Práctica 2: Sensores digitales.

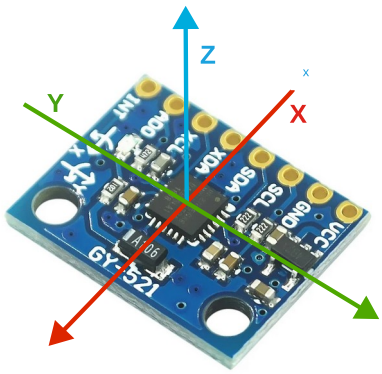
Adquisición y Procesamiento de Señal

Objetivo

El objetivo de esta práctica es la introducción al funcionamiento de los sensores digitales; al protocolo I2C, utilizado típicamente para la adquisición de datos de este tipo de sensores; y a la entrada/salida por interrupciones y sus diferencias con la entrada salida programada.

Sensor

Utilizaremos el sensor MPU6050, un chip con acelerómetro y giroscopio. El acelerómetro proporciona 3 valores de aceleración y el giroscopio 3 valores de velocidad de giro, en ambos casos correspondientes a los ejes X, Y y Z según la siguiente orientación:



Además, el sensor también proporciona la temperatura ambiente.

La **hoja técnica** (*datasheet*) y el **mapa de registros** del sensor están disponibles, respectivamente, en:

https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

En la hoja técnica se pueden encontrar las tablas con las especificaciones eléctricas del chip, las especificaciones de los sensores internos, los límites de funcionamiento, los protocolos de comunicaciones que utiliza, etc.

En el mapa de registros se encuentra la dirección y descripción de cada una de las posiciones de memoria (registros) que utiliza el sensor para su configuración y para el almacenamiento de las muestras que toma, en este caso de aceleración, velocidad de giro y temperatura. El microcontrolador interactúa con el sensor escribiendo y leyendo los valores almacenados en dichos registros.

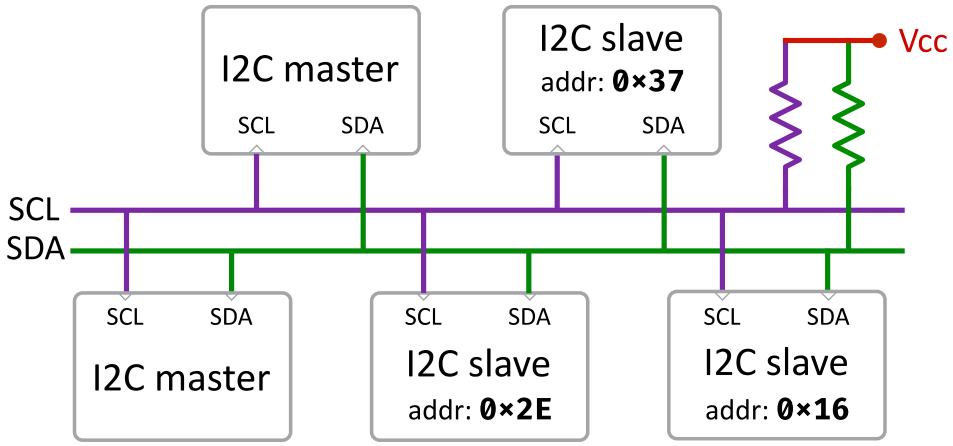
Para comunicarnos con el sensor desde el ESP32 necesitamos algún protocolo de comunicaciones que nos permita **enviar datos** al sensor (escribir en sus registros) o **recibir datos** del mismo (leer los registros). El ESP32 soporta múltiples protocolos (I2C, SPI, UART, TWAI…) pero el MPU6050 soporta únicamente I2C, por lo que será el protocolo en el que nos centraremos.

I2C

I2C (***I**nter-**I**ntegrated **C**ircuit*) es un protocolo de comunicaciones **serie** (i.e., bit a bit) que permite a **uno o varios *maestros*** (o controladores) comunicarse con **uno o varios *esclavos*** (o destinos) utilizando únicamente dos líneas de comunicación: SCL (***S**erial **CL**ock*) y SDA (***S**erial **DA**ta*).

SCL es la línea de reloj, una señal cuadrada periódica que marca el ritmo al que se transmiten los bits en la línea SDA. Cada ciclo (período) de SCL es un bit. Cuanto mayor sea su frecuencia, más rápido se leerán o escribirán los datos. Una señal de 400 kHz implica que se transmiten 400 kbit/s. El maestro siempre es el que escribe en la línea SCL, determinando la velocidad de las comunicaciones.

