Universitat de Lleida

Testing process for components in the Chemical plant project
Embedded and Ubiquitous systems

Task part of the second sprint

UDL - MINF - 2021 - Team 1

Content:

Objective

The main objective of the task is to ensure that every component of the project is working, doing simple tests.

List of components

1x Raspberry Pi
1x Arduino UNO
3x ESP-01
1x LCD
1x DHT11 sensor
1x Ultrasonic sensor (HC-SR04)
1x PCF
1x I2c module

Components already tested while developing: Arduino and Raspberry, so they won't need a specific test method.

Tests done

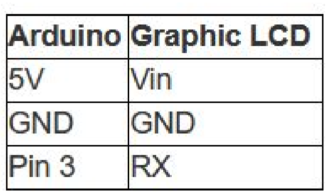
1. LCD

First you need to include the library for working with the LCD <SparkFunSerialGraphicLCD.h>. (via the Arduino IDE)

The test will be taken using the arduino as controller with the following code:

```
#include <SparkFunSerialGraphicLCD.h> //include  the Serial Graphic LCD library
#include <SoftwareSerial.h>

#define maxX 127//159
#define maxY 63 //127

LCD LCD;

void setup() {
  // put your setup code here, to run once:

  LCD.setHome();//set the cursor back to 0,0.
  LCD.clearScreen();//clear anything that may have been previously printed ot the screen.

  LCD.printStr("MINF UDL");
  delay(5000);

  LCD.setX((maxX/2)-18);
  LCD.setY((maxY/2)-4);
  LCD.printStr("Master Informatic Engineering");
