
# Technical document - Final delivery

## Ubiquitous and Embedded systems

## UDL MINF 2021

## Team 1

## Document structure:

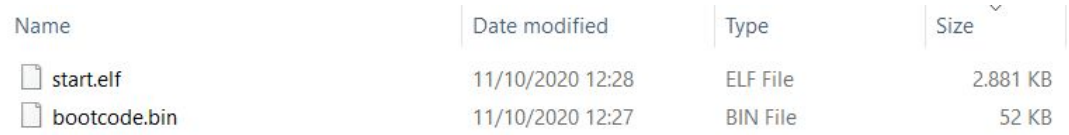
ChibiOS on Raspberry

Link to get some help (from the creator of the ChibiOS port for the RPi):
https://www.stevebate.net/chibios-rpi/GettingStarted.html

1. You need to install the arm cross compiler in order to compile the kernel
    a. If you are on Linux, sudo apt-get install gcc-arm-none-eabi
    b. If you are on Windows, developer.arm.com > tools & and Software > open source Software > GNU Toolchain > Embedded (or search for YAGARTO)

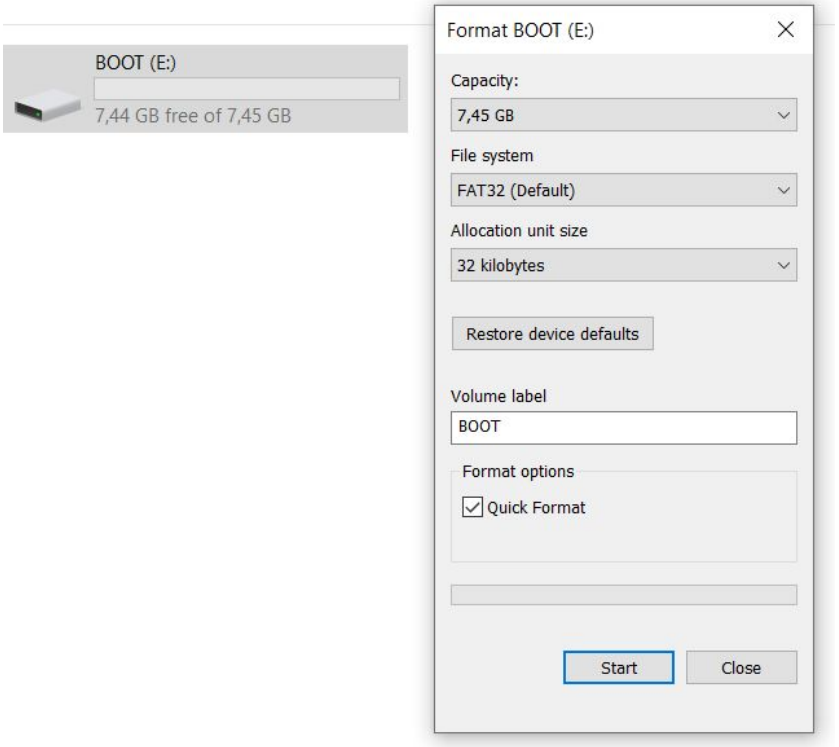
2. Download the Raspberry B ChibiOS Port from github:
   https://github.com/steve-bate/ChibiOS-RPi

3. Download the firmware files from the github:
    a. start.elf & bootcode.bin




   https://github.com/raspberrypi/firmware

4. Having everything, we need to connect the SD Card on the computer and format it in FAT32 (save a backup of the files previously inside it).



5. Now, navigate to the folder \ChibiOS-RPi-master\demos\ARM11-BCM2835-GCC of the ChibiOS Port. (This folder will compile a demo project for our Raspberry model).

6. Type ‘make’ to build the code, if everything worked, it will create a sub-folder called build with a file named ch.bin inside it.

7. Rename the ch.bin you just created to kernel.img.

8. Copy the start.elf, bootcode.bin and kernel.img to the SD card.

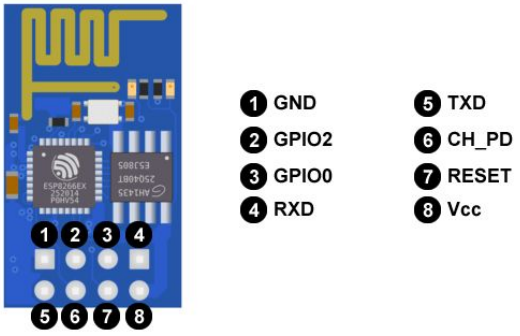
9. Now plug the SD card on the raspberry and it’s done, a demo project is prepared to be used. (We can use this demo project as a model to our needs, just modifying it.)

Universitat de Lleida

# Arduino and ESP8266 integration (Data Receiver)
# Embedded and Ubiquitous systems
# Task part of the second sprint
# UDL - MINF - 2021 - Team 1

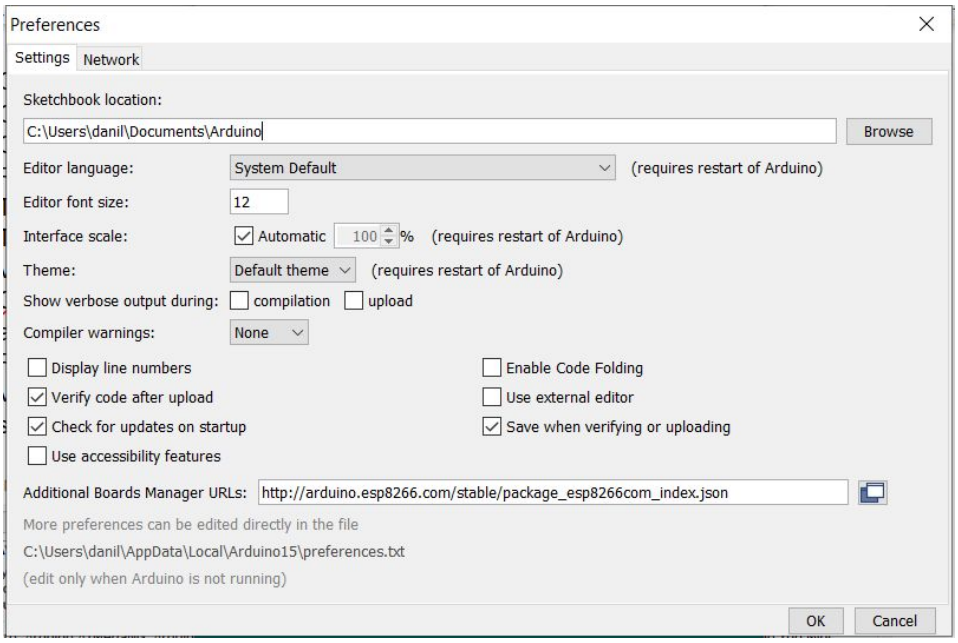
Content:






1. GND = ground
2. GPIO2 general purpose I/O. Its digital pin 2.
3. GPIO0 general purpose I/O. Its digital pin 0.
4. RXD es el pin por donde se van a recibir los datos del puerto serie. Trabaja a 3,3 V. También se puede utilizar como pin digital GPIO: sería el número 3.
5. TXD es el pin por donde se van a transmitir los datos del puerto serie. Trabaja a 3,3 V. También se puede utilizar como pin digital GPIO: sería el número 1.
6. CH_PD pin para apagar y encender el ESP-01: si lo ponemos a 0 V (LOW) se apaga, y a 3,3 V (HIGH) se enciende.
7. **RESET** pin to reset ESP-01: if receives 0 V (LOW) it resets.
8. Vcc is where we power the ESP-01. Works in 3,3 V allows a maximum of 3,6 V. The supplied power must be greater than 200 mA.

**Installing ESP-01 libraries on Arduino IDE:**




http://arduino.esp8266.com/stable/package_esp8266com_index.json


**Programming**:

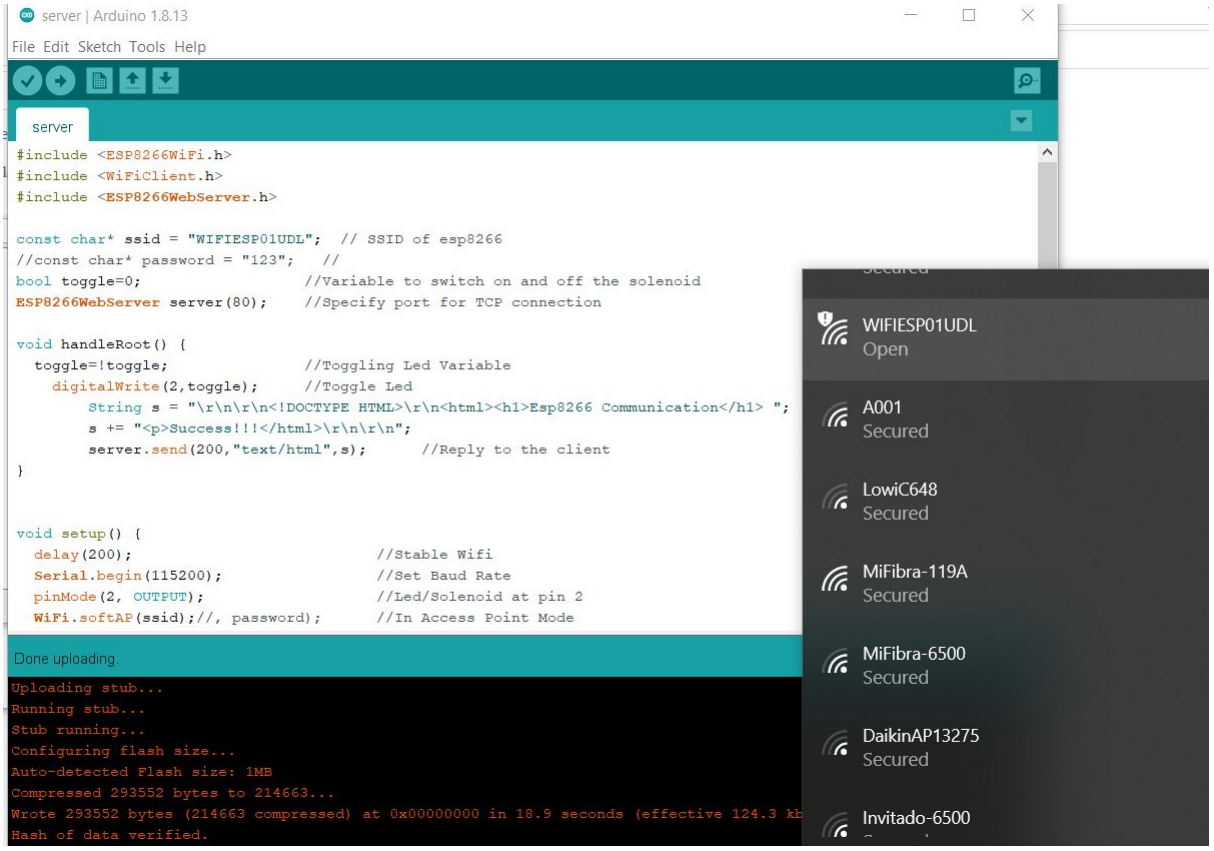
Since we have the USB to ESP-01 adapter (red board with USB), we can easily use that to program the ESP-01.

The idea here is to program (flash) every ESP-01 using this method, and then move it to the final implementation of the project

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.
7. Here you have a simple code for blinking the led (blue) of the ESP8266:
   a. https://learn.sparkfun.com/tutorials/esp8266-thing-hookup-guide/example-sketch-blink

ESP-01 + DHT11: https://www.youtube.com/watch?v=of_g89PQQqU

Testing the Arduino as a Acess Point:



## Setting it up to work as data receiver

1. The ESP-01 will communicate with the Arduino using serial connection, in this case, we are gonna to emulate this connection using the SoftwareSerial library.
2. That said, let's talk about the code:
    a. Arduino code (will need some minor changes for the final project):

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2, 3); // RX, TX

// Variables to handle the data from the ESP
const byte numChars = 32;
char receivedChars[numChars];
char tempChars[numChars];     // temporary array for use when parsing

// variables to hold the parsed data
char message[numChars] = {0};
float floatTemp = 0.0;
float floatHum = 0.0;

boolean newData = false;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  mySerial.begin(115200);
  delay(5000);
}

// Enter data in this style <HelloWorld, 12, 24.7>

void loop() {
       recvWithStartEndMarkers();
       if (newData == true) {
       strcpy(tempChars, receivedChars);
              // this temporary copy is necessary to protect the original data
              //   because strtok() used in parseData() replaces the commas with \0
       parseData();
       showParsedData();
       newData = false;
       }
}

//============

void recvWithStartEndMarkers() {
       static boolean recvInProgress = false;
       static byte ndx = 0;
       char startMarker = '<';
       char endMarker = '>';
       char rc;

       while (mySerial.available() > 0 && newData == false) {
       rc = mySerial.read();
```

```
        if (recvInProgress == true) {
                if (rc != endMarker) {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= numChars) {
                        ndx = numChars - 1;
                }
                }
                else {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
                }
        }

        else if (rc == startMarker) {
                recvInProgress = true;
        }
        }
}

//============

void parseData() {     // split the data into its parts

        char * strtokIndx; // this is used by strtok() as an index

        strtokIndx = strtok(tempChars,",");   // get the first part - the string
        strcpy(message, strtokIndx); // copy it to messageFromPC

        strtokIndx = strtok(NULL, ","); // this continues where the previous call left off
        floatTemp = atof(strtokIndx); // convert this part to a float

        strtokIndx = strtok(NULL, ",");
        floatHum = atof(strtokIndx);  // convert this part to a float

}

//============

void showParsedData() {
        Serial.print("Message: ");
        Serial.println(message);
        Serial.print("Temperature: ");
        Serial.println(floatTemp);
        Serial.print("Humidity: ");
        Serial.println(floatHum);
}
```

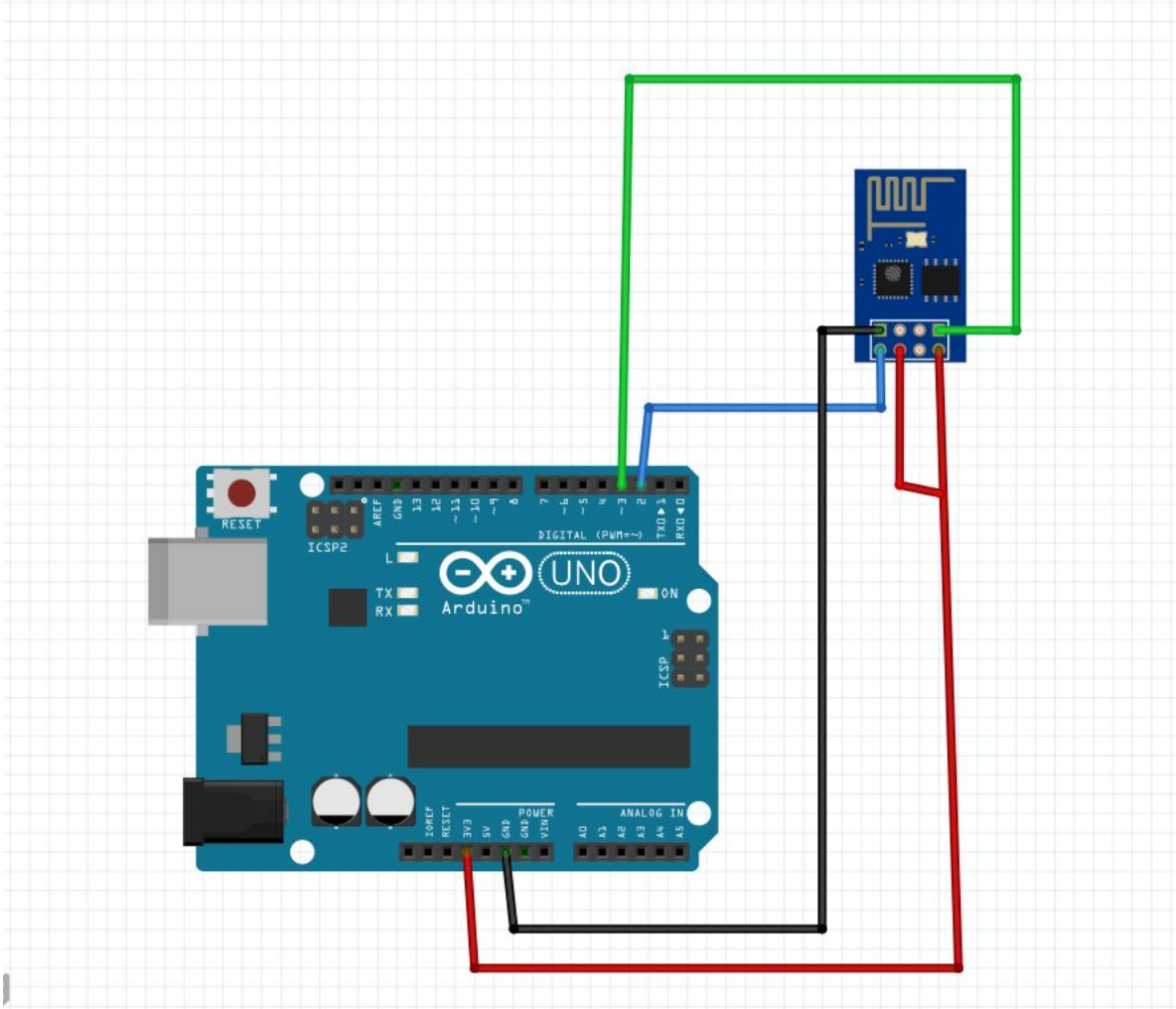
b. ESP-01 code (also pending of minor changes):

