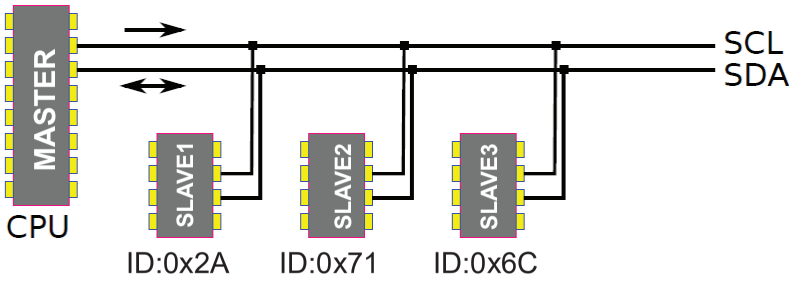
Interfaz de 2 hilos: I2C

Índice

El bus I2C

- I2C = Inter Integrated Circuit
- Desarrollado por Philips a principios de 1980
  - Control de varios chips en televisores de manera sencilla
  - Utilizado por otras empresas para buscar la compatibilidad
- TWI (Two Wires Interfaz) desarrollado por ATMEL
  - Evitar conflictos de licencias
- TWI ≅ I2C

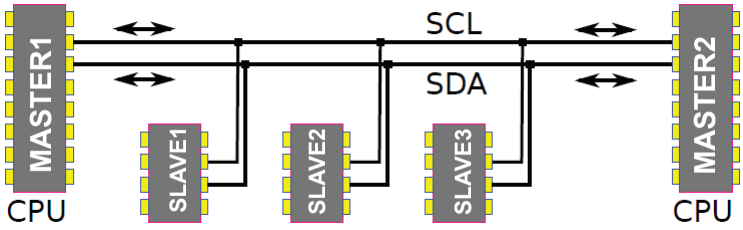
El bus I2C

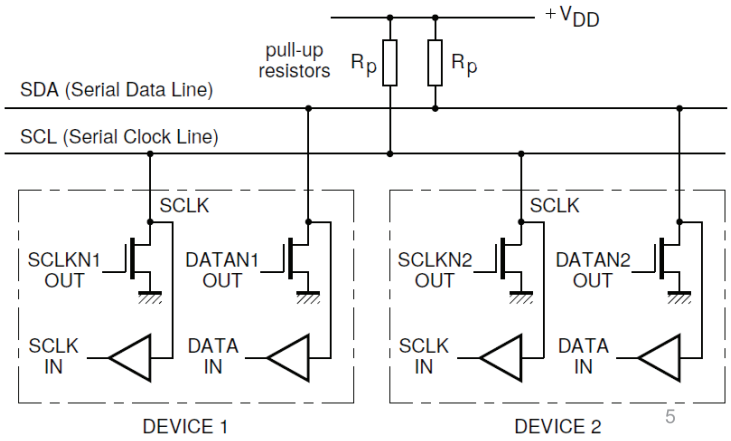


- Solo 2 cables
  - **SDA** (**S**erial **DA**ta)
  - **SCL** (**S**erial **CL**ock)
- Cada dispositivo conectado tiene una dirección única
  - ID de 7 o 10 bits
- En la comunicación existen:
  - **Transmisor**: el que envía datos
  - **Receptor**: el que recibe datos
  - **Maestro**: el que inicia la comunicación y genera SCL
  - **Esclavo**: el que es direccionado

El bus I2C

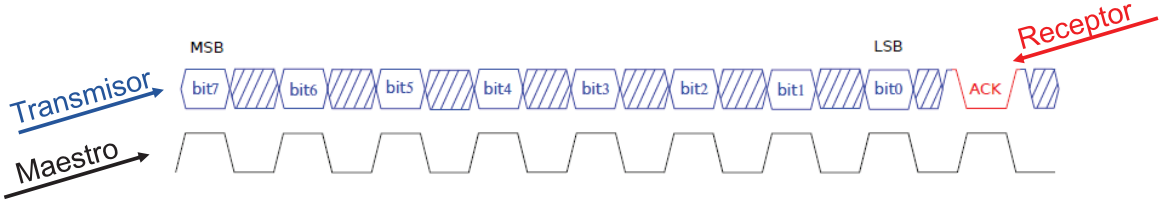
- Un solo maestro o múltiples
- SDA y SCL generadas con etapas en colector abierto
  - Se requieren resistencias de pull-up
- Velocidades típicas de SCL
  - Modo estándar: 100 KHz
  - Modo fast: 400 KHz
  - Hasta 3.4 MHz (no soportado en muchos μC)





