MASB | Arduino
Comunicación serie IV

#stm32 #c #biomedical #microcontroladores #programacion

Albert Álvarez Carulla
hello@thealbert.dev
https://thealbert.dev/

5 de abril de 2022

Índice

1. Comunicación serie - Parte IV (Arduino)

El I2C es un protocolo de comunicación serie más de los muchos que existen. Lo que hace que sea interesante aprender a utilizar el I2C es que es un protocolo muy extendido. **Se utiliza muchísimo.** En esta práctica vamos a ver cómo operar con él. Para ello, utilizaremos un sensor atmosférico, el BME280 de Bosch, que mide temperatura, humedad y presión atmosférica. Para poder trabajar con él, necesitaremos saber realizar operaciones de escritura y lectura a través del I2C, a la vez que jugamos con los bytes leídos para unirlos o desplazar sus bits.

Mientras que en MASB vemos cómo programar, siempre me gusta dar 4 pinceladas de los aspectos más básicos de, en este caso, una comunicación I2C a nivel de electrónica. Para saber qué estamos haciendo realmente cuando programamos.

La comunicación I2C es un tipo de comunicación entre un maestro y uno o varios esclavos. El maestro se encarga de gestionar el bus de comunicación arbitrando quién habla en cada momento. Cada dispositivo conectado al bus I2C tiene una dirección propia, que es la que utilizará el master para indicar quién debe de recibir los datos que se están transmitiendo a través del bus. Ese bus de comunicación está implementado en 2 líneas: SDA y SCL. La línea SCL es una línea en la que viaja una señal de reloj mientras se da una operación de escritura y lectura. Esa señal de reloj la genera el master y para que exista una comunicación, sea cual sea, debe de haber un reloj. Mientras no hay comunicación, el master fija la señal SCL a nivel alto.

La línea SDA igual. Si no se está transmitiendo información, esta se mantiene a nivel alto. Esta señal es la que transporta la información, los datos, propiamente dicha. El valor de SDA se lee en cada flanco de reloj.

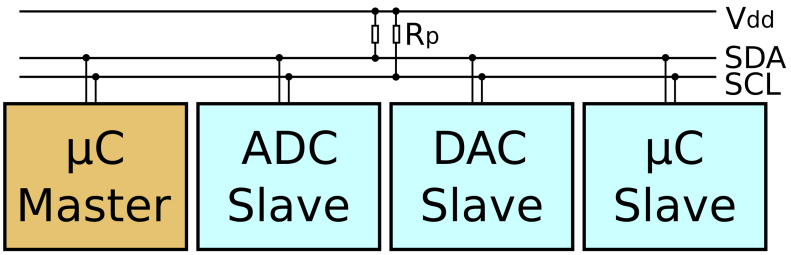

Figura 1: I2C.svg.png

> De en:user:Cburnett - Own work made with Inkscape, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1472017

Y por último (no necesitamos saber mucho más para poder operar con el I2C), el protocolo establece que las direcciones de los dispositivos han de ser de 7 bits. Esta dirección se envía en el primer byte

de la comunicación (en I2C se envían paquetes de bytes, que tienen 8 bits cada uno) y después se hacen llegar el resto de bytes que debe de recibir el esclavo indicado por la dirección suministrada en el primer byte.

"*Pero oye Albert, si se envían paquetes de 8 bits, ¿porque la dirección es de 7 bits?*" Qué buena pregunta, Alicia. Un +1. **El primer byte que contiene la dirección del esclavo utiliza los 7 bits más significativos para indicar esa dirección y el bit menos significativo para indicar si se está realizando una operación de escritura (un 0) o de lectura (un 1).**

Si la operación es de escritura, después del envío de la dirección, el master continúa enviando bytes. Si es una operación de lectura, el master envía el primer byte con la dirección y la operación a realizar, y no envía nada a través de SDA mientras sigue generando una señal de reloj en SCL para que el esclavo indicado conteste.

Pero, ¡eh! No os mareo más la perdiz. Con esto tenéis más que de sobra para poder empezar a programar el I2C en Arduino. ¡Vamos a ello!

1.1. Objetivos

- Introducción a la comunicación serie síncrona I2C en Arduino.
- Uso de la librería Wire de Arduino para la comunicación I2C.
- Union de bytes.
- Desplazamiento de bits.
- Refactorización y modularización.

1.2. Procedimiento

En esta práctica vamos a leer la temperatura, la presión atmosférica y la humedad con un BME280 de Bosch. En este enlace podéis encontrar el *datasheet* del componente. ¡Tenedlo a mano! La aplicación que implementaremos seguirá el siguiente flujo de ejecución:

```
flowchart TD
    subgraph setup["setup()"]
      id1["Init UART serial communication with PC (115200 8N1)"]-->id2["Init I2C communication with BME280"]
        id2-->id3["Enable each measurement with x16 oversampling and exit sleep mode"]
        id3-->id3.5["Get calibration coeficients"]
    end
    subgraph loop["loop()"]
        subgraph temperature["Temperature measurement"]
        direction TB
            id4["Get temperature reading"]
```

```
            id4-->id5["Compute temperature coeficient `t_fine` for next
                preassure and humidty measurements computation"]
            id5-->id6["Compute temperature measurement"]
        end
        subgraph preassure["Preassure measurement"]
            direction TB
            id7["Get preassure reading"]
            id7-->id8["Compute peassure measurement with `t_fine`
               compensation"]
        end
        subgraph humidity["Humidity measurement"]
            direction TB
            id9["Get humidity reading"]
            id9-->id10["Compute humidity measurement with `t_fine`
               compensation"]
        end
        id11["Send data to PC"]
    end
    setup-->loop
    temperature-->humidity-->preassure-->id11
```

1.2.1. Conexión del módulo y establecimiento de la comunicación

Vamos a empezar por el principio y vamos a conectar el módulo BME280 a nuestra placa de evaluación (EVB). Del módulo, utilizaremos 4 de sus pines: VIN, GND, SCK y SDI. Los conectaremos a nuestra placa de evaluación siguiendo la siguiente tabla.

> ¡Recuerda hacer las conexiones con la alimentación desconectada! Es decir, **desconectad el USB del ordenador**.

NUCLEO-F401RE	BME280	Descripción
3V3	VIN	Pin de alimentación. Hay que conectar una tensión de 3.0 a 5.0 V con la que se generará una tensión de 3.0 V para alimentar la electrónica del módulo.
GND	GND	Tensión de referencia tanto para la alimentación como para las señales digitales.
SCL/D15	SCK	El módulo opera tanto con SPI (otro tipo de comunicación síncrona) como con I2C. Este pin se comparte entre ambos tipos de comunicaciones y corresponde a la señal de reloj, lo que en I2C corresponde a la señal SCL. (Que no nos despiste que el fabricante haya querido llamar a este pin SCK utilizando la nomenclatura del SPI).

NUCLEO-F401RE	BME280	Descripción
SDA/D11	SDI	Lo mismo que con el pin SCK. Se comparte entre los dos tipos de comunicaciones disponibles: SPI e I2C. El fabricante del módulo ha decidido mantener la nomenclatura del SPI, pero este pin también es la señal SDA del I2C y corresponde a la señal en la que se envían los datos.

Una vez conectado, enchufamos ya la EVB al ordenador y vamos a hacer una comunicación mínima con el módulo para comprobar simplemente que lo hemos conectado correctamente y el módulo funciona. Para ello, intentaremos leer un registro del BME280 que contiene el ID del sensor. Si podemos leerlo y su valor es el correcto, podemos asegurar que el sensor está bien conectado y opera correctamente.