```
#include <ESP8266WiFi.h>
#include <espnow.h>

// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
        int id;
        float x;
        float y;
```

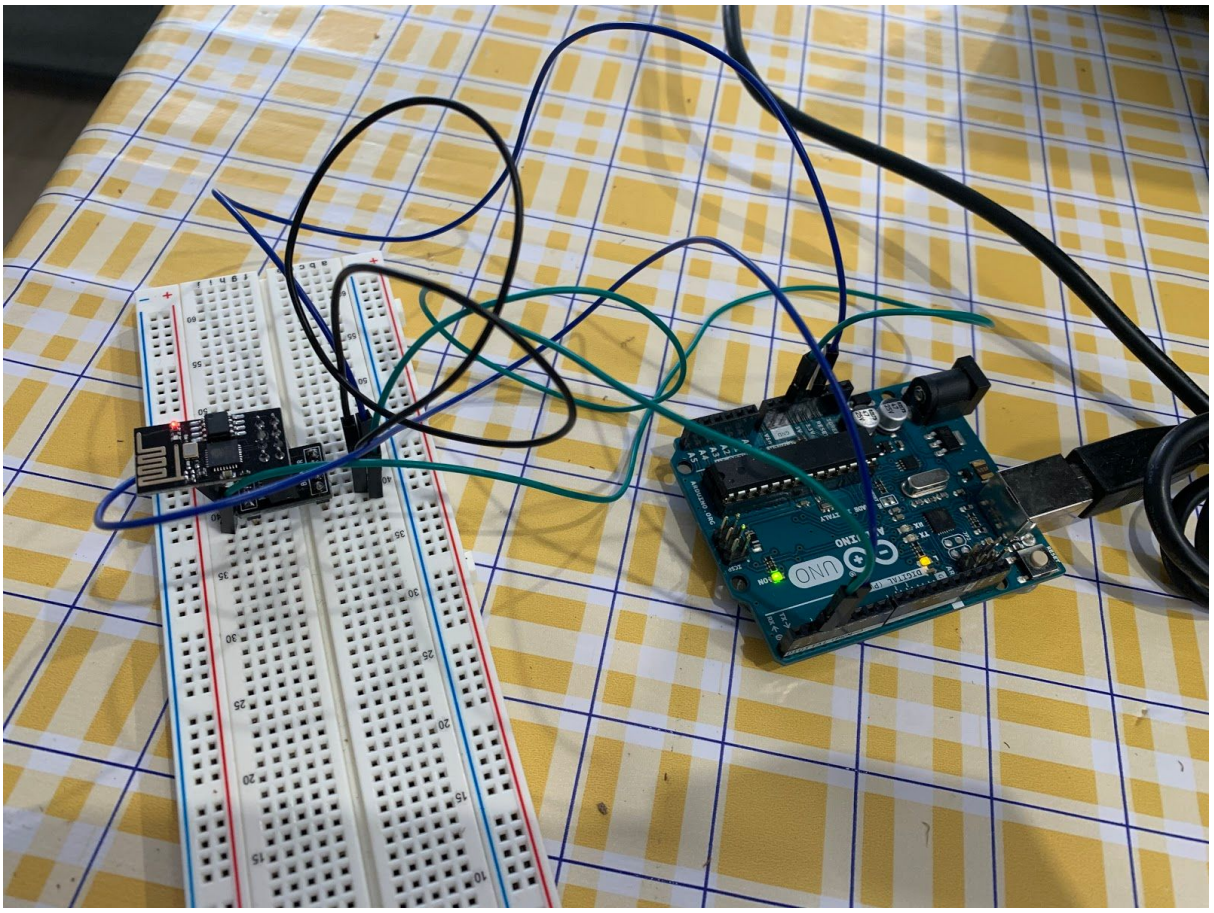
```
} struct_message;

// Create a struct_message called myData
struct_message myData;

// Create a structure to hold the readings from each board
struct_message board1;
struct_message board2;

// Create an array with all the structures
struct_message boardsStruct[2] = {board1, board2};

unsigned long lastTime = 0;
unsigned long timerDelay = 10000;

float board1Distance = 0.0;
float board2Temp = 0.0;
float board2Hum = 0.0;

// Callback function that will be executed when data is received
void OnDataRecv(uint8_t * mac_addr, uint8_t *incomingData, uint8_t len) {
  char macStr[18];
  //Serial.print("Packet received from: ");
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
        mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  //Serial.println(macStr);
  memcpy(&myData, incomingData, sizeof(myData));
  //Serial.printf("Board ID %u: %u bytes\n", myData.id, len);
  // Update the structures with the new incoming data
  boardsStruct[myData.id-1].x = myData.x;
  boardsStruct[myData.id-1].y = myData.y;
  boardsStruct[myData.id-1].id = myData.id;
  //Serial.printf("x value: %f \n", boardsStruct[myData.id-1].x);
  //Serial.printf("y value: %f \n", boardsStruct[myData.id-1].y);
  //Serial.println();
}

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();

  // Init ESP-NOW
  if (esp_now_init() != 0) {
        Serial.println("Error initializing ESP-NOW");
        return;
  }

  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_set_self_role(ESP_NOW_ROLE_SLAVE);
  esp_now_register_recv_cb(OnDataRecv);
}

void loop(){
   // loop
  if ((millis() - lastTime) > timerDelay) {
        // Access the variables for each board

        board1Distance = boardsStruct[0].x;
        float test = boardsStruct[0].y;
```

```
        board2Temp = boardsStruct[1].x;
        board2Hum = boardsStruct[1].y;



        // Send the information to the arduino.
        Serial.print("<DataProducers,");
        Serial.print(board2Temp);
        Serial.print(",");
        Serial.print(board2Hum);
        Serial.print(">");
        //
        Serial.print(board1Distance);
        Serial.print(",");
        Serial.print(test);
        Serial.print("\n");

        lastTime = millis();
  }
}
```

3. Electrical connections:




4. Building the setup:



5. Testing
    a. In order to test it, we need to implement at least another ESP-01 as data sender using the ESPNow, in this case, the test was made using the Dataproducer 2(DHT11 sensor) providing the information via ESPNow.

```
COM3
:31:50.266 -> Humidity: 39.00
:32:00.274 -> Message: DataProducer1
:32:00.274 -> Temperature: 20.00
:32:00.274 -> Humidity: 39.00
:32:10.276 -> Message: DataProducer1
:32:10.276 -> Temperature: 19.00
:32:10.276 -> Humidity: 38.00
:32:20.250 -> Message: DataProducer1
:32:20.250 -> Temperature: 20.00
:32:20.250 -> Humidity: 38.00
:32:30.272 -> Message: DataProducer1
:32:30.272 -> Temperature: 20.00
:32:30.272 -> Humidity: 38.00
:32:40.268 -> Message: DataProducer1
:32:40.268 -> Temperature: 20.00
:32:40.268 -> Humidity: 38.00
:32:50.264 -> Message: DataProducer1
:32:50.264 -> Temperature: 20.00
:32:50.264 -> Humidity: 39.00
:33:00.274 -> Message: DataProducer1
:33:00.274 -> Temperature: 20.00
:33:00.274 -> Humidity: 39.00
:33:10.251 -> Message: DataProducer1
:33:10.251 -> Temperature: 20.00
:33:10.285 -> Humidity: 39.00
:33:20.271 -> Message: DataProducer1
:33:20.271 -> Temperature: 20.00
:33:20.271 -> Humidity: 39.00
Autoscroll  Show timestamp
```

```
ESP_NOW_Receiver_modified_ARDUINO.ino

//============

void parseData() {      // split the data into its parts

    char * strtokIndx; // this is used by strtok() as an index

    strtokIndx = strtok(tempChars,",");      // get the first part - the string
    strcpy(message, strtokIndx); // copy it to messageFromPC

    strtokIndx = strtok(NULL, ","); // this continues where the previous call left off
    floatTemp = atof(strtokIndx);     // convert this part to a float

    strtokIndx = strtok(NULL, ",");
    floatHum = atof(strtokIndx);     // convert this part to a float

}

//============

void showParsedData() {
    Serial.print("Message: ");
    Serial.println(message);
    Serial.print("Temperature: ");
    Serial.println(floatTemp);
    Serial.print("Humidity: ");
    Serial.println(floatHum);
}
```


# 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:

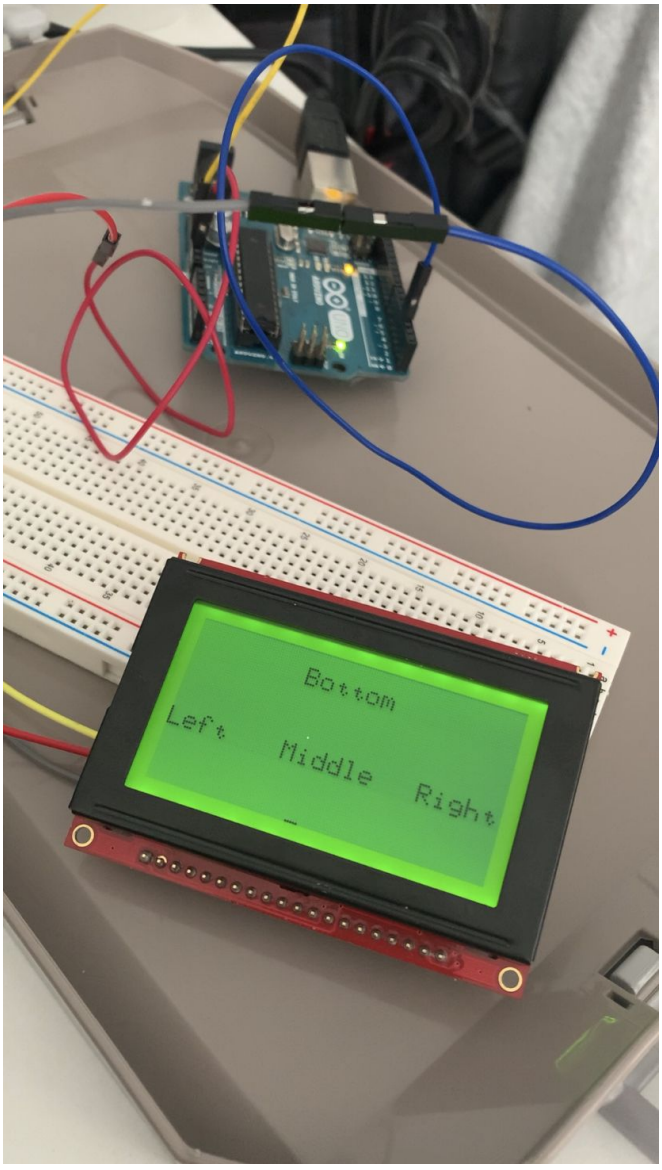
```
#include <SparkFunSerialGraphicLCD.h> //include the Serial Graphic LCD library
#include <SoftwareSerial.h>

#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).

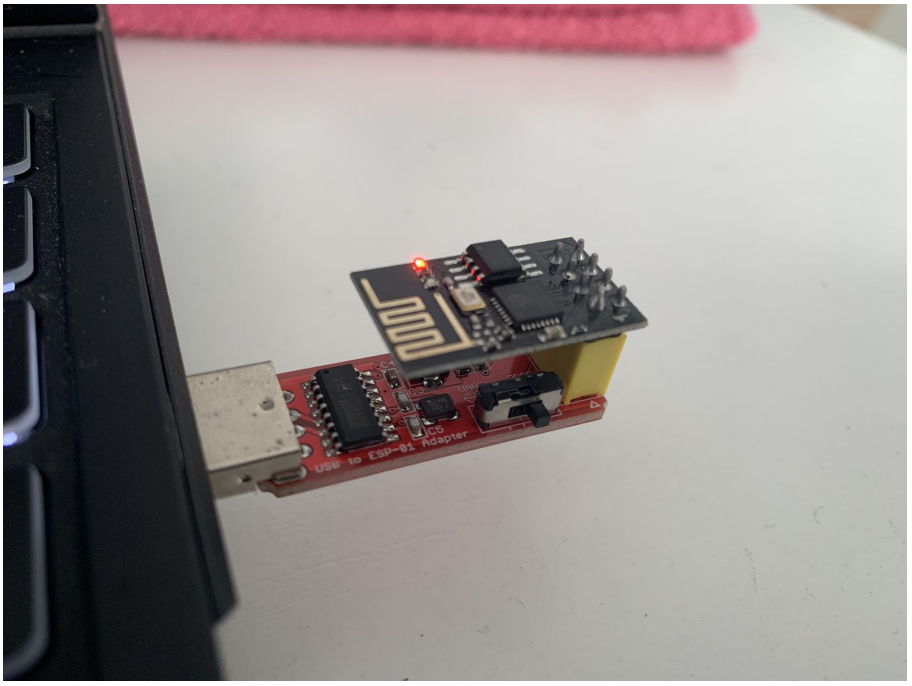
1. On the adapter, switch to the PROG mode (the other is UART)
2. Connect the adapter on computer and install its drivers:
   USB Adapter Driver: http://www.wch.cn/download/CH341SER_EXE.html
3. After that, you can connect your ESP-01 on it and plug on the computer
4. In the Arduino IDE go Tools > Board > 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:

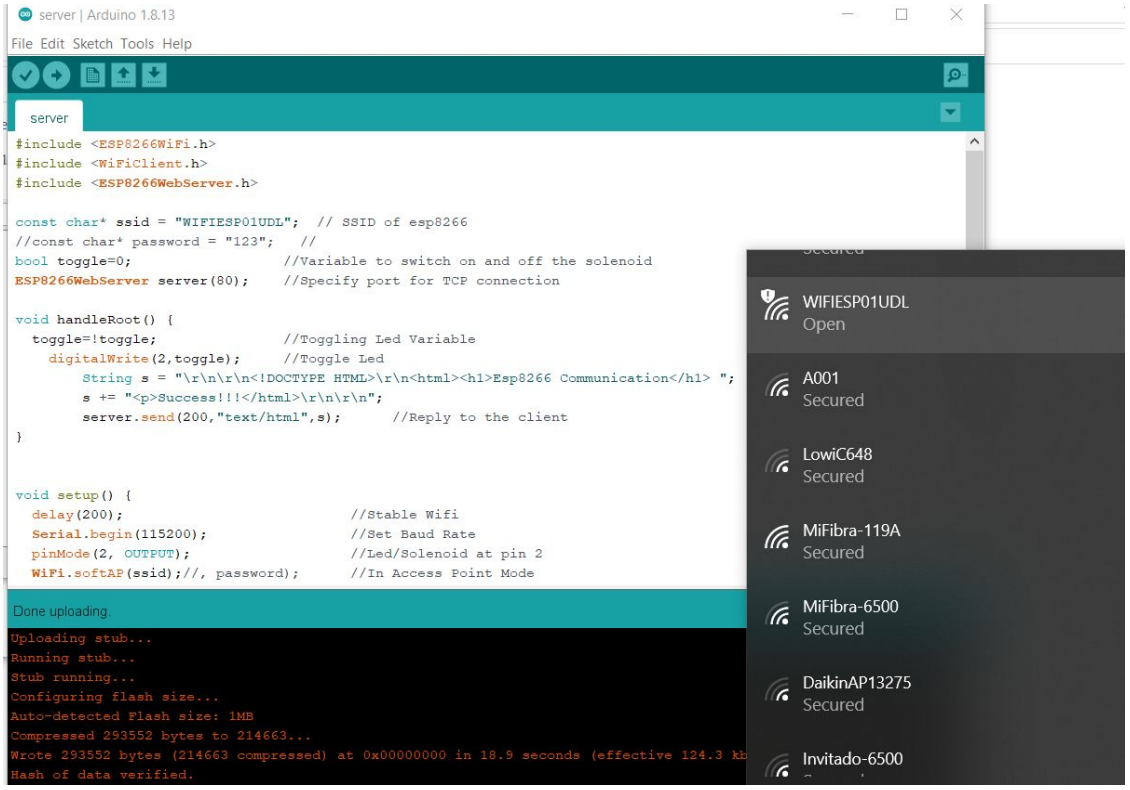
b. https://learn.sparkfun.com/tutorials/esp8266-thing-hookup-guide/example-sketch-blink

