2. I2C - INTER-INTEGRATED CIRCUIT

El protocolo I2C (Inter-Integrated Circuit) es un protocolo de comunicación serial síncrono utilizado para comunicar dispositivos electrónicos entre sí. Fue desarrollado por Philips Semiconductor (ahora NXP Semiconductors) en los años 80 y actualmente es ampliamente utilizado en sistemas embebidos y de comunicaciones.

El protocolo I2C se utiliza para conectar dispositivos a través de dos líneas de señal, llamadas SDA (Serial Data Line) y SCL (Serial Clock Line). La línea SDA es bidireccional y se utiliza para transmitir datos entre los dispositivos, mientras que la línea SCL es unidireccional y se utiliza para sincronizar la transmisión de datos.

Los dispositivos conectados a la red I2C se identifican mediante un número de dirección único, que se utiliza para seleccionar un dispositivo específico para la comunicación. Cada dispositivo tiene una dirección de 7 bits, que puede ser configurada por el fabricante o por el usuario.

El protocolo I2C utiliza dos tipos de transacciones: lecturas y escrituras. Durante una transacción de escritura, el maestro (el dispositivo que inicia la transacción) envía una dirección de 7 bits del dispositivo esclavo seguida de los datos que se van a enviar. Durante una transacción de lectura, el maestro envía la dirección de 7 bits del dispositivo esclavo seguida de una señal de reinicio y luego lee los datos que se envían desde el dispositivo esclavo.

El protocolo I2C también permite que los dispositivos esclavos inicien una transacción enviando una señal de inicio al maestro. Esto se utiliza comúnmente para solicitar datos del maestro.

2.1 CONEXIONES

Este protocolo soporta la posibilidad de conectar una gran cantidad de dispositivos, llamados esclavos o slaves a un mismo maestro o dispositivo master.

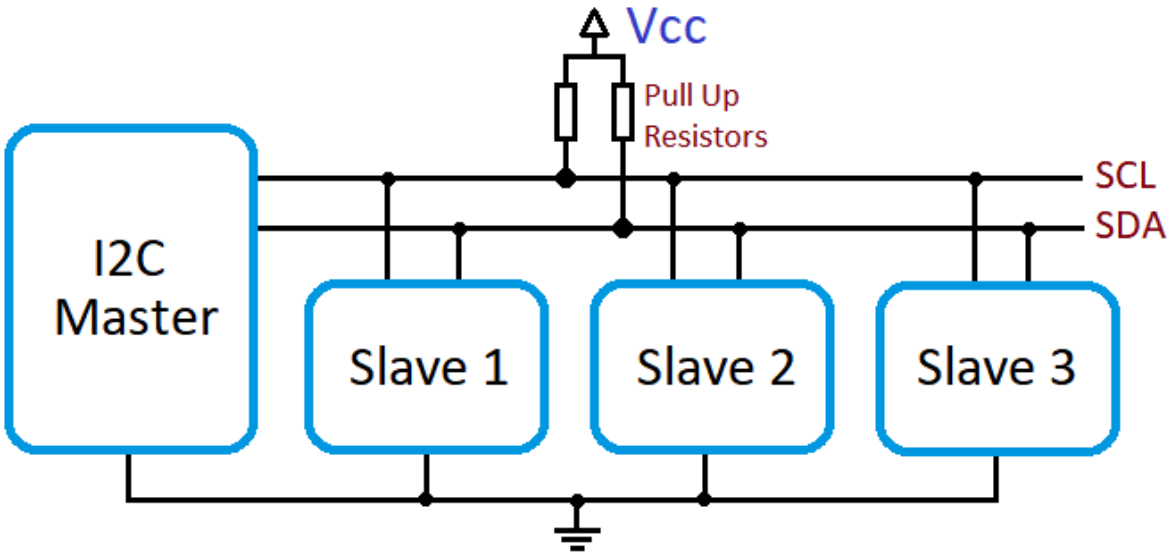

Figura 1: Conexión de protocolo I2C

Todas las líneas de SDA y SCL son compartidas entre todos los dispositivos y deben tener una resistencia de pull-up que asegura que mientras no se estén utilizando las líneas o no se estén transmitiendo datos, estas permanezcan en estado alto. Cuando se van a utilizar, el dispositivo que inicia la transferencia fuerza las líneas a estado bajo para comenzar la comunicación.

