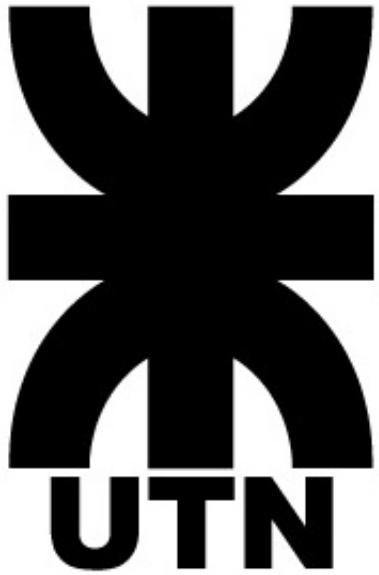


Universidad Tecnológica Nacional
Facultad Regional de Córdoba

# Trabajo Práctico De Laboratorio $N°3$

Navarro, Facundo 63809

Curso: 6r4
Grupo $N°5$

## Técnicas Digitales IV

Docentes:
Ing. Cayuela, Pablo
Ing. Olmedo, Sergio

June 23, 2020

# Contents

# 1 Introducción

En el siguiente trabajo práctico se investigan e implementan protocolos de comunicación síncronos, en donde $R_X$ y $T_X$ operan en conjunto con un señal de *clock*, los cuales son $I^2C$, $SPI$, $PS2$.

# 2 Personal system version 2 (PS2)

# 3 Serial peripheral interface (SPI)

# 4 Inter intergrated circuit ($I^2C$)

Es un bus serial de comunicación de 8 bits entre circuitos integrados que se encuentran próximos entre ellos (normalmente en la misma placa). Utiliza tan solo dos cables (mas GND de referencia) y tiene cuatro modos estandarizados de velocidad

- standard (100 kbps)
- fast (400 kbps)
- fast-plus (1 Mbps)
- high-speed (3,3 Mbps)

El bus $I^2C$ consisten en dos cables, llamados *SCL* (serial clock) y *SDA* (serial data), los cuales interconectan un *master* con un número de unidades *esclavas*.Cada dispositivo en el bus tiene una única dirección y puede operar como transmisor o receptor, poniendo o adquiriendo datos en la linea *SDA*, es decir que esta linea es bidireccional. El clock es unidireccional y es generado por el master, quien además es el encargado de iniciar y finalizar la transferencia de datos. Ejemplos de familias de integrados actualmente fabricados con soporte $I^2C$ son: memorias flash y EEPROM, conversores A/D y D/A, circuitos RTC, sensores de temperatura, etc.

Las salidas de SCL y SDA son a drenador abierto, por lo que son necesarias resistencias del tipo pull-up ($R_{UP}$), normalmente en el rango de los 1,5 a 33 $k\Omega$. El valor de $R_{UP}$ depende de la capacitancia total del nodo, en caso de buses muy largos con muchos esclavos adheridos a él, entonces las resistencias deben ser pequeñas para alcanzar el mínimo "rise time" definido en la especificaciones del $I^2C$.

El número de dispositivos compartiendo el mismo bus puede alcanzar hasta 128 (direcciones de 7-bit) o 1024 (direcciones de 10-bit). Más de un *master* es permitido, el tal caso el protocolo $I^2C$ administra el bus, considerando como master principal al primero que ponga el bajo el voltaje de *SDA*.

## 4.1 Operación de la interfaz $I^2C$

**Diagrama de tiempos.** El diagrama de tiempos de una típica transferencia de datos se muestra en la figura 4. Ambas líneas están en alto cuando el bus esta en estado "inactivo". El master inicia la transferencia creando una condición de arranque (S), en la cual SDA cambia de alto a bajo mientras que SCL se mantiene en alto. Luego genera la señal clock sobre SCL. Dependiendo del tipo de transacción, o el maestro o el esclavo seleccionado introducen datos sobre SDA. Los datos deben ser estables cuando SCL este en alto y el cambio de dato solo puede ocurrir cuando SCL este en bajo. Estas definiciones aseguran que tanto las condiciones de arranque como parada nunca puedan ser confundidas como datos.