## 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:






**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");
}
```

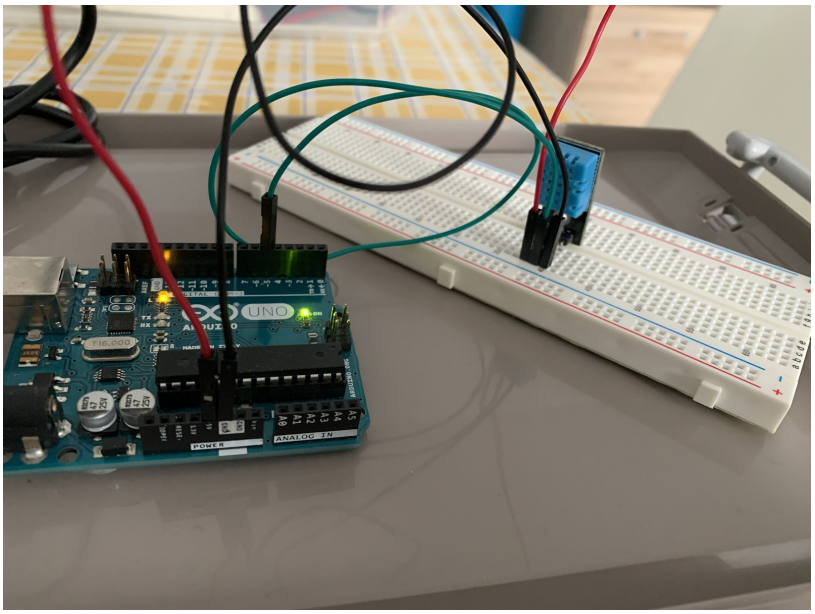
## Electrical Wiring:

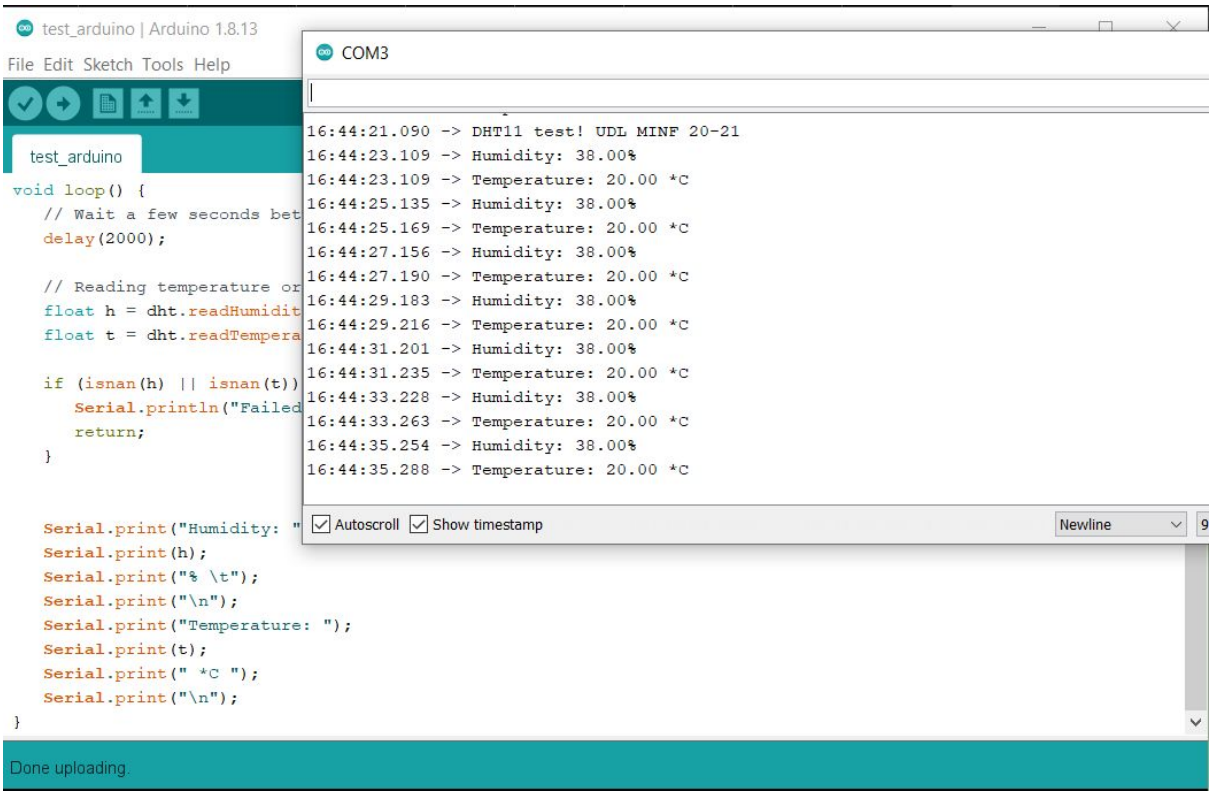
DHT11 Vcc -> Arduino 5V

DHT11 Output ->one of the digital pins of Arduino (5 per example)

DHT11 GND ->Arduino GND (its on the side of the 5V)

## Photos:





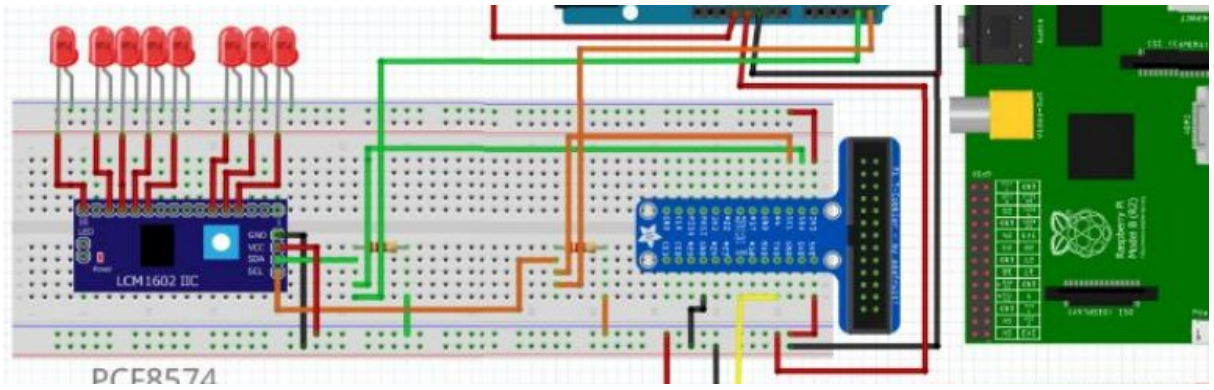

**Conclusion**: This test showed us that the sensor is functioning correctly and showing the values as it should.

### 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:

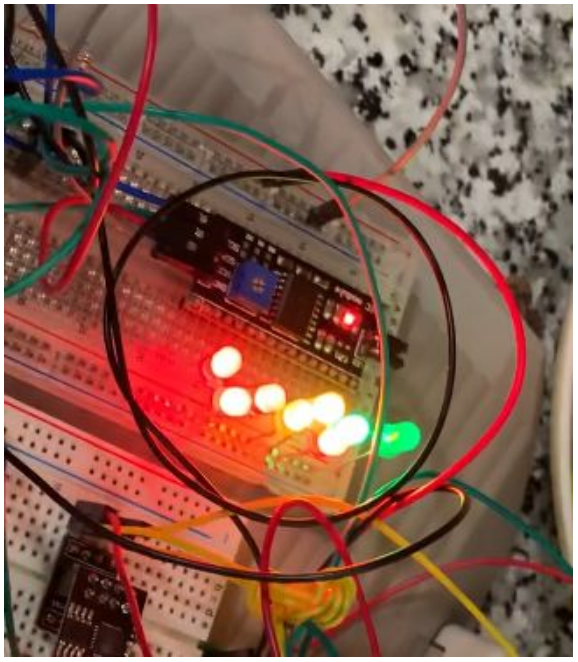



The address of the PCF8574 in the i2c connection is: 0x27

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.

Universitat de Lleida

# Data Producer 1 development
# Embedded and Ubiquitous systems
# Task part of the second sprint
# UDL - MINF - 2021 - Team 1

Reference guide: https://randomnerdtutorials.com/esp-now-many-to-one-esp8266-nodemcu/

1. The purpose of this document is to explain the process in developing the data producer 2.
   For this project we are using a library called ESPNOW that facilitates the communication between multiple ESP-01. In this case, we are gonna use a system of 1 receiver / multiple senders, better said: ESP-01 on Arduino is the receiver, and the 2 ESP-01 of the data producers are senders.
   a. That said, we have this setup:
   b. Arduino-ESP-01 slave (Receiver)
   c. Data producer 1 ESP-01 master (ESP+Ultrasonic sensor) Board #1 (Sender)
   d. Data producer 2 ESP-01 master (ESP+Temperature sensor) Board #2 (Sender)

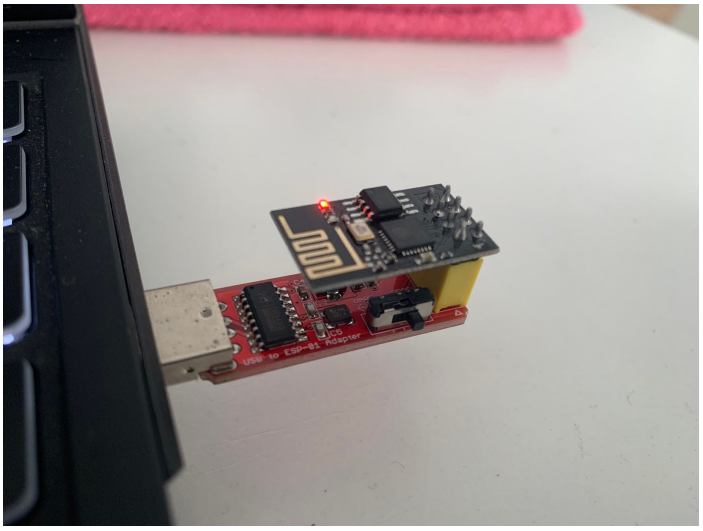
2. It's important to know in which ESP you are working, to adjust the code. So let's start on the Data Producer 2.
3. Open the Arduino IDE, select the board to Generic ESP 8266 Module and the correct port.
   The first step is to get the MAC Address of the receiver ESP-01, which means the ESP-01 that you are gonna use in the main part of the project (remember, the receiver), you need to execute this code on it to discover its Mac Address (See step 4 of how connect the ESP-01 to the computer to compile on it):

```
#include <ESP8266WiFi.h>
#endif

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

```
   Serial.println();
   Serial.print("ESP Board MAC Address:  ");

}

void loop(){
   Serial.println(WiFi.macAddress());
}
```

Don’t forget to disconnect the Adapter after compiling, change the switch to UART, and insert it again! Then your program should run.
Open the COM monitor on the Baud rate 115200 to see your mac address, and store it somewhere.
This part is essential because the library needs the mac address of the receiver to send data.

4. With this MAC Address in hand you can start programming the code into the ESP-01 of the Data Producer 1, for doing this we are using the ESP to USB adapter in the Switch in PROG:



5. Open the Arduino IDE, select the board to Generic ESP 8266 Module and to the correct port.
6. In the following code you will need to change this parameter with the MAC Address that you got from your receiver ESP-01:

```
// REPLACE WITH RECEIVER MAC Address (ESP-01 that is connected with the Arduino)
      uint8_t broadcastAddress[] = {0xA4, 0xCF, 0x12, 0xBF, 0x15, 0xEE};
```

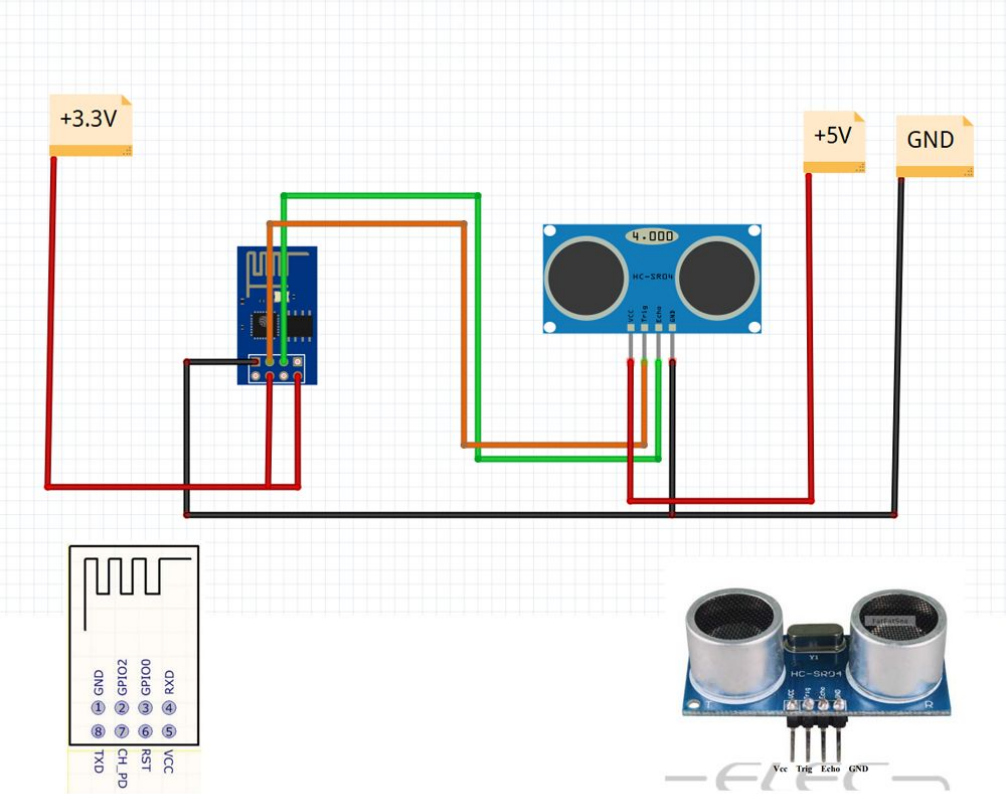
7. Code (also avaliable on Github):

```
#include <ESP8266WiFi.h>
#include <espnow.h>

// REPLACE WITH RECEIVER MAC Address (ESP-01 thats connected with the Arduino)
uint8_t broadcastAddress[] = {0xA4, 0xCF, 0x12, 0xBF, 0x15, 0xEE};

// pins of the esp8266 connected to the sensor
#define trigPin 2  //GPIO2
#define echoPin 0  //GPIO0

// intialize the variables for storing the measurement
float timeElapsed;
float distance;

// Set your Board ID (ESP32 Sender #1 = BOARD_ID 1, ESP32 Sender #2 = BOARD_ID 2, etc)
#define BOARD_ID 1

// Structure to send data
// Must match the receiver structure
typedef struct struct_message {
        int id;
        float x;
        float y;
} struct_message;

// Create a struct_message called test to store variables to be sent
struct_message myData;

unsigned long lastTime = 0;
unsigned long timerDelay = 10000;

// Callback when data is sent
void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("\r\nLast Packet Send Status: ");
  if (sendStatus == 0){
        Serial.println("Delivery success");
  }
  else{
        Serial.println("Delivery fail");
  }
}

void setup() {
  // Init Serial Monitor
  Serial.begin(115200);

  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();

  // Init ESP-NOW
  if (esp_now_init() != 0) {
        Serial.println("Error initializing ESP-NOW");
        return;
  }
  // Set ESP-NOW role
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);

  // Once ESPNow is successfully init, we will register for Send CB to
```