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

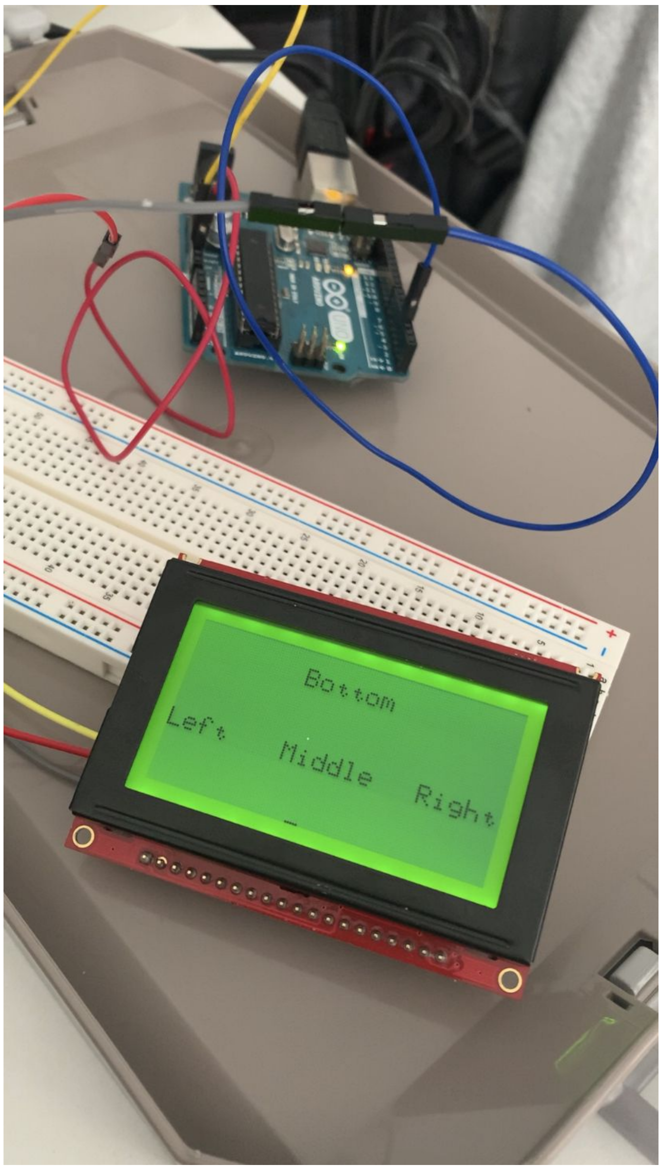
Electrical wiring:



Arduino	Graphic LCD
5V	Vin
GND	GND
Pin 3	RX

It will write some things based on the position (X,Y).

Photo:



Conclusion: This test was enough to start knowing the LCD behaviour and also ensuring it's working.

2. **ESP-01**

In this test we will program the ESP-01 to act as a wifi server. (We must repeat this test for the 3 ESP-01).

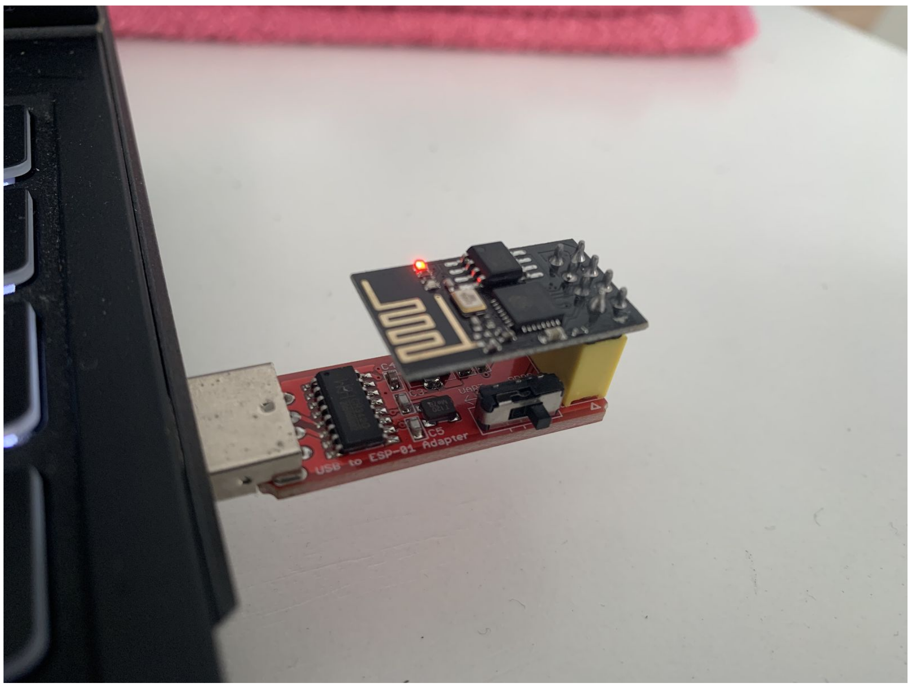
1. On the adapter, switch to the PROG mode (the other is UART)
2. Connect the adapter on computer and install its drivers:
   USB Adapter Driver: http://www.wch.cn/download/CH341SER_EXE.html
3. After that, you can connect your ESP-01 on it and plug on the computer
4. In the Arduino IDE go Tools > Board > ESP8266 boards and then select Generic ESP8266 Module.
5. Tools > Port > Select the port the USB is in (check in the devices page of windows)
6. Now you can Verify and then compile your code.
   a. Here you have a simple code for blinking the led (blue) of the ESP8266:

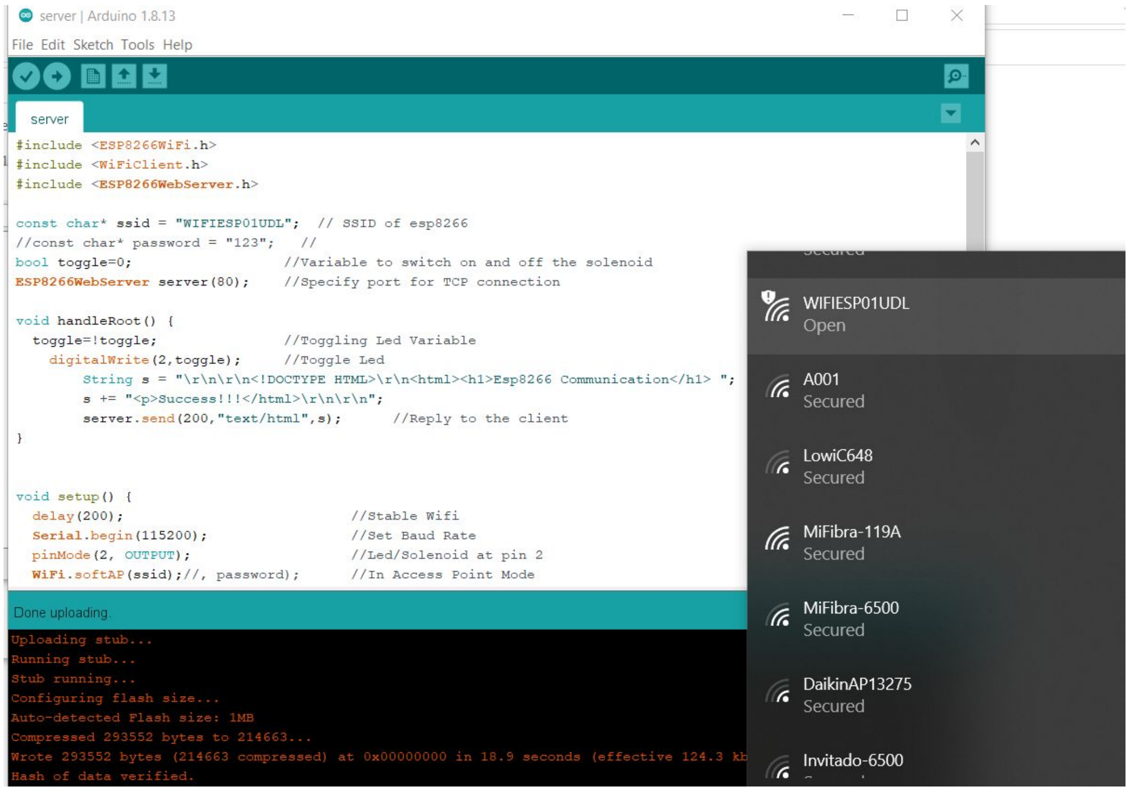
b. https://learn.sparkfun.com/tutorials/esp8266-thing-hookup-guide/example-sketch-blink

Code for the testing:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char* ssid = "WIFIESP01UDL";  // SSID of esp8266
//const char* password = "123";   //
bool toggle=0;                          //Variable to switch on and off the solenoid
ESP8266WebServer server(80);                        //Specify port for TCP connection

void handleRoot() {
 String s = "\r\n\r\n<!DOCTYPE HTML>\r\n<html><h1>Esp8266 Communication</h1> ";
 s += "<p>Success!!!</html>\r\n\r\n";
 server.send(200,"text/html",s);                    //Reply to the client
}


void setup() {
 delay(200);                            //Stable Wifi
 Serial.begin(115200);                  //Set Baud Rate
 //pinMode(2, OUTPUT);                              //Led/Solenoid at pin 2
 WiFi.softAP(ssid);//, password);                   //In Access Point Mode

 IPAddress myIP = WiFi.softAPIP();                  //Check the IP assigned. Put this Ip in the client host.
 Serial.print("AP IP address: ");
 Serial.println(myIP);                  //Print the esp8266-01 IP(Client must also be on the save IP series)
 server.on("/Led", handleRoot);                     //Checking client connection
 server.begin();                        // Start the server
 Serial.println("Server started");
}

void loop() {
 // Check if a client has connected. On first connection switch on the Solenoid on next switch off.
 server.handleClient();
}
```