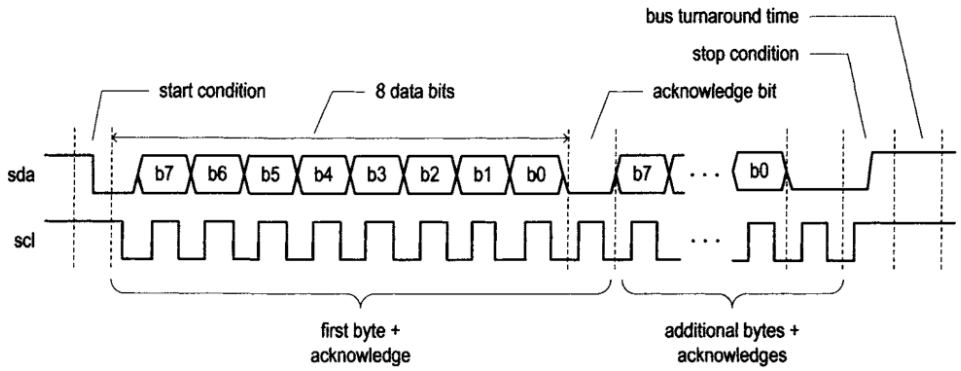


Figure 1: Diagrama de tiempos genérico de una transferencia por $I^2C$

La transferencia esta hecha byte-a-byte con el MSB primero. Cada byte es seguido con un "acknowledge" bit en el noveno pulso de reloj. El número de bytes en una transferencia no está restringido. Después de terminado, el master finaliza la transferencia creando una condición de parada (P), en la cual SDA cambia de bajo a alto mientras SCL se mantiene en alto. Luego de que la transferencia esta completada el bus debe esperara un pequeño tiempo (turnaround time) antes de inicializar otra transacción.

**Características eléctricas.** La conexión del bus $I^2C$ se muestra en la figura **2**, usa la tecnología "drenador-abierto", lo que significa que en la etapa de salida de un dispositivo tiene que tener la estructura de drenador-abierto. Tanto la linea *sda* como *scl* están conectadas a $V_{DD}$ a través de las resistencias pull-up las cuales representan un estado en alto cuando el bus esta en alta impedancia. La línea se vuelve en bajo cuando la salida de alguno de los dispositivos se vuelve bajo. Se dice que realiza la función de "and-cableada".

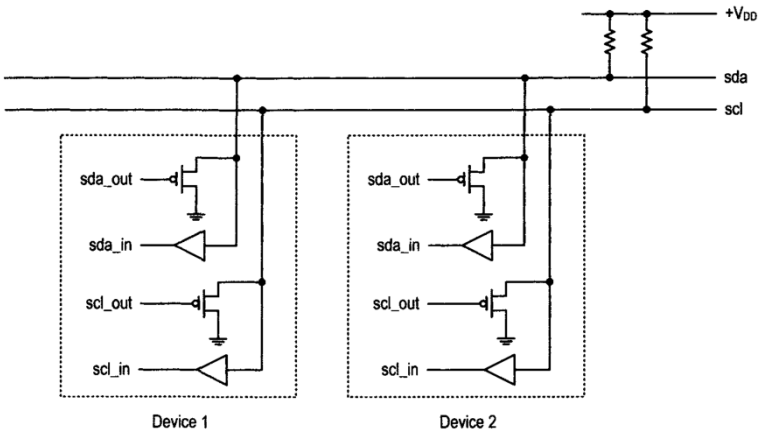


Figure 2: Diagrama conceptual de un bus $I^2C$.