SDA es la línea que lleva los datos binarios del maestro al esclavo (en operaciones de **escritura**) o del esclavo al maestro (en operaciones de **lectura**). I2C es, por lo tanto, un protocolo ***half-duplex***: no permite escribir y leer al mismo tiempo.



Para que la comunicación sea posible habiendo múltiples destinos conectados a las mismas 2 líneas, I2C utiliza **direcciones**, generalmente de 7 bits. El maestro, que siempre es el que inicia la comunicación, escribe la dirección bit a bit en la línea de datos (SDA) **al principio de cada transacción** para dirigirse a un esclavo concreto. No puede haber más de un dispositivo con la misma dirección I2C en el mismo bus.

Todos los esclavos monitorizan el bus. El esclavo al que se dirige el maestro, al identificar su dirección en el bus, sabe que debe atender la solicitud de escritura o lectura. El tipo de operación la marca el bit transmitido justo tras la dirección: **0 ≡ escritura (W)**, **1 ≡ lectura (R)**. Por ejemplo, para realizar una operación de escritura sobre un dispositivo que tiene la dirección de 7 bits **0010110** (0x16 en hexadecimal), el maestro escribiría en SDA el byte **00101100** (0x2C), mientras que para una operación de lectura escribiría **00101101** (0x2D). Si la operación es de escritura, el maestro continuará escribiendo en SDA los datos que desee pasar al esclavo, y si es de lectura, el maestro dejará de transmitir y será el esclavo el que pase a escribir en la línea SDA los datos que serán leídos por el maestro.

Cada byte enviado va seguido de **un bit**, escrito en SDA **por el receptor** (maestro o esclavo), que sirve para confirmar la recepción del byte (ACK/NACK). Es decir, se necesitan 9 ciclos de reloj de SCL para la transmisión de un byte entre dos dispositivos.

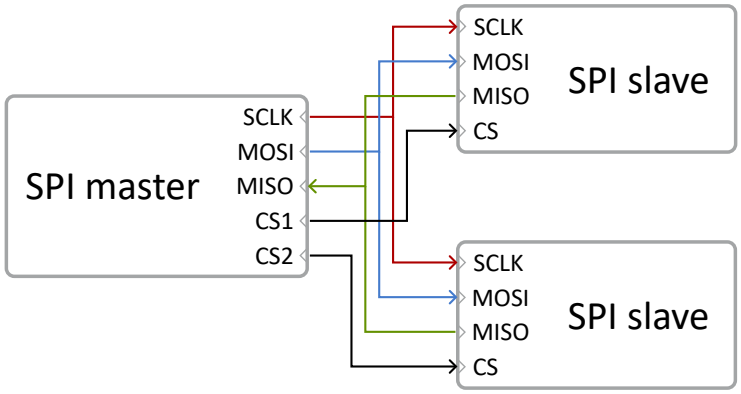
La velocidad de transmisión (frecuencia de la señal SCL) depende del modo de funcionamiento. Las velocidades más comunes soportadas por sensores son 100 kbit/s (*standard mode*) o 400 kbit/s (*fast mode*). Existen modos menos comunes que permiten hasta 3.4 Mbit/s (*high speed mode*), o, con algunas limitaciones con respecto a los otros modos, hasta 5 Mbit/s (*ultra-high speed mode*).

Otros protocolos

SPI

SPI (*Serial Peripheral Interface*) es un protocolo de comunicaciones serie que permite a **un** maestro comunicarse con **uno o varios** esclavos. Al contrario que I2C, el protocolo (en general) es ***full-duplex***: permite leer y escribir al mismo tiempo. Para ello, además de la línea de reloj que realiza la misma función que en I2C (generalmente denominada SCL, SCK, SCLK o CLK), utiliza 2 líneas diferentes de datos para escrituras y lecturas: MOSI (*Master-Out-Slave-In*) y MISO (*Master-In-Slave-Out*), respectivamente.