> **Registros** Pensad en los registros como una tabla en la que cada celda tiene una dirección asociada que la identifica dentro de la tabla. Cada celda contiene información que queremos leer (en el caso de querer obtener los resultados de una medición, o el ID del sensor, por ejemplo) o bien escribir (para establecer la configuración del sensor). Cuando se quiera leer/escribir en una de estas celdas o registros, deberemos de proveer primeramente la dirección de esa celda/registro y seguidamente proveer el valor a escribir en él en el caso de realizar una operación de escritura, o leer el valor devuelto por el sensor en el caso de realizar una operación de lectura. Acordaros que se indica el tipo de operación (lectura/escritura) mediante el último bit del primer byte (en el que también se envía la dirección del esclavo).

El registro con el ID del sensor tiene la dirección `0xD0` y su valor es `0x60`. Por lo tanto, si intentamos leer ese registro y obtenemos el valor esperado, podemos deducir que está todo conectado correctamente. Vamos a ello. El código que utilizaremos es el siguiente:

```
1  //Incluimos la librria Wire que se encarga de gestionar la comunicacion
       I2C
2  #include <Wire.h>
3
4  // En lugar de escribir la direccion de los registros a mano,
5  // utilizamos macros para mejorar la legibilidad y mantenibilidad
6  // del codigo
7  #define BME280_ADDRESS 0x77    // Direccion del sensor
8  #define BME280_REG_ID 0xD0     // Direccion del registro con el ID del
       sensor
9
10 void setup() {
11
12     // Inicializamos la comunicacion UART con el ordenador
13   Serial.begin(115200);
14
```

```
15      // Inicializamos la libreria de comunicacion I2C
16    Wire.begin();
17
18      // Iniciamos la comunicacion con el sensor
19    // Es necesario indicar la direccion del sensor (no confundir con la
20    // direccion que tienen cada uno de los registros del sensor)
21    Wire.beginTransmission(BME280_ADDRESS);
22
23      // Escribimos/indicamos la direccion del registro que queremos leer
24    Wire.write(BME280_REG_ID);
25
26      // Finalizamos la comunicacion
27    Wire.endTransmission();
28
29    // Solicitamos que se nos envie el valor del registro que hemos
          indicado
30    // anteriormente
31      // Para ello, indicamos tambien el numero de bytes que esperamos (
            en este
32    // caso, solo 1)
33    Wire.requestFrom(BME280_ADDRESS, 1);
34
35      // Esperamos que este disponible ese byte que esperamos
36    while(Wire.available() < 1) {};
37
38      // Leemos el valor recibido
39    uint8_t id = Wire.read();
40
41      // Comprobamos si el valor recibido es el esperado e indicamos
42      // el resultado por terminal serie
43    if (id == 0x60) {
44      Serial.println("BME280 connected!");
45    } else {
46      Serial.println("No BME280 found...");
47    }
48  }
49
50  // Aqui no hacemos nada de momento...
51  void loop() {
52
53  }
```

El proceso de lectura es sencillo. Primero enviamos la dirección del registro que queremos leer. Esto lo hacemos con la instrucción `Wire.beginTransmission` que, automáticamente, envía la dirección del esclavo y el bit de operación igual a 0 (escritura). Después, con la instrucción `Wire.write`, se envía la dirección del registro que queremos que luego se nos devuelva. Finalizamos la comunicación con, oh sorpresa, `Wire.endTransmission`.

```
1  ...
2  Wire.beginTransmission(BME280_ADDRESS);
```

```
Wire.write(BME280_REG_ID);
Wire.endTransmission();
...
```

Luego, una vez hemos "escrito" la dirección del registro que queremos leer, vamos a "leer" su contenido. Lo hacemos con la instrucción `Wire.requestFrom`. Esta función se encarga de, automáticamente, enviar la dirección del esclavo y el bit de operación igual a 1 (lectura).

```
...
Wire.requestFrom(BME280_ADDRESS, 1);
while(Wire.available() < 1) {};
uint8_t id = Wire.read();
...
```

En `Wire.requestFrom` indicamos la dirección del dispositivo y luego el número de bytes que queremos recibir. En este caso solo 1, pero podríamos solicitar más de uno. Si este fuera el caso, se nos devolvería primeramente el byte en la dirección indicada y los siguiente bytes serían los valores de los registros consecutivos. Es decir, si quisiéramos leer los registros `0x01`, `0x02` y `0x03`, podríamos hacer:

```
...
// Valor registro 0x01
Wire.beginTransmission(BME280_ADDRESS);
Wire.write(0x01);
Wire.endTransmission();

Wire.requestFrom(BME280_ADDRESS, 1); // Esperamos 1 byte
while(Wire.available() < 1) {};

uint8_t value1 = Wire.read();

// Valor registro 0x02
Wire.beginTransmission(BME280_ADDRESS);
Wire.write(0x02);
Wire.endTransmission();

Wire.requestFrom(BME280_ADDRESS, 1); // Esperamos 1 byte
while(Wire.available() < 1) {};

uint8_t value2 = Wire.read();

// Valor registro 0x03
Wire.beginTransmission(BME280_ADDRESS);
Wire.write(0x03);
Wire.endTransmission();

Wire.requestFrom(BME280_ADDRESS, 1); // Esperamos 1 byte
while(Wire.available() < 1) {};

```

```
uint8_t value3 = Wire.read();
...
```

o bien podemos hacer de manera más compacta:

```
...
// Valor registro 0x01, 0x02 y 0x03
Wire.beginTransmission(BME280_ADDRESS);
Wire.write(0x01);
Wire.endTransmission();

Wire.requestFrom(BME280_ADDRESS, 3); // Esperamos 3 bytes
while(Wire.available() < 3) {};

uint8_t value1 = Wire.read();
uint8_t value2 = Wire.read();
uint8_t value3 = Wire.read();
...
```

Fácil, ¿cierto? Sobretodo, ¡tened en cuenta que esto solo sirve para **registros consecutivos**!

Bueno, si nos ha funcionado el código y vemos un "BME280 connected!" seguimos adelante. Si no es el caso, revisa las conexiones.

1.2.2. Lectura de la temperatura

Vamos a proceder con la primera lectura de una medida. En este caso, la temperatura. Para ello, lo primero que tenemos que hacer es habilitar esa medida y poner el sensor en modo "Normal". Y es que, por defecto, las mediciones vienen deshabilitadas y en modo *sleep* (bajo consumo) cuando se enciende el sensor.

1.2.2.1. Configuración de la temperatura El registro en el que indicar la configuración que habilite la medición de la temperatura y haga que el sensor opere en modo "Normal" es el `ctrl_meas`, con dirección `0xF4`. Este registro contiene la configuración de diferentes aspectos del sensor (es muy común que diferentes configuraciones compartan un mismo registro para evitar sobredimensionar la memoria necesaria para almacenar esos registros). Cada bit o grupo de bytes del registro configuran diferentes aspectos del sensor. En el caso del registro `ctrl_meas`, las diferentes configuraciones son:

Bits	Nombre	Descripción
7, 6, 5	osrs_t[2:0]	Configura el modo de operación de la medición de temperatura.

Bits	Nombre	Descripción
4, 3, 2	osrs_p[2:0]	Configura el modo de operación de la medición de presión.
1, 0	mode[1:0]	Configura el modo de operación del sensor.