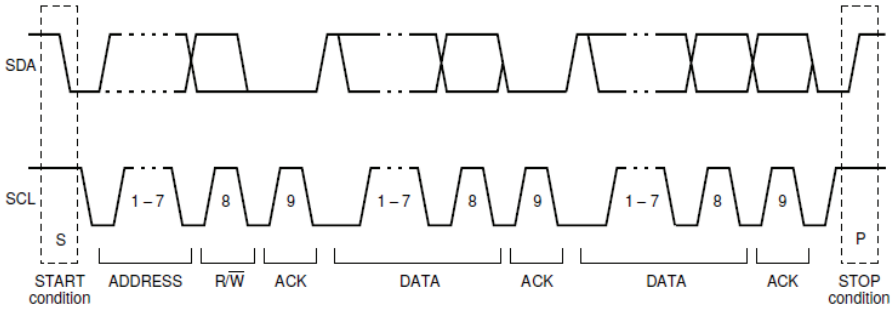
El bus I2C

- Se transmiten bytes, siempre de 8 bits
- El MSB en primer lugar
- Sin restricción en el número de bytes que se pueden enviar
- Cada byte es seguido por un bit de reconocimiento (ACK):
  - El transmisor no fuerza la línea SDA
  - El receptor pone la línea SDA a 0
- Dato estable durante el nivel alto del reloj SCL

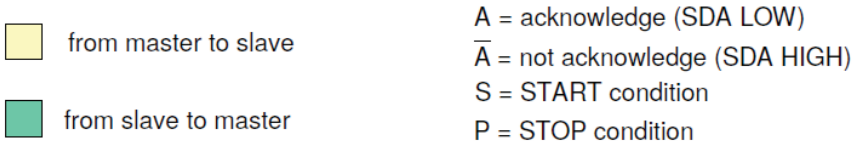


El bus I2C

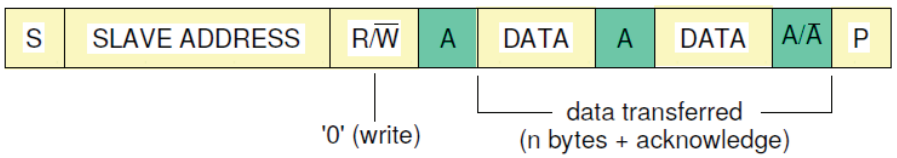
- Transferencia completa:
  - El **master** establece quién es el esclavo y en qué sentido se realiza la transferencia (R=\W). El master genera el SCL y la condición de inicio **S**
  - El **esclavo** escribe el primer ACK
  - El **transmisor** emite el resto de bits, salvo los ACK, que los escribe el **receptor**
  - El **master** genera la condición de parada **P**