SPI no utiliza direcciones, por lo que son necesarias líneas adicionales entre el maestro y cada uno de los esclavos para seleccionar a uno en concreto. Estas líneas generalmente se denominan CS (*Chip Select*) o SS (*Slave Select*). El maestro activa la línea correspondiente al esclavo con el que quiere comunicarse. El resto de esclavos, que no tienen activas sus respectivas líneas CS, deben ignoran los datos transmitidos en MISO y MOSI entre el maestro y el esclavo seleccionado.

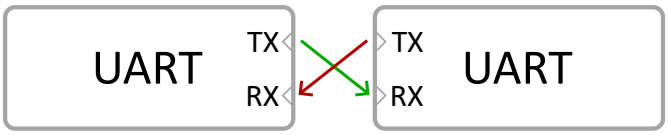


SPI es un protocolo mucho más simple y menos estructurado que I2C. Existen variantes que solo utilizan una línea de datos para ambos sentidos (*half-duplex*) o que utilizan más de una línea para transmitir datos en paralelo (*Dual* SPI, *Quad* SPI, *Octal* SPI...). Esto permite a SPI alcanzar velocidades de transmisión mucho más elevadas (>100 Mbit/s), lo que lo hace idóneo para periféricos que requieren alta velocidad de transferencia, como memorias RAM externas, memorias flash o pantallas.

UART

UART (*Universal Asynchronous Receiver-Transmitter*) es otro protocolo serie, típicamente utilizado para la comunicación entre el ordenador y el microcontrolador, pero que también puede ser utilizado entre el microcontrolador y un periférico.

Al contrario que I2C y SPI, es **punto a punto**: solo permite la comunicación entre dos dispositivos. También al contrario que los anteriores, es **asíncrono**: no tiene línea de reloj (I2C y SPI son **síncronos**). Solo es necesaria una línea de datos para cada sentido (*full-duplex*):



Al ser asíncrono y no tener línea de reloj que marque la velocidad de transmisión los dispositivos tienen que **conocer de antemano la velocidad** del otro dispositivo para que la comunicación sea posible. Las velocidades más típicas soportadas por dispositivos UART son, en bits/s, 9600, 19200, 38400, 57600 o 115200, aunque se pueden alcanzar velocidades de varios Mbit/s.

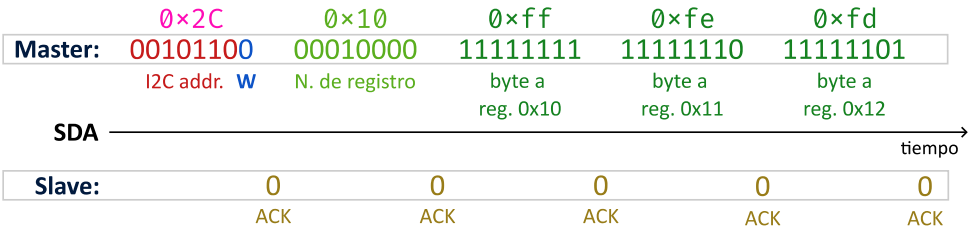
Comunicación entre microcontrolador y sensor

Independientemente del protocolo de comunicaciones utilizado (I2C, SPI, etc.), los sensores (o cualquier otro tipo de periférico) deben definir un protocolo para la escritura y lectura de sus **registros**.

Cuando se usa I2C para el envío y la recepción de datos, la mayoría de los sensores utilizan el siguiente protocolo para la comunicación con un microcontrolador:

- Si el microcontrolador quiere **escribir en uno o varios registros consecutivos** del sensor, el microcontrolador (maestro) inicia una **operación I2C de escritura (W).** Como datos, el microcontrolador envía en primer lugar **el número del primer registro** en el que desea empezar a escribir, y a continuación el **valor o valores que desea escribir**.

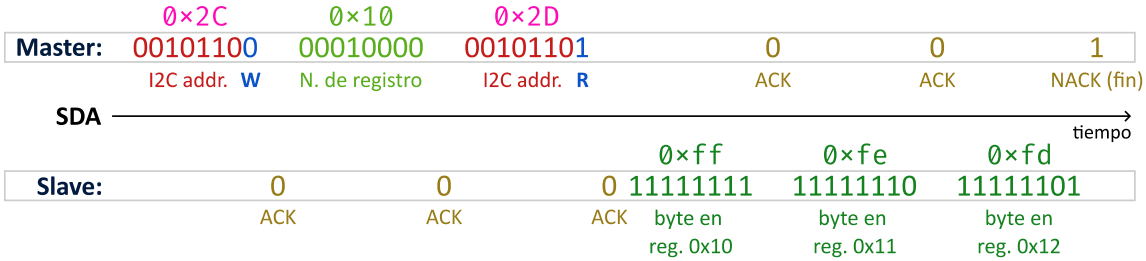
  Por ejemplo, si el microcontrolador quiere escribir los bytes 0xff 0xfe 0xfd en los registros 0x10 0x11 y 0x12 (**consecutivos**) de un sensor con dirección I2C 0x16 (0010110), podría hacerlo con una sola transacción:



  Si quisiese escribir en varios registros **no consecutivos**, tendría que hacer varias operaciones de escritura.

- Para **leer uno o varios registros consecutivos**, el microcontrolador le debe enviar al sensor el registro desde el que desea empezar a leer. Por ello son necesarias **dos operaciones I2C**: una de **escritura (W)**, donde el microcontrolador le envía al sensor la dirección de dicho registro, seguida de una de **lectura (R)**, en la que el sensor le pasará al microcontrolador los datos.

  Por ejemplo, si el microcontrolador desea ahora leer los datos de los registros 0x10 0x11 y 0x12 lo haría de la siguiente forma:



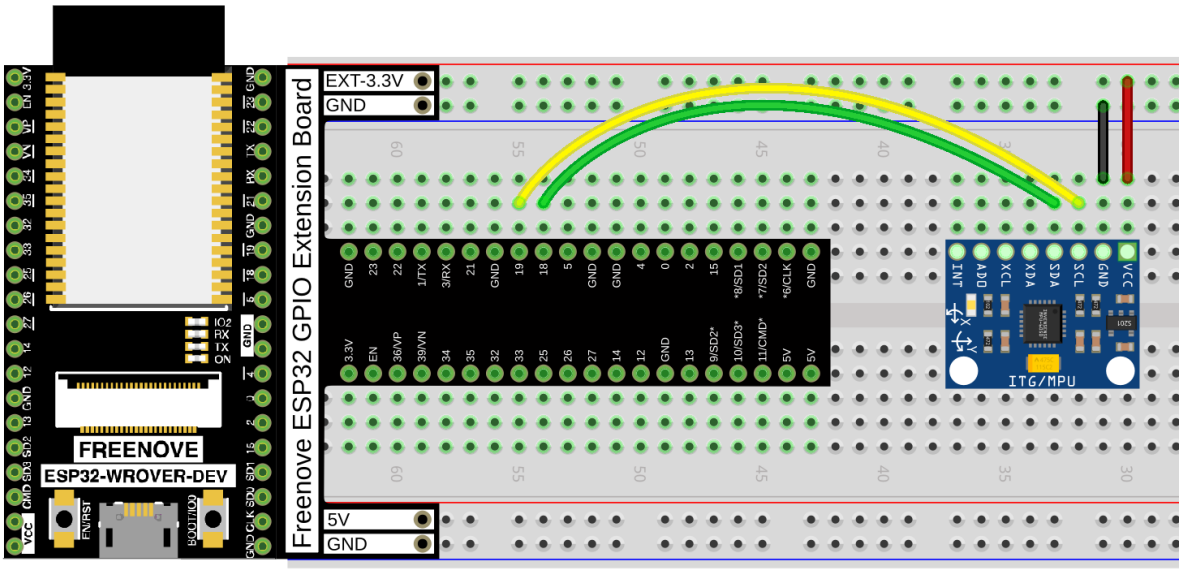
  El microcontrolador informa al sensor de que no quiere recibir más datos con un NACK.

El MPU6050 utiliza este protocolo y lo define en su hoja técnica (9.3 I^2C Communications Protocol—Communications).

Existen otros sensores con direcciones de registros de 16 bits (en lugar de 8 bits) o con registros de 16 o 32 bits (es decir, los datos en cada registro ocupan dos o cuatro bytes), o que definen protocolos más complejos (por ejemplo, intercalando bytes de CRC para verificar la correcta transmisión de datos a través del bus). El protocolo anterior es fácilmente adaptable a estas variaciones.

