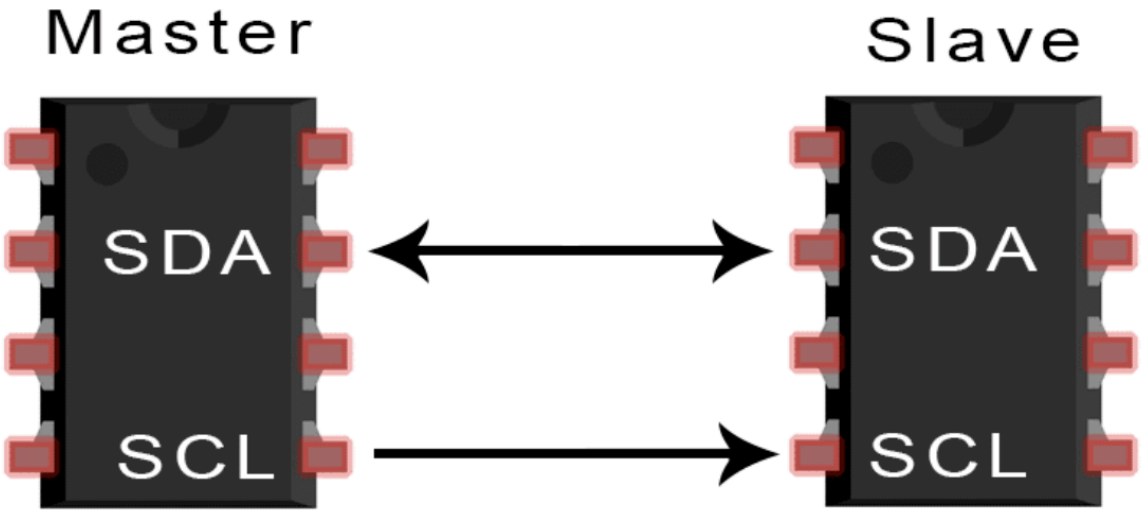
MEMORIA DE PROYECTO I2C:



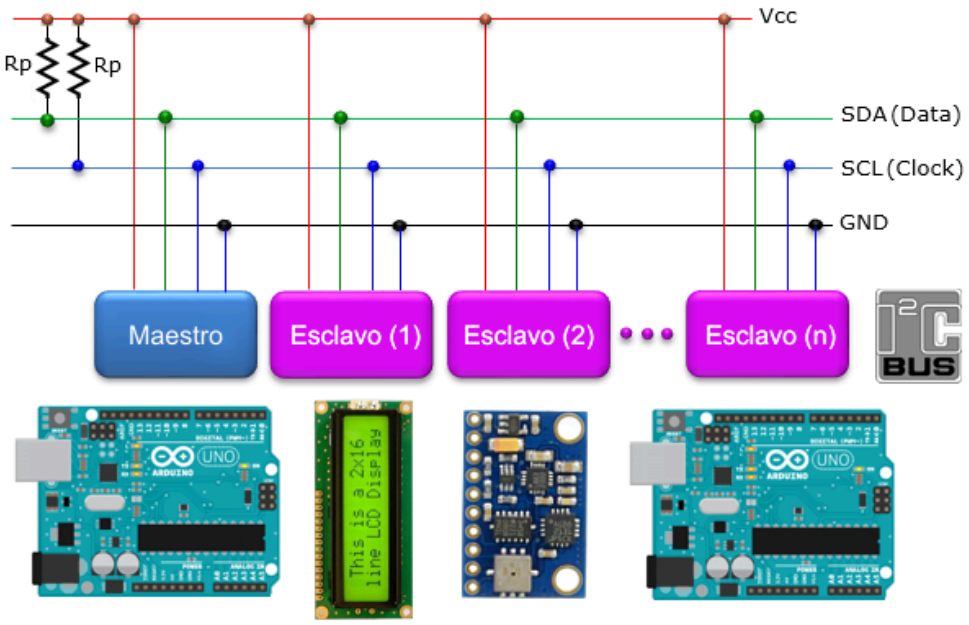
Oscar Gandía Iglesias
Iker Ubide Muñoz
Diego Ruíz Roca
Miquel Benlloch Armijo

INTRODUCCIÓN A I2C:

Nuestro trabajo se basa en el estudio y desarrollo del protocolo de comunicación serial I2C(Inter-Integrated Circuit) que fue creado en 1982 por la empresa Philips Semiconductors. I2C ofrece la comunicación a través de un bus de datos en lo que se denomina comunicación maestro esclavo en el que un único dispositivo ordena el envío de información de uno de sus esclavos, esta comunicación es siempre unidireccional. Básicamente el maestro decide con quién comunicarse y ordena que debe hacer cada esclavo y que quiere recibir de ellos uno por uno.

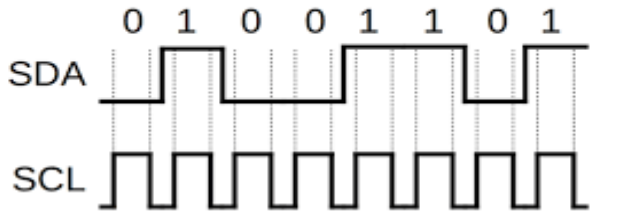
Esta comunicación tiene interacción activa por parte de los dos interlocutores, cada vez que el maestro o el esclavo hablan tiene que confirmar la otra parte que ha recibido la información, lo que da seguridad a la comunicación. Se detalla más adelante los procesos de comunicación.

La conexión entre esclavo y maestro se realiza mediante dos conectores, SDA que contiene el bus de datos que se transmite y SCL el reloj que coordina los pulsos, este reloj proviene del maestro que marca el ritmo de la comunicación.