En teoría, debido a que el protocolo utiliza un sistema de direcciones de 7 bits, esto permitiria conectar hasta 128 dispositivos esclavos distintos mientras todos tengan direcciones diferentes. En la práctica, conectar demasiados dispositivos puede cargar la línea haciéndola más lenta, y, si bien algunos dispositivos tienen la posibilidad de configurar su dirección, muchos vienen con una dirección ya fija.

2.2 TRANSMISIÓN DE DATOS

Como se mencionó anteriormente, dos líneas son necesarias para este protocolo, la línea de datos o SDA y la de SCL. Ambas líneas permanecen en estado alto (VCC) mientras ningún dispositivo las esté usando.

2.1.1 LÍNEA DE CLOCK (SCL)

La existencia de esta señal es lo que hace que sea un protocolo sincrónico. Esta es una línea compartida y es generada desde el dispositivo maestro. Esta señal permite capturar en cada flanco ascendente o descendente el valor de la línea SDA para que el dispositivo que recibe los datos sepa que contiene el paquete transmitido.

La frecuencia de la señal de clock depende de la velocidad a la que se quiera enviar datos. Algunas velocidades estandard para dispositivos que usen I2C son:

- **Standard mode (Modo estándar)**: Velocidad máxima de transmisión de 100 kbps (kilobits por segundo).
- **Fast mode (Modo rápido)**: Velocidad máxima de transmisión de 400 kbps.
- **Fast mode plus (Modo rápido plus)**: Velocidad máxima de transmisión de 1 Mbps (megabits por segundo).
- **High-speed mode (Modo de alta velocidad)**: Velocidad máxima de transmisión de 3.4 Mbps.
- **Ultra-fast mode (Modo ultra rápido)**: Velocidad máxima de transmisión de 5 Mbps.

Si hay más de un dispositivo conectado al bus I2C, la velocidad quedará limitada por la velocidad máxima del dispositivo más lento que esté conectado.

2.1.2 LÍNEA DE DATOS (SDA)

Esta es una línea bidireccional compartida entre todos los dispositivos. Cualquiera de los dispositivos, ya sea maestro o esclavo puede tomar control sobre esta línea y enviar información. Esto implica que todos los dispositivos pueden ver lo que se está enviando, pero solo los afectados por la dirección emitida hacen caso.

Este protocolo es half-duplex, es decir, que solo se puede enviar o recibir, no pueden realizarse ambas operaciones en simultáneo, ya que solo existe una línea de datos compartida.

2.1.3 FORMATO DE TRANSMISIÓN

Como se mencionó anteriormente, las líneas de SCL y SDA están en estado alto mientras no haya transmisión en el bus. Cuando se inicia una transmisión, esta consiste en:

- Condición de inicio o start. Esto es que la línea de SDA pasa a estado bajo seguida de la de SCL.
- Luego el dispositivo maestro procede a enviar los 7 bits de dirección del esclavo al que quiere dirigirse. Se envían en orden del MSB al LSB.
- El octavo bit consiste en un bit de lectura o escritura (R/W) en donde se indica si el maestro quiere leer (1) o escribir (0) el esclavo.
- Luego viene un bit de parte del dispositivo esclavo llamado acknowledgement o ACK que consiste en un estado bajo por parte del esclavo para indicar que el mensaje se recibió correctamente. De otra manera, la línea permanece en estado alto y el dispositivo maestro puede tomar medidas para corregir el error.
- Luego de ese primer byte, los siguientes tienen un formato similar, dependiendo si se hizo una lectura o escritura:

    a. **Lectura**: el maestro procede a enviar 8 o 16 bits que forman una dirección de memoria que se desea leer dentro del esclavo. Cada transmisión de 8 bits es seguida de un bit de ACK. Posterior a esto, el esclavo envía la información solicitada en paquetes de 8 bits como lo hizo el maestro.
    b. Escritura: el maestro procede a enviar 8 o 16 bits que forman una dirección de memoria del esclavo que se desea escribir. Luego de elegir el registro a escribir, el maestro envía los bytes necesarios para escribir el registro. Cada paquete enviado tiene su bit de ACK al final.