Tarea 1: Comunicación a través de I2C

Conecta el sensor de acuerdo al siguiente montaje:



Utiliza la clase `I2C` de `machine` para comunicarte con el sensor:

https://docs.micropython.org/en/latest/library/machine.I2C.html

Cuando crees el objeto I2C, pasa los parámetros `id` (igual a 0, solo tendremos un bus I2C) y `scl` y `sda` (iguales a los pines utilizados en el montaje). Ten en cuenta que `id` es un parámetro posicional (primer argumento), y `sda` y `scl` son parámetros con nombre. No es necesario pasar la frecuencia; por defecto es 400 kHz (*fast mode*) que es soportada por el MPU6050.

Haz un programa que obtenga una lista de los dispositivos I2C conectados al ESP32 (`scan`) y la imprima. Si aparece alguna de las direcciones posibles de un MPU6050 entre los dispositivos conectados, lee el **registro de identificación** (**WHO_AM_I**) y verifica que devuelve el valor correspondiente a dicho sensor. Consulta la **hoja técnica** (9.2. I^2C Interface) para conocer las posibles direcciones I2C del MPU6050 y el **mapa de registros** para conocer el número de registro **WHO_AM_I** y el valor que debe tener en un MPU6050.

Nota: Como se indica en la sección anterior, para leer un registro del MPU6050 (y en general de cualquier periférico I2C) es necesario realizar dos operaciones: una de escritura del número de registro seguida de una de lectura en la que el sensor devuelve los datos. En la clase `I2C` estas dos operaciones están ya implementadas en una única función: `readfrom_mem`, a la que se le pasa como parámetros la dirección I2C del dispositivo, el número de registro que se desea leer y el número de bytes a leer, y devuelve un **array de bytes** del tamaño especificado con el resultado de la lectura.

Preguntas

1. Conecta el pin marcado “AD0” del sensor a la línea 3.3V. ¿Qué cambia con respecto a no conectarlo? ¿Para qué puede ser útil esta funcionalidad?

Tarea 2: Inicio de medida y lectura de datos

Completa el código de la tarea anterior para leer, 5 veces por segundo (**frecuencia de muestreo** de 5 Hz), los valores **crudos** (sin procesar) de aceleración, temperatura y velocidad de giro.

Para ello, en primer lugar debes configurar el sensor para que comience a medir muestras de acelerómetro y giroscopio. El MPU6050 se inicia en modo de bajo consumo, en el que no toma muestras de giroscopio. Para que comience a tomarlas es necesario escribir en el registro **PWR_MGMT_1** el valor 0x01.

Nota: Para escribir en un registro se puede usar la función `writeto_mem` de la clase `I2C`, a la que se le pasa como parámetros la dirección I2C, la dirección del registro que se desea escribir y un **array de bytes** con los valores a escribir comenzando desde ese registro. En este caso el array sólo contendrá un elemento, pues solo queremos escribir un byte en el registro **PWR_MGMT_1**.

A continuación, ya en el bucle principal del programa, debes obtener (con `readfrom_mem`) un array de bytes que contenga los valores. Los valores medidos por el sensor se encuentran en los registros que van desde **ACCEL_XOUT_H** hasta **GYRO_ZOUT_L**. Dado que los registros son consecutivos, se pueden leer en una sola operación de lectura del número de bytes apropiado.

Imprime los bytes obtenidos del sensor y verifica que los valores cambian significativamente al mover o girar la placa de pruebas.

Preguntas

1. Teniendo en cuenta cómo se realiza la lectura de datos en I2C, ¿cuántos ciclos de reloj son necesarios para leer una muestra completa (i.e., aceleración + temperatura + velocidad de giro) del sensor? ¿Cuánto tiempo supone con I2C configurado a 400 kHz?
2. Dado lo anterior, ¿cuál es la máxima frecuencia de muestreo teórica que nos permitiría el sensor con el bus I2C configurado a 400 kHz?

Tarea 3: Descodificación de datos

Crea una función `raw2val` que transforme los valores crudos (array de bytes) obtenidos en la tarea anterior en valores que representen la aceleración, la temperatura y la velocidad de giro. Imprime los valores por pantalla.