En este caso, de momento solo nos preocupa los bits osrs*t[2:0] y mode[1:0]. Como podemos ver en el* datasheet_ del sensor, para habilitar la temperatura necesitamos indicar alguno de estos valores:

osrs_t[2:0]	Modo de medición de la temperatura
000	Deshabilitado (por defecto)
001	Oversampling x1
010	Oversampling x2
011	Oversampling x4
100	Oversampling x8
>=101	Oversampling x16

Con que no estuviera en modo 000 nos sería más que suficiente para hacer que funcionase la medición de temperatura. *Oversampling* quiere decir que toma una serie de puntos y hace una media de ellos con la intención de reducir el ruido en la medición. Vamos a escoger el *oversampling* máximo de x16 (es decir, tomará 16 mediciones de temperatura y se nos devolverá su media). Para ello, escogeremos osrs_t[2:0] = 101.

Para configurar el modo “Normal”, siguiendo las indicaciones del *datasheet*, indicaremos mode[1:0]=11.

Vamos a ello en el código:

```
#include <Wire.h>

#define BME280_ADDRESS 0x77
#define BME280_REG_CTRL_MEAS 0xF4  // Direccion del registro
    configuracion

void setup() {
  Serial.begin(115200);

  Wire.begin();

    // Escribimos la configuracion en el registro
```

```
12    Wire.beginTransmission(BME280_ADDRESS);
13
14    Wire.write(BME280_REG_CTRL_MEAS);  // Indicamos la direccion del
          registro
15    Wire.write(0b10100011);            // Valor que queremos almacenar
16                                         // en el registro
17
18    Wire.endTransmission();
19  }
20
21  void loop() {
22
23  }
```

Como podemos ver, podemos indicar un valor en binario. Esto lo hacemos indicando `0b` seguido del número en binario que queremos indicar. Podemos ver que hemos indicado `0b10100011`, es decir, `101 000 11`, siendo, osrs_t[2:0], osrs_p[2:0] y mode[1:0], respectivamente. Con esto ya tenemos el sensor despierto y con la temperatura habilitada. Vamos a leer las mediciones que va realizando el sensor.

1.2.2.2. Medición de la temperatura Antes de meternos en código, vamos a ver qué hay que hacer para obtener la medición. El sensor captura una medida de su sensor de temperatura, pero esta no se puede usar directamente para indicar la temperatura. Es necesario aplicarle una transformación para obtener la temperatura de verdad. Esa transformación utiliza una serie de coeficientes que están almacenados en la memoria/registros del dispositivo. Estos valores son medidos/calculados durante la fabricación de cada dispositivo y son grabados en su memoria. Por lo tanto, cada dispositivo tiene valores de calibración únicos. Una vez leídos esos valores de calibración, se los aplicamos a la lectura del sensor de temperatura y podemos obtener la temperatura real. ¿Qué fórmula/expresión/transformación hemos de aplicar? Eso nos lo dice el *datasheet*. Tenemos diferentes fórmulas de "compensación" (como las llama el *datasheet*) en función de la resolución/precisión deseada. Nosotros vamos a utilizar la compensación con mayor precisión que está disponible aquí.

Es un carro de expresión que **no debemos de intentar "comprender"**. Es una expresión que el fabricante nos indica aplicar y cuyo origen conoce solo él. Lo aplicamos y listos. Vamos a ello. Primeramente vemos el código completo y seguidamente vamos a ir viendo qué vamos haciendo en cada sección:

```
1  #include <Wire.h>
2
3  #define BME280_ADDRESS 0x77
4  #define BME280_REG_CTRL_MEAS 0xF4
5  #define BME280_REG_TEMP 0xFA // Direccion del registro con la medida
6  #define BME280_REG_DIG_T 0x88 // Direccion del registro de compensacion
7
8  // En la funcion setup, nada nuevo que ver
```

```
9  void setup() {
10   Serial.begin(115200);
11
12   Wire.begin();
13
14   Wire.beginTransmission(BME280_ADDRESS);
15
16   Wire.write(BME280_REG_CTRL_MEAS);
17   Wire.write(0b10100011);
18
19   Wire.endTransmission();
20 }
21
22 void loop() {
23
24     // Leemos el valor de la medicion de la temperatura
25   Wire.beginTransmission(BME280_ADDRESS);
26
27   Wire.write(BME280_REG_TEMP);
28
29   Wire.endTransmission();
30
31   Wire.requestFrom(BME280_ADDRESS, 3);
32
33   while (Wire.available() < 3) {};
34
35   int32_t adc_T = ((int32_t) Wire.read()) << 12;
36   adc_T = adc_T | (((int32_t) Wire.read()) << 4);
37   adc_T = adc_T | (((int32_t) Wire.read()) >> 4);
38
39     // Leemos el valor de los registros de calibracion de la
       temperatura
40   Wire.beginTransmission(BME280_ADDRESS);
41
42   Wire.write(BME280_REG_DIG_T);
43
44   Wire.endTransmission();
45
46   Wire.requestFrom(BME280_ADDRESS, 6);
47
48   while (Wire.available() < 6) {};
49
50   uint16_t dig_T1 = ((uint16_t) Wire.read());
51   dig_T1 = dig_T1 | (((uint16_t) Wire.read()) << 8);
52
53   int16_t dig_T2 = ((int16_t) Wire.read());
54   dig_T2 = dig_T2 | (((int16_t) Wire.read()) << 8);
55
56   int16_t dig_T3 = ((int16_t) Wire.read());
57   dig_T3 = dig_T3 | (((int16_t) Wire.read()) << 8);
58
```

```
59      // Calulamos el coeficiente de calibracion `t_fine`
60    double var1 = (((double) adc_T) / 16384.0 - ((double) dig_T1) /
          1024.0) * ((double) dig_T2);
61    double var2 = ((((double) adc_T) / 131072.0 - ((double) dig_T1) /
          8192.0) *
62      (((double) adc_T) / 131072.0 - ((double) dig_T1) / 8192.0)) * ((
          double) dig_T3);
63
64    int32_t t_fine = (int32_t)(var1 + var2);
65
66      // Calculamos la temperatura
67    double temperature = t_fine / 5120.0;
68
69      // La enviamos al ordenador
70    Serial.print("T: ");
71    Serial.println(temperature);
72
73    delay(500);
74  }
```

1.2.2.2.1. Obtención de la temperatura Ahora vamos a ver el detalle. Centrándonos en la función `loop()` (ya que la función `setup()` es la misma), empezamos pidiendo al sensor la medición de temperatura que se encuentra en los registros `0xFA`, `0xFB` y `0xFC`. El sensor realiza una medición de 20 bits sin signo (solo números enteros positivos). Esos 20 bit necesitan al menos 3 registros ya que cada registro solo tiene 8 bits. Por eso se necesitan más de uno para almacenarla medida. La medición de la temperatura es recogida en el *datasheet* como ut[19:0], donde ut[19:12] corresponde a los bits del registro `0xFA`, ut[11:4] a los del registro `0xFB`, y u[3:0] a los bits 7, 6 ,5 y 4 del registro `0xFC`. Puesto que los registros de la temperatura son consecutivos podemos leer el primer registro `0xFA` y esperar a recibir 3 bytes.

```
1  ...
2          Wire.beginTransmission(BME280_ADDRESS);
3
4          Wire.write(BME280_REG_TEMP);
5
6          Wire.endTransmission();
7
8          Wire.requestFrom(BME280_ADDRESS, 3);
9
10         while (Wire.available() < 3) {};
11 ...
```

Desplazamiento y unión de bits

Que no os despiste la operación de unir los bits de los diferentes registros para obtener la medición de temperatura. Una vez recibidos los 3 bytes, vamos a proceder a leer el primero y lo guardamos en una

variable de 32 bits llamada `adc_T`. Esta variable contendrá la medición de la temperatura resultado de unir los bits de los 3 registros.