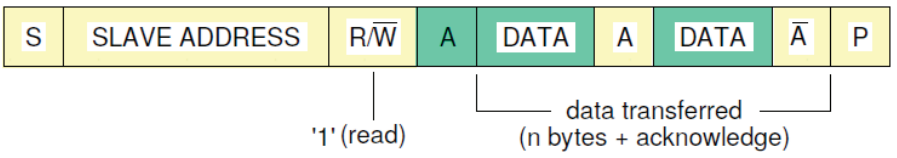
El bus I2C



Escritura en el esclavo

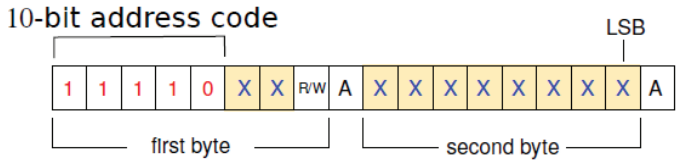


Lectura del esclavo



El bus I2C

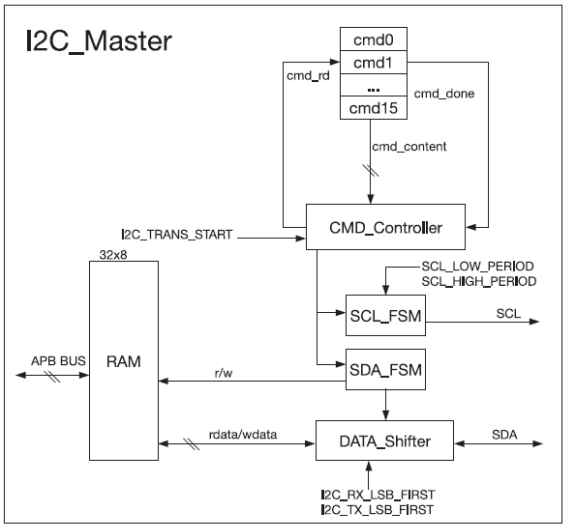
- I2C con direcciones de 10 bits (en lugar de 7):
  - La dirección en el primer byte tras el START es 11110XX
  - El bit R/#W de ese primer byte indica el sentido de la transferencia
  - Los 8 bits del siguiente byte, junto con los bits XX del primero forman una dirección de 10 bits.

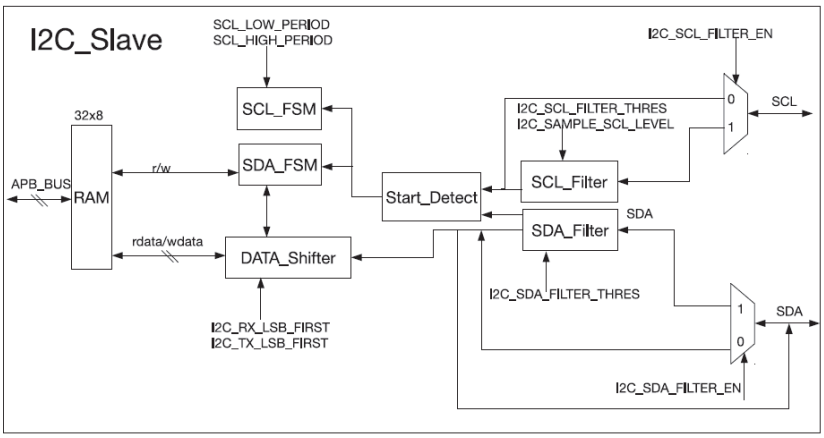


I2C en ESP32

- ESP32 dispone de 2 controladores I2C independientes
- Cada controlador I2C tiene las siguientes características:
  - SDA y SCL se puede conectar a cualquier pin del ESP32
    - Conexión por defecto del M5Stack: SDA pin 21 y SCL al 22
  - configurable como master o slave
  - modo standard (100 kbit/s) y modo fast (400 kbit/s)
  - direccionamiento de 7 y 10 bits
  - Incluyen un filtro digital programable para eliminar glitches

I2C en ESP32





RAM de 32x8 bits, mapeada en la memoria del ESP32

I2C en ESP32

- Configuración de la frecuencia de reloj:

Table 57: SCL Frequency Configuration

I2C_SCL_FILTER_EN	I2C_SCL_FILTER_THRES	SCL_Low_Level_Cycles	SCL_High_Level_Cycles
0	Don't care	I2C_SCL_LOW_PERIOD+1	I2C_SCL_HIGH_PERIOD+7
1	[0,2]		I2C_SCL_HIGH_PERIOD+8
	[3,7]		I2C_SCL_HIGH_PERIOD+6+I2C_SCL_FILTER_THRES

$$f_{scl} = \frac{80\ \text{MHz}}{\text{SCL\_Low\_Level\_Cycles} + \text{SCL\_High\_Level\_Cycles}}$$

*Esp32 technical reference Manual (pag. 287)

Comunicación I2C con Arduino

- Disponible con funciones de la librería "**Wire**":
  - begin() : configura como master (por defecto, sin parámetro de entrada) o slave (hay que pasarle la dirección del dispositivo)
  - begin(SDA,SCL) : configura los pines del puerto I2C
  - beginTransmission(address)
  - write()
  - endTransmission()

  Escritura del master en un slave
  - requestFrom(address, quantity)
  - read()
  - available()