FUNCIONAMIENTO DEL BUS DE DATOS:

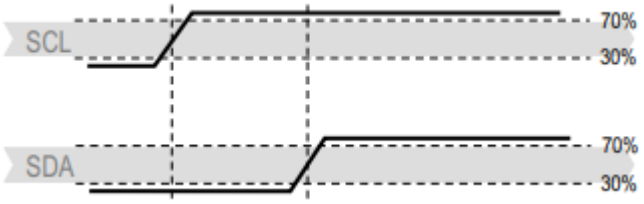
Decimal	Hexadecimal	Binario
77	0x4D	0b01001101



Cuando se configura un dispositivo como maestro, emite una señal de reloj a la frecuencia deseada. También manda una señal SDA inicial a nivel alto que significa que no está enviando información.
De forma abstracta podemos entender la comunicación i2c como un conjunto de instrucciones que se envían para mandar a los esclavos.

START: Toda comunicación empieza con un start, ocurre cuando el SDA cambia de nivel alto abajo a la vez que CLK está a nivel a nivel alto.

STOP: Al finalizar la comunicación SDA mandará una señal de stop que consiste en mandar una señal de nivel alto cuando SCL está a nivel alto.



ACK: Tanto el esclavo como el maestro confirman si han recibido o no el byte de información. Cuando se ha enviado un Byte el emisor deja en alto el SDA y espera que el receptor lo ponga en alto para confirmarlo. Si no lo hace en un tiempo de espera se finalizará la conexión.

NACK: En contraparte NACK se utiliza cuando no recibe un mensaje de forma correcta y deja SDA a nivel alto, también ocurre cuando se ha finalizado una lectura o escritura.

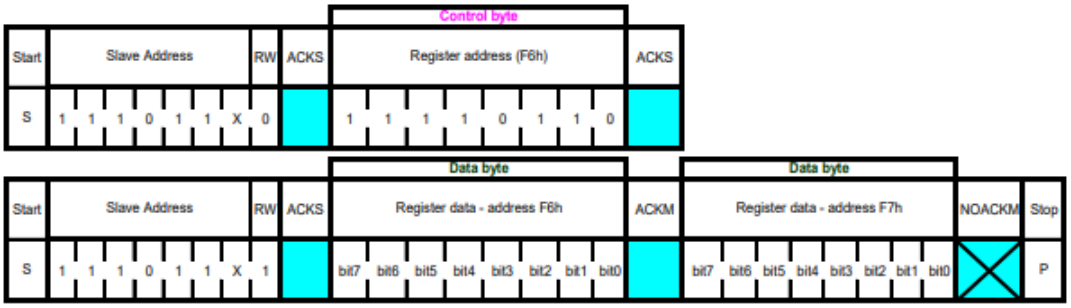
Con estos procedimientos podemos establecer operaciones de lectura y escritura para comunicarnos con otros dispositivos.

COMUNICACIÓN A TRAVÉS DE I2C:

El maestro puede comunicarse con los distintos dispositivos conectados a través de su dirección de esclavo. Esta dirección es propia de cada dispositivo y solo se puede modificar mediante su pineado.

Para comunicarse con un dispositivo el maestro deberá en primer lugar empezar un start y escribir en el bus SDA la dirección del esclavo, si éste existe en la línea SDA y SCL le mandará un ACK para confirmarlo.
A partir de ahora podemos comunicarnos con este dispositivo en cada uno de sus registros. Para acceder a un registro escribiremos la dirección de ese registro de la misma forma que lo hicimos con la dirección de esclavo. Seguidamente podemos ordenar la escritura o lectura de ese registro.

Para ordenar escribir o leer pondremos la dirección de ese esclavo (8 bits) en el primer bit debe ser 0 si queremos escribir o 1 si queremos leer. Por lo general se nos proporciona una dirección del dispositivo específico y la desplazamos un bit, el bit vacío será ese bit de lectura o escritura.



En este ejemplo del sensor BMP280 podemos ver como se escribe la dirección del esclavo seguida del registro, después es necesario volver a hacer un start porque ahora queremos leer, entonces le volvemos a pasar la dirección del esclavo en modo lectura y comenzamos a recibir datos.

La comunicación acaba con un NACK y un STOP que ponen el bus en nivel alto de nuevo.

DESCRIPCIÓN DEL PROYECTO:

En el proyecto se ha elaborado una estación meteorológica, esta cuenta con varios sensores y una pantalla que son los esclavos de una ESP32 S3 que será el maestro.

Entre los dispositivos se encuentran un sensor de presión, de temperatura y humedad, de luminosidad, una brújula y una pantalla oled. Estos se detallarán más abajo.

La lógica interna del ESP32 inicializa los sensores y la comunicación I2C y elige uno a uno con quién quiere comunicarse. Los datos se manejan correctamente mediante tareas y mutex para evitar condiciones de carrera. Estos se transmiten al dispositivo de la pantalla y se envían mediante un servidor Bluetooth.

DESCRIPCIÓN DE LA PROGRAMACIÓN EN ESP IDF:
En la programación del proyecto se ha optado por la ESP32 y el uso de las librerías oficiales que se ofrecen para el lenguaje C.
En este apartado se explicarán las funcionalidades que ofrece la librería para su correcta programación.

```
#include "i2c.h"
```

Lo inicial para la comunicación en I2C es configurar el la ESP32 en modo maestro.

```
i2c_config_t config = {
    .mode = MODE,
    .sda_io_num = GPIO_SDA,
    .scl_io_num = GPIO_CLK,
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_pullup_en = GPIO_PULLUP_ENABLE,

    #if MODE ==  I2C_MODE_MASTER
    .master.clk_speed = I2C_MASTER_FREQ_HZ,
    #else
    config.slave.addr_10bit_en = ;
    config.slave.slave_addr = ;
    config.slave.maximun_speed = ;
    #endif
};

i2c_param_config(I2C_MASTER_NUM, &config);

i2c_driver_install(I2C_MASTER_NUM, config.mode, I2C_MASTER_RX_BUF_DISABLE, I2C_MASTER_TX_BUF_DISABLE, 0);
```

De la siguiente forma se configura la ESP32 en modo maestro. La función i2c_param_config recibe los parámetros de configuración, MODO (maestro/esclavo), pines SDA y SCL, pull ups para habilitar la recepción y emisión de datos y la velocidad de reloj. Los otros parámetros son los de la configuración de modo esclavo y no los usamos.
Por último instalamos los drivers con la configuración que hemos creado, también hay además configuración extra de buffer y de flags que no hemos implementado.