¿Qué significa ese (int32_t) y ese « 12? Como sabemos, el primer registro `0xFA` corresponde a los bits ut[19:11] de la medida de temperatura. Por ello, primero convertimos el byte recibido a un valor de 32 bits con `(int32_t)` y después desplazamos los bits hacia la izquierda 12 posiciones con « 12. Si, por ejemplo, el valor leído en `0xFA` fuera `0b11111111`, primeramente convertiríamos ese valor un valor de 32 bits y luego los desplazaríamos a la izquierda 12 posiciones:

```
...
    // 0b11111111
    uint8_t valor0xFA = Wire.read();

    // 0b00000000000000000000000011111111
    int32_t valor0xFA_32b = (int32_t) valor0xFA;

    // 0b00000000000011111111000000000000
    int32_t valor0xFA_32b_shifted = valor0xFA_32b << 12;
...
```

Todo eso es lo que hemos hecho en una sola línea con:

```
...
    int32_t adc_T = ((int32_t) Wire.read()) << 12;
...
```

En la siguientes instrucciones, incorporamos los otros 2 bytes a la medida:

```
...
    adc_T = adc_T | (((int32_t) Wire.read()) << 4);
    adc_T = adc_T | (((int32_t) Wire.read()) >> 4);
...
```

En cada instrucción hacemos lo comentado, convertir el valor recibido a un valor de 32 bits y lo desplazamos el número de posiciones que corresponde a cada registro y lo incorporamos al valor de temperatura sumándoselo (la operación | es la suma binaria bit a bit).

1.2.2.2.2. Obtención de los coeficientes de compensación Ahora toca hacer los mismo, pero con los coeficientes grabados en la memoria del dispositivo para aplicar la fórmula de compensación. Primero solicitamos los coeficientes. Puesto que son consecutivos, podemos pedirlos del tirón esperando 6 bytes.

```
...
    Wire.beginTransmission(BME280_ADDRESS);

    Wire.write(BME280_REG_DIG_T);
```

```
5
6       Wire.endTransmission();
7
8       Wire.requestFrom(BME280_ADDRESS, 6);
9
10      while (Wire.available() < 6) {};
11  ...
```

Luego, hacemos lo mismo que con la temperatura y unimos los bits para obtener los coeficientes:

```
1   ...
2       uint16_t dig_T1 = ((uint16_t) Wire.read());
3       dig_T1 = dig_T1 | (((uint16_t) Wire.read()) << 8);
4
5       int16_t dig_T2 = ((int16_t) Wire.read());
6       dig_T2 = dig_T2 | (((int16_t) Wire.read()) << 8);
7
8       int16_t dig_T3 = ((int16_t) Wire.read());
9       dig_T3 = dig_T3 | (((int16_t) Wire.read()) << 8);
10  ...
```

1.2.2.2.3. Cálculo del coeficiente de compensación `t_fine` A partir de la medida de la temperatura y los coeficientes de compensación, el *datasheet* propone calcular una variable intermedia llamada `t_fine`. Este nuevo coeficiente se utiliza para calcular la presión y la humedad puesto que ambas medidas dependen también de la temperatura.

```
1   ...
2       double var1 = (((double) adc_T) / 16384.0 - ((double) dig_T1) /
            1024.0) * ((double) dig_T2);
3       double var2 = ((((double) adc_T) / 131072.0 - ((double) dig_T1) /
            8192.0) *
4         (((double) adc_T) / 131072.0 - ((double) dig_T1) / 8192.0)) * ((
              double) dig_T3);
5
6       int32_t t_fine = (int32_t)(var1 + var2);
7   ...
```

1.2.2.2.4. Cálculo de la temperatura compensada Ahora sí. Aplicamos la fórmula de compensación que nos indica el *datasheet* y que, repito, no hace falta entender/comprender de dónde viene, nos la da directamente el fabricante.

```
1   ...
2       double temperature = t_fine / 5120.0;
3   ...
```

Una vez calculada, la enviamos al ordenador. También añadimos un *delay* de 500 ms para poder reducir

la velocidad con la que van saliendo nuevas medidas por el terminal y así poderlos leer cómodamente.

```
...
    Serial.print("T: ");
    Serial.println(temperature);

    delay(500);
}
```

Ahora mismo debéis de tener mostrando la temperatura del ambiente por el terminal serie, pero sobretodo aseguraros de entender qué hemos ido haciendo en el código puesto que son operaciones básicas de manipulación de bytes/bits muy comunes en la programación de microcontroladores y que necesitaréis durante el resto de la práctica y el proyecto final.

1.2.3. Lectura de la humedad

Ahora vamos con la humedad. Mismo procedimiento que con la temperatura, pero son sus respectivos registros y fórmulas de compensación (acordaros de activar la medida con un *oversampling* x16). La lectura de la humedad *se* **incorpora** a la de temperatura, por lo que el código contiene ambas medidas. Aquí no se irá tan al detalle con la explicación. Este es el código.

```
#include <Wire.h>

#define BME280_ADDRESS 0x77
#define BME280_REG_CTRL_MEAS 0xF4

// Anadimos el registro para habilitar la humedad
#define BME280_REG_CTRL_HUM 0xF2

#define BME280_REG_TEMP 0xFA
#define BME280_REG_DIG_T 0x88
#define BME280_REG_HUM 0xFD

// Los registros de los coeficientes de compensacion de la humedad no
    son
// consecutivos, estan desperdigados
// Por eso necesitamos leer en diferentes puntos/registros
#define BME280_REG_DIG_H1 0xA1
#define BME280_REG_DIG_H2_H3_H4 0xE1
#define BME280_REG_DIG_H5 0xE5

void setup() {
  Serial.begin(115200);

  Wire.begin();