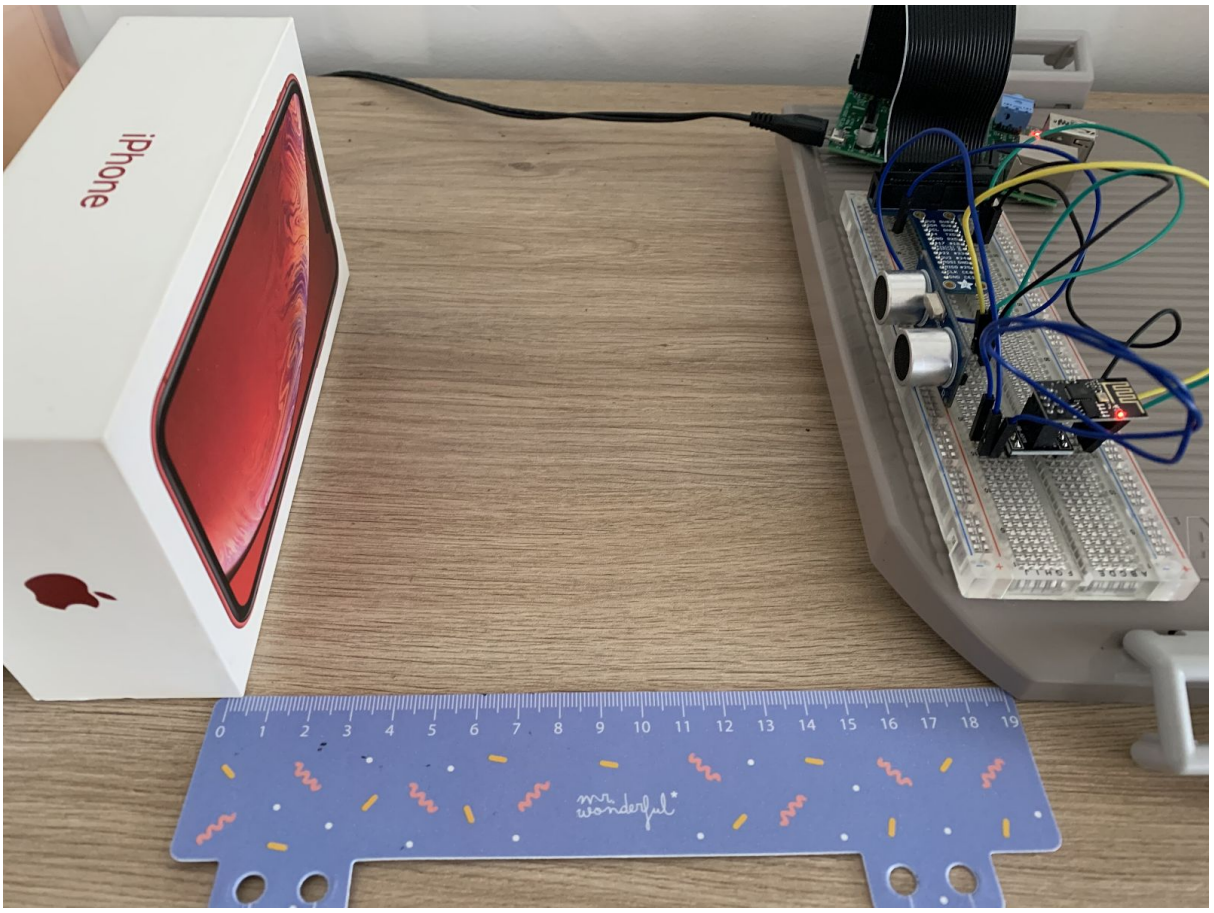
```
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);

}

void loop() {
  // do the measurements of the sensor
   digitalWrite(trigPin, LOW);  //para generar un pulso limpio ponemos a LOW 4us
   delayMicroseconds(4);

   digitalWrite(trigPin, HIGH);  //generamos Trigger (disparo) de 10us
   delayMicroseconds(10);
   digitalWrite(trigPin, LOW);

   timeElapsed = pulseIn(echoPin, HIGH);
   distance = timeElapsed/58.3;

   delay(1000);

  if ((millis() - lastTime) > timerDelay) {
        // Set values to send
        myData.id = BOARD_ID;
        myData.x = distance;
        myData.y = timeElapsed;

        // Send message via ESP-NOW
        esp_now_send(0, (uint8_t *) &myData, sizeof(myData));
        lastTime = millis();
  }
}
```

Now the code for your ESP-01 is done, the next step is to build the electrical connections with the sensor and power supply.

8. Connections



It should look like this (in this case i am using the raspberry pi as power supplly (5V, 3.3V and GND):




Unfortunately, in order to test it you will need to also setup the Receiver ESP-01 using the ESP-Now code to receive the measurements.

Universitat de Lleida

# Data Producer 2 development
# Embedded and Ubiquitous systems
# Task part of the second sprint
# UDL - MINF - 2021 - Team 1

Reference guide: https://randomnerdtutorials.com/esp-now-many-to-one-esp8266-nodemcu/

1. The purpose of this document is to explain the process in developing the data producer 2.
   For this project we are using a library called ESPNOW that facilitates the communication between multiple ESP-01. In this case, we are gonna use a system of 1 receiver / multiple senders, better said: ESP-01 on Arduino is the receiver, and the 2 ESP-01 of the data producers are senders.
   a. That said, we have this setup:
   b. Arduino-ESP-01 slave (Receiver)
   c. Data producer 1 ESP-01 master (ESP+Ultrasonic sensor) Board #1 (Sender)
   d. Data producer 2 ESP-01 master (ESP+Temperature sensor) Board #2 (Sender)

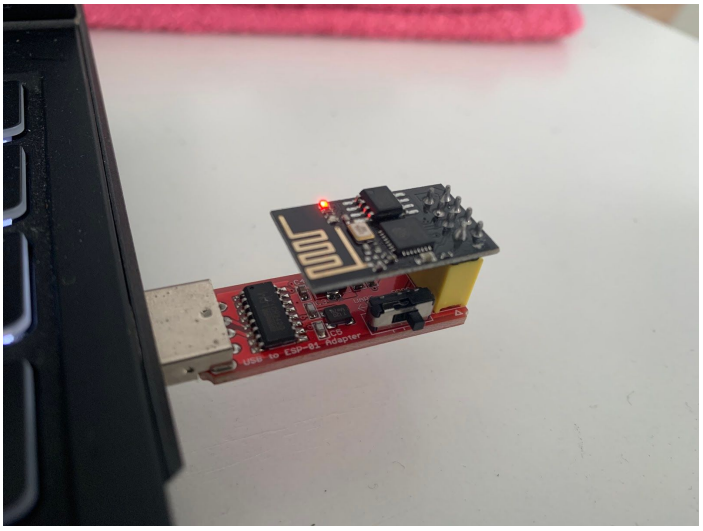
2. It's important to know in which ESP you are working, to adjust the code. So let's start on the Data Producer 2.
3. Open the Arduino IDE, select the board to Generic ESP 8266 Module and the correct port.
   The first step is to get the MAC Address of the receiver ESP-01, which means the ESP-01 that you are gonna use in the main part of the project (remember, the receiver), you need to execute this code on it to discover its Mac Address (See step 4 of how connect the ESP-01 to the computer to compile on it):

```
#include <ESP8266WiFi.h>
#endif

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

```
   Serial.println();
   Serial.print("ESP Board MAC Address:  ");

}

void loop(){
   Serial.println(WiFi.macAddress());
}
```

Don’t forget to disconnect the Adapter after compiling, change the switch to UART, and insert it again! Then your program should run.
Open the COM monitor on the Baud rate 115200 to see your mac address, and store it somewhere.
This part is essential because the library needs the mac address of the receiver to send data.

4. With this MAC Address in hand you can start programming the code into the ESP-01 of the Data Producer 2, for doing this we are using the ESP to USB adapter in the Switch in PROG:



5. Open the Arduino IDE, select the board to Generic ESP 8266 Module and to the correct port.
6. In the following code you will need to change this parameter with the MAC Address that you got from your receiver ESP-01:

```
// REPLACE WITH RECEIVER MAC Address (ESP-01 that is connected with the Arduino)
uint8_t broadcastAddress[] = {0xA4, 0xCF, 0x12, 0xBF, 0x15, 0xEE};
```

7. Code (also avaliable on Github):

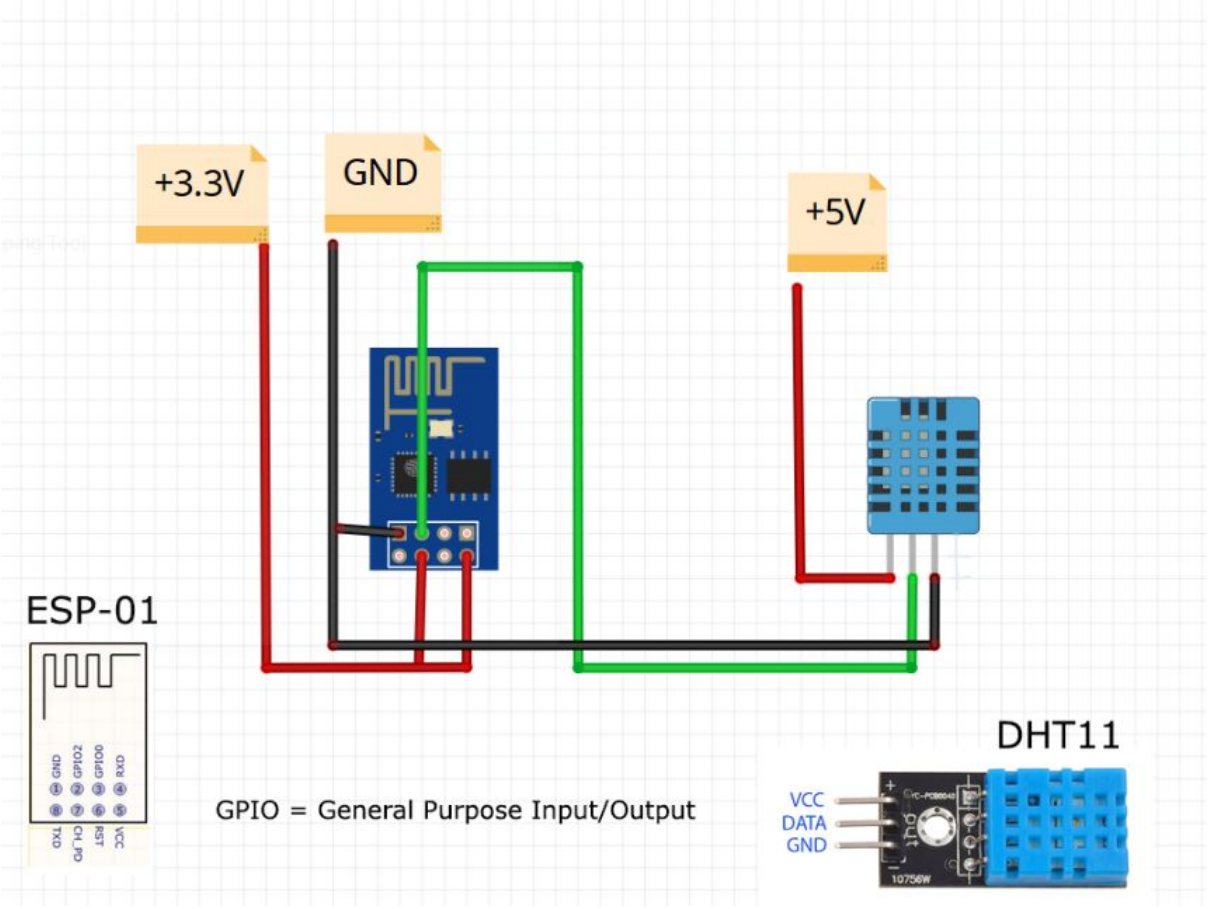
```
#include <ESP8266WiFi.h>
#include <espnow.h>
```

```
#include "DHT.h"

#define DHTTYPE DHT11   // DHT 11

// REPLACE WITH RECEIVER MAC Address (ESP-01 thats connected with the Arduino)
uint8_t broadcastAddress[] = {0xA4, 0xCF, 0x12, 0xBF, 0x15, 0xEE};
// We say the sensor is connected on the pin 2
uint8_t DHTPin = 2;

// Initialize the sensor
DHT dht(DHTPin, DHTTYPE);

float Temperature;
float Humidity;

// Set your Board ID (ESP32 Sender #1 = BOARD_ID 1, ESP32 Sender #2 = BOARD_ID 2, etc)
#define BOARD_ID 2

// Structure example to send data
// Must match the receiver structure
typedef struct struct_message {
        int id;
        float x;
        float y;
} struct_message;

// Create a struct_message called test to store variables to be sent
struct_message myData;

unsigned long lastTime = 0;
unsigned long timerDelay = 10000;

// Callback when data is sent
void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("\r\nLast Packet Send Status: ");
  if (sendStatus == 0){
        Serial.println("Delivery success");
  }
  else{
        Serial.println("Delivery fail");
  }
}

void setup() {
  // Init Serial Monitor
  Serial.begin(115200);

  pinMode(DHTPin, INPUT);
  dht.begin();

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();

  // Init ESP-NOW
  if (esp_now_init() != 0) {
        Serial.println("Error initializing ESP-NOW");
        return;
  }
  // Set ESP-NOW role
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);

  // Once ESPNow is successfully init, we will register for Send CB to
  // get the status of Transmitted packet
```

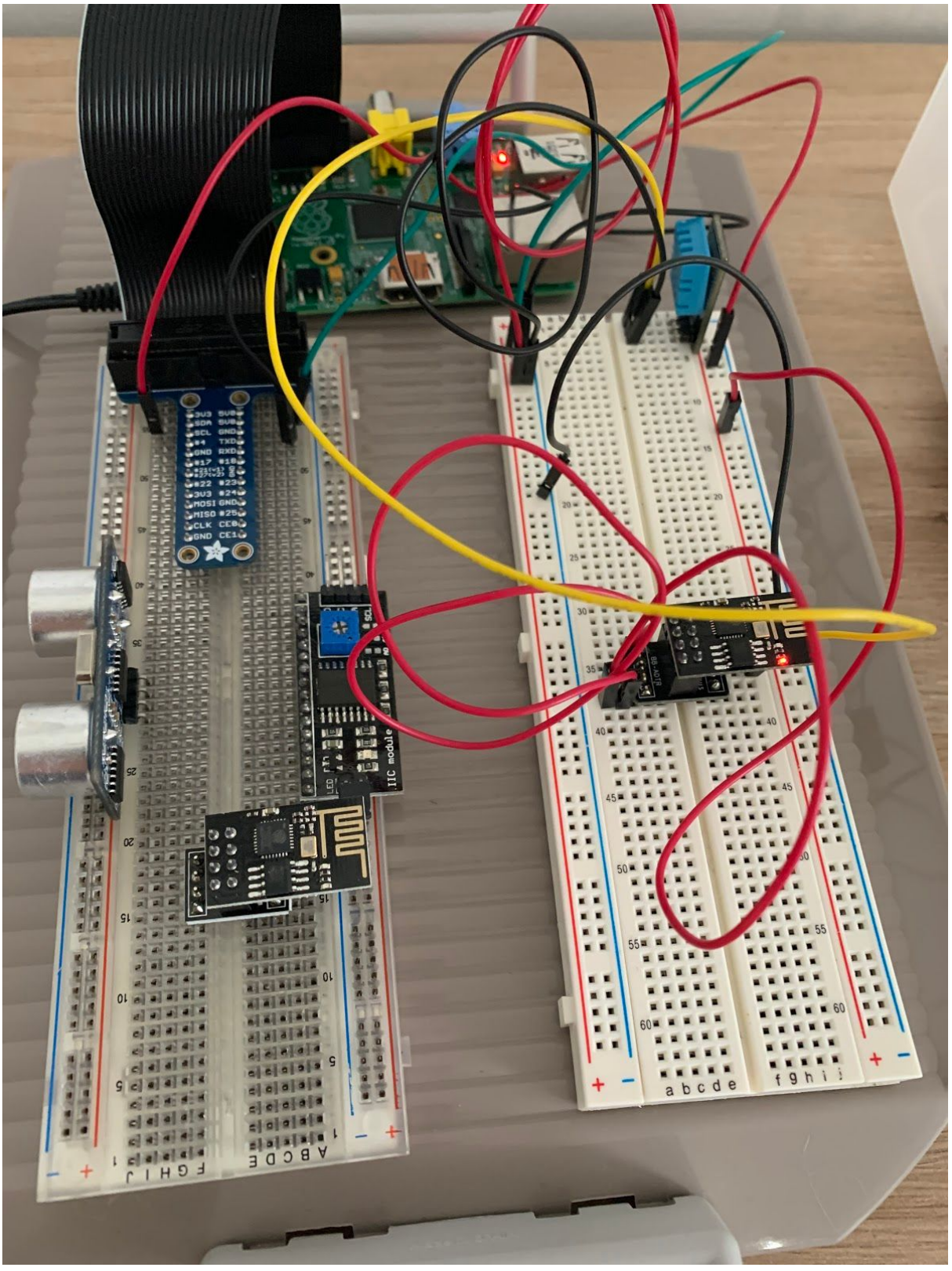
```
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);

}

void loop() {
  if ((millis() - lastTime) > timerDelay) {
        Temperature = dht.readTemperature(); // Gets the values of the temperature
        Humidity = dht.readHumidity(); // Gets the values of the humidity