A nivel más bajo podemos encontrar las funciones que usamos para enviar órdenes de escritura o lectura, son las siguientes.

```
i2c_cmd_handle_t cmd = i2c_cmd_link_create();
```

Esta función inicializa el I2C y tiene listo un buffer de tamaño variable para almacenar toda la información que se va a enviar.

Las siguientes funciones que involucran cmd guardan en el buffer los distintos bytes que se van a enviar:

```
esp_err_t i2c_master_start(i2c_cmd_handle_t cmd_handle)
```

Comienza con un start.

```
esp_err_t i2c_master_write_byte(i2c_cmd_handle_t cmd_handle, uint8_t data, bool ack_en)
```

Escribe un solo byte, recibe como parámetros el cmd, el byte a escribir y la confirmación del ACK.

```
esp_err_t i2c_master_write(i2c_cmd_handle_t cmd_handle, const uint8_t *data, size_t data_len,
bool ack_en)
```

Escribe varios bytes seguidos, recibe un puntero a un array de bytes y la longitud que queremos leer además.

```
esp_err_t i2c_master_read(i2c_cmd_handle_t cmd_handle, uint8_t *data, size_t data_len,
i2c_ack_type_t ack)
```

Lee un varios bytes en un registro ya especificado, recibe el puntero a ese array que va a escribir la función y su longitud.

```
esp_err_t i2c_master_stop(i2c_cmd_handle_t cmd_handle)
```

Stop, finaliza comunicación.

```
esp_err_t i2c_master_cmd_begin(i2c_port_t i2c_num, i2c_cmd_handle_t cmd_handle, TickType_t
ticks_to_wait)
```

Esta función envía todos los datos encolados a través del maestro. Esta función está protegida por MUTEX. Además también establece el timeout máximo para la espera del ACK.

```
void i2c_cmd_link_delete(i2c_cmd_handle_t cmd_handle)
```

Borra el cmd para ahorrar memoria.

Con ello un código de lectura del sensor BMP280 se vería así:

```
esp_err_t err=0;
i2c_cmd_handle_t cmd = i2c_cmd_link_create();
uint8_t buffer_read[6] = {0, 0, 0, 0, 0, 0};
err +=i2c_master_start(cmd);
err +=i2c_master_write_byte(cmd, ADDRS_BMP280 << 1 | I2C_MASTER_WRITE, true);
err +=i2c_master_write_byte(cmd, ADDRS_READ, true);
err +=i2c_master_start(cmd);
err +=i2c_master_write_byte(cmd, ADDRS_BMP280 << 1 | I2C_MASTER_READ, true);
err+= i2c_master_read(cmd, buffer_read, 6, I2C_MASTER_LAST_NACK);
err+=i2c_master_stop(cmd);
err+=i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, TIME_OUT / portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd);
```

La librería i2c.h de Espressif también ofrece otras funciones que abstraen estos procedimientos como:

```
esp_err_t i2c_master_write_to_device(i2c_port_t i2c_num, uint8_t device_address, const uint8_t
*write_buffer, size_t write_size, TickType_t ticks_to_wait)
```

```
esp_err_t i2c_master_write_read_device(i2c_port_t i2c_num, uint8_t device_address, const uint8_t
*write_buffer, size_t write_size, uint8_t *read_buffer, size_t read_size, TickType_t
ticks_to_wait)
```

Ambas leen y escriben en los registros combinando las funciones anteriores.

ESP32S3 - I2C

En el Esp32-s3 está configurado por defecto SDA-gpio 8, SCL-GPIO 9. Sin embargo es posible utilizar cualquier pin GPIO para la comunicación i2c.Desde el ESP32S3 se puede elegir el modo de comunicación (esclavo o Master) permitiendo la comunicación por i2c entre varios microcontroladores, cuenta con dos modos de velocidad:
Standard mode(100 kbits/s) y Fast Mode(400 kbit/s).

Además no solo se limita a las 7-bit slave address, que son las predominantes, sino que también prevé 10-bit slave address, lo que permite aumentar el número de dispositivos conectados a la vez.

Cuenta con memoria RAM asignada para I2C, TX RAM guarda los datos que el controlador tiene que enviar por I2C. Mientras que RX RAM se encarga de guardar los datos que se reciben mediante la comunicación I2C.