```

```
25      // Habilitamos la medicion de humedad (oversampling x16)
26      // El datasheet nos dice que debemos de configurar este registro
27    // antes de configurar el registro BME280_REG_CTRL_MEAS
28    Wire.beginTransmission(BME280_ADDRESS);
29
30      Wire.write(BME280_REG_CTRL_HUM);
31    Wire.write(0b00000101);
32
33    Wire.endTransmission();
34
35    Wire.beginTransmission(BME280_ADDRESS);
36
37    Wire.write(BME280_REG_CTRL_MEAS);
38    Wire.write(0b10100011);
39
40    Wire.endTransmission();
41  }
42
43  void loop() {
44
45    Wire.beginTransmission(BME280_ADDRESS);
46
47    Wire.write(BME280_REG_TEMP);
48
49    Wire.endTransmission();
50
51    Wire.requestFrom(BME280_ADDRESS, 3);
52
53    while (Wire.available() < 3) {};
54
55    int32_t adc_T = ((int32_t) Wire.read()) << 12;
56    adc_T = adc_T | (((int32_t) Wire.read()) << 4);
57    adc_T = adc_T | (((int32_t) Wire.read()) >> 4);
58
59    Wire.beginTransmission(BME280_ADDRESS);
60
61    Wire.write(BME280_REG_DIG_T);
62
63    Wire.endTransmission();
64
65    Wire.requestFrom(BME280_ADDRESS, 6);
66
67    while (Wire.available() < 6) {};
68
69    uint16_t dig_T1 = ((uint16_t) Wire.read());
70    dig_T1 = dig_T1 | (((uint16_t) Wire.read()) << 8);
71
72    int16_t dig_T2 = ((int16_t) Wire.read());
73    dig_T2 = dig_T2 | (((int16_t) Wire.read()) << 8);
74
75    int16_t dig_T3 = ((int16_t) Wire.read());
```

```
76    dig_T3 = dig_T3 | (((int16_t) Wire.read()) << 8);
77
78    double var1 = (((double) adc_T) / 16384.0 - ((double) dig_T1) /
          1024.0) * ((double) dig_T2);
79    double var2 = ((((double) adc_T) / 131072.0 - ((double) dig_T1) /
          8192.0) *
80      (((double) adc_T) / 131072.0 - ((double) dig_T1) / 8192.0)) * ((
            double) dig_T3);
81
82    int32_t t_fine = (int32_t)(var1 + var2);
83
84    double temperature = t_fine / 5120.0;
85
86    // Empezamos a medir humedad
87    Wire.beginTransmission(BME280_ADDRESS);
88
89    Wire.write(BME280_REG_HUM);
90
91    Wire.endTransmission();
92
93    // En este caso, la medicion de humedad es de 16 bit y solo
94    // requiere 2 registros
95    Wire.requestFrom(BME280_ADDRESS, 2);
96
97    while (Wire.available() < 2) {};
98
99    int32_t adc_H = ((int32_t) Wire.read()) << 8;
100   adc_H = adc_H | ((int32_t) Wire.read());
101
102   Wire.beginTransmission(BME280_ADDRESS);
103
104   Wire.write(BME280_REG_DIG_H1);
105
106   Wire.endTransmission();
107
108   Wire.requestFrom(BME280_ADDRESS, 1);
109
110   while (Wire.available() < 1) {};
111
112   uint8_t dig_H1 = Wire.read();
113
114   Wire.beginTransmission(BME280_ADDRESS);
115
116   Wire.write(BME280_REG_DIG_H2_H3_H4);
117
118   Wire.endTransmission();
119
120   Wire.requestFrom(BME280_ADDRESS, 5);
121
122   while (Wire.available() < 5) {};
123
```

```
124   int16_t dig_H2 = (int16_t) Wire.read();
125   dig_H2 = dig_H2 | (((int16_t) Wire.read()) << 8);
126
127   uint8_t dig_H3 = Wire.read();
128
129   int16_t dig_H4 = ((int16_t) Wire.read()) << 4;
130
131   // Del siguiente registro solo queremos los ultimos 4 bits
132   dig_H4 = dig_H4 | (((int16_t) Wire.read()) & 0x000F);
133
134   Wire.beginTransmission(BME280_ADDRESS);
135
136   Wire.write(BME280_REG_DIG_H5);
137
138   Wire.endTransmission();
139
140   Wire.requestFrom(BME280_ADDRESS, 3);
141
142   while (Wire.available() < 3) {};
143
144   // Del siguiente registro no queremos los ultimos 4 bits
145   int16_t dig_H5 = (((int16_t) Wire.read()) & 0xFFF0) >> 4;
146   dig_H5 = dig_H5 | (((int16_t) Wire.read()) << 4);
147
148   int8_t dig_H6 = (int8_t) Wire.read();
149
150   // Aplicamos la formula de compensacion
151   double humidity = (((double) t_fine) - 76800.0);
152   humidity = (adc_H - (((double) dig_H4) * 64.0 + ((double) dig_H5) /
          16384.0 * humidity)) * (((double) dig_H2) / 65536.0 * (1.0 + ((
          double) dig_H6) / 67108864.0 * humidity * (1.0 + ((double) dig_H3)
           / 67108864.0 * humidity)));
153   humidity = humidity * (1.0 - ((double) dig_H1) * humidity / 524288.0)
          ;
154
155   if (humidity > 100.0) {
156     humidity = 100.0;
157   } else if (humidity < 0.0) {
158     humidity = 0.0;
159   }
160
161   Serial.print("T: ");
162   Serial.print(temperature);
163   Serial.print(", H: ");
164   Serial.println(humidity);
165
166   delay(500);
167 }
```

¿Algo a destacar respecto a la medición de temperatura? Pues que los coeficientes de compensación están desperdigados por la memoria del dispositivo y hay que hacer varias operaciones de lectura en

lugar de leer todo del tirón, y que algún registro se comparte entre dos coeficientes de compensación (por lo que es necesario utilizar el producto bit a bit &).

Medición de la humedad, *achieved*.

1.2.4. Refactorización de código

Me imagino que ya os debe pasar a estas alturas de la película que os duelan los ojos al ver tantas líneas de código.... Brfff... Vamos a hacer una refactorización y vamos a crear funciones para cada funcionalidad del código para poder mejorar su legibilidad.

Vamos a empezar por algo sencillo: dos funciones, una para escribir en los registros del BME280 y otra para leer. Las llamaremos: `BME280_Write()` y `BME280_Read()`.

```
void BME280_Write(uint8_t address, uint8_t regAddress, uint8_t * buffer
    ,
    uint8_t bytesToWrite) {

  Wire.beginTransmission(address);
    Wire.write(regAddress);

  for (uint8_t i = 0; i < bytesToWrite; i++) {
    Wire.write(buffer[i]);
  }

  Wire.endTransmission();
}
```

Nada raro. Una función a la que le pasamos la dirección del dispositivo, el registro en el que queremos escribir, un buffer o array con los bytes a escribir, y un número que indica cuántos bytes se van enviar. El código es sencillo. ¿Hace falta comentarlo?

```
void BME280_Read(uint8_t address, uint8_t regAddress, uint8_t * buffer,
    uint8_t bytesToRead) {

  Wire.beginTransmission(address);
    Wire.write(regAddress);
    Wire.endTransmission();

    Wire.requestFrom(address, bytesToRead);

    while (Wire.available() < bytesToRead) {};

  for (uint8_t i = 0; i < bytesToRead; i++) {
    buffer[i] = Wire.read();
  }
}
```