  Lectura de un slave por parte del master
  - setClock(): fija la frecuencia de reloj de SCL en Hz. Por defecto está configurado a 100 KHz

https://www.arduino.cc/en/Reference/Wire

Comunicación I2C con ESP-IDF

- Control completo de los recursos de los controladores I2C
- ESP32 dispone de 2 puertos I2C independientes
  - Nombres: I2C_NUM_0 e I2C_NUM_1
- Funciones de la librería **i2c.h**
  - i2c_param_config(): configuración del puerto I2C.
  - I2c_driver_install(): instalación del driver
  - i2c_cmd_link_create() y i2c_cmd_link_delete(): crear y borrar un manejador para incluir los comandos
  - i2c_master_cmd_begin(): ejecuta la lista de comandos

https://esp-idf.readthedocs.io/en/latest/api-reference/peripherals/i2c.html

Comunicación I2C con ESP-IDF

- Funciones de la librería **i2c.h**
  - i2c_master_start(): commando start.
  - i2c_master_write(), i2c_master_write_byte: escribir un buffer o un byte
  - i2c_master_read() y i2c_master_read_byte (): leer un buffer o un byte
  - i2c_master_stop(): commando stop

  - i2c_isr_register(): manejador para gestión de interrupciones
  - …

https://esp-idf.readthedocs.io/en/latest/api-reference/peripherals/i2c.html

Comunicación I2C con ESP-IDF

1. Configuración del puerto I2C con i2c_param_config():

- Dispone de dos parámetros:
  - Nombre del puerto: I2C_NUM_0 / I2C_NUM_1
  - Puntero a la estructura (i2c_config_t) que contiene los detalles de la configuración:
    - mode (I2C_MODE_MASTER, I2C_MODE_SLAVE)
    - sda_io_num, slc_io_num: pines de SDA e SCL
    - sda_pullup_en, scl_pullup_en: configuración de resistencia de pull-up
    - clk_speed: velocidad del reloj si es master

Ejemplo

```
i2c_config_t conf;
conf.mode = I2C_MODE_MASTER;
conf.sda_io_num = 25;
conf.scl_io_num = 26;
conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
conf.master.clk_speed = 100000;

i2c_param_config(I2C_NUM_0, &conf);
```

Comunicación I2C con ESP-IDF

2. Instalar el driver i2c_driver_install():

- Dispone de los parámetros:
  - I2c_port_t: I2C_NUM_0 ,I2C_NUM_1
  - I2c_mode_t: I2C_MODE_MASTER, I2C_MODE_SLAVE
  - Tamaño de los buffers en los que alojar los datos recibidos y enviados (sólo para el modo SLAVE)
  - Flags para alojar la interrupción (en caso de usarla)

Ejemplos

Modo MASTER:

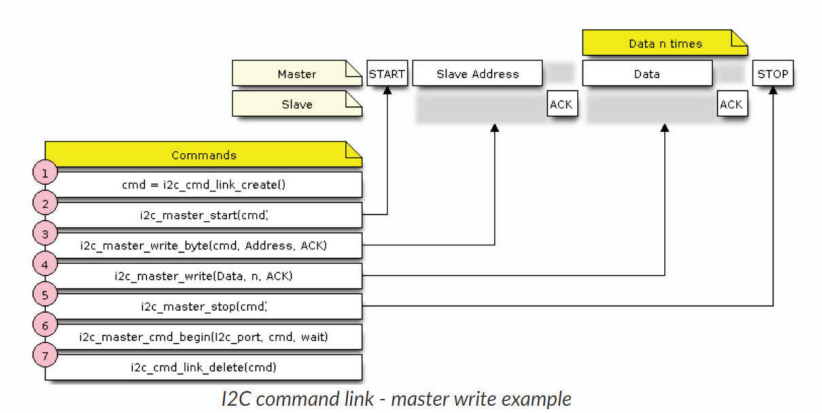
```
i2c_driver_install(I2C_NUM_0,
I2C_MODE_MASTER, 0, 0, 0);
```

Modo SLAVE:

```
i2c_driver_install(
I2C_NUM_0, I2C_MODE_SLAVE, 4, 4,
ESP_INTR_FLAG_IRAM );
```


Comunicación I2C con ESP-IDF

3. Comenzar la comunicación (escritura en modo master)



I2C command link - master write example

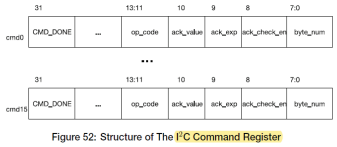


Figure 52: Structure of The I²C Command Register

Los comandos de comunicación se escriben en los registros (cmd0…cmd15) y se transfieren al pasar la función i2C_master_cmd_begin()