SENSORES:
A continuación ofrecemos una breve explicación del funcionamiento de cada uno de los sensores

VEML7700

Este sensor se encarga de devolver la luz ambiental con una resolución de 16 bits.Funciona desde 10kHz hasta 400kHz.
Dirección de esclavo:0x10

Para comenzar a utilizarlo hace falta configurar los distintos parámetros con los que cuenta para poder medir bien la luz en las diferentes situaciones.Para ello, nos hemos creado una estructura de datos que cuenta con todo lo necesario:

```
typedef struct
{
    uint16_t gain;              /*!< Ganancia del sensor*/
    uint16_t integration_time;  /*!< Sensor integration time configuration */
    uint16_t persistance;       /*!< Last result persistance on-sensor configuration */
    uint16_t interrupt_enable;  /*!< Enable/disable interrupts */
    uint16_t shutdown;          /*!< Shutdown command configuration */
    float resolution;           /*!< Current resolution and multiplier */
    uint32_t maximum_lux;       /*!< Current maximum lux limit */
}Config_veml7700;

//Variables globales
Config_veml7700 sensor_luz;
```

En la función config_veml7700() hemos configurado cada parámetro para que sea lo más óptimo para medir bien al aire libre, como se puede observar a continuación:

```
sensor_luz.gain=VEML7700_GAIN_1_8;
sensor_luz.integration_time=VEML7700_IT_100MS;
sensor_luz.persistance=VEML7700_PERS_1;
sensor_luz.interrupt_enable=false;
sensor_luz.shutdown=VEML7700_POWERSAVE_MODE1;
sensor_luz.resolution=0.5376;
sensor_luz.maximum_lux=35232;
```

Para la resolución y la máxima luz del sensor hay que mirar la tabla del VEML7700 ya que según la ganancia y el integration time que elijas se puede modificar los valores.

Una vez añadido todos los datos,hemos metido los datos en un uint16 siguiendo el datasheet(ir a datasheet VEML7700 Table 1-Configuration Register #0), y lo hemos enviado al registro 0X00 que es el que se encarga de la configuración

```
//METO los datos del sensor que voy a enviar para configurarlo
uint16_t config_data =(
    (sensor_luz.gain<<11) |
    (sensor_luz.integration_time<<6) |
    (sensor_luz.persistance<<4) |
    (sensor_luz.interrupt_enable<<1) |
    (sensor_luz.shutdown<<0)
);

i2c_cmd_handle_t cmd = i2c_cmd_link_create();
error+=i2c_master_start(cmd);
error+=i2c_master_write_byte(cmd, (veml7700_slave_address << 1) | I2C_MASTER_WRITE, true);
error+=i2c_master_write_byte(cmd,command_config, true);

uint8_t write_data[2];
write_data[0] = config_data&0xff;
write_data[1] = (config_data>>8)&0xff;
error+=i2c_master_write(cmd, write_data, 2, true);

i2c_master_stop(cmd);

error+= i2c_master_cmd_begin(master_num, cmd, 1000 / portTICK_PERIOD_MS);
```

Una vez configurado el sensor tan solo hace falta implementar una función que lea el sensor y guarde el valor en el puntero que le pasamos como parámetro. Como el sensor envía los datos con resolución de 16 bits, es necesario crear un vector/puntero que pueda guardar la información. En nuestro caso hemos usado un vector uint8_t de tamaño 2.

Para que funcione, primero tenemos que mandar la dirección del esclavo y escribir qué dirección de registro queremos leer, en este caso utilizamos la 0x04 (véase el datasheet).

```
    error+=i2c_master_write_byte(cmd, (veml7700_slave_address << 1) | I2C_MASTER_WRITE, tr
    error+=i2c_master_write_byte(cmd, command_read, true);
```

Ahora, necesitaremos mandar de nuevo la dirección de esclavo para decirle que queremos leer y en la función “i2c_master_read” pasaremos como puntero nuestro vector:

```
  error+=i2c_master_start(cmd);
  error+=i2c_master_write_byte(cmd, (veml7700_slave_address << 1) | I2C_MASTER_READ, t


  error+=i2c_master_read(cmd, read_data, 2, I2C_MASTER_LAST_NACK);
```

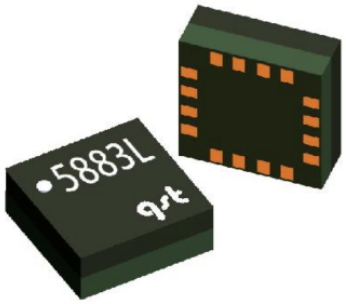
Finalmente hará falta hacer una pequeña transformación al valor leído. primero ordenando los uint8_t del MSB a LSB, y en segundo lugar multiplicando el valor por la resolución que hayamos configurado:

```
valor_leido = read_data[0] | (read_data[1]<<8);
*lux=valor_leido * sensor_luz.resolution;
```

GY-271:

El sensor GY-271 incluye dos sensores, el primero es el HT20 que ya usamos en otro sensor. El segundo sensor es el QMC5883L. Es un sensor de brújula diseñado para aplicaciones de alta precisión como navegación, drones y robots entre muchas otras.

El QMC5883L está basado en el efecto Hall, diseñado para medir campos magnéticos en tres dimensiones. Es capaz de detectar el campo magnético terrestre y proporcionar información sobre la dirección del norte magnético.



Configuración y características:

- **Sensibilidad**: El sensor tiene una alta sensibilidad que le permite detectar incluso pequeñas variaciones en el campo magnético.

- **Rangos de medición:** Puede medir campos magnéticos en un rango de ±2 y ±8 Gauss.

- **Resolución**: Ofrece una alta resolución de lectura (2 bytes), lo que permite mediciones precisas.

- **Alimentación**: entre 1.65 v y 3.6 v.

- **Modos de funcionamiento**: El QMC5883L admite varios modos de funcionamiento, como el modo de medición continuo y el modo de medición de un solo disparo. El modo de medición continua permite obtener lecturas continuas del campo magnético, mientras que el modo de un solo disparo realiza una sola medición y luego entra en modo de espera.

I2C en el QMC5883L:

El sensor tiene como dirección por defecto la 0x0D, y dispone de varios registros, unos de los cuales (del 0 al 5) son para leer la información del sensor y otros para configurarlo a tu gusto escribiendo bits en ciertas posiciones de cada registro.

Primero : ¿cuales son los pasos para leer y escribir?, nos dan un buen esquema en el Datasheet.