Más de lo mismo. Una función a la que le pasamos la dirección del dispositivo, el registro que queremos leer, un buffer o array donde almacenar los bytes a leer, y un número que indica cuántos bytes se van a leer. Solo con estas dos funciones, vamos a ver como queda el código:

```
#include <Wire.h>

#define BME280_ADDRESS 0x77
#define BME280_REG_CTRL_HUM 0xF2
#define BME280_REG_CTRL_MEAS 0xF4
#define BME280_REG_TEMP 0xFA
#define BME280_REG_DIG_T 0x88
#define BME280_REG_HUM 0xFD
#define BME280_REG_DIG_H1 0xA1
#define BME280_REG_DIG_H2_H3_H4 0xE1
#define BME280_REG_DIG_H5 0xE5

uint8_t TxBuffer[32] = {0}, RxBuffer[32] = {0};

void BME280_Write(uint8_t address, uint8_t regAddress, uint8_t * buffer
    ,
  uint8_t bytesToWrite) {

  Wire.beginTransmission(address);
  Wire.write(regAddress);

  for (uint8_t i = 0; i < bytesToWrite; i++) {
    Wire.write(buffer[i]);
  }

  Wire.endTransmission();
}

void BME280_Read(uint8_t address, uint8_t regAddress, uint8_t * buffer,
  uint8_t bytesToRead) {

  Wire.beginTransmission(address);
  Wire.write(regAddress);
  Wire.endTransmission();

  Wire.requestFrom(address, bytesToRead);

  while (Wire.available() < bytesToRead) {};

  for (uint8_t i = 0; i < bytesToRead; i++) {
    buffer[i] = Wire.read();
  }
}

void setup() {
  Serial.begin(115200);
```

```
46
47    Wire.begin();
48
49      TxBuffer[0] = 0b00000101;
50    BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_HUM, TxBuffer, 1);
51
52    TxBuffer[0] = 0b10100011;
53    BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_MEAS, TxBuffer, 1);
54
55  }
56
57  void loop() {
58
59    BME280_Read(BME280_ADDRESS, BME280_REG_TEMP, RxBuffer, 3);
60
61    int32_t adc_T = ((int32_t) RxBuffer[0]) << 12;
62    adc_T = adc_T | (((int32_t) RxBuffer[1]) << 4);
63    adc_T = adc_T | (((int32_t) RxBuffer[2]) >> 4);
64
65    BME280_Read(BME280_ADDRESS, BME280_REG_DIG_T, RxBuffer, 6);
66
67    uint16_t dig_T1 = ((uint16_t) RxBuffer[0]);
68    dig_T1 = dig_T1 | (((uint16_t) RxBuffer[1]) << 8);
69
70    int16_t dig_T2 = ((int16_t) RxBuffer[2]);
71    dig_T2 = dig_T2 | (((int16_t) RxBuffer[3]) << 8);
72
73    int16_t dig_T3 = ((int16_t) RxBuffer[4]);
74    dig_T3 = dig_T3 | (((int16_t) RxBuffer[5]) << 8);
75
76    double var1 = (((double) adc_T) / 16384.0 - ((double) dig_T1) /
         1024.0) * ((double) dig_T2);
77    double var2 = ((((double) adc_T) / 131072.0 - ((double) dig_T1) /
         8192.0) *
78      (((double) adc_T) / 131072.0 - ((double) dig_T1) / 8192.0)) * ((
           double) dig_T3);
79
80    int32_t t_fine = (int32_t)(var1 + var2);
81
82    double temperature = t_fine / 5120.0;
83
84    BME280_Read(BME280_ADDRESS, BME280_REG_HUM, RxBuffer, 2);
85
86    int32_t adc_H = ((int32_t) RxBuffer[0]) << 8;
87    adc_H = adc_H | ((int32_t) RxBuffer[1]);
88
89    BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H1, RxBuffer, 1);
90
91    uint8_t dig_H1 = RxBuffer[0];
92
93    BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H2_H3_H4, RxBuffer, 5);
```

```
94
95    int16_t dig_H2 = (int16_t) RxBuffer[0];
96    dig_H2 = dig_H2 | (((int16_t) RxBuffer[1]) << 8);
97
98    uint8_t dig_H3 = RxBuffer[2];
99
100   int16_t dig_H4 = ((int16_t) RxBuffer[3]) << 4;
101
102   dig_H4 = dig_H4 | (((int16_t) RxBuffer[4]) & 0x000F);
103
104   BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H5, RxBuffer, 3);
105
106   int16_t dig_H5 = (((int16_t) RxBuffer[0]) & 0xFFF0) >> 4;
107   dig_H5 = dig_H5 | (((int16_t) RxBuffer[1]) << 4);
108
109   int8_t dig_H6 = (int8_t) RxBuffer[2];
110
111   double humidity = (((double) t_fine) - 76800.0);
112   humidity = (adc_H - (((double) dig_H4) * 64.0 + ((double) dig_H5) /
          16384.0 * humidity)) * (((double) dig_H2) / 65536.0 * (1.0 + ((
          double) dig_H6) / 67108864.0 * humidity * (1.0 + ((double) dig_H3)
           / 67108864.0 * humidity)));
113   humidity = humidity * (1.0 - ((double) dig_H1) * humidity / 524288.0)
          ;
114
115   if (humidity > 100.0) {
116     humidity = 100.0;
117   } else if (humidity < 0.0) {
118     humidity = 0.0;
119   }
120
121   Serial.print("T: ");
122   Serial.print(temperature);
123   Serial.print(", H: ");
124   Serial.println(humidity);
125
126   delay(500);
127 }
```

Mejor, ¿no?… ¿¡No!? Espero estar oyendo un “sí” a todo volumen… Ahora vamos a ir un poco más allá. Los coeficientes de compensación son siempre los mismos. Son valores constantes. No hace falta leerlos todo el rato. Por ello, los leeremos solo en la función `setup()` al principio de la ejecución del programa. Parte del código quedaría así:

```
1  ...
2  uint16_t dig_T1 = 0;
3  int16_t dig_T2 = 0, dig_T3 = 0, dig_H2 = 0, dig_H4 = 0, dig_H5 = 0;
4  uint8_t dig_H1 = 0, dig_H3 = 0;
5  int8_t dig_H6 = 0;
6
```

```
...

void setup() {
  Serial.begin(115200);

  Wire.begin();

    TxBuffer[0] = 0b00000101;
  BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_HUM, TxBuffer, 1);

  TxBuffer[0] = 0b10100011;
  BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_MEAS, TxBuffer, 1);

  BME280_Read(BME280_ADDRESS, BME280_REG_DIG_T, RxBuffer, 6);

  dig_T1 = ((uint16_t) RxBuffer[0]);
  dig_T1 = dig_T1 | (((uint16_t) RxBuffer[1]) << 8);

  dig_T2 = ((int16_t) RxBuffer[2]);
  dig_T2 = dig_T2 | (((int16_t) RxBuffer[3]) << 8);

  dig_T3 = ((int16_t) RxBuffer[4]);
  dig_T3 = dig_T3 | (((int16_t) RxBuffer[5]) << 8);

  BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H1, RxBuffer, 1);

  dig_H1 = RxBuffer[0];

  BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H2_H3_H4, RxBuffer, 5);

  dig_H2 = (int16_t) RxBuffer[0];
  dig_H2 = dig_H2 | (((int16_t) RxBuffer[1]) << 8);

  dig_H3 = RxBuffer[2];

  dig_H4 = ((int16_t) RxBuffer[3]) << 4;

  dig_H4 = dig_H4 | (((int16_t) RxBuffer[4]) & 0x000F);

  BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H5, RxBuffer, 3);

  dig_H5 = (((int16_t) RxBuffer[0]) & 0xFFF0) >> 4;
  dig_H5 = dig_H5 | (((int16_t) RxBuffer[1]) << 4);

  dig_H6 = (int8_t) RxBuffer[2];

}
...
```

No os siguen doliendo los ojos? Vamos a modular el código en funciones.

```
...
uint16_t dig_T1 = 0;
int16_t dig_T2 = 0, dig_T3 = 0, dig_H2 = 0, dig_H4 = 0, dig_H5 = 0;
uint8_t dig_H1 = 0, dig_H3 = 0;
int8_t dig_H6 = 0;

...

void BME280_GetDigT(void) {
  BME280_Read(BME280_ADDRESS, BME280_REG_DIG_T, RxBuffer, 6);

  dig_T1 = ((uint16_t) RxBuffer[0]);
  dig_T1 = dig_T1 | (((uint16_t) RxBuffer[1]) << 8);

  dig_T2 = ((int16_t) RxBuffer[2]);
  dig_T2 = dig_T2 | (((int16_t) RxBuffer[3]) << 8);

  dig_T3 = ((int16_t) RxBuffer[4]);
  dig_T3 = dig_T3 | (((int16_t) RxBuffer[5]) << 8);
}

void BME280_GetDigH(void) {
  BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H1, RxBuffer, 1);

  dig_H1 = RxBuffer[0];

  BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H2_H3_H4, RxBuffer, 5);

  dig_H2 = (int16_t) RxBuffer[0];
  dig_H2 = dig_H2 | (((int16_t) RxBuffer[1]) << 8);

  dig_H3 = RxBuffer[2];

  dig_H4 = ((int16_t) RxBuffer[3]) << 4;

  dig_H4 = dig_H4 | (((int16_t) RxBuffer[4]) & 0x000F);

  BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H5, RxBuffer, 3);

  dig_H5 = (((int16_t) RxBuffer[0]) & 0xFFF0) >> 4;
  dig_H5 = dig_H5 | (((int16_t) RxBuffer[1]) << 4);

  dig_H6 = (int8_t) RxBuffer[2];
}

void setup() {
  Serial.begin(115200);

  Wire.begin();

    TxBuffer[0] = 0b00000101;
```

```
52    BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_HUM, TxBuffer, 1);
53
54    TxBuffer[0] = 0b10100011;
55    BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_MEAS, TxBuffer, 1);
56
57    BME280_GetDigT();
58    BME280_GetDigH();
59  }
```