Photos:





```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char* ssid = "WIFIESP01UDL";  // SSID of esp8266
//const char* password = "123";   //
bool toggle=0;                    //Variable to switch on and off the solenoid
ESP8266WebServer server(80);      //Specify port for TCP connection

void handleRoot() {
  toggle=!toggle;                 //Toggling Led Variable
    digitalWrite(2,toggle);       //Toggle Led
        String s = "\r\n\r\n<!DOCTYPE HTML>\r\n<html><h1>Esp8266 Communication</h1> ";
        s += "<p>Success!!!</html>\r\n\r\n";
        server.send(200,"text/html",s);       //Reply to the client
}


void setup() {
  delay(200);                             //Stable Wifi
  Serial.begin(115200);                   //Set Baud Rate
  pinMode(2, OUTPUT);                     //Led/Solenoid at pin 2
  WiFi.softAP(ssid);//, password);        //In Access Point Mode
```

```
Done uploading.
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 1MB
Compressed 293552 bytes to 214663...
Wrote 293552 bytes (214663 compressed) at 0x00000000 in 18.9 seconds (effective 124.3 kb
Hash of data verified.
```

Conclusion: This test was enough to ensure the functionality of the ESP-01.

3. DHT11

In this test we will see if our temperature sensor component (DHT11 is working).

First, we need to add the following libraries on the Arduino IDE:
https://github.com/adafruit/DHT-sensor-library
and
https://github.com/adafruit/Adafruit_Sensor

You can do that manually or going into the Arduino IDE > Tools > Manage Libraries > and then search and install the libraries.

Code:

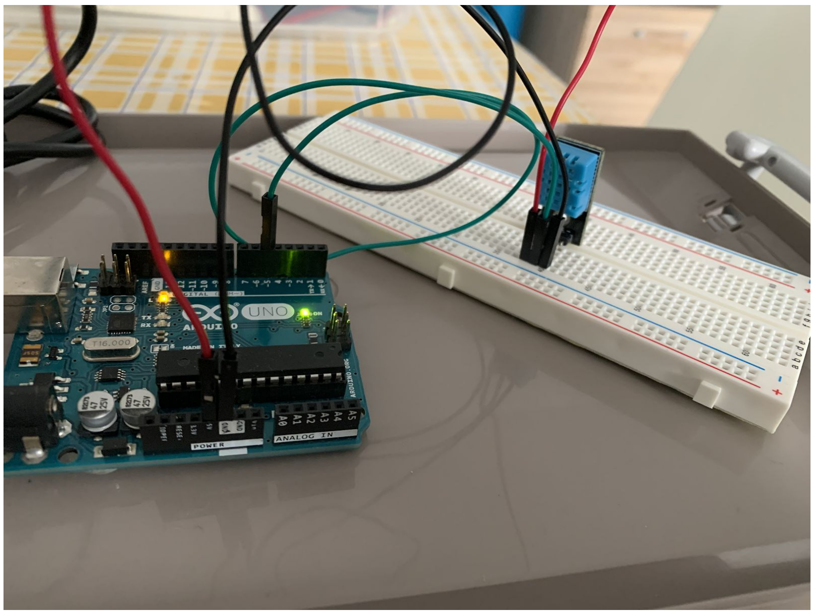
```
#include "DHT.h"

// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor
```