Table 11. I²C Write

START	Slave Address							R W	SACK	Register Address (0x09)								SACK	Data (0x01)								SACK	STOP
	0	0	0	1	1	0	1	0		0	0	0	0	1	0	0	1		0	0	0	0	0	0	0	1		

Table 12. I²C Read

START	Slave Address							R W	SACK	Register Address (0x00)								SACK										
	0	0	0	1	1	0	1	0		0	0	0	0	0	0	0	0											
START	Slave Address							R W	SACK	Data (0x00)								MACK	Data (0x01)									
	0	0	0	1	1	0	1	1		0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	1		
MACK	Data (0x02)								MACK	....								MACK	Data (0x07)								NACK	STOP
	0	0	0	0	0	0	1	0		....									0	0	0	0	0	1	1	1		

Configuración de registros:

```
esp_err_t err = 0;
i2c_cmd_handle_t cmd = i2c_cmd_link_create();
i2c_master_start(cmd);
err += i2c_master_write_byte(cmd, QMC5883L_ADDR << 1 | I2C_MASTER_WRITE, true);
err += i2c_master_write_byte(cmd, QMC5883L_CONFIG2, true);
err+=i2c_master_write_byte(cmd, 0x41, true);
i2c_master_stop(cmd);
err += i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, TIME_OUT / portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd);
```

En el registro 0A escribimos 0x, para habilitar el ROL_PNT y el INT_ENB.

Table 19. Control Register 2

Addr.	7	6	5	4	3	2	1	0
0AH	SOFT_RST	ROL_PNT						INT_ENB

Luego ponemos la configuración deseada en el registro 09, y la escribimos:

```
uint8_t controlregister1 = QMC5883L_CONFIG_OS256 | QMC5883L_CONFIG_2GAUSS | QMC5883L_CONFIG_100HZ | QMC5883L_CONFIG_CONT;
```

Table 18. Control Register 1

Addr	7	6	5	4	3	2	1	0
09H	OSR[1:0]		RNG[1:0]		ODR[1:0]		MODE[1:0]	

Reg.	Definition	00	01	10	11
Mode	Mode Control	Standby	Continuous	Reserve	Reserve
ODR	Output Data Rate	10Hz	50Hz	100Hz	200Hz
RNG	Full Scale	2G	8G	Reserve	Reserve
OSR	Over Sample Ratio	512	256	128	64

Ahora la escribimos:

```
i2c_cmd_handle_t cmd2 = i2c_cmd_link_create();
err +=i2c_master_start(cmd2);
err +=i2c_master_write_byte(cmd2, QMC5883L_ADDR << 1 | I2C_MASTER_WRITE, true);
err +=i2c_master_write_byte(cmd2, QMC5883L_CONFIG, true);
err+=i2c_master_write_byte(cmd2, controlregister1, true);
err+=i2c_master_stop(cmd2);
err+=i2c_master_cmd_begin(I2C_MASTER_NUM, cmd2, TIME_OUT / portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd2);
```

Lectura de datos:

Para obtener lecturas del campo magnético en los tres ejes (x, y, z), se leen los valores de los registros correspondientes utilizando una secuencia de comandos I2C. Los valores leídos se interpretan como la intensidad del campo magnético en cada eje.

Lo primero es **calibrar** el sensor, esto significa calcular un “offset” en X y en Y para que las medidas sean más correctas :

```
void calculo_offset()

    for (int i = 0; i < 1000; i++)
    {
        read_data_bytes();
        if (xbytes > x_max) x_max = xbytes;
        if (xbytes < x_min) x_min = xbytes;
        if (ybytes > y_max) y_max = ybytes;
        if (ybytes < y_min) y_min = ybytes;
        if (zbytes > z_max) z_max = zbytes;
        if (zbytes < z_min) z_min = zbytes;

        printf("%d \n", (int)xbytes);
        printf("%d \n", (int)ybytes);
        printf("%d \n", (int)zbytes);

        vTaskDelay(pdMS_TO_TICKS(10));
    }

    x_offset = (x_max + x_min) / 2;
    y_offset = (y_max + y_min) / 2;
    z_offset = (z_max + z_min) / 2;
```

-Lo que se hace es llamar a la función que lee los datos dentro de un for, y al dato leído compararlo con un máximo y un mínimo. Ponemos un delay para que le dé tiempo a que varíen las medidas. Al acabar calculamos el offset como una media entre el máximo y el mínimo.

Es **importante realizar una calibración inicial** del sensor para compensar las interferencias magnéticas locales y garantizar mediciones precisas. Esto implica realizar movimientos circulares con el sensor en todas las direcciones para mapear el campo magnético circundante y calcular los valores de compensación necesarios.

Antes de leer los datos hay que **comprobar si hay datos nuevos a leer**, y esto lo sabemos si en el registro 06 en el primer bit tenemos un uno o un cero:

```
err +=i2c_master_start(cmd);
err +=i2c_master_write_byte(cmd, QMC5883L_ADDR << 1 | I2C_MASTER_WRITE, true);
err +=i2c_master_write_byte(cmd, QMC5883L_STATUS, true);
err +=i2c_master_start(cmd);

err +=i2c_master_write_byte(cmd, QMC5883L_ADDR << 1 | I2C_MASTER_READ, true);
// err+=i2c_master_read_byte(cmd, data, I2C_MASTER_LAST_NACK);
err+= i2c_master_read(cmd, data, 1, I2C_MASTER_LAST_NACK);
err+=i2c_master_stop(cmd);
err+=i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, TIME_OUT / portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd);

//data &= 0x01;

data[0] = data[0] << 7;


return data[0];
```

Aquí seguimos la secuencia de lectura, la información leída se guarda en el buffer data y lo leemos. Esta función llama check status retorna true si se puede leer datos y false si no.

