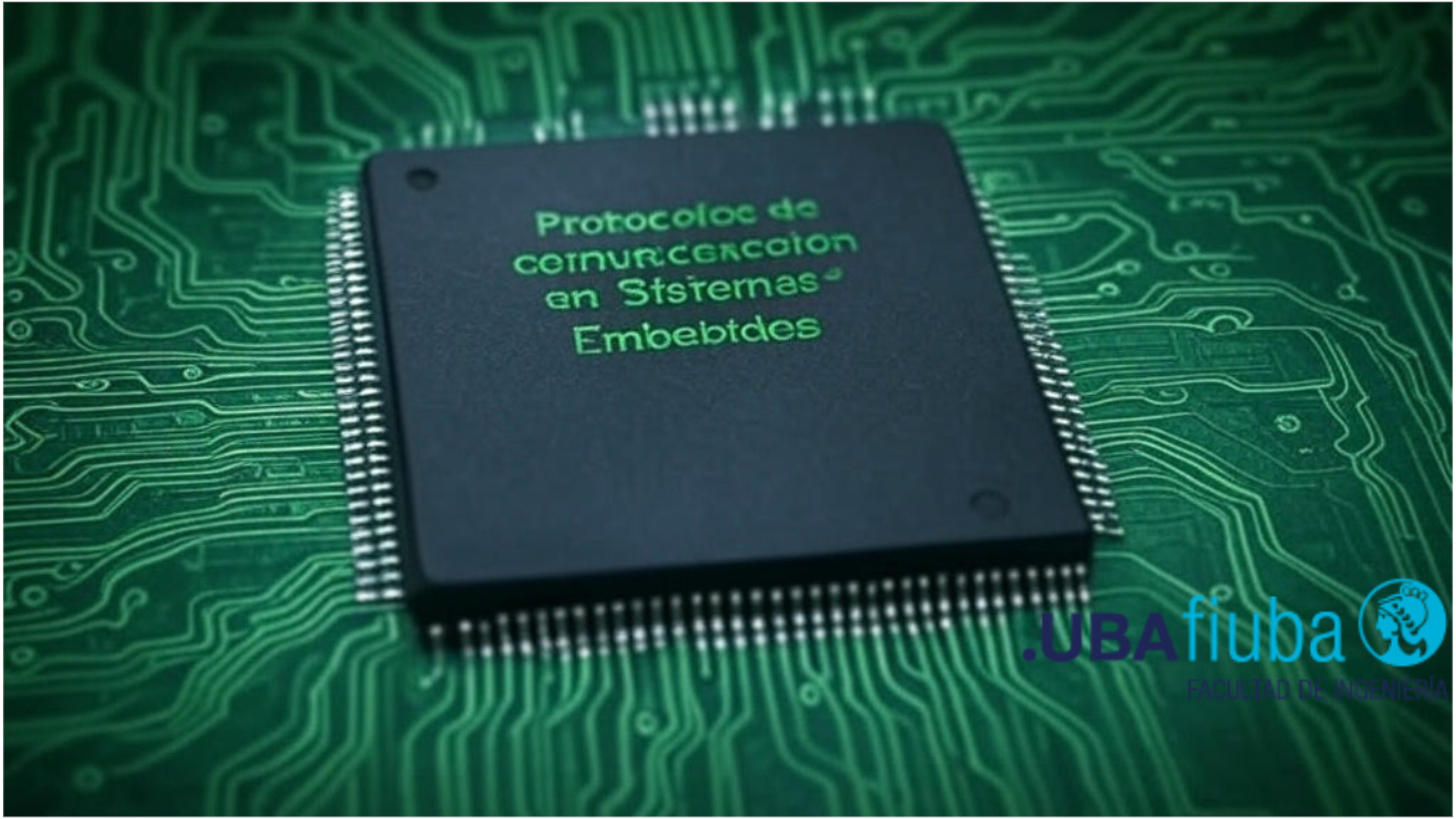


Protocoloc de
cemnurcexcaion
en Stsfemas
Embebtdes
.UBA fiuba
FACULTAD DE INGENIERÍA