Para ello, en primer lugar, es necesario transformar los datos crudos del sensor en tipos de datos que puedan ser operados por el microcontrolador (int, float, etc.). El MPU6050 tiene un ADC con **resolución de 16 bits**, de ahí que cada muestra ocupe 2 bytes. Para cada valor, el registro con sufijo **_H** contiene el byte más significativo y el **_L** el menos significativo. El ADC proporciona valores **con signo** (enteros) pues, al contrario que el ADC del ESP32, que medía una magnitud positiva (voltaje entre 0 y V_{ref}), la aceleración y la velocidad de giro pueden tomar valores negativos. Si el ADC del ESP32 proporcionaba 2^{12} posibles valores entre 0 y $2^{12} - 1$ (4095), el del MPU6050 proporciona 2^{16} posibles valores entre -2^{15} y $2^{15} - 1$ (−32768 y +32767).

Para convertir el array de bytes en una lista de 7 valores enteros se puede utilizar la función `unpack` del módulo `struct` (https://docs.python.org/3/library/struct.html), pasando como primer argumento de formato una cadena que indique que los datos son de 16 bits con signo (short), y en formato *big-endian* (el orden de los bytes en el array va de más significativo a menos significativo).

Una vez obtenidos los valores enteros del ADC, es necesario convertirlos a valores en las unidades adecuadas. Vamos a imprimir la aceleración en **g** ($1\ g = 9.806\ m/s^2$), la velocidad de giro en **°/s** ($1°/s = 2\pi/360$ rad/s) y la temperatura en **grados centígrados**. Para convertirlos es necesario tener en cuenta que el ADC, en su configuración por defecto, mide aceleraciones en el rango ±2 g y velocidades de giro en el rango ±250°/s. Los valores extremos de medida se corresponden con los valores extremos del

ADC. Para el sensor de temperatura, consulta la fórmula necesaria para la conversión en la sección 4.18 del mapa de registros.

Preguntas

1. ¿Qué valores de aceleración para X, Y y Z mide con el sensor en horizontal? ¿Y si lo giras sobre un lado? ¿Por qué? ¿Qué está midiendo el acelerómetro con el sensor en reposo?
2. ¿Qué valores de velocidad de giro mide con el sensor en reposo? ¿Son exactamente los esperados?
3. Implementa una función de calibración que corrija el error de *offset* (desplazamiento). La función debe capturar 50 valores del acelerómetro y del giroscopio en reposo horizontal, hacer la media, y corregir los valores mostrados en el bucle principal. Cuando el sensor esté horizontal y sin moverse debe aparecer una aceleración de 0 g en los ejes X e Y y +1 g en el eje Z, y una velocidad de giro de 0°/s en los 3 ejes.

Tarea 4: Configuración del rango del ADC y de la frecuencia de muestreo

Los sensores digitales suelen permitir la configuración de diferentes parámetros, lo que generalmente se realiza antes del bucle principal. Entre los parámetros más habituales se encuentran la frecuencia de muestreo y el rango de medida.

Para los rangos de acelerómetro y giroscopio, el MPU6050 utiliza los bits 3 y 4 de los registros **GYRO_CONFIG** y **ACCEL_CONFIG**. Consulta la documentación para los posibles rangos de cada uno y los correspondientes valores en los registros.

Para la frecuencia de muestreo, el MPU6050 utiliza el registro **SMPRT_DIV**. El registro divide la frecuencia de muestreo interna del giroscopio (8 kHz o 1 kHz, dependiendo de la configuración) entre el valor de dicho registro + 1. Por ejemplo, si la frecuencia interna es 1 kHz, podemos obtener una frecuencia de muestreo de 100 Hz escribiendo el valor 9 en el registro.

Configura el giroscopio para un rango de ±1000°/s y el acelerómetro para un rango de ±4 g. Recuerda adaptar la función `raw2val` a los nuevos rangos para que calcule los valores correctos.

Configura también la frecuencia de muestreo a 20 Hz mediante **SMPRT_DIV**. Ahora, con la frecuencia de muestreo configurada en el sensor, en lugar de esperar con `time.sleep` en el bucle principal del programa podemos esperar a que el MPU6050 nos informe de que ha generado una nueva muestra. Para ello, se debe leer el registro **INT_STATUS**, que tendrá el último bit igual a 1 cuando haya una nueva muestra disponible para leer. Si el último bit es 0 el sensor aún no ha generado una nueva muestra y es necesario esperar (bucle `while`). Para activar esta funcionalidad debes escribir un 1 en el último bit del registro **INT_ENABLE** antes del bucle principal.