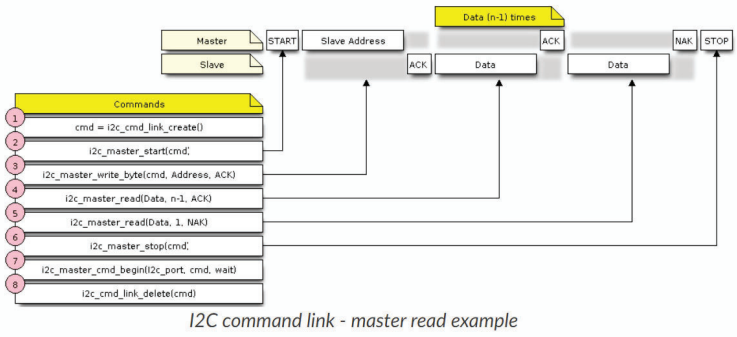
Incluimos la información de si lee o escribe en la palabra Address, concatenando la dirección del esclavo con el bit R/W

Ejemplo (BMP180)

```
i2c_cmd_handle_t cmd =i2c_cmd_link_create();
 i2c_master_start(cmd);
 i2c_master_write_byte(cmd,
(device_addr << 1) | I2C_MASTER_WRITE, true);
 i2c_master_write(cmd, data, n, true);
 i2c_master_stop(cmd);
 i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000 /
portTICK_RATE_MS);
 i2c_cmd_link_delete(cmd);
```

Comunicación I2C con ESP-IDF

3. Comenzar la comunicación (ejemplo de lectura modo Master)



I2C command link - master read example

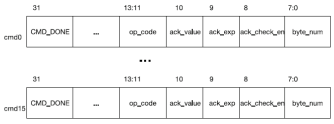


Figure 52: Structure of The I²C Command Register

Los comandos de comunicación se escriben en los registros (cmd0…cmd15) y se transfieren al pasar la función i2C_master_cmd_begin()

Ejemplo (BMP180)

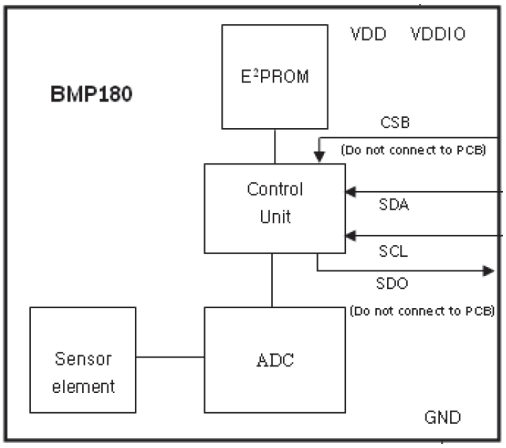
```
i2c_cmd_handle_t cmd =i2c_cmd_link_create();
 i2c_master_start(cmd);
 i2c_master_write_byte(cmd, (device_addr << 1)
| I2C_MASTER_READ, true);
 i2c_master_read(cmd, &data, n-1, true);
 i2c_master_read(cmd, &data+n-1, 1, false);
 i2c_master_stop(cmd);
 i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000
/ portTICK_RATE_MS);
 i2c_cmd_link_delete(cmd);
```


Control de sensores I2C con esp32

- Sensor inteligente
  - Tiene capacidad de realizar alguna/s de las siguientes funciones:
    - pre-procesamiento
    - comunicación digital/protocolo
    - toma de decisiones
    - memorización de configuración
  - Lo forman un dispositivo HW + driver SW
  - Contienen registros direccionables de:
    - Control
    - Status
    - Datos

BMP180
Digital pressure sensor

- Sensor digital de presión y temperatura
- Compuesto por:
  - Sensor piezo-resistivo
  - Memoria E²PROM
  - Conversor ADC
  - Interfaz I2C
- Genera datos no compensados de presión (UP 16-19 bits) y temperatura (UT de 16 bits)
- E²PROM almacena valores de calibración para compensar UP y UT



BMP180
Digital pressure sensor

- Mapa de memoria

Registers:	Control registers	Calibration registers	Data registers	Fixed
Type:	read / write	read only	read only	read only

Register Name	Register Adress	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
out_xlsb	F8h	adc_out_xlsb<7:3>					0	0	0	00h
out_lsb	F7h	adc_out_lsb<7:0>								00h
out_msb	F6h	adc_out_msb<7:0>								80h
ctrl_meas	F4h	oss<1:0>		sco	measurement control					00h
soft reset	E0h	reset								00h
id	D0h	id<7:0>								55h
calib21 downto calib0	BFh *down* to AAh	calib21<7:0> *down* to calib0<7:0>								n/a

Sco (register F4h <5>): Start of conversion. The value of this bit stays "1" during conversion and is reset to "0" after conversion is complete (data registers are filled).