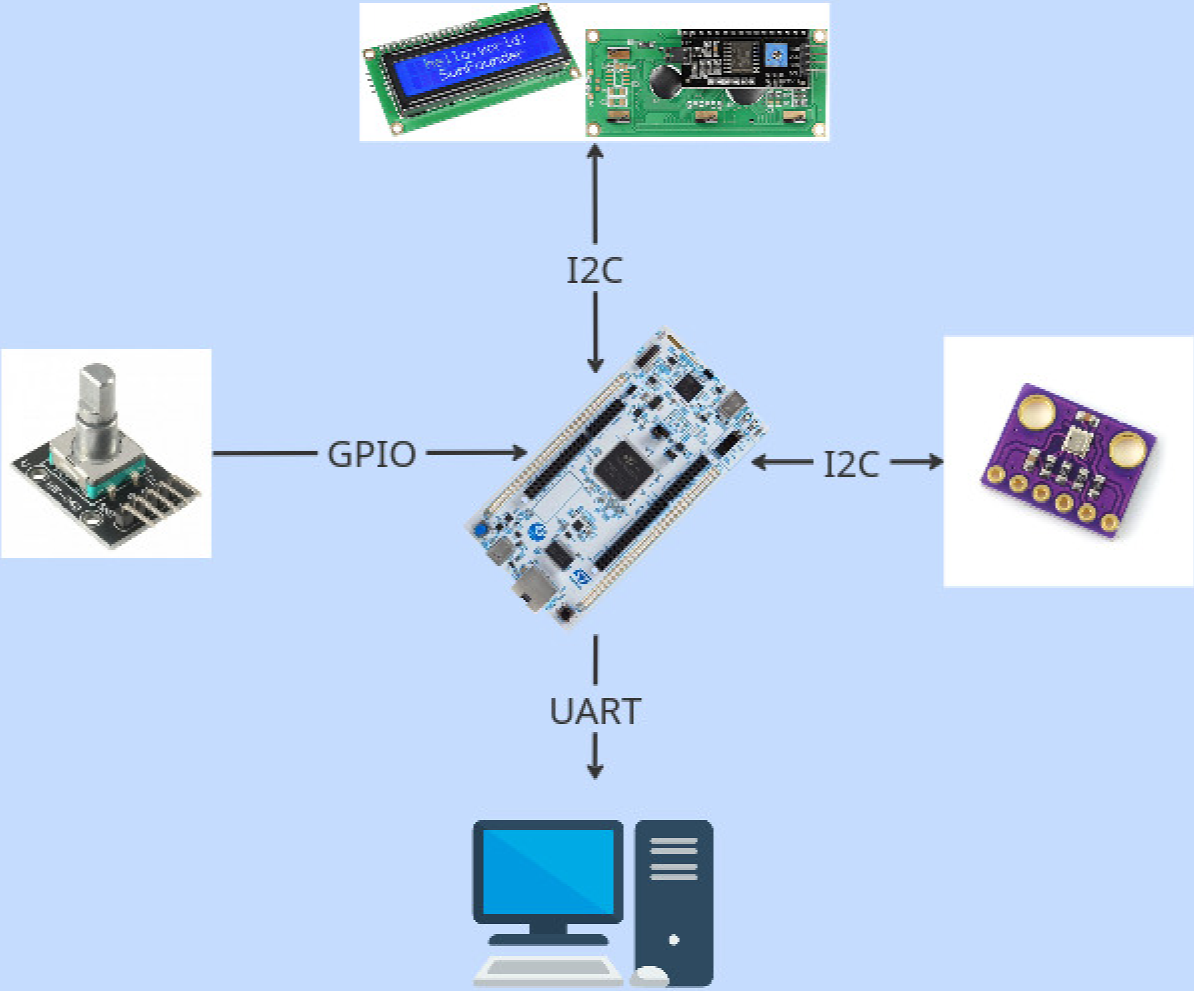
Protocolos de comunicación en sistemas embebidos