Ahora vamos a **leer los datos** “crudos”:

```
if (check_status() > 0)
{


    err +=i2c_master_start(cmd);
    err +=i2c_master_write_byte(cmd, QMC5883L_ADDR << 1 | I2C_MASTER_WRITE, true);
    err +=i2c_master_write_byte(cmd, QMC5883L_X_LSB, true);
    err +=i2c_master_start(cmd);

    err +=i2c_master_write_byte(cmd, QMC5883L_ADDR << 1 | I2C_MASTER_READ, true);
    err+= i2c_master_read(cmd, buffer_read, 6, I2C_MASTER_LAST_NACK);
    err+=i2c_master_stop(cmd);
    err+=i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, TIME_OUT / portTICK_PERIOD_MS);
    i2c_cmd_link_delete(cmd);

}


xbytes = buffer_read[1] << 8 | buffer_read[0] ;
ybytes = buffer_read[3] << 8 | buffer_read[2] ;
zbytes = buffer_read[5] << 8 | buffer_read[4] ;
```

Si es mayor que cero el resultado de check status es cuando puedo leer datos nuevos.

En cada posición del buffer se guarda un byte, y como podemos ver en la tabla de abajo cada coordenada lee de dos registros, siendo el primero q lee LSB y el siguiente MSB, por esto en las variables int16 donde me guardo las coordenadas enteras tengo que hacer un or y mover el MSB 8 posiciones para ponerlo a la derecha:

Table 13. Register Map

Addr.	7	6	5	4	3	2	1	0	Access
00H	Data Output X LSB Register XOUT[7:0]								Read only
01H	Data Output X MSB Register XOUT[15:8]								Read only
02H	Data Output Y LSB Register YOUT[7:0]								Read only
03H	Data Output Y MSB Register YOUT[15:8]								Read only
04H	Data Output Z LSB Register ZOUT[7:0]								Read only
05H	Data Output Z MSB Register ZOUT[15:8]								Read only

Por último realizamos la **conversión a grados:**

Una vez obtenidos los valores de los ejes x e y, se utilizan para calcular el ángulo de dirección del campo magnético. Esto se hace típicamente utilizando la función `atan2`, que proporciona el ángulo en radianes. Luego, este ángulo se convierte a grados para obtener la dirección de la brújula.

- Primero se normalizan los datos con la calibración.

```
xbytes -= x_offset ;
ybytes -= y_offset;
```

- Luego pasamos a radianes:

```
float heading = atan2((double)ybytes, (double)xbytes) * (180.0 / M_PI);
```

- Cuando ya lo tenemos en radianes pasamos a grados y comprobamos que no sean negativos ni superen los 360º:

```
float declination_angle = (DECLINATION_ANGLE_DEGREES + (DECLINATION_ANGLE_MINUTES / 60.0)) / (180 / M_PI);
heading += declination_angle;
    if (heading < 0.0f)
{
    heading += 360.0f;
}
if (heading > 360.0f)
{
    heading -= 360.0f;
}
```

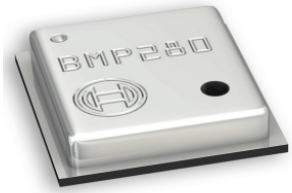
Nota : el declination angle es según el lugar de la tierra hay que sumar unos grados y unos minutos a la dirección.

Conclusión:

El QMC5883L es un sensor de brújula versátil y preciso que proporciona mediciones confiables del campo magnético terrestre.

BMP280:

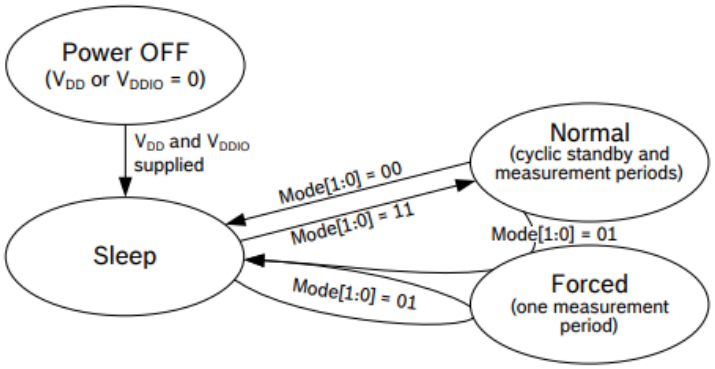
El sensor BMP280 es un sensor digital que se encarga de medir tanto temperatura como presión.



El sensor cumple con la siguientes características:
Suministro de voltaje : 1.7 - 3.6 v.
Intensidad máxima : 42uA.

Rango de temperatura : -40ºC - 85ºC
Rango de presión: 300 - 1100 hpa.

Este sensor opera en diferentes estados:
Sleep Mode, Normal Mode y Forced Mode.
En el Modo Sleep el dispositivo no hace ninguna medida y entra en modo ahorro de energía consumiendo menos corriente.
Al configurar el sensor se puede pasar al modo Normal o al modo Forced.
El modo normal establece una alternancia entre el modo Forced que es el que salta a la medición y el modo Sleep que se ejecuta periódicamente.



El sensor además ofrece las siguientes configuraciones:

-t standby: Tiempo entre cada medida en el modo Normal. (0.5 ms - 500 ms …)
-Resolución:
-Oversampling: Número de muestras en cada muestreo de presión (reduce ruido).
-Filtro IIr: Coeficiente de filtro para medidas de presión.

Muchas de estas medidas se han puesto por su valor por defecto especificado en el Datasheet para una medida forzada.