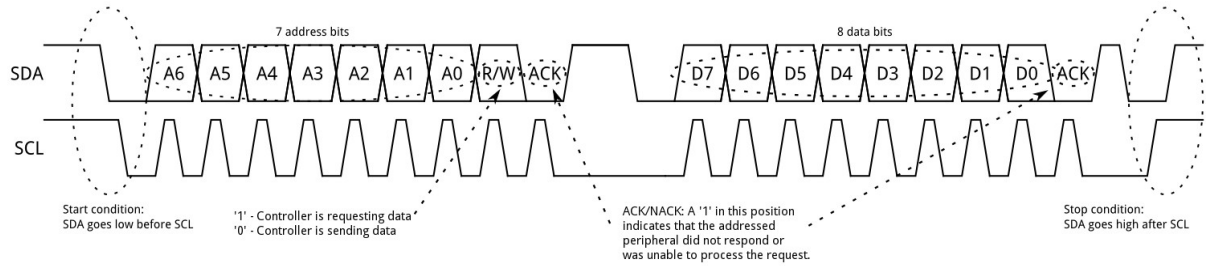

Figura 2: Formato de transmisión I2C

Cuando la comunicación se termina, se realiza una condición de stop que consiste en que la línea de SCL vuelve a estado alto seguida del SDA.

2.3 GLOSARIO

A continuación hay una lista de términos usados con sus respectivas definiciones para aclarar cualquier posible confusión.

- **I2C**: Inter-Integrated Circuit.
- **Half-duplex**: comunicación donde un dispositivo puede solo enviar o recibir datos. Tiene línea de datos compartida para ambas operaciones.
- **SDA**: línea de datos.
- **SCL**: línea de clock.

- **MSB**: bit más significativo.
- **LSB**: bit menos significativo.

2.4 REFERENCIAS ADICIONALES

A continuación se deja material adicional para el que desee profundizar más en el tema que se describió.

- I2C Primer: What is I2C? (Part 1)
- I2C Timing: Definition and Specification Guide (Part 2)

ANEXO I: I2C EN LA RASPBERRY PI PICO

La Raspberry Pi Pico tiene dos módulos de I2C disponibles, referidos en el SDK como *i2c0* e *i2c1*. La mayoría de los GPIO pueden hacer uso de ambos I2C, debe elegirse un GPIO apropiado revisando el pinout.

Para configurar el I2C, puede hacerse lo siguiente:

```
// Configuro el I2C0 a 100 KHz de clock
i2c_init(i2c0, 100 * 1000);
// Elijo GPIO4 como linea de SDA
gpio_set_function(4, GPIO_FUNC_I2C);
// Elijo GPIO5 como linea de SCL
gpio_set_function(5, GPIO_FUNC_I2C);
// Activo pull-up en ambos GPIO, son debiles por lo que
// es recomendable usar pull-ups externas
gpio_pull_up(4);
gpio_pull_up(5);
```

Luego, para escribir o leer con el I2C se puede hacer lo siguiente:

```
// Array para guardar 6 bytes leidos
uint8_t buf[6];
// Byte con la direccion de un registro
uint8_t reg = REG_PRESSURE_MSB;
// Escribo al dispositivo con la direccion ADDR, un byte
// (la direccion en reg) y mantiene el control del bus el master
i2c_write_blocking(i2c_default, ADDR, &reg, 1, true);
// Leo del dispositivo con la direccion ADDR, 6 bytes, los guardo
// en la variable buf y termino la transmision
i2c_read_blocking(i2c_default, ADDR, buf, 6, false);
```