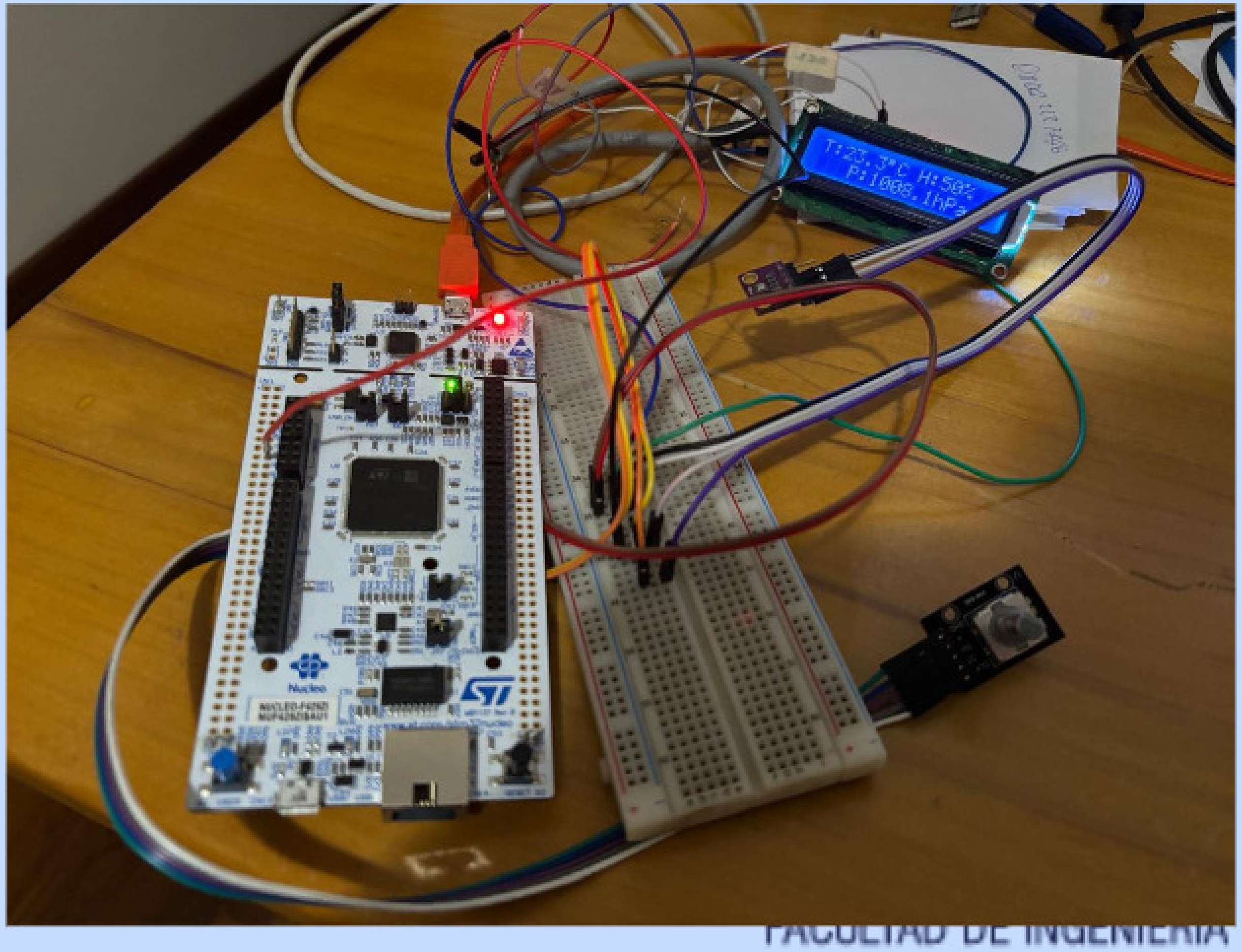
Docente: Israel Pavelek

Alumno: Fernando Folmer

.UBAfiuba

FACULTAD DE INGENIERÍA



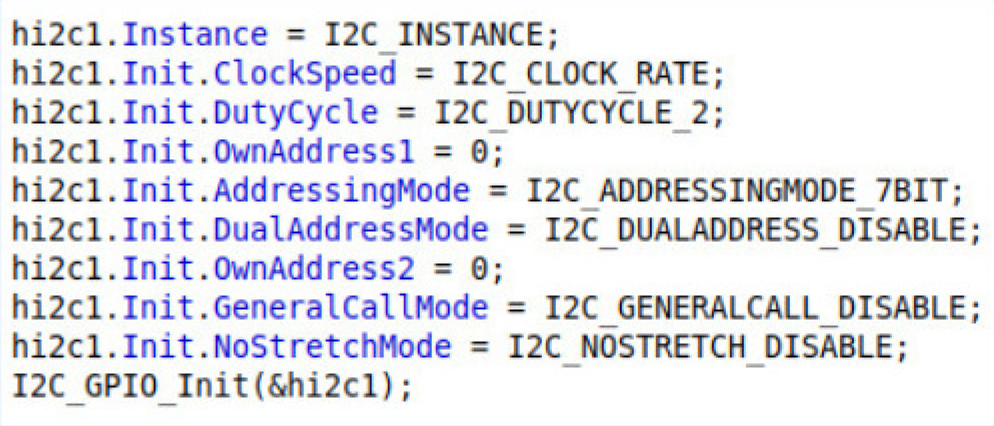


FACULTAD DE INGENIERIA

Configuración de Protocolos

I2C

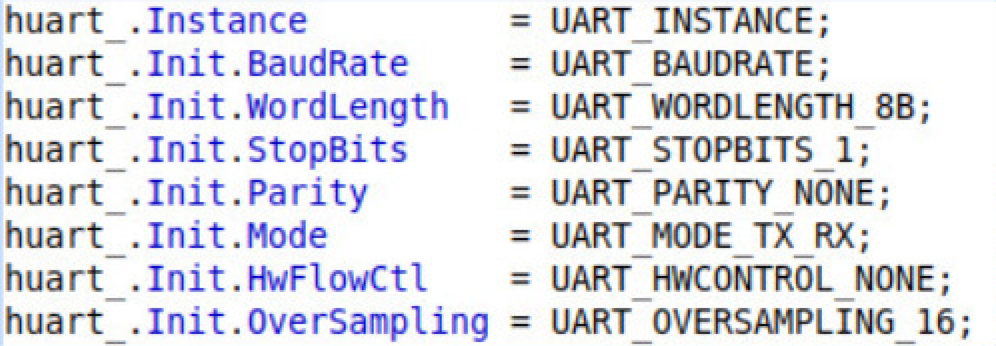
- **Frecuencia:** 100 kHz
- **Duty Cycle:** 1:1(Se ignora en 100kHz)
- **Direccionamiento:** 7 bits
- **Dual address:** Desactivado
- **Llamada general:** Deshabilitada
- **Stretching:** Habilitado



```
hi2c1.Instance = I2C_INSTANCE;
hi2c1.Init.ClockSpeed = I2C_CLOCK_RATE;
hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
hi2c1.Init.OwnAddress1 = 0;
hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2 = 0;
hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
I2C_GPIO_Init(&hi2c1);
```

UART

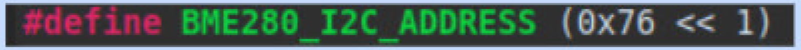
- **Baud Rate:** 115200 baudios
- **Longitud palabra:**8 bits
- **Bits de stop:** 1 bit
- **Bit de paridad:** NO
- **Modo:** Tx-Rx
- **Oversampling:** x16