        // Set values to send
        myData.id = BOARD_ID;
        myData.x = Temperature;
        myData.y = Humidity;

        // Send message via ESP-NOW
        esp_now_send(0, (uint8_t *) &myData, sizeof(myData));
        lastTime = millis();
  }
```

Now the code for your ESP-01 is done, the next step is to build the electrical connections with the sensor and power supply.

8. Connections




It should look like this (in this case i am using the raspberry pi as power supplly (5V, 3.3V and GND):




Unfortunately, in order to test it you will need to also setup the Receiver ESP-01 using the ESP-Now code.

Universitat de Lleida

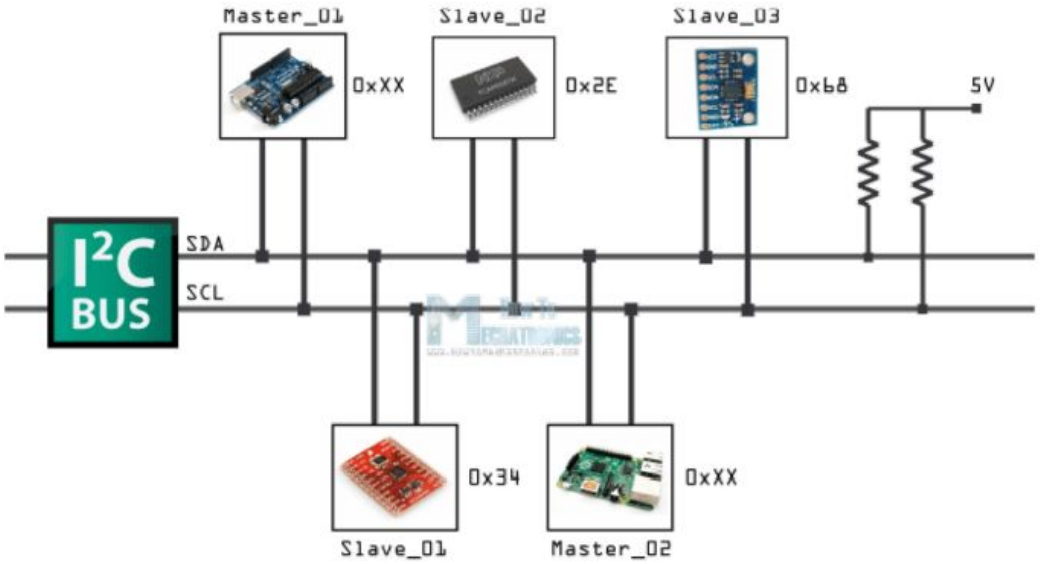
# I2c <-> Arduino documentation (Sprint 3)

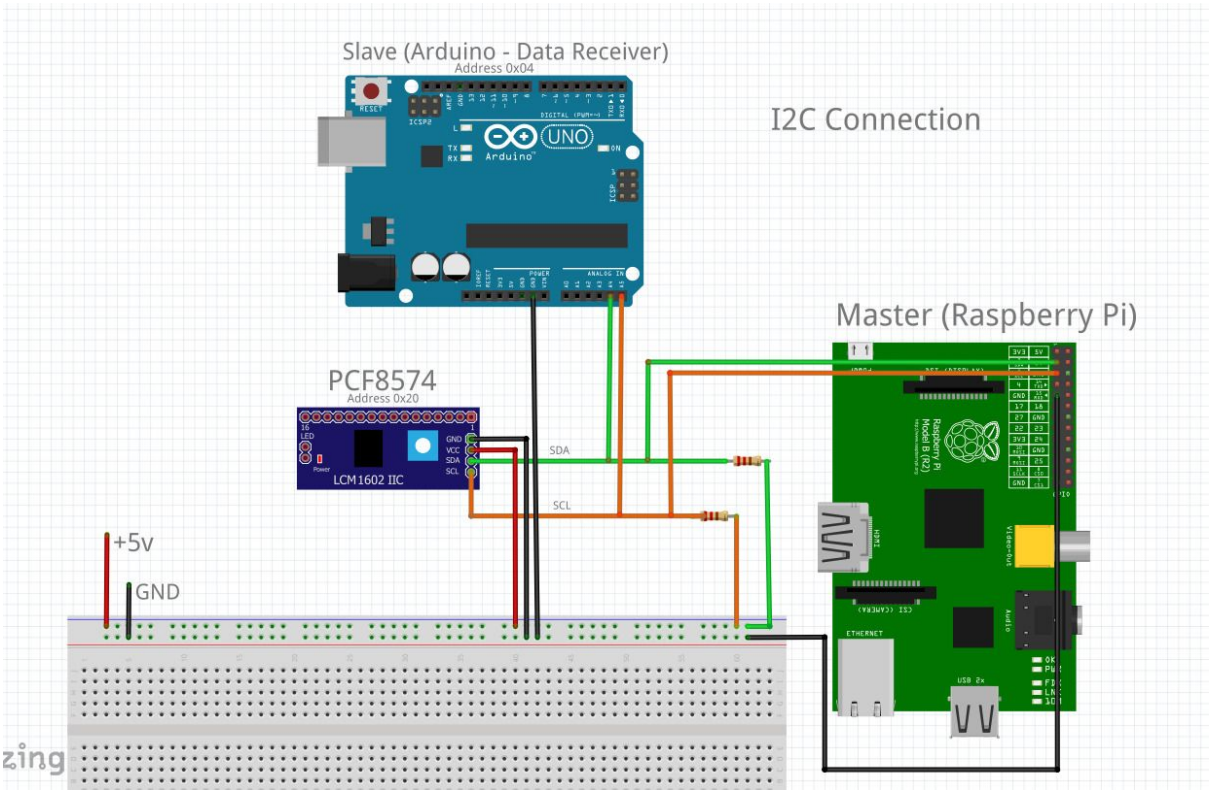
MINF UDL 20-21

Ubiquitous and embedded systems

Team 1

1. **Use Wire Library (Arduino)**
2. **Wiring connection**






Arduino A4 <> SDA

Arduino A5 <> SCL

Raspberry SDA (Pin 2) <> SDA

Raspberry SCL (Pin 3) <> SCL

GND (All) <> GND (All)

### 3. Code (Arduino)

a. Initializing Arduino as a Slave and sending information (3 floats) via the sendHandler function.

b. We initialize the i2c bus with the `Wire.begin(I2C_ADDR);`

c. Then we answer to the master's i2c request with the function `Wire.onRequest(sendData_handler);`

d. After that the data is sent (collected by the ESP receiver) `Wire.write((byte *) sensorData, sizeof sensorData);`

```
#include <SoftwareSerial.h>
#include <Wire.h>
#define I2C_ADDR 0x04

uint8_t data;

SoftwareSerial mySerial(2, 3); // RX, TX
```

```
// Variables to handle the data from the ESP
const byte numChars = 64;
char receivedChars[numChars];
char tempChars[numChars];        // temporary array for use when
parsing

// variables to hold the parsed data
char message[numChars] = {0};
float floatTemp = 0.0;
float floatHum = 0.0;
float floatDistance = 0.0;
float floatTime = 0.0;
char distanceBuffer[7];
char tempBuffer[7];
char humBuffer[7];
float sensorData[3];

boolean newData = false;

void setup() {
  // put your setup code here, to run once:
  Wire.begin(I2C_ADDR);

  Serial.begin(115200);
  mySerial.begin(115200);

  Wire.onRequest(sendData_handler);
  delay(5000);
}

// Enter data in this style <HelloWorld, 12, 24.7>

void loop() {
    recvWithStartEndMarkers();
    if (newData == true) {
        strcpy(tempChars, receivedChars);
            // this temporary copy is necessary to protect the
original data
            //   because strtok() used in parseData() replaces the
commas with \0
        parseData();
        showParsedData();
```

```
        newData = false;
    }
}

//============

void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

    while (mySerial.available() > 0 && newData == false) {
        rc = mySerial.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= numChars) {
                    ndx = numChars - 1;
                }
            }
            else {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                newData = true;
            }
        }

        else if (rc == startMarker) {
            recvInProgress = true;
        }
    }
}

//============

void parseData() {      // split the data into its parts
```

```
    char * strtokIndx; // this is used by strtok() as an index

    strtokIndx = strtok(tempChars,",");      // get the first part -
the string
    strcpy(message, strtokIndx); // copy it to

    strtokIndx = strtok(NULL, ","); // this continues where the
previous call left off
    floatDistance = atof(strtokIndx);     // convert this part to a
float

    strtokIndx = strtok(NULL, ",");
    floatTime = atof(strtokIndx);     // convert this part to a float

    strtokIndx = strtok(NULL, ","); // this continues where the
previous call left off
    floatTemp = atof(strtokIndx);     // convert this part to a float

    strtokIndx = strtok(NULL, ",");
    floatHum = atof(strtokIndx);     // convert this part to a float

}

//============

void showParsedData() {
    Serial.print("Message: ");
    Serial.println(message);
    Serial.print("Distance (HC-SR04): ");
    Serial.println(floatDistance);
    Serial.print("Time (HC-SR04): ");
    Serial.println(floatTime);
    Serial.print("Temperature (DHT11): ");
    Serial.println(floatTemp);
    Serial.print("Humidity (DHT11): ");
    Serial.println(floatHum);
}

void sendData_handler (){
  sensorData[0] = floatDistance;
  sensorData[1] = floatTemp;
  sensorData[2] = floatHum;
```

```
  Wire.write((byte *) sensorData, sizeof sensorData);

  delay(100);

}
```

4. **Code (Raspberry)**

```
#include "ch.h"
#include "hal.h"
#include "chprintf.h"

static const uint8_t arduino_address = 0x04;
static const uint8_t pcf_address = 0x27;

static WORKING_AREA(waThread_I2C, 128);
static msg_t Thread_I2C(void *p)
{
  (void)p;
  chRegSetThreadName("SerialPrintI2C");
  uint8_t request[] = {0, 0};
  uint8_t result[3];
  uint8_t aux;
  uint8_t bit = 0b10000000;

  msg_t status;

  // Some time to allow slaves initialization
  chThdSleepMilliseconds(2000);

  while (TRUE)
  {

    // Request values
    i2cMasterTransmit(&I2C0, arduino_address, request, 2,
                      &result, 3);

    // Attempt to send the bits to the pcf
    i2cMasterTransmit(&I2C0, pcf_address, &bit, sizeof(bit),
                      &aux, 0);
```

```
    chThdSleepMilliseconds(10);

    sdPut(&SD1, (int8_t)0x7C);
    sdPut(&SD1, (int8_t)0x18);
    sdPut(&SD1, (int8_t)0x00);
    chThdSleepMilliseconds(10);

    sdPut(&SD1, (int8_t)0x7C);
    sdPut(&SD1, (int8_t)0x19);
    sdPut(&SD1, (int8_t)0x20);
    chThdSleepMilliseconds(10);

    chprintf((BaseSequentialStream *)&SD1, "Data: %u %u %u",
result[0], result[1], result[2]);

    request[1]++;
    if (request[1] > 10)
    {
      request[1] = 0;
      request[0]++;
    }

    chThdSleepMilliseconds(2000);
  }
  return 0;
}

int main(void)
{
  halInit();
  chSysInit();

  // Initialize Serial Port
  sdStart(&SD1, NULL);

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

```
  chThdCreateStatic(waThread_I2C, sizeof(waThread_I2C),
                    HIGHPRIO, Thread_I2C, NULL);

  // Blocks until finish
  chThdWait(chThdSelf());

  return 0;
}
```

5. Considerations:
    a.

Universitat de Lleida

LCD <> ChibiOS communication (Sprint 3)

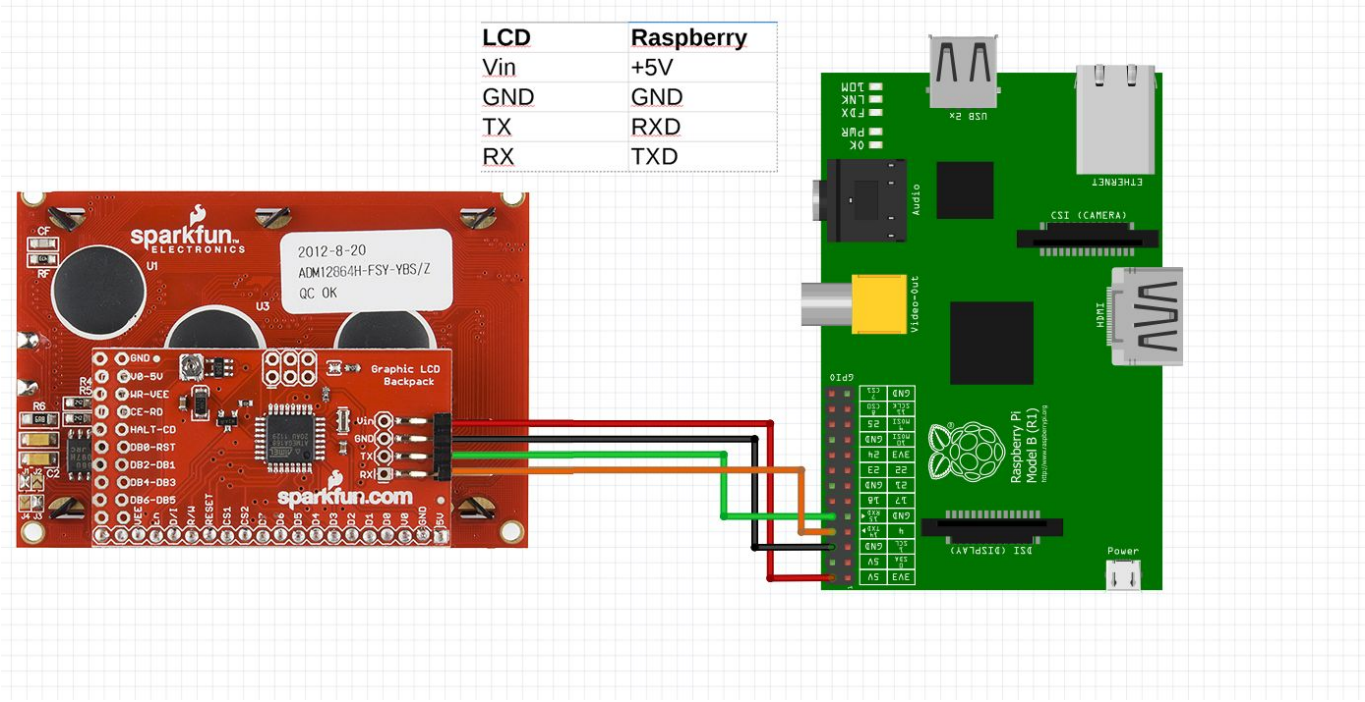
MINF UDL 20-21

Ubiquitous and embedded systems

Team 1

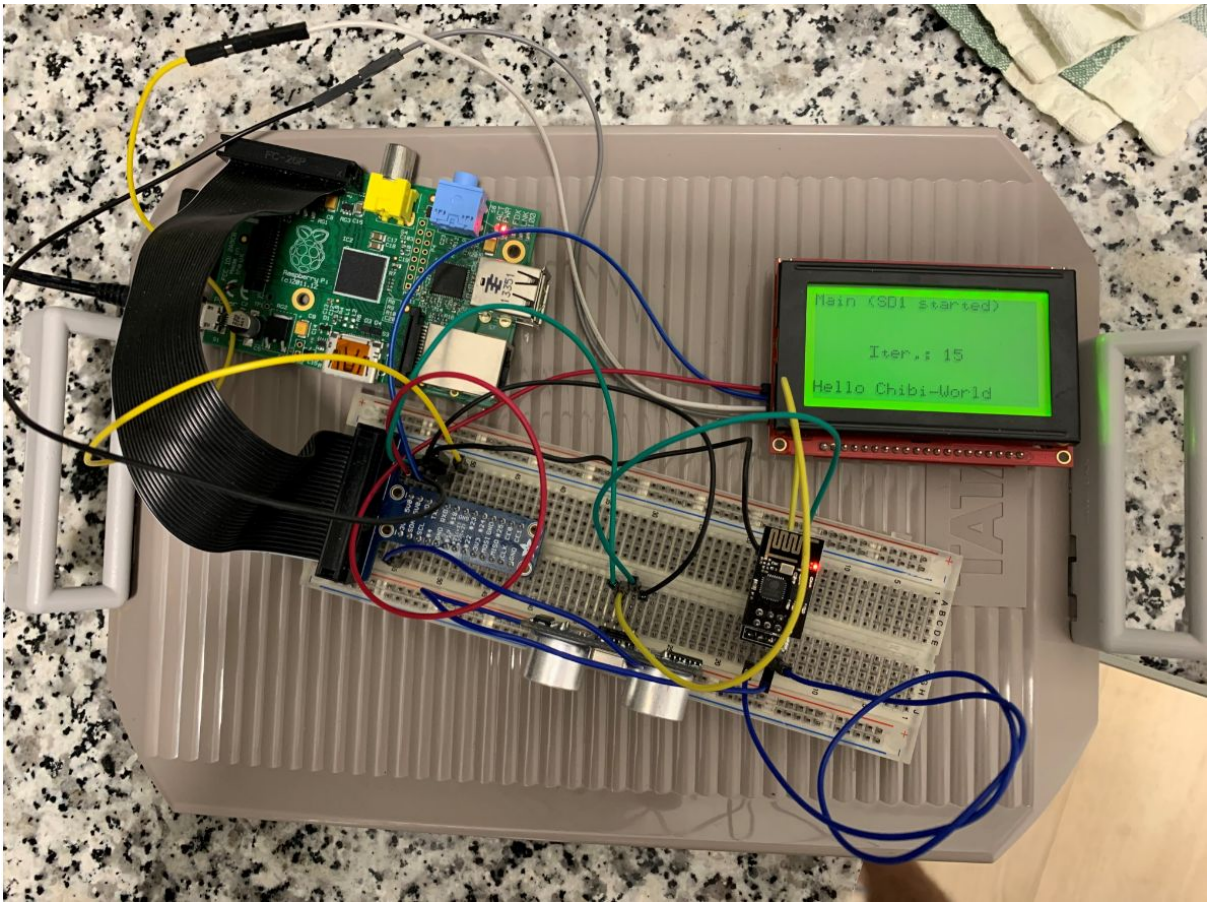
**Objective:** Make the chibiOS works with the LCD Screen, because this feature will be needed in the final sprint (data representation).