**Protocolo de bus.** Cuando el bus esta en *idle*, cualquier dispositivo puede comenzar la transferencia y convertirse en el *master*. Una tipica transferencia de datos esta compuesta de varias partes:

- Start.
- Dirección del esclavo mas bit de dirección.
- Datos.
- Stop.

El *master* inicia la condición de *start* y luego envía la dirección de 7-bits del esclavo junto con el bit que indica el sentido del flujo de datos, el cual es un 1 para la operación de lectura (del slave al master) y un 0 para la operación de escritura (del master al slave). El maestro libera la línea de *sda* y el *slave* con la

dirección correspondiente debe reconocer tirando a bajo la línea *sda* durante el noveno pulso del reloj. La transferencia de datos puede proceder luego byte-a-byte según lo especificado por el bit de dirección. En la figura **3** se muestran las secuencias completas de escritura y lectura de 2 bytes. Cabe destacar que el master debe generar un bit "reconocimiento negado" en el noveno ciclo de reloj de la operación de lectura luego de recibir el ultimó byte de datos.

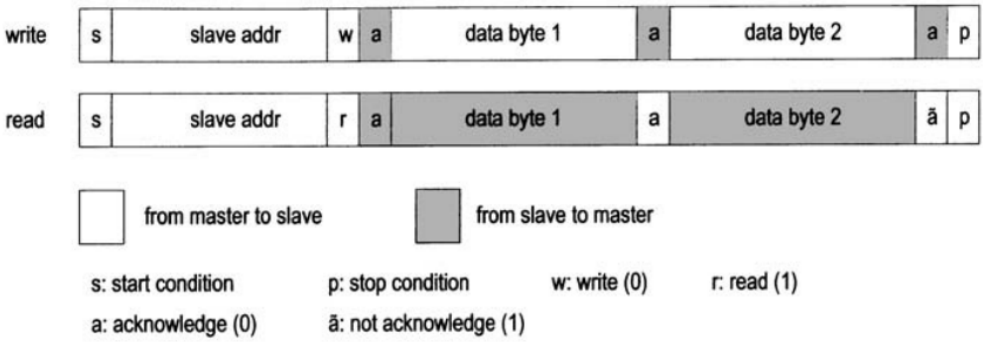


Figure 3: Secuencia completa de escritura y lectura de dos bytes.

### 4.1.1 Módulo ds3231.

El modulo utilizado es el basado en el integrado RTC ds3231. El ds3231 es un reloj de tiempo real (RTC) extremadamente preciso con un oscilador integrado con compensación de temperatura de cristal (TCXO). El dispositivo incorpora la posibilidad de adicionarle una batería y mantener el cronometraje preciso cunado se interrumpe la alimentación principal del módulo. Ademas incluye una memoria EEPROM AT24C32 de 32Kb y un sensor de temperatura.

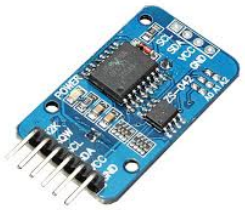

Figure 4: Módulo ds3231.