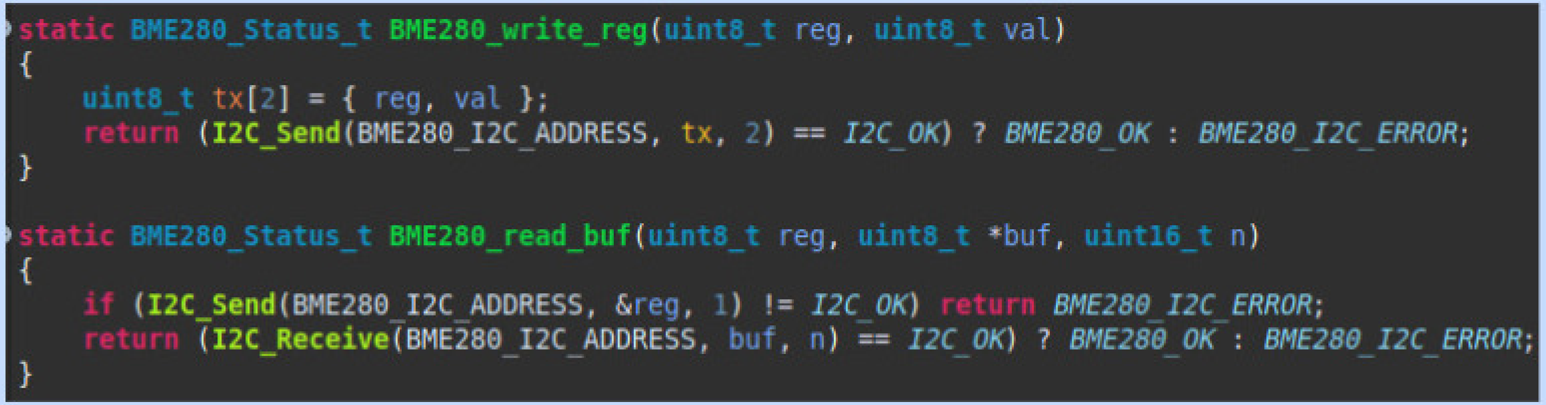
```
huart_.Instance          = UART_INSTANCE;
huart_.Init.BaudRate     = UART_BAUDRATE;
huart_.Init.WordLength   = UART_WORDLENGTH_8B;
huart_.Init.StopBits     = UART_STOPBITS_1;
huart_.Init.Parity       = UART_PARITY_NONE;
huart_.Init.Mode         = UART_MODE_TX_RX;
huart_.Init.HwFlowCtl    = UART_HWCONTROL_NONE;
huart_.Init.OverSampling = UART_OVERSAMPLING_16;
```

FACULTAD DE INGENIERÍA

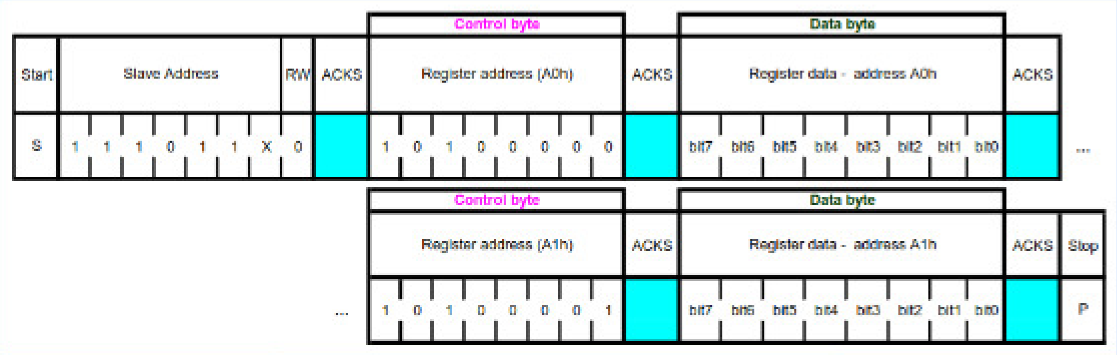
BME280 (Bosch)



```c
#define BME280_I2C_ADDRESS (0x76 << 1)
```