Oss (register F4h <7:6>): controls the oversampling ratio of the pressure measurement (00b: single, 01b: 2 times, 10b: 4 times, 11b: 8 times).

Soft reset (register E0h): Write only register. If set to 0xB6, will perform the same sequence as power on reset.

Chip-id (register D0h): This value is fixed to 0x55 and can be used to check whether communication is functioning.

Control register values

Measurement	Control register value (register address 0xF4)	Max. conversion time [ms]
Temperature	0x2E	4.5
Pressure (oss = 0)	0x34	4.5
Pressure (oss = 1)	0x74	7.5
Pressure (oss = 2)	0xB4	13.5
Pressure (oss = 3)	0xF4	25.5

Calibration coefficients

	BMP180 reg adr	
Parameter	MSB	LSB
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

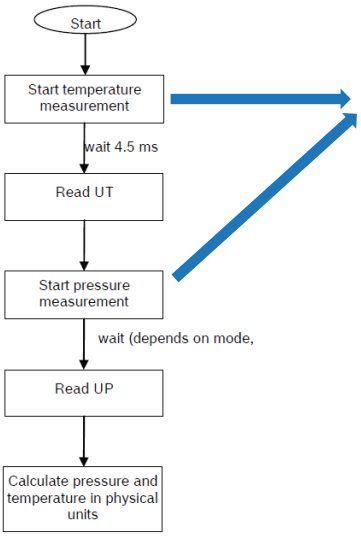
BMP180
Digital pressure sensor

BPM180 module address

A7	A6	A5	A4	A3	A2	A1	W/R
1	1	1	0	1	1	1	0/1

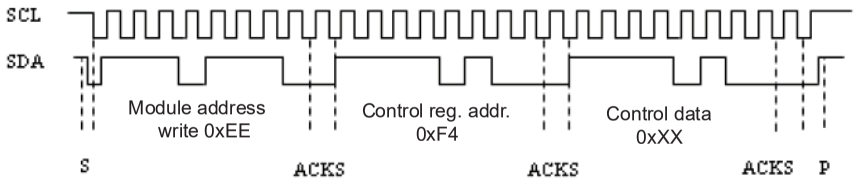
0xEF read
0xEE write

- Flujo de medida



5.4 Start temperature and pressure measurement

The timing diagrams to start the measurement of the temperature value UT and pressure value UP are shown below. After start condition the master sends the device address write, the register address and the control register data. The BMP180 sends an acknowledgement (ACKS) every 8 data bits when data is received. The master sends a stop condition after the last ACKS.



Control register values

Measurement	Control register value (register address 0xF4)	Max. conversion time [ms]
Temperature	0x2E	4.5
Pressure (oss = 0)	0x34	4.5
Pressure (oss = 1)	0x74	7.5
Pressure (oss = 2)	0xB4	13.5
Pressure (oss = 3)	0xF4	25.5

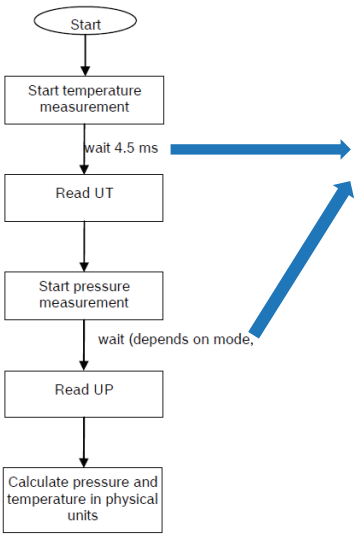
BMP180
Digital pressure sensor

BPM180 module address

A7	A6	A5	A4	A3	A2	A1	W/R
1	1	1	0	1	1	1	0/1

0xEF read
0xEE write

- Flujo de medida



Espera a que el ADC termine de digitalizar el valor sensado:

- Introducir un retardo
- Leer el bit SCO del registro de control (actúa como "flag" de "status")

Sco (register F4h <5>): Start of conversion. The value of this bit stays "1" during conversion and is reset to "0" after conversion is complete (data registers are filled).