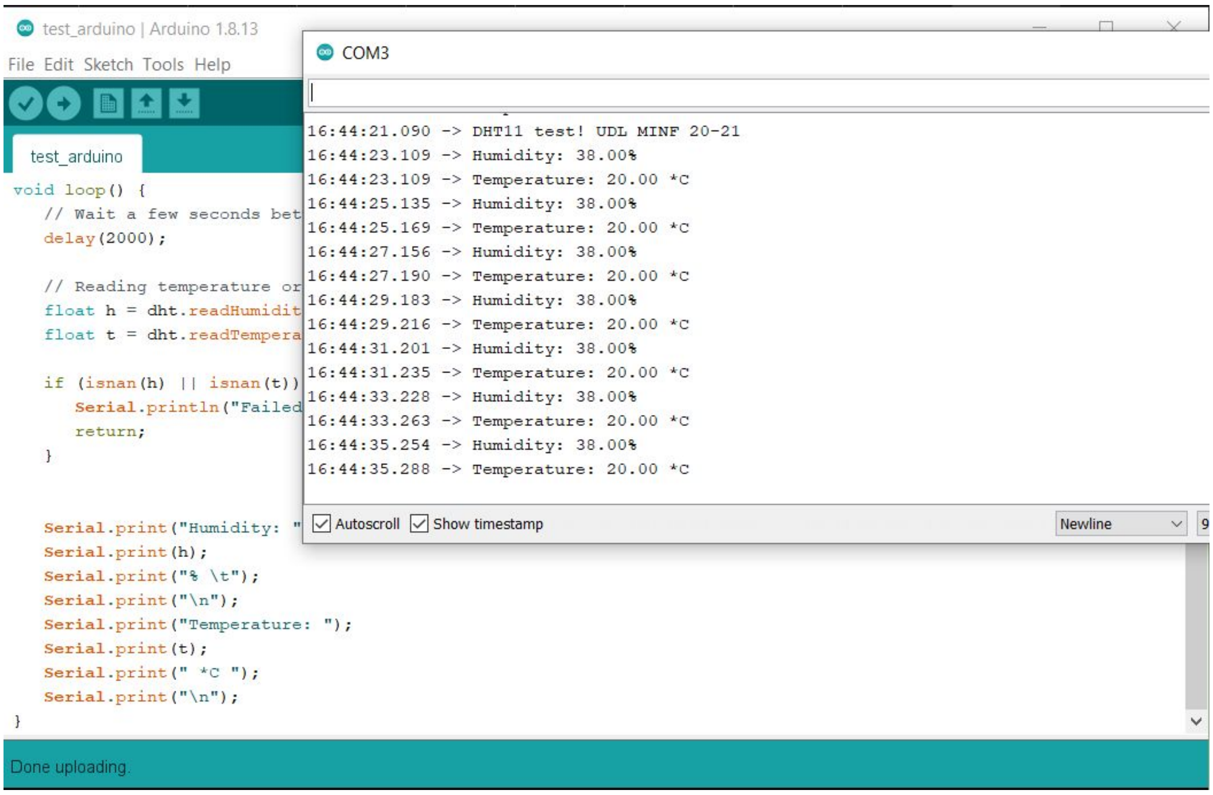
```
const int DHTPin = 5;                // what digital pin we're connected to

DHT dht(DHTPin, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHT11 test! UDL MINF 20-21");

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (isnan(h) || isnan(t)) {
                        Serial.println("Failed to read from DHT sensor!");
                        return;
  }


  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print("% \t");
  Serial.print("\n");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" *C ");
  Serial.print("\n");
}
```

Electrical Wiring:

DHT11 Vcc -> Arduino 5V
DHT11 Output ->one of the digital pins of Arduino (5 per example)
DHT11 GND ->Arduino GND (its on the side of the 5V)

Photos:





```
void loop() {
  // Wait a few seconds bet
  delay(2000);

  // Reading temperature or
  float h = dht.readHumidit
  float t = dht.readTempera

  if (isnan(h) || isnan(t))
    Serial.println("Failed
    return;
  }


  Serial.print("Humidity: "
  Serial.print(h);
  Serial.print("% \t");
  Serial.print("\n");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" *C ");
  Serial.print("\n");
}
```

```
16:44:21.090 -> DHT11 test! UDL MINF 20-21
16:44:23.109 -> Humidity: 38.00%
16:44:23.109 -> Temperature: 20.00 *C
16:44:25.135 -> Humidity: 38.00%
16:44:25.169 -> Temperature: 20.00 *C
16:44:27.156 -> Humidity: 38.00%
16:44:27.190 -> Temperature: 20.00 *C
16:44:29.183 -> Humidity: 38.00%
16:44:29.216 -> Temperature: 20.00 *C
16:44:31.201 -> Humidity: 38.00%
16:44:31.235 -> Temperature: 20.00 *C
16:44:33.228 -> Humidity: 38.00%
16:44:33.263 -> Temperature: 20.00 *C
16:44:35.254 -> Humidity: 38.00%
16:44:35.288 -> Temperature: 20.00 *C
```