Comprobemos que todo funciona. Para hacerlo, comprobad que actualmente estas son vuestras expresiones:

wow.webp

Si no es el caso, volver al principio, anda…

Ahora ya vamos a por todas. Vamos a añadir en funciones la lectura de las medidas y los cálculos de optimización.

```
1  ...
2  int32_t BME280_GetTemperature(void) {
3    BME280_Read(BME280_ADDRESS, BME280_REG_TEMP, RxBuffer, 3);
4
5    int32_t adc_T = ((int32_t) RxBuffer[0]) << 12;
6    adc_T = adc_T | (((int32_t) RxBuffer[1]) << 4);
7    adc_T = adc_T | (((int32_t) RxBuffer[2]) >> 4);
8
9    return adc_T;
10 }
11
12 int32_t BME280_CalculateTFine(int32_t adc_T) {
13   double var1 = (((double) adc_T) / 16384.0 - ((double) dig_T1) /
         1024.0) * ((double) dig_T2);
14   double var2 = ((((double) adc_T) / 131072.0 - ((double) dig_T1) /
         8192.0) *
15     (((double) adc_T) / 131072.0 - ((double) dig_T1) / 8192.0)) * ((
         double) dig_T3);
16
17   return (int32_t)(var1 + var2);
18 }
19
20 double BME280_CalculateTemperature(int32_t t_fine, int32_t adc_T) {
21   double var1 = (((double) adc_T) / 16384.0 - ((double) dig_T1) /
         1024.0) * ((double) dig_T2);
22   double var2 = ((((double) adc_T) / 131072.0 - ((double) dig_T1) /
         8192.0) *
23     (((double) adc_T) / 131072.0 - ((double) dig_T1) / 8192.0)) * ((
         double) dig_T3);
24
25   return t_fine / 5120.0;
26 }
```

```
27
28 int32_t BME280_GetHumidity(void) {
29   BME280_Read(BME280_ADDRESS, BME280_REG_HUM, RxBuffer, 2);
30
31   int32_t adc_H = ((int32_t) RxBuffer[0]) << 8;
32   adc_H = adc_H | ((int32_t) RxBuffer[1]);
33
34   return adc_H;
35 }
36
37 double BME280_CalculateHumidity(int32_t t_fine, int32_t adc_H) {
38   double humidity = (((double) t_fine) - 76800.0);
39   humidity = (adc_H - (((double) dig_H4) * 64.0 + ((double) dig_H5) /
         16384.0 * humidity)) * (((double) dig_H2) / 65536.0 * (1.0 + ((
         double) dig_H6) / 67108864.0 * humidity * (1.0 + ((double) dig_H3)
          / 67108864.0 * humidity)));
40   humidity = humidity * (1.0 - ((double) dig_H1) * humidity / 524288.0)
         ;
41
42   if (humidity > 100.0) {
43     humidity = 100.0;
44   } else if (humidity < 0.0) {
45     humidity = 0.0;
46   }
47   return humidity;
48 }
49
50 void setup() {
51   Serial.begin(115200);
52
53   Wire.begin();
54
55     TxBuffer[0] = 0b00000101;
56   BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_HUM, TxBuffer, 1);
57
58   TxBuffer[0] = 0b10100011;
59   BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_MEAS, TxBuffer, 1);
60
61   BME280_GetDigT();
62   BME280_GetDigH();
63
64 }
65
66 void loop() {
67
68   int32_t adc_T = BME280_GetTemperature();
69   int32_t t_fine = BME280_CalculateTFine(adc_T);
70   double temperature = BME280_CalculateTemperature(t_fine, adc_T);
71
72   int32_t adc_H = BME280_GetHumidity();
73   double humidity = BME280_CalculateHumidity(t_fine, adc_H);
```

```
74
75    Serial.print("T: ");
76    Serial.print(temperature);
77    Serial.print(", H: ");
78    Serial.println(humidity);
79
80    delay(500);
81  }
```

¿¡Qué locura es esta!? ¿¡¡Pero habéis visto que entendible es ahora lo que hacen las funciones `setup()` y `loop()` ahora!!? Esto sí que es código legible. Además, no se si os habéis dado cuenta, pero lo que habéis hecho y lo que tenéis delante se llama

Figura 2: Bob Esponja.

Obviamente, muy rudimentaria y mejorable, pero una librería al fin y al cabo. ¡Enhorabuena!