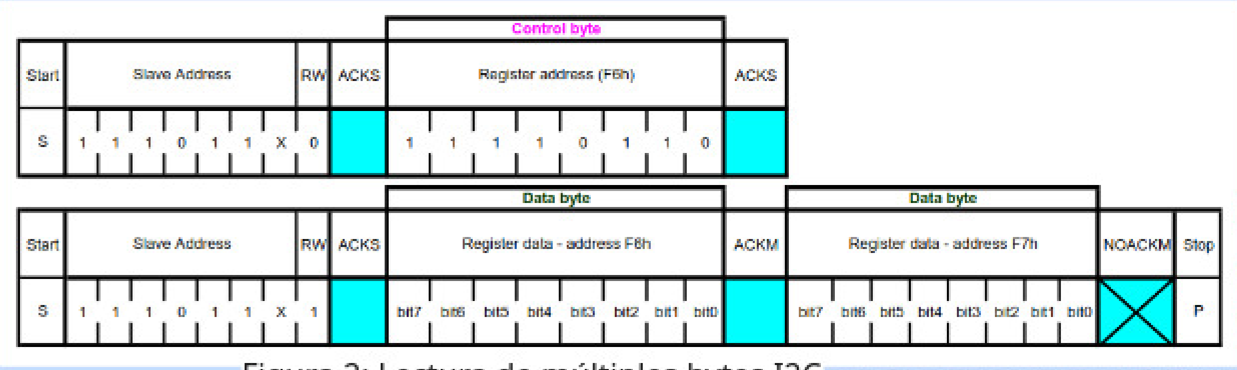
```c
static BME280_Status_t BME280_write_reg(uint8_t reg, uint8_t val)
{
    uint8_t tx[2] = { reg, val };
    return (I2C_Send(BME280_I2C_ADDRESS, tx, 2) == I2C_OK) ? BME280_OK : BME280_I2C_ERROR;
}

static BME280_Status_t BME280_read_buf(uint8_t reg, uint8_t *buf, uint16_t n)
{
    if (I2C_Send(BME280_I2C_ADDRESS, &reg, 1) != I2C_OK) return BME280_I2C_ERROR;
    return (I2C_Receive(BME280_I2C_ADDRESS, buf, n) == I2C_OK) ? BME280_OK : BME280_I2C_ERROR;
}
```



				Control byte		Data byte		
Start	Slave Address	RW	ACKS	Register address (A0h)	ACKS	Register data - address A0h	ACKS	
S	1 1 1 0 1 1 X	0		1 0 1 0 0 0 0 0		bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0		...

	Control byte		Data byte		
	Register address (A1h)	ACKS	Register data - address A1h	ACKS	Stop
...	1 0 1 0 0 0 0 1		bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0		P

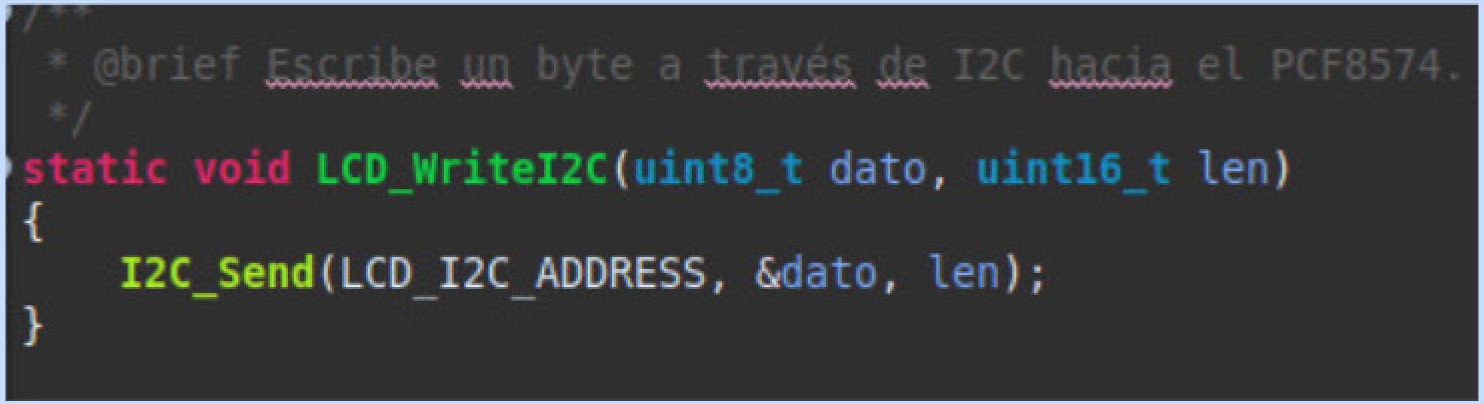
Figura 1: Escritura de múltiples bytes I2C