El módulo tiene la ventaja que ambos dispositivos (memoria y RTC) se comunican a través del protocolo $I^2C$, las direcciones de memoria se puede observar en la figura **5**, la dirección de memoria es mapeable a dos direcciones posibles debido a una compatibilidad de la susodicha con un modelo anterior de memoria.

```
#A2, A1, A0 = 111
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- 55 -- -- -- -- -- -- -- 5d -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --

#A2, A1, A0 = 101
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- 57 -- -- -- -- -- -- -- 5f
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 5: Direcciones de memoria en hexadecimal.

La dirección del RTC es 0x68 y la memoria comparte tanto 0x5d como 0x55, esta puede ser modificada cerrando los puentes A0, A1, A2 que se encuentra disponible en el modulo y como se observan en el circuito de la figura 6, de allí que hayan dos tablas de direcciones.

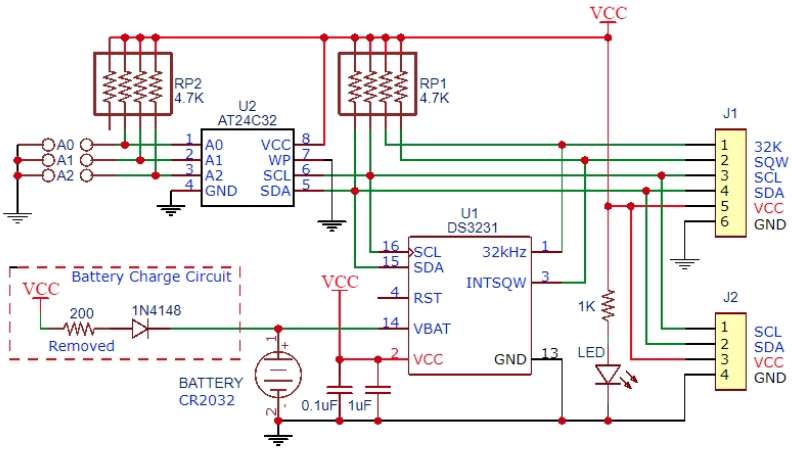


Figure 6: Esquemático del módulo.

El PCB está diseñado para que se coloque una pila CR2032 recargable, en caso que se use una que no lo sea es recomendable desoldar o re posicionar el circuito de carga de batería.

### 4.1.2 Estructura del proyecto

El proyecto fue desarrollado en el SW Vivado 2019.2.1 ya que se utilizó la plataforma de entrenamiento de Digilent, Basys3.

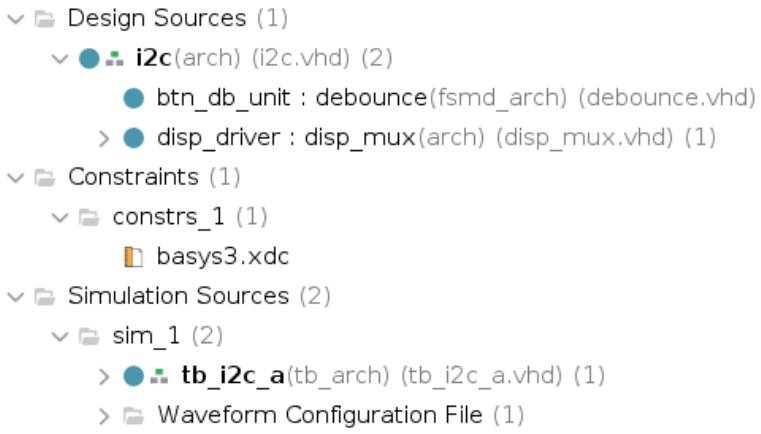


Figure 7: Árbol del proyecto.

## 4.2 Protocolo $I^2C$

Para el desarrollo del protocolo $I^2C$ se plantea una maquina de estado que avance deacuerdo a las etapas que indita el datasheet del integrado ds3231 como se ve en la figura 8