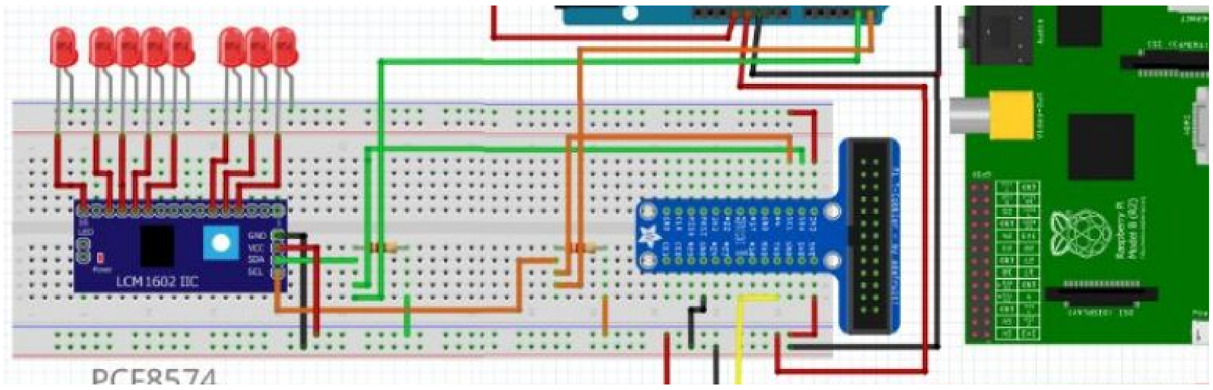
Conclusion: This test showed us that the sensor is functioning correctly and showing the values as it should.

4. HC-SR04

The testing was done through the data producer 2 development, all the process is in its respective documentation.

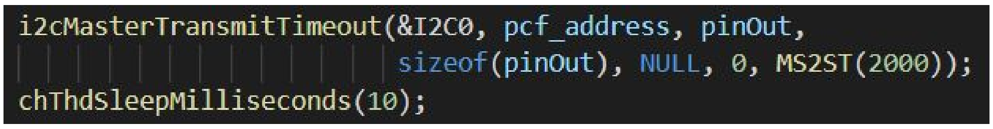
5. PCF

To test the PCF, we needed to connect 8 LED's to its I/O ports and also, the SDA and SCL lines in the i2c connected, so that way the PCF would be able to receive data from the i2c master (the raspberry in this case), the connections are as follow:

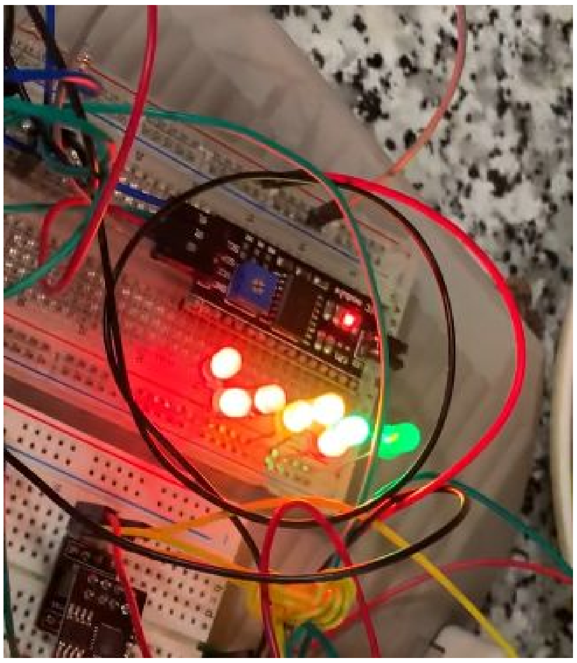


The address of the PCF8574 in the i2c connection is: 0x27

With the wiring done, we can use the ChibiOS code to send data to the PCF over i2c, this data will indicate to the PCF8574 which LEDs he needs to turn on (which digital ports he need to activate)


```
i2cMasterTransmitTimeout(&I2C0, pcf_address, pinOut,
                         sizeof(pinOut), NULL, 0, MS2ST(2000));
chThdSleepMilliseconds(10);
```

With this, we managed to send data to the PCF and also confirm its functionality.