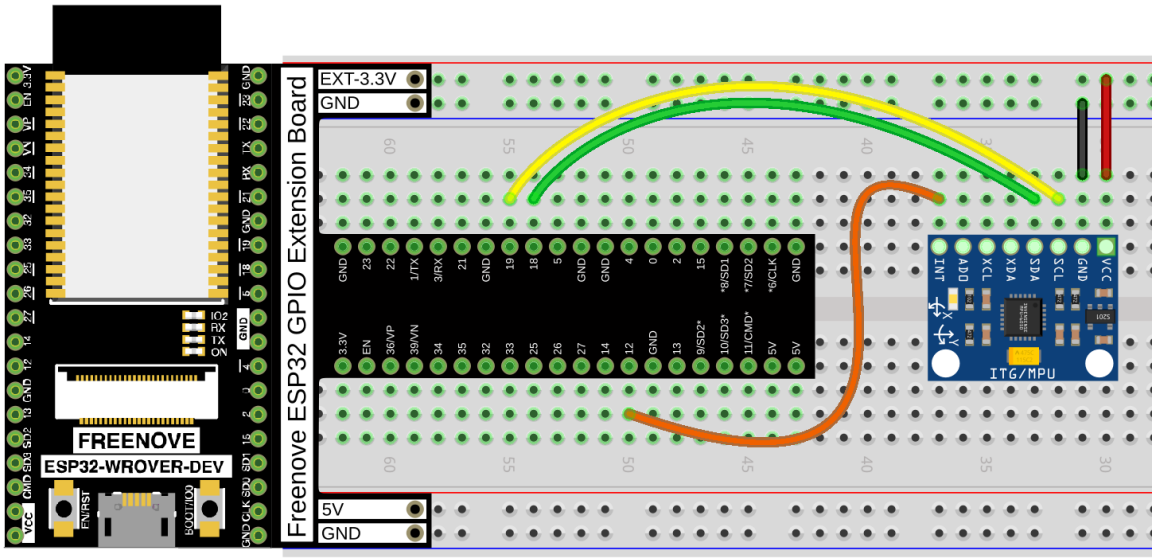
Tarea 5: Interrupciones

En el apartado anterior el microcontrolador está comprobando constantemente si existe una nueva muestra del sensor que deba procesar leyendo el registro **INT_STATUS**. Este tipo de entrada/salida, llamada **entrada/salida programada** o **por encuesta (*polling*)**, consume ciclos de reloj y mantiene ocupado el bus I2C, lo que no es deseable.

Para solucionar estos problemas se suele usar **entrada/salida por interrupciones**, que consiste en que el periférico (en este caso el sensor) notifica directamente al microcontrolador cuando haya una nueva muestra, sin que este deba comprobarlo.

Cuando ocurre una interrupción, la CPU del microcontrolador salta al **manejador (*handler*) de la interrupción**, un fragmento de código que se encarga de atenderla. Cuando sale del manejador continúa con la ejecución normal del programa.

El MPU6050 dispone de un pin (**INT**) que pone a nivel alto cuando tiene una nueva muestra disponible. Podemos aprovecharlo para generar una interrupción. Realiza el siguiente montaje:



Configura el pin 12 del microcontrolador como pin de entrada, y utiliza la función `irq` de la clase `Pin` para definir el manejador de la interrupción (https://docs.micropython.org/en/latest/library/machine.Pin.html).

El manejador de interrupción será una función que será llamada cada vez que el pin cambie de estado alto a bajo o o de bajo a alto (configurable mediante el parámetro `trigger` de la función `irq`, que puede tomar los valores `Pin.IRQ_FALLING` o `Pin.IRQ_RISING`, respectivamente). El manejador debe ser una función que reciba un argumento, que será el objeto `Pin` que ha causado la interrupción.

Cambia el código del bucle principal para realizar la lectura de los datos solo cuando el MPU6050 genere una interrupción. Para ello, utiliza el manejador de interrupción para cambiar el estado de una **variable global booleana** (`new_sample`) que condicione la lectura de datos en el bucle principal.

Preguntas

1. Desconecta y conecta el cable del pin 12 durante la lectura de datos. ¿Qué ocurre?
2. Añade al principio del bucle principal el siguiente código:

   ```
   if not new_sample and new_sample:
       print('Esto es imposible')
       break
   ```

   ¿Por qué se imprime la línea, si la condición del `if` debería ser siempre falsa?