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]				filter[2:0]			spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]				osrs_p[2:0]		mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x58
calib25...calib00	0xA1…0x88	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Revision	Reset
Type:	do not write	read only	read / write	read only	read only	read only	write only

Aquí se muestra el mapa de memoria del sensor. Para asegurar la medida se ha configurado el sensor escribiendo en los registros 0xF4 y 0xF5.
La lectura se realiza entre los registros 0xF7 y 0xFC, además se requiere para calcular las medidas de los datos leídos entre los registros 0x88 y 0xA1.

```
void init_BMP280()
{
    esp_err_t err=0;
    err=set_config_byte();
    if(err!=ESP_OK)
    {
        //printf("Error en config");
        return;
    }
    err=get_calibration_data();
    if(err!=ESP_OK)
    {
        //printf("Error en get_calibration");
        return;
    }
    is_not_init=err;
}
```

```
void get_BMP280_packet()
{
    if(is_not_init==ESP_OK)
    {
        esp_err_t err=0;

        err+=write_command_bytes();
        if(err!=ESP_OK){
           // printf("Error en write\n");
            return;
        }
        //printf("Write correcto\n");
        err+=read_data_bytes();
        if(err){
            //printf("Error en read\n");
            return;
        }
        //printf("Read correcto\n");
        calculate_measure();
    }
    else
    init_BMP280();
}
```

La estructura de la librería para manejar el sensor funciona de esta manera.
Se inicia el sensor escribiendo en el registro 0xF5, posteriormente se leen los datos de calibración.
Cada vez que se va a medir se escribe en el registro 0xF5 y posteriormente se leen los datos.Para calcular, el datasheet ofrece una función para calcular la temperatura y presión utilizando los valores leídos del sensor y los parámetros obtenidos en la inicialización.

Oled SSD1306:

La Oled ssd1306 es una pantalla que cuenta 126x64 pixeles para imprimir datos, esta pantalla en concreto es perfecta para desarrollar el proyecto ya que tiene la posibilidad de conectarse al esp32-s3 mediante comunicación i2c.
Para ello utilizaremos una plantilla proporcionada por Espressif llamada **i2c_oled**.

Analicemos el código necesario para hacer esta conexión posible.
Para que la pantalla funcionase utilizamos la librería **lvgl.h** (Light and Versatile Graphics Library).

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "esp_timer.h"
#include "driver/i2c.h"
#include "esp_err.h"
#include "esp_log.h"
#include "lvgl.h"
#include "esp_lvgl_port.h"
```

Esta biblioteca se encuentra declarada en el fichero .h de la pantalla oled junto a otras bibliotecas necesarias para desarrollar correctamente esta implementación.

Estas otras bibliotecas son **FreeRTOS** para manipular tareas y **driver/i2c** para manipular la conexión mediante i2c entre otras.

1-Definición de datos:

Lo primero que nos encontramos en este código es la definición de algunos parámetros importantes.

```
#define EXAMPLE_LCD_PIXEL_CLOCK_HZ     (400 * 1000)
#define EXAMPLE_PIN_NUM_RST            -1
#define EXAMPLE_I2C_HW_ADDR            0x3C

// The pixel number in horizontal and vertical
#if CONFIG_EXAMPLE_LCD_CONTROLLER_SSD1306
#define EXAMPLE_LCD_H_RES              128
#define EXAMPLE_LCD_V_RES              64
#elif CONFIG_EXAMPLE_LCD_CONTROLLER_SH1107
#define EXAMPLE_LCD_H_RES              64
#define EXAMPLE_LCD_V_RES              128
#endif
// Bit number used to represent command and parameter
#define EXAMPLE_LCD_CMD_BITS           8
#define EXAMPLE_LCD_PARAM_BITS         8
```

Estas definiciones son los píxeles de la pantalla Oled y la dirección i2c del dispositivo.

A continuación tenemos un par de variables globales y un mutex para protegerlas.

```
lv_disp_t *disp;
extern float hum,temp,press,lux_als,heading;
extern uint8_t valor;

portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;
```

Estas variables son cruciales ya que son las que imprimimos por la pantalla oled, contienen los datos captados por los sensores, mientras que **lv_disp_t *disp** es un puntero a una estructura que contiene los datos necesarios para imprimir texto por la pantalla.

2-Función de impresión:

La primera función que nos topamos en este .c es la que imprime el texto por la pantalla, esto se hace mediante una función llamada

```
lv_label_set_text(label2, hum);
```

Esta función recibe dos parámetros, el primero es una “etiqueta” que es una estructura de tipo **lv_obj_t** la cual contiene todas las especificaciones sobre el texto y el segundo es una cadena de texto con el mensaje a enseñar.

La cadena de texto la generamos y la almacenamos en un buffer ya que contiene una variable global que va cambiando en función de lo que miden los sensores. Esto se consigue mediante un **snprintf();** y tiene este aspecto:

```
snprintf(buffer_hum, sizeof(buffer_hum), "Hum: %0.2f", humedad);
```

Para seleccionar el mensaje que se imprimirá por la pantalla oled implementamos un switch con la variable global valor.

```
switch(valor)
{
    case 0:
```

Esta variable global varía en el archivo Bluetooth.c, cambia en función de la cadena de texto recibida por una aplicación para el móvil que se conecta con el esp32s3 mediante bluetooth.

3-Función de notificación:

La siguiente función, **notify_lvgl_flush_ready** es un callback que notifica a la biblioteca lvgl.h que la operación de actualización de la pantalla ha sido completada

```
static bool notify_lvgl_flush_ready(esp_lcd_panel_io_handle_t panel_io, esp_lcd_panel_io_event_data_t *edata, void *user_ctx)
{
    lv_disp_t * disp = (lv_disp_t *)user_ctx;
    lvgl_port_flush_ready(disp);
    return false;
}
```

4-Tarea en bucle:

Para ir actualizando la información que se imprime por la pantalla se crea una tarea desde la función **main()** llamada

```
xTaskCreate(&oled,"oled",4096,NULL,2,NULL);
```

esta tarea llama en bucle a otra función que se encuentra en Oled.c llamada **imprimir_oled();**