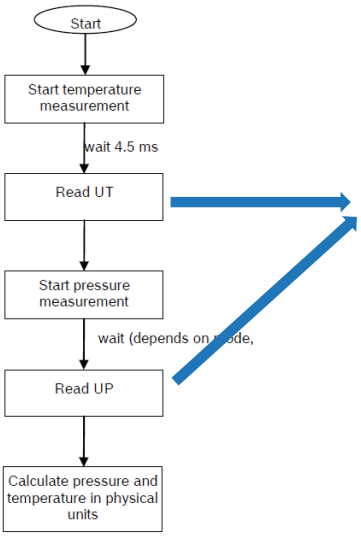
BMP180
Digital pressure sensor

BPM180 module address

A7	A6	A5	A4	A3	A2	A1	W/R
1	1	1	0	1	1	1	0/1

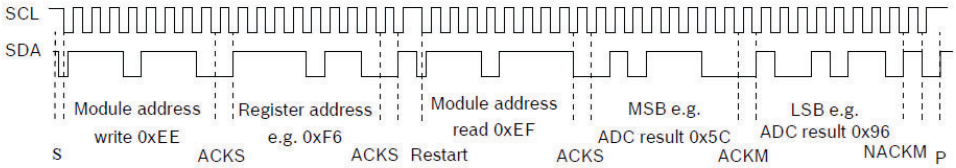
0xEF read
0xEE write

- Flujo de medida



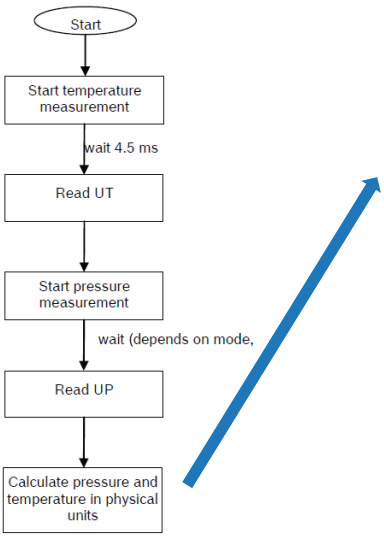
5.5 Read A/D conversion result or E^2PROM data

Then the master sends a restart condition followed by the module address read that will be acknowledged by the BMP180 (ACKS). The BMP180 sends first the 8 MSB, acknowledged by the master (ACKM), then the 8 LSB. The master sends a "not acknowledge" (NACKM) and finally a stop condition.



BMP180
Digital pressure sensor

- Flujo de medida



- Leer los datos de calibración (tabla de 0xAA a 0xBE):
- Realizar los cálculos de temperatura y presión

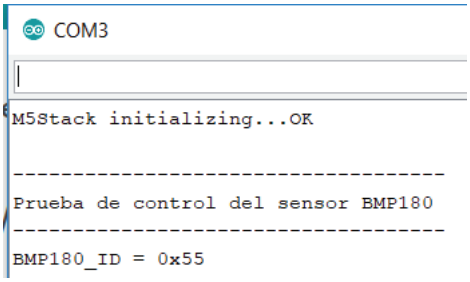
Read calibration data from the E^2PROM of the BMP180	
read out E^2PROM registers, 16 bit, MSB first	
AC1 (0xAA, 0xAB)	(16 bit)
AC2 (0xAC, 0xAD)	(16 bit)
AC3 (0xAE, 0xAF)	(16 bit)
AC4 (0xB0, 0xB1)	(16 bit)
AC5 (0xB2, 0xB3)	(16 bit)
AC6 (0xB4, 0xB5)	(16 bit)
B1 (0xB6, 0xB7)	(16 bit)
B2 (0xB8, 0xB9)	(16 bit)
MB (0xBa, 0xBB)	(16 bit)
MC (0xBC, 0xBD)	(16 bit)
MD (0xBE, 0xBF)	(16 bit)