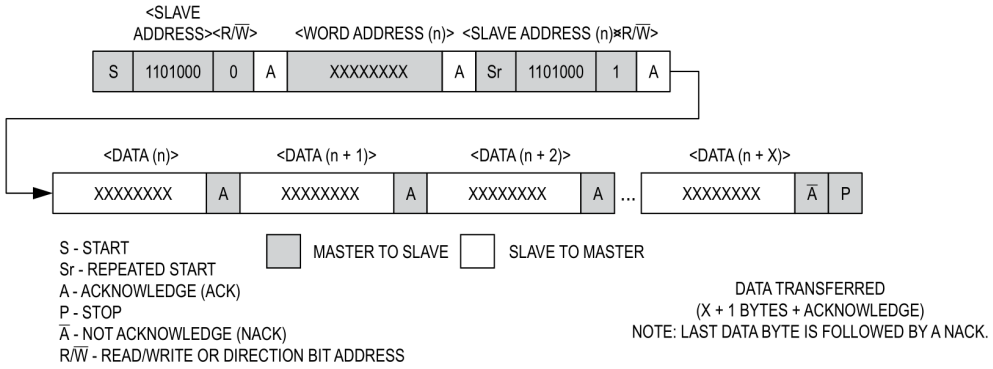


Figure 8: Secuencia para lectura a partir de un registro del integrado ds3231.

Las etapas de la máquina de estado planteada es la siguiente:

1. La FPGA genera la secuencia de arranque.
2. Se envía la dirección del slave junto con un 0 para indicar la escritura.
3. El master espera un ack (0) por parte del slave.
4. Se envía el registro a escribir.
5. El master espera un ack (0) por parte del slave.
6. Sin dejar el bus, se manda una condición de re-start.
7. Se envía nuevamente la dirección del slave, esta vez junto a un 1 para indicar la operación de lectura.
8. El master espera un ack (0) por parte del slave.
9. Se lee un byte transferido por el slave.
10. El master genera un ack (0).
11. Los dos anteriores se repite 2 veces mas, salvo que en el último byte el master genera un nack (1).
12. El master genera la secuencia de stop y libera el bus.

Tanto para la generación de datos como de los casos especiales de start, stop la trama se trabaja con un contador, para garantizar el cumplimiento de los tiempos necesarios para la comunicación. En la figura **9** se observa como se forman cada una de estas condiciones.

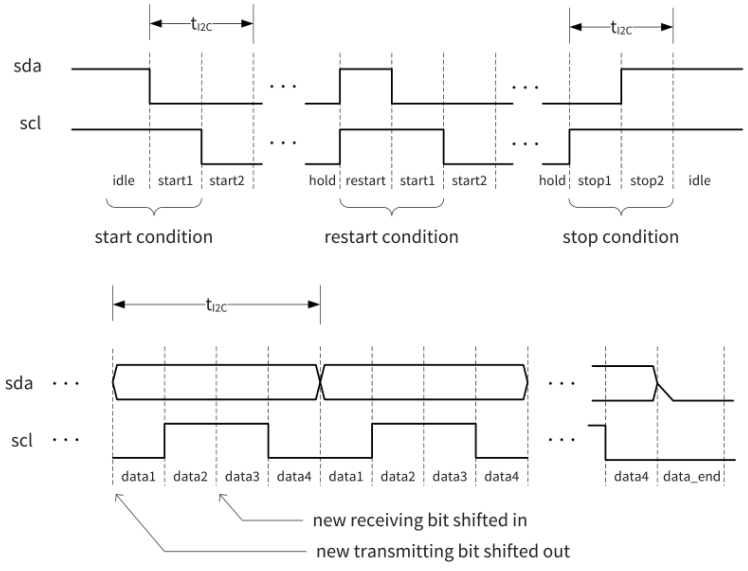


Figure 9: Divisiones de los estados del $I^2C$

Se trabaja con una velocidad de transmicion de 100 Kb, por este motivo y lo visto previamente se determinan dos constantes *QUTR* y *HALF* que representan respectivamente un cuarto y la mitad de un ciclo de reloj completo.