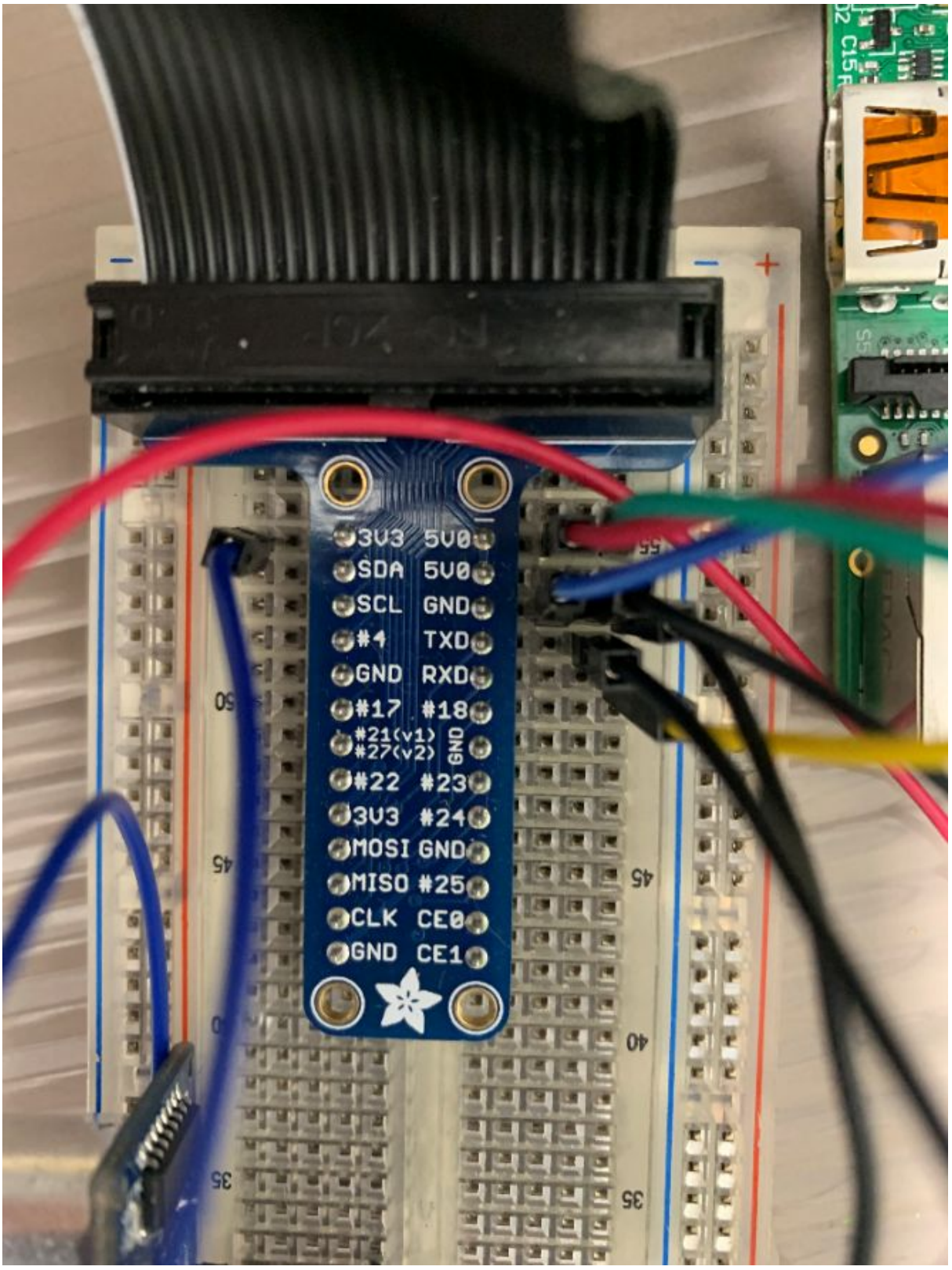
## 1. Wiring connection:




Connect the pins carefully to not invert the TX and RX.

2. **It should look like this:**
    a. (In my case the Data Producer 1 is also connected to the Raspberry, please ignore it)




3V3 5V0
SDA 5V0
SCL GND
#4 TXD
GND RXD
#17 #18
#21(v1) #27(v2) GND
#22 #23
3V3 #24
MOSI GND
MISO #25
CLK CE0
GND CE1

## 3. Code:

a. In this case, we are using a code example as base to build our final code, this code will print some strings to the LCD and also an iteration.

b. The communication goes like this: We are using the driver SerialDriver of the ChibiOS to communicate with the LCD Screen, using the SD1 which means the UART pins of Raspberry (In our case, is the TXD and RXD pins).

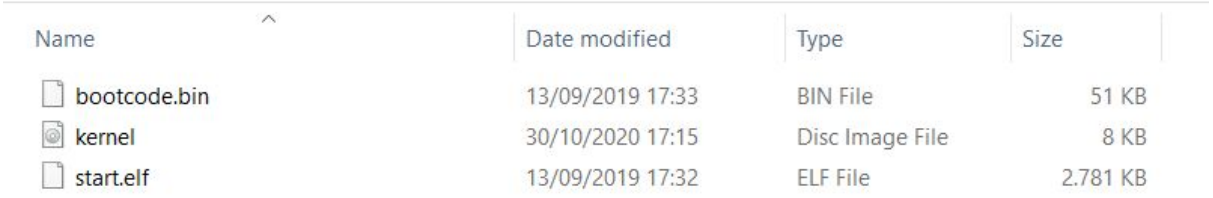
c. In the future, we will implement in this code the feature to print in the LCD Screen the numbers we have received via i2c.

```
#include "ch.h"
#include "hal.h"
#include "chprintf.h"

static WORKING_AREA(waThread_LCD, 128);
static msg_t Thread_LCD(void *p)
{
  (void)p;
  chRegSetThreadName("SerialPrint");
  uint16_t iteration = 0;
  while (TRUE)
  {
    sdPut(&SD1, (uint8_t)0x7C);
    sdPut(&SD1, (uint8_t)0x18);
    sdPut(&SD1, (uint8_t)0x20);
    chThdSleepMilliseconds(10);

    sdPut(&SD1, (uint8_t)0x7C);
    sdPut(&SD1, (uint8_t)0x19);
    sdPut(&SD1, (uint8_t)0x20);
    chThdSleepMilliseconds(10);

    chprintf((BaseSequentialStream *)&SD1, "Iter.: %u", iteration);
    iteration++;
```

```
    chThdSleepMilliseconds(2000);
  }
  return 0;
}

/*
 * Application entry point.
 */
int main(void)
{
  halInit();
  chSysInit();

  char txbuf[] = "Hello Chibi-World - UDL-MINF";

  // Initialize Serial Port
  sdStart(&SD1, NULL);

  // First Message
  chprintf((BaseSequentialStream *)&SD1, "Main (SD1 started)");

  // Coordinates
  sdPut(&SD1, (uint8_t)0x7C);
  sdPut(&SD1, (uint8_t)0x18);
  sdPut(&SD1, (uint8_t)0x00);
  chThdSleepMilliseconds(10);

  sdPut(&SD1, (uint8_t)0x7C);
  sdPut(&SD1, (uint8_t)0x19);
  sdPut(&SD1, (uint8_t)0x0A);
  chThdSleepMilliseconds(10);

  // Second message
  chprintf((BaseSequentialStream *)&SD1, txbuf);

  // Start thread
  chThdCreateStatic(waThread_LCD, sizeof(waThread_LCD), HIGHPRIO,
Thread_LCD, NULL);

  /*
   * Events servicing loop.
   */
```

```
  chThdWait(chThdSelf());

  return 0;
}
```

4. Put the code on the SD card (there's already an example done in the Github, under /documentation/ called boot_LCD.rar
    a. Remember, the SD should look like this:



| Name | Date modified | Type | Size |
|---|---|---|---|
| bootcode.bin | 13/09/2019 17:33 | BIN File | 51 KB |
| kernel | 30/10/2020 17:15 | Disc Image File | 8 KB |
| start.elf | 13/09/2019 17:32 | ELF File | 2.781 KB |

5. Put the SD on the Raspberry, turn it on and test!

Universitat de Lleida

# Data log and Screen representation (Sprint 4)

MINF UDL 20-21

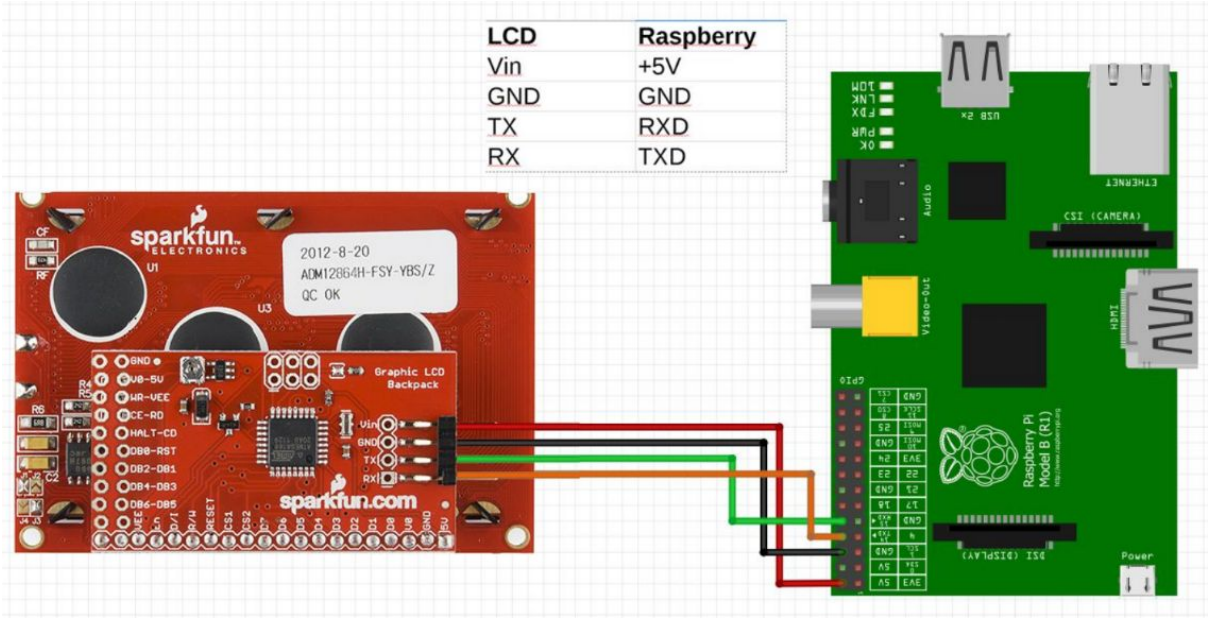
Ubiquitous and embedded systems

Team 1

## 1. Objective

a. The objective is to make a graphical representation of the obtained temperature and humidity on the LCD Screen, being a relation of value x time (24h), the code will be present in the chibiOS.

## 2. Wiring




Note: in this case the RXD -> TX cable is not necessary, onyl the TXD <> RX since the LCD is not sending anything to the Raspberry.

## 3. Code (ChibiOS)

### a. Thread

```
static WORKING_AREA(waThread_LCD, 512);
static msg_t Thread_LCD(void *p)
{
  (void)p;
  chRegSetThreadName("SerialPrint");
  drawStructure();

  while (TRUE)
  {
    if (screenNeedsRefresh == 1)
    {
      if (needsClear == 1)
        clearScreen();

      if (screenToShow == 0)
        drawGraphLineTemp();
      else if (screenToShow == 1)
        drawGraphLineHum();

      screenNeedsRefresh = 0;
    }
    chThdSleepMilliseconds(2000);
  }
  return 0;
}
```

### b. Auxiliar functions

```
void stackHandler()
{
  if (aux_counter == 63)
    aux_counter = 0;
  if (aux_counter == 0)
  {
    stackLineTemp[0][0] = 18;
    stackLineTemp[0][1] = 14 + temperature;
    stackLineTemp[0][2] = 18 + 1;
    stackLineTemp[0][3] = 14 + temperature;
```

```
    stackLineHum[0][0] = 18;
    stackLineHum[0][1] = 14 + roundNo(humidity / 2);
    stackLineHum[0][2] = 18 + 1;
    stackLineHum[0][3] = 14 + roundNo(humidity / 2);
  }
  else
  {
    stackLineTemp[aux_counter][0] = stackLineTemp[aux_counter - 1][2];
    stackLineTemp[aux_counter][1] = stackLineTemp[aux_counter - 1][3];
    stackLineTemp[aux_counter][2] = stackLineTemp[aux_counter - 1][2] + 1;

    if (temperature > 38)
      stackLineTemp[aux_counter][3] = 14 + 38;
    else
      stackLineTemp[aux_counter][3] = 14 + temperature;

    stackLineHum[aux_counter][0] = stackLineHum[aux_counter - 1][2];
    stackLineHum[aux_counter][1] = stackLineHum[aux_counter - 1][3];
    stackLineHum[aux_counter][2] = stackLineHum[aux_counter - 1][2] + 1;
    if (humidity > 76)
      stackLineHum[aux_counter][3] = 14 + 38;
    else
      stackLineHum[aux_counter][3] = 14 + roundNo(humidity / 2);
  }
  lcdCounter += 1;
  aux_counter += 1;
  screenNeedsRefresh = 1;

  // function to control when the graphics of LCD is going to change
  if (lcdCounter == 4)
  {
    if (screenToShow == 1)
      screenToShow = 0;
    else if (screenToShow == 0)
      screenToShow = 1;

    needsClear = 1;
    firstEnter = 1;
    lcdCounter = 0;
  }
}

void drawGraphLineTemp()
{
  int value = temperature;

  if (value > 38)
    value = 38;

  // values
  lcdPrintf(105, 47, "%d", temperature);
  lcdPrintf(105, 38, "%d", humidity);
```

```
  if (firstEnter == 1)
  {
    // title
    lcdPrintf(25, 61, "Temperature", 0);
    lcdPrintf(4, 61, "C", 0);

    // legend
    lcdPrintf(7, 22, "%u", 8);
    lcdPrintf(1, 32, "%u", 18);
    lcdPrintf(1, 42, "%u", 29);
    lcdPrintf(1, 52, "%u", 38);

    // 32/38 pixels alçada, i
    //int startX = 18;
    //int startY = 14;

    int i = 0;
    for (i = 0; i < aux_counter; i++) // draw all previous lines
    {
      drawLine(stackLineTemp[i][0],
               stackLineTemp[i][1],
               stackLineTemp[i][2],
               stackLineTemp[i][3]);
    }
    firstEnter = 0;
  }
  drawLine(stackLineTemp[aux_counter - 1][2],
           stackLineTemp[aux_counter - 1][3],
           stackLineTemp[aux_counter - 1][2] + 1,
           14 + value); // draw current line
}

void drawGraphLineHum()
{
  int value = humidity;

  if (value > 38)
    value = 38;

  // values
  lcdPrintf(105, 47, "%d", temperature);
  lcdPrintf(105, 38, "%d", humidity);

  if (firstEnter == 1)
  {
    // title
    lcdPrintf(25, 61, "Humidity", 0);
    lcdPrintf(4, 61, "%%", 0);

    // legend
    lcdPrintf(7, 22, "%u", 19);
```

```
    lcdPrintf(1, 32, "%u", 38);
    lcdPrintf(1, 42, "%u", 57);
    lcdPrintf(1, 52, "%u", 76);

    int i = 0;

    for (i = 0; i < aux_counter; i++)
    {
      drawLine(stackLineHum[i][0],
               stackLineHum[i][1],
               stackLineHum[i][2],
               stackLineHum[i][3]); // draw all previous lines
    }
    firstEnter = 0;
  }
  drawLine(stackLineHum[aux_counter - 1][2],
           stackLineHum[aux_counter - 1][3],
           stackLineHum[aux_counter - 1][2] + 1,
           14 + roundNo(value / 2)); // draw current line
}


void drawStructure()
{
  // info
  lcdPrintf(92, 47, "T:", 0);
  lcdPrintf(92, 38, "H:", 0);
  lcdPrintf(92, 27, "D:", 0);
  //
  lcdPrintf(10, 11, "%u", 0);
  lcdPrintf(118, 47, "C", 0);
  lcdPrintf(118, 38, "%%", 0);
  //mainframe
  drawLine(17, 13, 17, 52, 0);
  drawLine(18, 13, 87, 13, 0);
  //
  drawLine(14, 52, 17, 52, 0);
  drawLine(14, 42, 17, 42, 0);
  drawLine(14, 32, 17, 32, 0);
  drawLine(14, 22, 17, 22, 0);
  //
  drawLine(30, 12, 30, 10, 0);
  drawLine(59, 12, 59, 10, 0);
  drawLine(87, 12, 87, 10, 0);
  //
  drawBox(90, 49, 125, 29, 0);
  // legend
  lcdPrintf(30, 8, "%u", 8);
  lcdPrintf(59, 8, "%u", 16);
  lcdPrintf(87, 8, "%u", 24);
  lcdPrintf(101, 8, "h", 0);
  //
```