				Control byte	
Start	Slave Address	RW	ACKS	Register address (F6h)	ACKS
S	1 1 1 0 1 1 X	0		1 1 1 1 0 1 1 0	

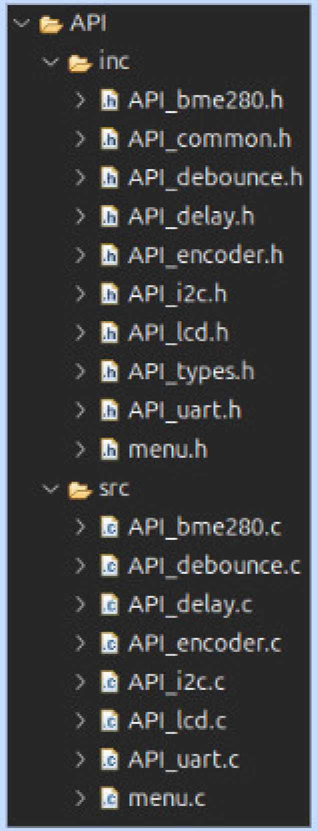
				Data byte		Data byte		
Start	Slave Address	RW	ACKS	Register data - address F6h	ACKM	Register data - address F7h	NOACKM	Stop
S	1 1 1 0 1 1 X	1		bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0		bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0	☒	P

Figura 2: Lectura de múltiples bytes I2C

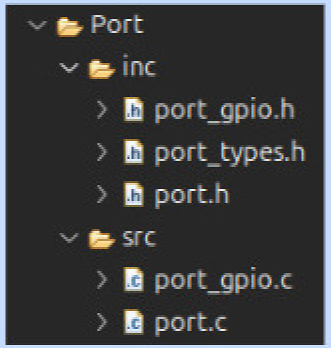


```
/**
 * @brief Escribe un byte a través de I2C hacia el PCF8574.
 */
static void LCD_WriteI2C(uint8_t dato, uint16_t len)
{
    I2C_Send(LCD_I2C_ADDRESS, &dato, len);
}
```

.UBAfiuba
FACULTAD DE INGENIERÍA



```
API
  inc
    API_bme280.h
    API_common.h
    API_debounce.h
    API_delay.h
    API_encoder.h
    API_i2c.h
    API_lcd.h
    API_types.h
    API_uart.h
    menu.h
  src
    API_bme280.c
    API_debounce.c
    API_delay.c
    API_encoder.c
    API_i2c.c
    API_lcd.c
    API_uart.c
    menu.c
```



```
Port
  inc
    port_gpio.h
    port_types.h
    port.h
  src
    port_gpio.c
    port.c
```

Enlaces a material de demostración

Video demostrativo LCD + BME280 (I2C)

- https://photos.app.goo.gl/9nqMM9WJit4xygb56

Captura puerto serie PC (UART)

- https://photos.app.goo.gl/zm392q2AU1JLYJzv7

.UBAfiuba
FACULTAD DE INGENIERÍA