```
architecture arch of i2c is
    -- Constants
    constant HALF           : integer := 500;   -- 10us/10ns/2 = 500
    constant QUTR           : integer := 250;   -- 10us/10ns/4 = 250
```

Figure 10: Constantes para 100 Kb.

Tanto para la escritura como la recepción, los datos se parten en 4 segmentos, como se puede apreciar en la figura **11**, esto es para poder construir los flancos ascendentes y descendentes del clock, como así también set capaz de recibir en la mitad del periodo del clock el dato que envíe el slave.

```
when data1_wr =>
    sdat_out <= data_reg(15);
    sclk_out <= '0';
    if (c_reg = QUTR) then
        c_next <= (others => '0');
        state_next <= data2_wr;
    end if;
when data2_wr =>
    sdat_out <= data_reg(15);
    if (c_reg = QUTR) then
        c_next <= (others => '0');
        state_next <= data3_wr;
    end if;
when data3_wr =>
    sdat_out <= data_reg(15);
    if (c_reg = QUTR) then
        c_next <= (others => '0');
        state_next <= data4_wr;
    end if;
when data4_wr =>
    sdat_out <= data_reg(15);
    sclk_out <= '0';
    if (c_reg = QUTR) then
        c_next <= (others => '0');
        if (bit_reg = 7) then
            state_next <= ack1_wr;
        else
            data_next <= data_reg(14 downto 0) & '0';
            bit_next <= bit_reg + 1;
            state_next <= data1_wr;
        end if;
    end if;
```

```
when data1_rd =>
    sclk_out <= '0';
    if (c_reg = QUTR) then
        c_next <= (others => '0');
        state_next <= data2_rd;
    end if;
when data2_rd =>
    if (c_reg = QUTR) then
        c_next <= (others => '0');
        state_next <= data3_rd;
        rx_next    <= rx_reg(22 downto 0) & i2c_sdat;
    end if;
when data3_rd =>
    if (c_reg = QUTR) then
        c_next <= (others => '0');
        state_next <= data4_rd;
    end if;
when data4_rd =>
    sclk_out <= '0';
    if (c_reg = QUTR) then
        c_next <= (others => '0');
        if (bit_reg = 7) then
            if(byte_reg = (BYTES_2_READ - 1)) then   -- done with 3 bytes
               state_next <=  nack1;
            else
                state_next <= ack1_rd;
            end if;
        else
            bit_next <= bit_reg + 1;
            state_next <= data1_rd;
        end if;
    end if;
```

Figure 11: Escritura y recepción de datos.

### 4.2.1 Simulación

Para la simulación se simulo la primer trama de envío del address del dispositivo que se va a comportar como slave, para ello se definen como constantes la direccion del address, se utilizan dos registros, uno para escritura y otro para lectura.

```
signal DEV_ADDR_WRITE   : std_logic_vector(7 downto 0) := x"D0"; -- 110_1000_0 (address + direction)
signal DEV_ADDR_READ    : std_logic_vector(7 downto 0) := x"D1"; -- 110_1000_1 (address + direction)
```

Figure 12: Adress del ds3231 para escritura y lectura.

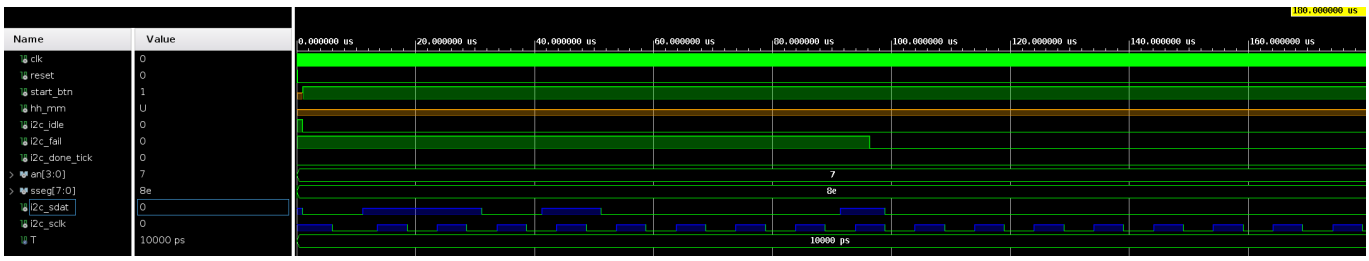


Figure 13: Simulacion del envio del address de escritura.

### 4.3 Multimedia

Se adjunta un dos links de unos vídeos demostrativos:

- Presentación: `https://youtu.be/1az8Dk07Ik4`
- Demo: `https://youtu.be/NCIMGwVY518`