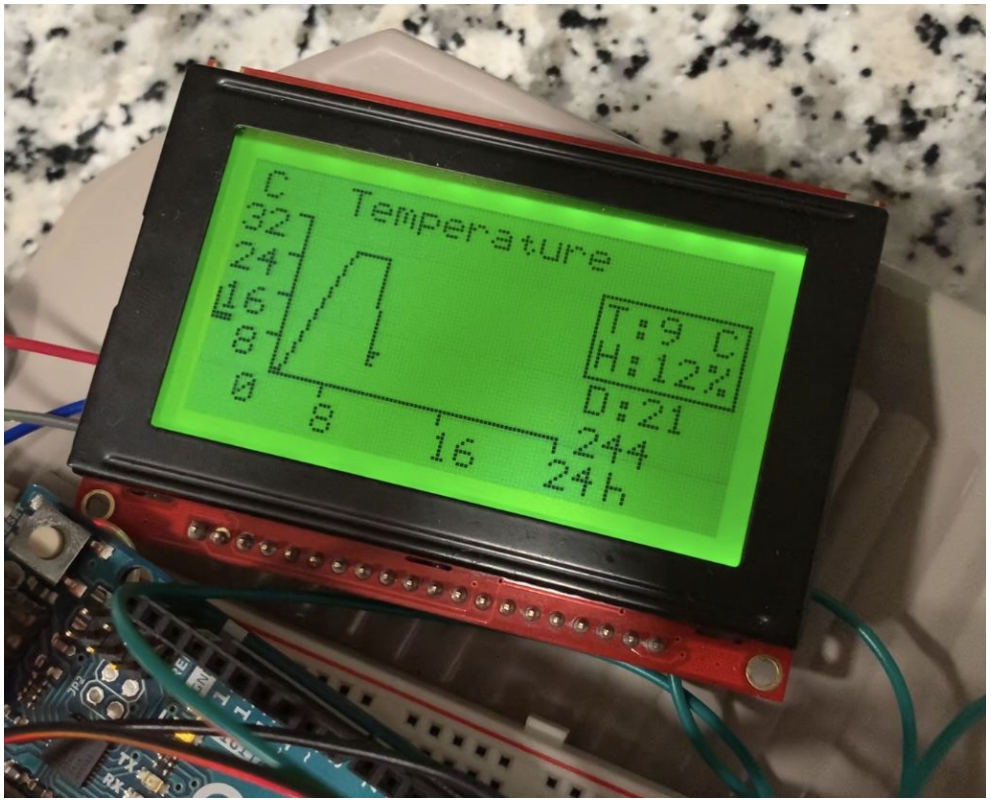
```
}

//  Prints a text in the LCD screen
void drawLine(int x1, int y1, int x2, int y2, int set)
{
  //draws a line from two given points.
  sdPut(&SD1, (uint8_t)0x7C);
  sdPut(&SD1, (uint8_t)0x0C);
  sdPut(&SD1, (uint8_t)x1);
  sdPut(&SD1, (uint8_t)y1);
  sdPut(&SD1, (uint8_t)x2);
  sdPut(&SD1, (uint8_t)y2);
  sdPut(&SD1, (uint8_t)0x01);

  chThdSleepMilliseconds(10);
}

void drawBox(int x1, int y1, int x2, int y2, int set)
{
  //draws a box from two given points.  sdPut(&SD1, (uint8_t)0x7C);
  sdPut(&SD1, (uint8_t)0x0F);
  sdPut(&SD1, (uint8_t)x1);
  sdPut(&SD1, (uint8_t)y1);
  sdPut(&SD1, (uint8_t)x2);
  sdPut(&SD1, (uint8_t)y2);
  sdPut(&SD1, (uint8_t)0x01);

  chThdSleepMilliseconds(10);
}

int roundNo(float num)
{
  return num < 0 ? num - 0.5 : num + 0.5;
}

void clearScreen()
{
  sdPut(&SD1, (uint8_t)0x7C);
  sdPut(&SD1, (uint8_t)0);

  drawStructure();
}

void lcdPrintf(int x, int y, char text[], int value)
{

  sdPut(&SD1, (uint8_t)0x7C);
  sdPut(&SD1, (uint8_t)0x18);
  sdPut(&SD1, (uint8_t)x);
  chThdSleepMilliseconds(10);

  sdPut(&SD1, (uint8_t)0x7C);
```

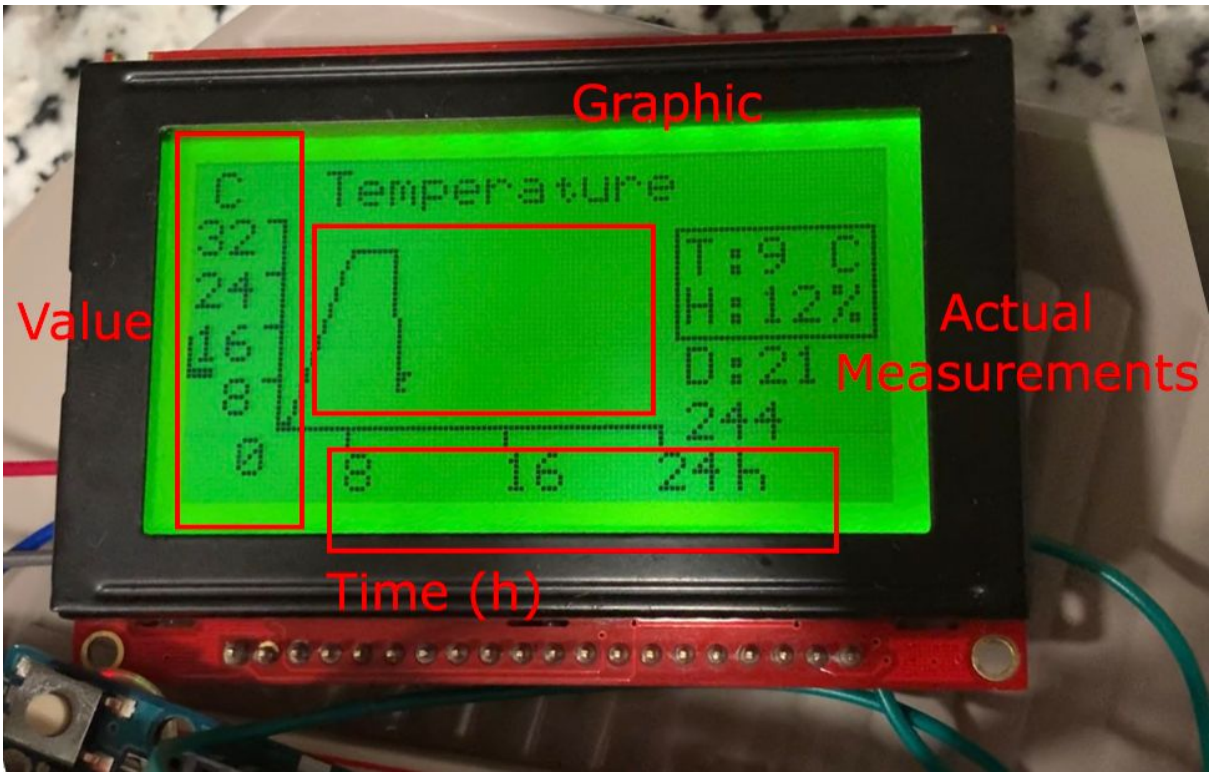
```
  sdPut(&SD1, (uint8_t)0x19);
  sdPut(&SD1, (uint8_t)y);
  chThdSleepMilliseconds(10);

  chprintf((BaseSequentialStream *)&SD1, text, value);
  chThdSleepMilliseconds(10);
}
```

## 4. Final result

a. Every 5 iterations, the screen switch between showing the Temperature and the Humidity

b. All values for the lines are saved on an array of coordinates, so we can keep showing the graphics for 24h.






Reference video: https://www.youtube.com/watch?v=WJ3wlqHZit0

**Data Storage structure:**

The selected structure for the data log is a 2-dimensional array, this way we can easily simulate the coordinates for the lines to print in the LCD.

```
int stackLineTemp[64][4];
int stackLineHum[64][4];
```

With this data which is treated in the stackHandler function (as seen before), we can easily plot the graphics with the acquired data in the LCD Thread (as seen previously)

```
else
{
  stackLineTemp[aux_counter][0] = stackLineTemp[aux_counter - 1][2];
  stackLineTemp[aux_counter][1] = stackLineTemp[aux_counter - 1][3];
  stackLineTemp[aux_counter][2] = stackLineTemp[aux_counter - 1][2] + 1;

  if (temperature > 38)
    stackLineTemp[aux_counter][3] = 14 + 38;
  else
    stackLineTemp[aux_counter][3] = 14 + temperature;
```

The aux_counter is a auxiliar counter created and incremented in the stackHandler() function, this variable controls mostly of the LCD Data representation frequency.

Example of drawing a line the graphic:

```
drawLine(stackLineHum[aux_counter - 1][2],
         stackLineHum[aux_counter - 1][3],
         stackLineHum[aux_counter - 1][2] + 1,
         14 + roundNo(value / 2)); // draw current line
```

Universitat de Lleida

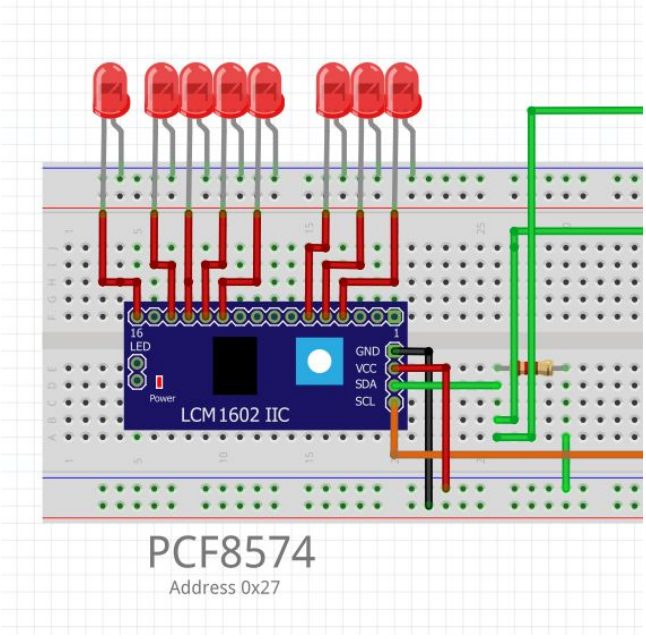
# Ultrasonic LED Bar representation (Sprint 4)

MINF UDL 20-21

Ubiquitous and embedded systems

Team 1

1. **Objective**
    a. The objective is to work with the PCF8574 to turn on LEDs on a LED Bar to represent the level of the container, it receives the distance from the sensor and makes the calculation to display the correct amount of LED's turned on.
2. **Wiring**




In this case the PCF is connected to the Raspberry via the SDA and SCL lines (with a transistor at pullup of these lines).

### 3. Code (ChibiOS)

a. Considerations: In order to the Raspberry to command the PCF, we must follow these steps:

i. Initialize the PCF

1. We send a 0x00 to the PCF to activate.

```
i2cled_init(pcf_address, 0x00);
```

ii. Send the information to the PCF

1. Now we have to send to the PCF 2 things: One is its own address + 0 (to indicate it's a write operation) + the data in which will tell which pins will be lighted up.

a. The address we are using

```
#define pcf_address 0x27        //device address
#define pcf_address_write 0x4E // pcf address + 0 bit write
```

b. The data we are sending, its 8 bits

```
0b11110110
```

c. Send both information in a array of size 2

```
  request[0] = pcf_address_write;
  request[1] = data;

 msg_t status = i2cMasterTransmit(
      &I2C0, device_address, request, 2,
      NULL, 0);
```

b. Full code:

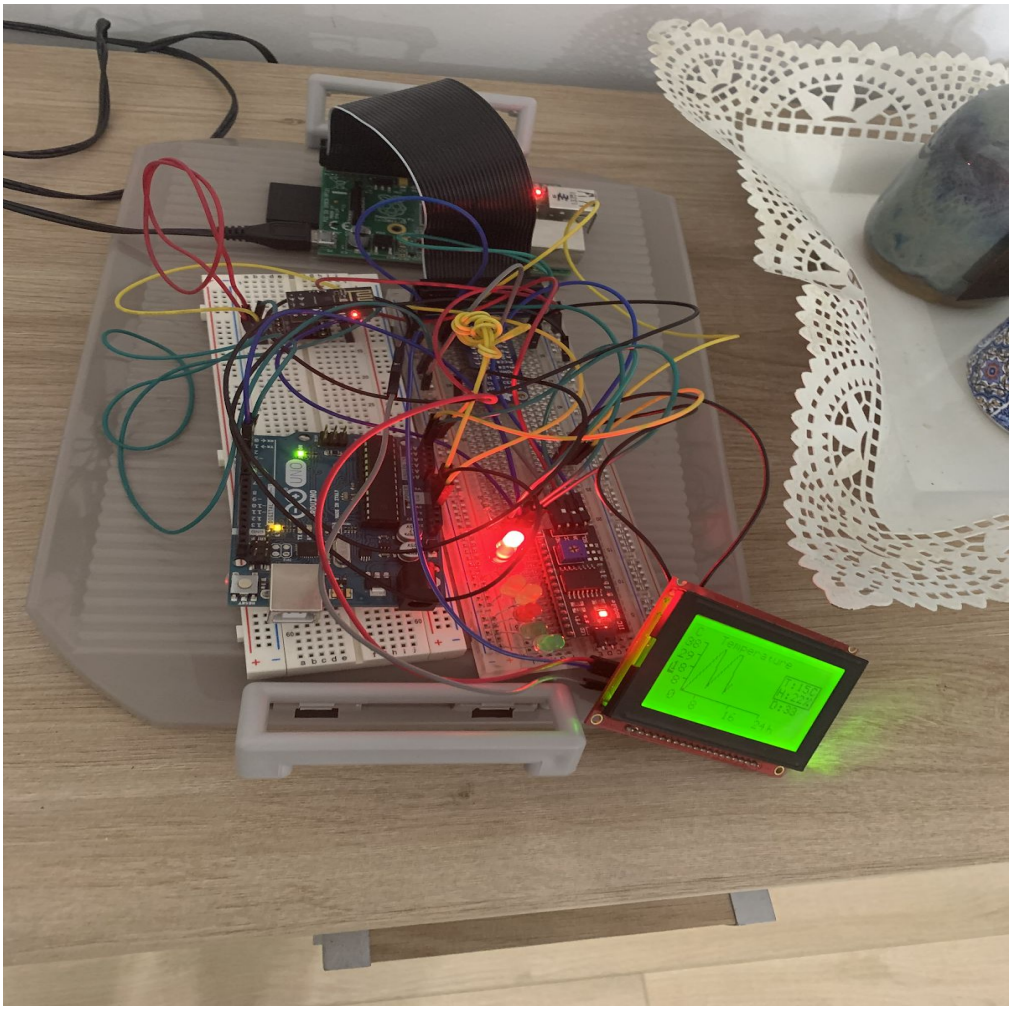
```
static WORKING_AREA(waThread_PCF, 128);
static msg_t Thread_PCF(void *p)
{
  (void)p;
  chRegSetThreadName("PCF");
  chThdSleepMilliseconds(2000);
  int distanceHandler = 0;

  while (TRUE)
  {
    chBSemWait(&smph);
```

```
    palSetPad(ONBOARD_LED_PORT, ONBOARD_LED_PAD);

    distanceHandler = handleDistance(distance);
    pinOut = (uint8_t)handleMeasure(distanceHandler);

    //i2cMasterTransmit(&I2C0, pcf_address_write, (uint8_t)pinOut,
sizeof(pinOut), NULL, 0);
    i2cled_write(pcf_address, pcf_address_write, pinOut);

    //lcdPrintf(92, 17, "%u", pinOut);

    palClearPad(ONBOARD_LED_PORT, ONBOARD_LED_PAD);
    aux_counter += 1;
    aux_pcf += 1;
    //temperature += 3;
    //humidity += 5;

    if (aux_pcf == 8)
      aux_pcf = 0;
    if (aux_counter % 15 == 0)
    {
      temperature = 6;
      humidity = 7;
    }
    chThdSleepMilliseconds(2000);
    chBSemSignal(&smph);
  }

  return 0;
}
static msg_t i2cled_write(uint8_t device_address,
                          uint8_t register_address,
                          uint8_t data)
{
  uint8_t request[2];
  request[0] = register_address;
  request[1] = data;

  msg_t status = i2cMasterTransmit(
      &I2C0, device_address, request, 2,
      NULL, 0);
  chThdSleepMilliseconds(50);

  if (status != RDY_OK)
    //chprintf((BaseSequentialStream *)&SD1, "Error while writing to
i2cled: %d\r\n", status);
```

```
    return status;
}

static void i2cled_init(uint8_t device_address, uint8_t dirmask)
{
  msg_t status = i2cled_write(device_address,
                              pcf_address_write, // direction register.
                              dirmask);
  chThdSleepMilliseconds(50);

  if (status != RDY_OK)
  //chprintf((BaseSequentialStream *)&SD1, "Error while setting direction
mask: %d\r\n", status);
}
```