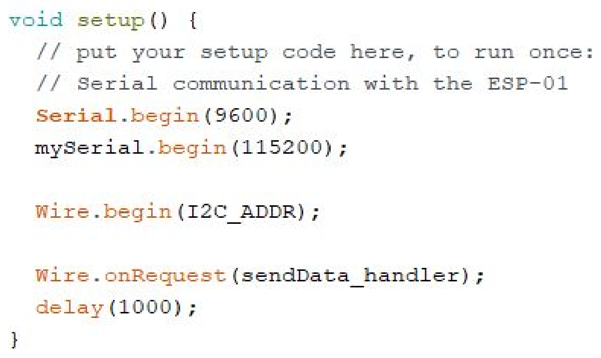
6. I2c module

In this specific module, we needed first to build the code for the corresponding devices which will be connecting through i2c protocol.

The devices used were:
Arduino (address 0x04 slave)
Raspberry Pi (master)
PCF8574 (address 0x27 slave)

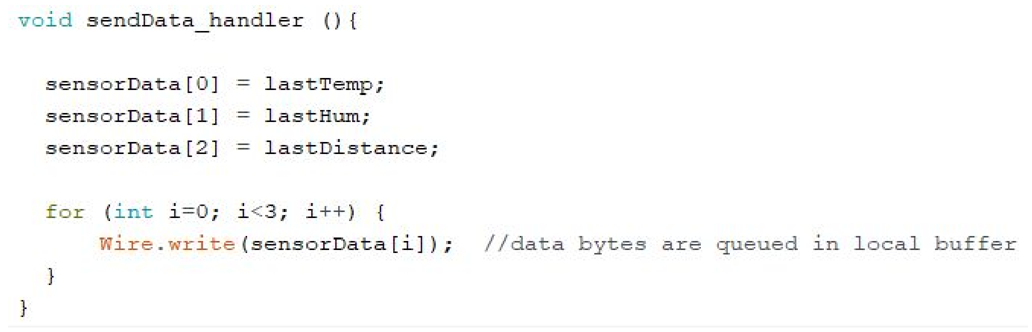
First, we need to register in Arduino the function to enter the i2c protocol as a slave, for that we are using the **Wire** library

#include <Wire.h>



```
void setup() {
  // put your setup code here, to run once:
  // Serial communication with the ESP-01
  Serial.begin(9600);
  mySerial.begin(115200);

  Wire.begin(I2C_ADDR);

  Wire.onRequest(sendData_handler);
  delay(1000);
}
```

We also register a callback function to be executed when the master of the i2c asks for data

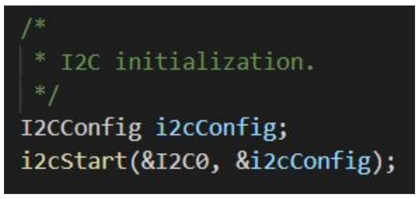


```
 void sendData_handler (){

   sensorData[0] = lastTemp;
   sensorData[1] = lastHum;
   sensorData[2] = lastDistance;

   for (int i=0; i<3; i++) {
       Wire.write(sensorData[i]);  //data bytes are queued in local buffer
   }
 }
```

This will send 3 ints to the raspberry through i2c connection.

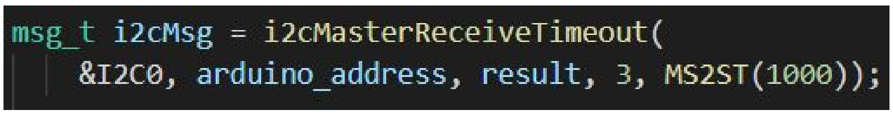
Next we need to provide the Raspberry (chibiOS) code, its very similar with the arduino but we have different functions for that.

First, we initialize the i2c connection as a master:



```
/*
 * I2C initialization.
 */
I2CConfig i2cConfig;
i2cStart(&I2C0, &i2cConfig);
```

Then, the next thing we do is just asks for data through this function:



```
msg_t i2cMsg = i2cMasterReceiveTimeout(
    &I2C0, arduino_address, result, 3, MS2ST(1000));
```

This will tell the Arduino to send data with a size of (3).

The pins used for the i2c conections is as follow:

Arduino A4 -> Raspberry SDA
Arduino A5 -> Raspberry SDL
with both lines pulled high +5v with a resistor.

Through this process, we managed to send data through the i2c protocol and so we consider this topic as properly tested.