
21    .scl_pad_i (i_i2c_scl),
22    .scl_pad_o (o_i2c_scl),
23    .scl_padoen_o (o_i2c_scl_oe),
24    .sda_pad_i (i_i2c_sda),
25    .sda_pad_o (o_i2c_sda),
26    .sda_padoen_o (o_i2c_sda_oe)
27 );
28
29 // Expandimos la salida de 8 bits a 32 bits
30 assign wb_s2m_i2c_accel_dat = {24'd0, i2c_rdt};
31
32 // Que no se usen en el módulo master de I2C
33 assign wb_s2m_i2c_accel_err = 1'b0;
34 assign wb_s2m_i2c_accel_rty = 1'b0;
35
```

*Instancia en el Swerlvolf core*

- Dirección: 32'h00002800
- Mask: 32'hffff000
- PIN L14 (SCL)/ PIN M14 (SDA)

```
1 assign io_i2c_scl    = i2c_scl_oe ? 1'bz : i2c_scl_o;
2 assign io_i2c_sda    = i2c_sda_oe ? 1'bz : i2c_sda_o;
3
4 assign i2c_scl_i = io_i2c_scl;
5 assign i2c_sda_i = io_i2c_sda;
```

*Conexiones en RVFPGA nexys*

## Diagrama I<sup>2</sup>C Secundario (*Slave*)



*Módulo a implementar i2cSlaveTop.v*

# Módulos I<sup>2</sup>C Secundario (*Slave*)

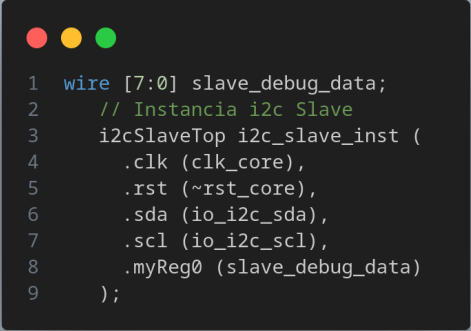
## registerInterface

- Gestiona un conjunto de registros direccionables para lectura y escritura.
- Conecta la lógica de comunicación serial con la lógica funcional del sistema.

## serialInterface

- Interpreta el protocolo I<sup>2</sup>C, detectando condiciones START, STOP y decodificando direcciones.
- Genera señales de control para acceder a los registros internos (regAddr, writeEn, dataOut).

## Implementación I<sup>2</sup>C Secundario (*Slave*)

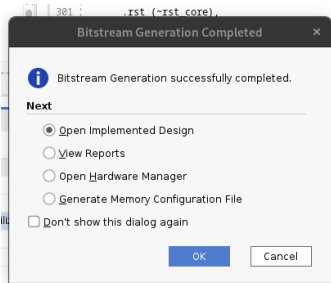


```
1 wire [7:0] slave_debug_data;  
2 // Instancia i2c Slave  
3 i2cSlaveTop i2c_slave_inst (  
4     .clk (clk_core),  
5     .rst (~rst_core),  
6     .sda (io_i2c_sda),  
7     .scl (io_i2c_scl),  
8     .myReg0 (slave_debug_data)  
9 );
```

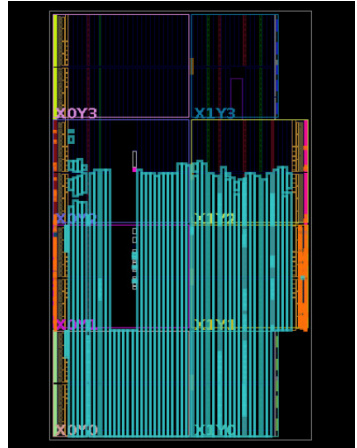
*Instancia del Slave*



# Síntesis e Implementación del I<sup>2</sup>C en la Nexys



*Generación Bitstream correcto*



*Implementación Final*

# Código de prueba del I<sup>2</sup>C

```
1  int main(void) {
2
3      WRITE(I2C_PRER_LO, 249);
4      WRITE(I2C_PRER_HI, 0x00);
5      WRITE(I2C_CTR, 0x80);    // Habilitar modulo I2C
6
7      // Configurar todos los pines GPIO como salidas
8      WRITE(GPIO_INOUT, 0xFFFF);
9
10     unsigned char value = 0;
```

```
1  while (1) {
2      // Enviar valor al registro 0 del esclavo I2C
3      i2c_write_byte(I2C_SLAVE_ADDR, 0x00, value);
4
5      // Mostrar mismo valor en los LEDs de la FPGA
6      WRITE(GPIO_LEDS, value);
7
8      // Incrementar valor
9      value++;
10
11     // Espera para que el cambio sea visible
12     delay(1000000);
13 }
```

# Código de Barrido por medio de SWs y 7SegDisplay

```
1 int main(void) {  
2     u16_t sw_values;  
3     int sw_count;  
4     WRITE(GPIO_INOUT, 0xFFFF);  
5     WRITE(SegEn_ADDR, 0x0);  
6  
7     WRITE(I2C_PRER_L0, 249); // PRER = (50e6 / (5 * 100e3)) - 1 = 249  
8     WRITE(I2C_PRER_HI, 0x00);  
9     WRITE(I2C_CTR, 0x80);
```

```
1 while (1) {  
2     sw_values = (READ(GPIO_SWs) >> 16) & 0xFFFF;  
3     WRITE(GPIO_LEDS, sw_values);  
4     sw_count = count_active_switches(sw_values);  
5     WRITE(SegDig_ADDR, sw_count);  
6     i2c_write_byte(I2C_SLAVE_ADDR, 0x00, (unsigned char)sw_count);  
7     delay();  
8 }
```

# Conclusiones

- Se logró implementar un módulo maestro I<sup>2</sup>C funcional, respetando el protocolo y controlando líneas SDA/SCL desde la FPGA.
- Se permite comunicación con otros componentes por medio de I<sup>2</sup>C para realizar barridos de voltaje por medio de comunicación de pines SDA/SCL.
- Se estableció una arquitectura modular, lista para integrarse con software en C/RISC-V.