Aquí os dejo el código completo.

```
1  #include <Wire.h>
2
3  #define BME280_ADDRESS 0x77
4  #define BME280_REG_CTRL_MEAS 0xF4
5  #define BME280_REG_CTRL_HUM 0xF2
6  #define BME280_REG_TEMP 0xFA
7  #define BME280_REG_DIG_T 0x88
8  #define BME280_REG_HUM 0xFD
9  #define BME280_REG_DIG_H1 0xA1
10 #define BME280_REG_DIG_H2_H3_H4 0xE1
11 #define BME280_REG_DIG_H5 0xE5
12
13 uint8_t TxBuffer[32] = {0}, RxBuffer[32] = {0};
14 uint16_t dig_T1 = 0;
15 int16_t dig_T2 = 0, dig_T3 = 0, dig_H2 = 0, dig_H4 = 0, dig_H5 = 0;
16 uint8_t dig_H1 = 0, dig_H3 = 0;
```

```
17 int8_t dig_H6 = 0;
18
19 void BME280_Write(uint8_t address, uint8_t regAddress, uint8_t * buffer
       ,
20   uint8_t bytesToWrite) {
21
22   Wire.beginTransmission(address);
23   Wire.write(regAddress);
24
25   for (uint8_t i = 0; i < bytesToWrite; i++) {
26     Wire.write(buffer[i]);
27   }
28
29   Wire.endTransmission();
30 }
31
32 void BME280_Read(uint8_t address, uint8_t regAddress, uint8_t * buffer,
33   uint8_t bytesToRead) {
34
35   Wire.beginTransmission(address);
36   Wire.write(regAddress);
37   Wire.endTransmission();
38
39   Wire.requestFrom(address, bytesToRead);
40
41   while (Wire.available() < bytesToRead) {};
42
43   for (uint8_t i = 0; i < bytesToRead; i++) {
44     buffer[i] = Wire.read();
45   }
46 }
47
48 void BME280_GetDigT(void) {
49   BME280_Read(BME280_ADDRESS, BME280_REG_DIG_T, RxBuffer, 6);
50
51   dig_T1 = ((uint16_t) RxBuffer[0]);
52   dig_T1 = dig_T1 | (((uint16_t) RxBuffer[1]) << 8);
53
54   dig_T2 = ((int16_t) RxBuffer[2]);
55   dig_T2 = dig_T2 | (((int16_t) RxBuffer[3]) << 8);
56
57   dig_T3 = ((int16_t) RxBuffer[4]);
58   dig_T3 = dig_T3 | (((int16_t) RxBuffer[5]) << 8);
59 }
60
61 void BME280_GetDigH(void) {
62   BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H1, RxBuffer, 1);
63
64   dig_H1 = RxBuffer[0];
65
66   BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H2_H3_H4, RxBuffer, 5);
```

```
67
68    dig_H2 = (int16_t) RxBuffer[0];
69    dig_H2 = dig_H2 | (((int16_t) RxBuffer[1]) << 8);
70
71    dig_H3 = RxBuffer[2];
72
73    dig_H4 = ((int16_t) RxBuffer[3]) << 4;
74
75    dig_H4 = dig_H4 | (((int16_t) RxBuffer[4]) & 0x000F);
76
77    BME280_Read(BME280_ADDRESS, BME280_REG_DIG_H5, RxBuffer, 3);
78
79    dig_H5 = (((int16_t) RxBuffer[0]) & 0xFFF0) >> 4;
80    dig_H5 = dig_H5 | (((int16_t) RxBuffer[1]) << 4);
81
82    dig_H6 = (int8_t) RxBuffer[2];
83  }
84
85  int32_t BME280_GetTemperature(void) {
86    BME280_Read(BME280_ADDRESS, BME280_REG_TEMP, RxBuffer, 3);
87
88    int32_t adc_T = ((int32_t) RxBuffer[0]) << 12;
89    adc_T = adc_T | (((int32_t) RxBuffer[1]) << 4);
90    adc_T = adc_T | (((int32_t) RxBuffer[2]) >> 4);
91
92    return adc_T;
93  }
94
95  int32_t BME280_CalculateTFine(int32_t adc_T) {
96    double var1 = (((double) adc_T) / 16384.0 - ((double) dig_T1) /
          1024.0) * ((double) dig_T2);
97    double var2 = ((((double) adc_T) / 131072.0 - ((double) dig_T1) /
          8192.0) *
98      (((double) adc_T) / 131072.0 - ((double) dig_T1) / 8192.0)) * ((
          double) dig_T3);
99
100   return (int32_t)(var1 + var2);
101 }
102
103 double BME280_CalculateTemperature(int32_t t_fine, int32_t adc_T) {
104   double var1 = (((double) adc_T) / 16384.0 - ((double) dig_T1) /
          1024.0) * ((double) dig_T2);
105   double var2 = ((((double) adc_T) / 131072.0 - ((double) dig_T1) /
          8192.0) *
106     (((double) adc_T) / 131072.0 - ((double) dig_T1) / 8192.0)) * ((
          double) dig_T3);
107
108   return t_fine / 5120.0;
109 }
110
111 int32_t BME280_GetHumidity(void) {
```

```
112   BME280_Read(BME280_ADDRESS, BME280_REG_HUM, RxBuffer, 2);
113
114   int32_t adc_H = ((int32_t) RxBuffer[0]) << 8;
115   adc_H = adc_H | ((int32_t) RxBuffer[1]);
116
117   return adc_H;
118 }
119
120 double BME280_CalculateHumidity(int32_t t_fine, int32_t adc_H) {
121   double humidity = (((double) t_fine) - 76800.0);
122   humidity = (adc_H - (((double) dig_H4) * 64.0 + ((double) dig_H5) /
          16384.0 * humidity)) * (((double) dig_H2) / 65536.0 * (1.0 + ((
          double) dig_H6) / 67108864.0 * humidity * (1.0 + ((double) dig_H3)
           / 67108864.0 * humidity)));
123   humidity = humidity * (1.0 - ((double) dig_H1) * humidity / 524288.0)
          ;
124
125   if (humidity > 100.0) {
126     humidity = 100.0;
127   } else if (humidity < 0.0) {
128     humidity = 0.0;
129   }
130   return humidity;
131 }
132
133 void setup() {
134   Serial.begin(115200);
135
136   Wire.begin();
137
138     TxBuffer[0] = 0b00000101;
139   BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_HUM, TxBuffer, 1);
140
141   TxBuffer[0] = 0b10100011;
142   BME280_Write(BME280_ADDRESS, BME280_REG_CTRL_MEAS, TxBuffer, 1);
143
144   BME280_GetDigT();
145   BME280_GetDigH();
146
147 }
148
149 void loop() {
150
151   int32_t adc_T = BME280_GetTemperature();
152   int32_t t_fine = BME280_CalculateTFine(adc_T);
153   double temperature = BME280_CalculateTemperature(t_fine, adc_T);
154
155   int32_t adc_H = BME280_GetHumidity();
156   double humidity = BME280_CalculateHumidity(t_fine, adc_H);
157
158   Serial.print("T: ");
```

```
159     Serial.print(temperature);
160     Serial.print(", H: ");
161     Serial.println(humidity);
162 
163     delay(500);
164 }
```

1.3. Reto

Tenemos un sensor que mide temperatura, humedad y presión, y hemos medido temperatura y humedad… ¿Te imaginas cuál será el reto? Pues sí, pequeño pony, el reto es incorporar la medida de presión atmosférica al código y ofrecerla por terminal junto a las otras dos medidas. Hazlo siguiendo la misma filosofía de modularidad: crea funciones para hacer cada operación como hemos hecho en la sección de refactorización.

Puedes imaginarte que será un código muy parecido, pero utilizando los registros pertinentes para la medición de presión. ¡Dale duro y a por ello! (Pero "sin presión", eh).

Figura 3: Ba-dum-tss!

También se puede calcular la altitud en función de la presión atmosférica, pero **esto NO os lo pido** y dejo a vuestro parecer implementarlo o no. Aunque es una fórmula muy simplona, que es la siguiente:

```
1 H = 44330 * [1 - (P/p0)^(1/5.255) ]
2 
3 H = altitude (m)
4 P = measured pressure (Pa) from the sensor
5 p0 = reference pressure at sea level (e.g. 1013.25hPa)
```

1.4. Evaluación

1.4.1. Entregables

Estos son los elementos que deberán de estar disponibles para el profesorado de cara a vuestra evaluación.

- ☐ **Commits** Se os deja a vuestro criterio cuándo realizar los *commits*. No hay número mínimo/máximo y se evaluará el uso adecuado del control de versiones (creación de ramas, *commits* en el momento adecuado, mensajes descriptivos de los cambios, …).
- ☐ **Reto**
- ☐ ~~**Informe**~~ No hay informe.

1.4.2. Pull Request

Finalizado el informe, acordaos de hacer el *push* pertinente y cread un *Pull Request* (PR) de vuestra rama a la `master`. Acordaos de ponerme como *Reviewer*. No hagáis *merge*.

1.4.3. Rúbrica

La rúbrica que utilizaremos para la evaluación la podéis encontrar en el CampusVirtual. Os recomendamos que le echéis un vistazo para que sepáis exactamente qué se evaluará y qué se os pide.

1.5. Conclusiones

Pues ya sabemos utilizar I2C en Arduino. Es un protocolo de comunicación muy, **MUY**, utilizado, por lo que saber cómo funciona nos dará la vida. Además, hemos visto cómo operar para leer y escribir en registros, ya que este suele ser el método más utilizado en I2C: leer registros, unir bytes, desplazar bits, etc. Y por último, hemos hecho una buena refactorización del código para mejorar su legibilidad. Si aún no me creéis, enseñadle el código de las funciones `setup()` y `loop()` a alguien que no sepa de qué va el código y seguro que sabe más o menos qué hace.

¡Vamos a por la siguiente parte de esta práctica en la que haremos lo mismo con STM32CubeIDE!