```
void imprimir_Oled()
{
        example_lvgl_demo_ui(disp);
        vTaskDelay(1000/portTICK_PERIOD_MS);
        lv_obj_clean(lv_scr_act());

}
```

Esta función llama a la primera función que hemos visto que imprimia texto por pantalla, tambien cuenta con una pausa de 1 segundo y a continuación llama a la función **lv_obj_clean(lv_scr_act());** la cual limpia todos los pixeles pintados anteriormente para dejar la pantalla en negro de nuevo.
De este modo lo que conseguimos es que los valores no se sobreescriban visualmente en la pantalla oled.

5-Configuración de la SSD1603:

Para que todo esto funcione solamente tenemos que configurar la pantalla con los defines nombrados anteriormente, para ello creamos una funcion llamada **configurar().**

```
ESP_LOGI(TAG, "Install panel IO");
    esp_lcd_panel_io_handle_t io_handle = NULL;
    esp_lcd_panel_io_i2c_config_t io_config = {
        .dev_addr = EXAMPLE_I2C_HW_ADDR,
        .control_phase_bytes = 1,
        .lcd_cmd_bits = EXAMPLE_LCD_CMD_BITS,
        .lcd_param_bits = EXAMPLE_LCD_CMD_BITS,
#if CONFIG_EXAMPLE_LCD_CONTROLLER_SSD1306
        .dc_bit_offset = 6,
```

Esta función también cuenta con manejadores de errores

```
ESP_ERROR_CHECK(esp_lcd_panel_reset(panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_init(panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(panel_handle, true));
```

y con la declaración de algunas estructuras necesarias para el almacenamiento de ciertos datos para la correcta impresión del texto por la pantalla oled.

```
const lvgl_port_display_cfg_t disp_cfg = {
    .io_handle = io_handle,
    .panel_handle = panel_handle,
    .buffer_size = EXAMPLE_LCD_H_RES * EXAMPLE_LCD_V_RES,
    .double_buffer = true,
    .hres = EXAMPLE_LCD_H_RES,
    .vres = EXAMPLE_LCD_V_RES,
    .monochrome = true,
    .rotation = {
        .swap_xy = false,
        .mirror_x = false,
        .mirror_y = false,
    }
};
```

Este código proporciona una solución eficiente para la visualización de datos de sensores en una pantalla OLED utilizando LVGL y ESP-IDF. La correcta configuración del hardware, junto con la robusta gestión de la interfaz gráfica, permite una visualización clara y actualizada de los datos en tiempo real.

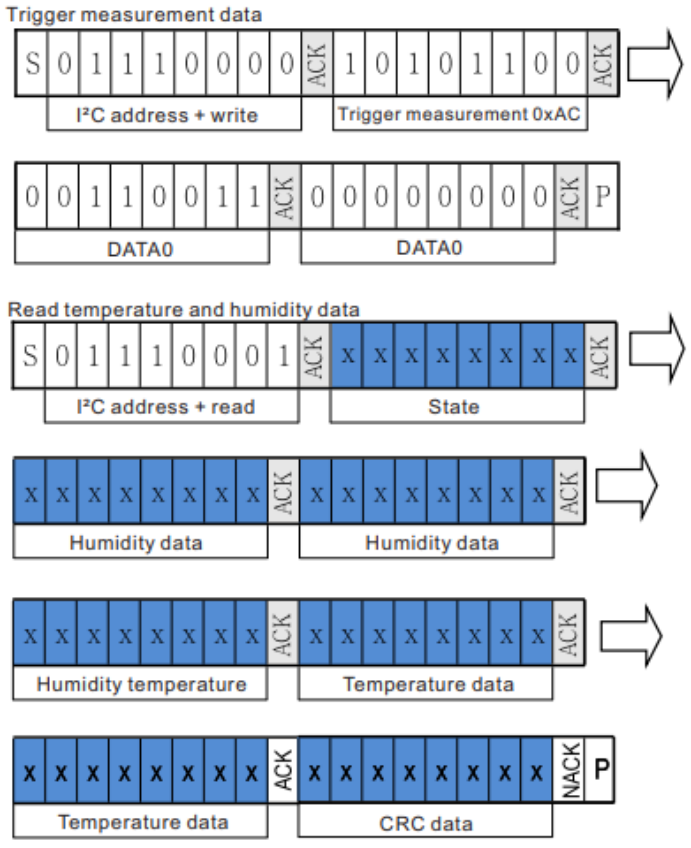
AHT20:

Descripción del Sensor:

El AHT20 es un sensor de alta precisión para medir la humedad y la temperatura, diseñado y fabricado por ASAIR. Este sensor se distingue por su alta precisión, rápida respuesta y estabilidad a largo plazo. Es un dispositivo digital que proporciona datos calibrados, linealizados y compensados a través de una interfaz de comunicación I2C, lo que lo hace ideal para aplicaciones en electrónica de consumo, automatización del hogar y control ambiental.

El AHT20 está encapsulado en un paquete pequeño, lo que facilita su integración en una variedad de dispositivos electrónicos. Además, ofrece una baja potencia de operación, lo que es crucial para aplicaciones alimentadas por baterías.

Este sensor es más simple y no requiere de inicialización para un correcto funcionamiento lo que nos lleva a hacer una lectura periódica controlada para leer la información



Simplemente introducimos el comando especificado por el datasheet 0xAC, 0X33 y 0x00.

Seguidamente podemos leer los siguientes valores, para la medición se guardan en 5 bytes consecutivos habiendo 2 y un medio para cada parámetro.

La medición es de forma lineal.

ANEXOS:

Datasheet de los sensores:

-GY-271
-VEMML7700
-BMP280
-AHT20
-LCD OLED

Librería Espressif: -ESP32 S3