calculate true temperature
$X1 = (UT - AC6) * AC5 / 2^{15}$
$X2 = MC * 2^{11} / (X1 + MD)$
$B5 = X1 + X2$
$T = (B5 + 8) / 2^4$

calculate true pressure
$B6 = B5 - 4000$
$X1 = (B2 * (B6 * B6 / 2^{12})) / 2^{11}$
$X2 = AC2 * B6 / 2^{11}$
$X3 = X1 + X2$
$B3 = (((AC1*4+X3) << oss) + 2) / 4$
$X1 = AC3 * B6 / 2^{13}$
$X2 = (B1 * (B6 * B6 / 2^{12})) / 2^{16}$
$X3 = ((X1 + X2) + 2) / 2^2$
$B4 = AC4 * (unsigend\ long)(X3 + 32768) / 2^{15}$
$B7 = ((unsigned\ long)UP - B3) * (50000 >> oss)$
if (B7 < 0x80000000) { p = (B7 * 2) / B4 }
else { p = (B7 / B4) * 2 }
$X1 = (p / 2^8) * (p / 2^8)$
$X1 = (X1 * 3038) / 2^{16}$
$X2 = (-7357 * p) / 2^{16}$
$p = p + (X1 + X2 + 3791) / 2^4$

Ejercicios

1. Utilizar el fichero Ejer_1_I2C.ino para realizar una función que lea 1 byte de un registro en un dispositivo utilizando las funciones de la librería Wire de Arduino. Utilizarlo para leer el ID del BMP180 cuando se pulsa el botón A del M5Stack y mostrarlo por el monitor serie.
   - Dirección del BMP180 0x77
   - El ID del BMP180 está en la dirección 0xD0 y tiene el valor 0x55
   - Función: uint8_t read_1byte(uint8_t device_addr, uint8_t read_addr);



Ejercicios

2. Visualizar y analizar con el analizador lógico las secuencias I2C generadas al leer el ID del BMP180 con la función realizada en el ejercicio anterior.
   - Dirección del BMP180 =x77
   - El ID del BMP180 está en la dirección 0xD0 y tiene el valor 0x55

Ejercicios

3. Utilizar el fichero Ejer_3_I2C.ino para realizar una función que escriba un byte en un registro de un dispositivo. Utilizarlo para escribir en el registro de control del BMP180 e inicializar la conversión de temperatura cuando se pulsa el botón A del M5Stack.

Una vez se ha escrito, leer el valor del registro de control y mostrarlo por el monitor serie para comprobar que se ha escrito correctamente. Utilizar el analizador lógico para visualizar y analizar las tramas I2C.

- Dirección del BMP180 0x77
- Dirección del registro de control 0xF4
- Control de inicio de medida de temperatura 0x2E
- Función: void write_1byte(uint8_t device_addr, uint8_t write_addr, uint8_t data);

Ejercicios

4. Utilizar el fichero Ejer_4_I2C.ino para realizar una función que lea una palabra de 16 bits (2 bytes) almacenada en 2 registros cuyas direcciones son consecutivas.

Utilizarlo para leer la el valor digitalizado de la temperatura del BMP180 cuando se pulsa el botón A del M5Stack y mostrarlo por el monitor serie. Utilizar el analizador lógico para visualizar y analizar las tramas I2C.

- Dirección del BMP180 0x77
- Dirección del registro con el byte MSB 0xF6
- Dirección del registro con el byte LSB 0xF7
- Función: uint16_t read_2bytes(uint8_t device_addr, uint8_t read_addr);

Ejercicios

5. Leer el código en C del driver del BMP180 (SFE_BMP180.ccp). Analizar el código teniendo en cuenta las operaciones de lectura y escritura en registros y el flujo de medida del dispositivo.

El driver está en PoliformaT/Recursos/Componentes

Ejercicios

6. Complete el fichero Ejer_6_I2C.ino para realizar una función que lea 1 byte de un registro en un dispositivo utilizando las funciones de la librería i2c.h del ESP-IDF de ESPRESSIF. Utilizarlo para leer el ID del BMP180 cuando se pulsa el botón A del M5Stack y mostrarlo por el monitor serie.
   - Dirección del BMP180 0x77
   - El ID del BMP180 está en la dirección 0xD0 y tiene el valor 0x55
   - Función: uint8_t read_1byte(uint8_t device_addr, uint8_t read_addr);