### 4. Final result

Reference video: https://www.youtube.com/watch?v=WJ3wlqHZit0




# Technical document - Transmission protocol - Final delivery

## Ubiquitous and Embedded systems

## UDL MINF 2021

## Team 1

## Document structure:



## 1. Data Producer 1 transmission protocol

**a. ESP-01 considerations:**

i. We make the communications between the ESP's using the library espnow. It registers 2 senders and 1 receiver.

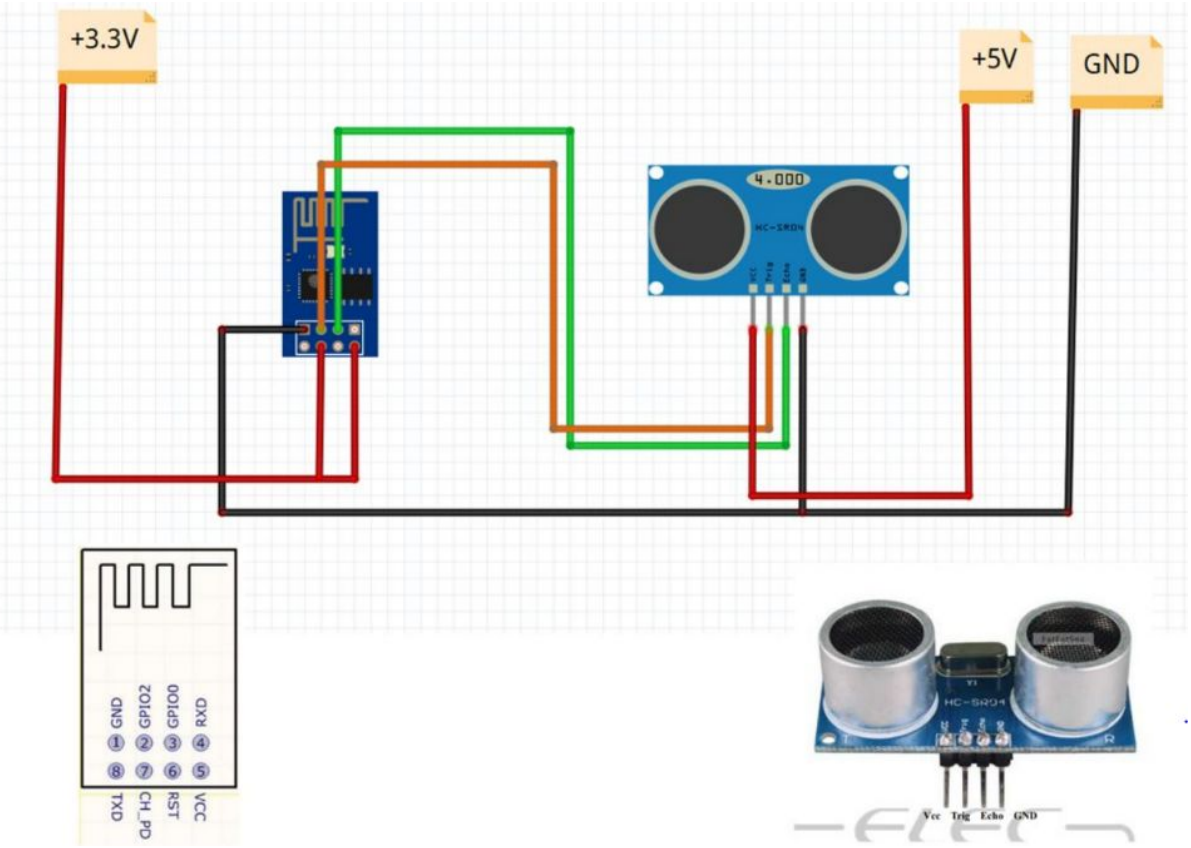
ii. The ESP reads the measures from the HCSR-04 sensor and then send this information to the receiver ESP, the format of the data is:

1. data.id = id of the board (1) int
2. data.x = distance, float
3. data.y = duration (dummy variable, not going to be used in the end), float

iii. Data is sent every 4 seconds

```
void loop() {
  // do the measurements of the sensor
   digitalWrite(trigPin, LOW);  //para generar un pulso limpio ponemos a LOW 4us
   delayMicroseconds(4);

   digitalWrite(trigPin, HIGH);  //generamos Trigger (disparo) de 10us
   delayMicroseconds(10);
   digitalWrite(trigPin, LOW);

   timeElapsed = pulseIn(echoPin, HIGH);
   distance = timeElapsed/58.3;

   delay(1000);

  if ((millis() - lastTime) > timerDelay) {
    // Set values to send
    myData.id = BOARD_ID;
    myData.x = distance;
    myData.y = timeElapsed;

    // Send message via ESP-NOW
    esp_now_send(0, (uint8_t *) &myData, sizeof(myData));
    lastTime = millis();
  }
}
```

## b. Schematics:



## 2. Data Producer 2 transmission protocol

### a. ESP-01 considerations:

i. We make the communications between the ESP's using the library espnow. It registers 2 senders and 1 receiver.

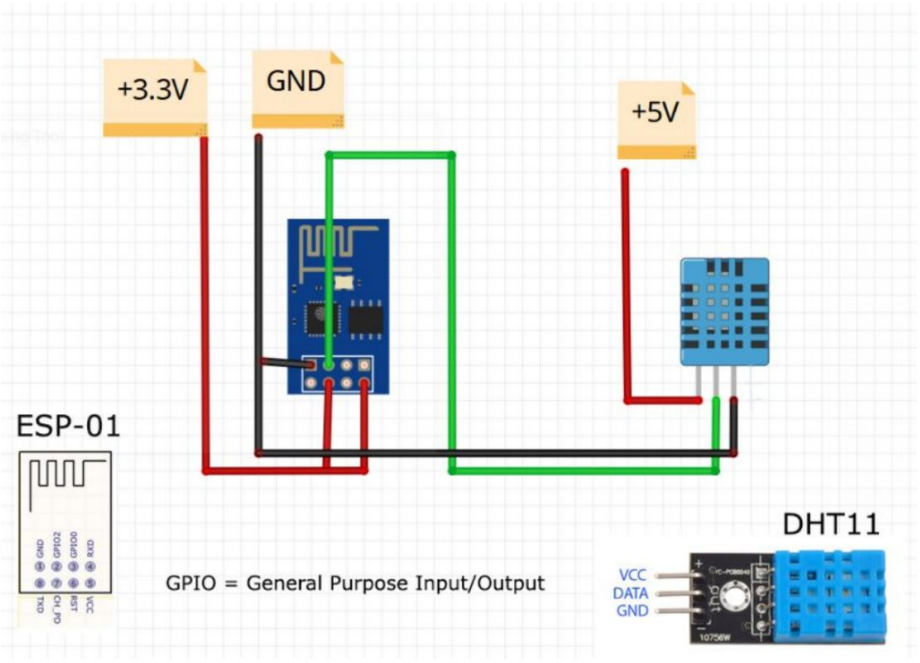
ii. The ESP reads the measures from the DHT11 sensor and then send this information to the receiver ESP, the format of the data is:

1. data.id = id of the board (2) int
2. data.x = temperature, float
3. data.y = humidity(dummy variable, not going to be used in the end), float

iii. Data is sent every time the data measured changes (in comparison with last value sent)

```
void loop() {
  Temperature = dht.readTemperature(); // Gets the values of the temperature
  Humidity = dht.readHumidity();

  if (Temperature != lastTemp && Humidity != lastHum ) {
    lastTemp = Temperature;
    lastHum = Humidity;

    // Set values to send
    myData.id = BOARD_ID;
    myData.x = lastTemp;
    myData.y = lastHum;

    // Send message via ESP-NOW
    esp_now_send(0, (uint8_t *) &myData, sizeof(myData));
  }
  delay(2000);
}
```

### b. Schematics:




## 3. Arduino transmission protocols

**a. Arduino considerations:**

i. Arduino here have 2 roles, receive the data from the ESP-01 receiver and send that data to the Raspberry through the i2c connection.

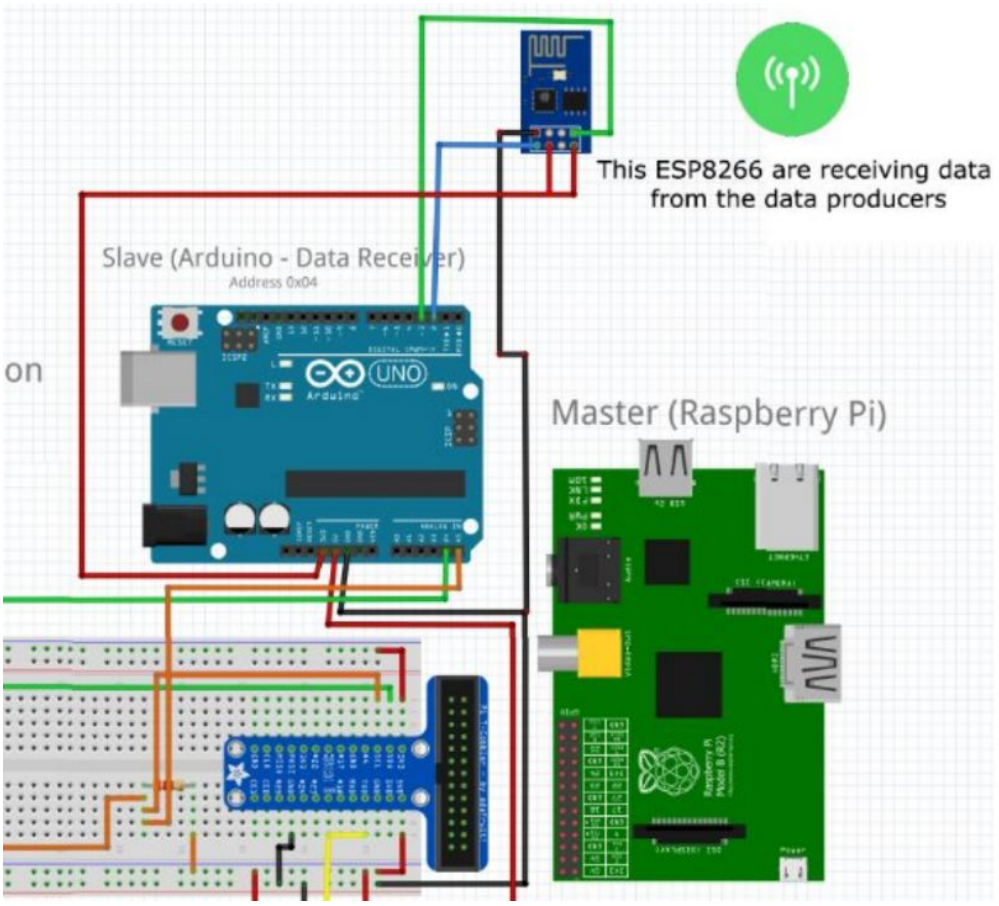
ii. The data is received from the ESP-01 through serial connection, using the softwareserial library.

iii. The data received is in this format:

1. **<Message,Distance,Duration,Temperature,Humidity>** char, float, float, float, float
2. The arduino receives this and then parses the data separating the variables, then the values are stored internally on the arduino.
3. This data is sent to the raspberry when requested through the i2c, using the function **Wire.onRequest(sendData_handler);**
4. The communication in i2c is made through pins A4 (SDA) and A5 (SDL)
5. The format of the data sent is 3 ints:

```
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
  }
}
```

**b. Schematics:**




## 4. ESP-01 receiver transmission protocols

**a. ESP-01 considerations**

i. This ESP-01 receives data periodically from the other 2 ESP's through the espnow connections, the MAC Address of this ESP is present on the code of the ESP senders in order for it to connect with each other.

ii. The data is stored and then sent to the arduino over the serial connection. (Pins TX and RX.

iii. Data is sent in this format whenever a new measure is received:

1. **<Message,Distance,Duration,Temperature,Humidity>** char, float, float, float, float

Receiver_ESP01

```
  //Serial.println(macStr);
  memcpy(&myData, incomingData, sizeof(myData));
  //Serial.printf("Board ID %u: %u bytes\n", myData.id, l
  // Update the structures with the new incoming data
  boardsStruct[myData.id-1].x = myData.x;
  boardsStruct[myData.id-1].y = myData.y;
  boardsStruct[myData.id-1].id = myData.id;

  board1Distance = int(boardsStruct[0].x);
  board1Time = int(boardsStruct[0].y);
  board2Temp = int(boardsStruct[1].x);
  board2Hum = int(boardsStruct[1].y);

  // send data to arduino
  Serial.print("<Data,");
  Serial.print(board1Distance);
  Serial.print(",");
  Serial.print(board1Time);
  Serial.print(",");
  Serial.print(board2Temp);
  Serial.print(",");
  Serial.print(board2Hum);
  Serial.print(">");
}
```

## 4. ChibiOS Raspberry receiver transmission protocols

### b. Raaspberry considerations

i. The raspberry act as a master in the i2c connection
ii. The raspberry is connected to a lcd screen through serial connection
iii. It sends a request to the arduino about the measures, then the arduino responds as previously explained **(3 ints)**
iv. Then, temperature and humidity are used to display in the LCD Screen.
v. The distance, is calculated and then used to send a instruction to the PCF8754 through i2c, the format of the data sent is: **(u_int8) 0b01110000** for example

```
// Request values
status = chBSemWait(&smph);

if (status == 0)
{
  msg_t i2cMsg = i2cMasterReceiveTimeout(
      &I2C0, arduino_address, result, 3, MS2ST(1000));

  if (i2cMsg == 0x00)
  {
    temperature = result[0];
    humidity = result[1];
    distance = result[2];

    stackHandler();

    ledLevel = aux_counter;

    if (ledLevel > 8)
      ledLevel = 0;
  }

  chBSemSignal(&smph);
}
chThdSleepMilliseconds(6000);
```

```
if (ledLevel != lastLedLevel)
{
  status = chBSemWait(&smph);

  if (status == 0)
  {
    pinOut[0] = (uint8_t)handleMeasure(ledLevel);

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

    lastLedLevel = ledLevel;

    chBSemSignal(&smph);
  }
}
chThdSleepMilliseconds(3000);
```


# Final Assembly (Sprint 4)

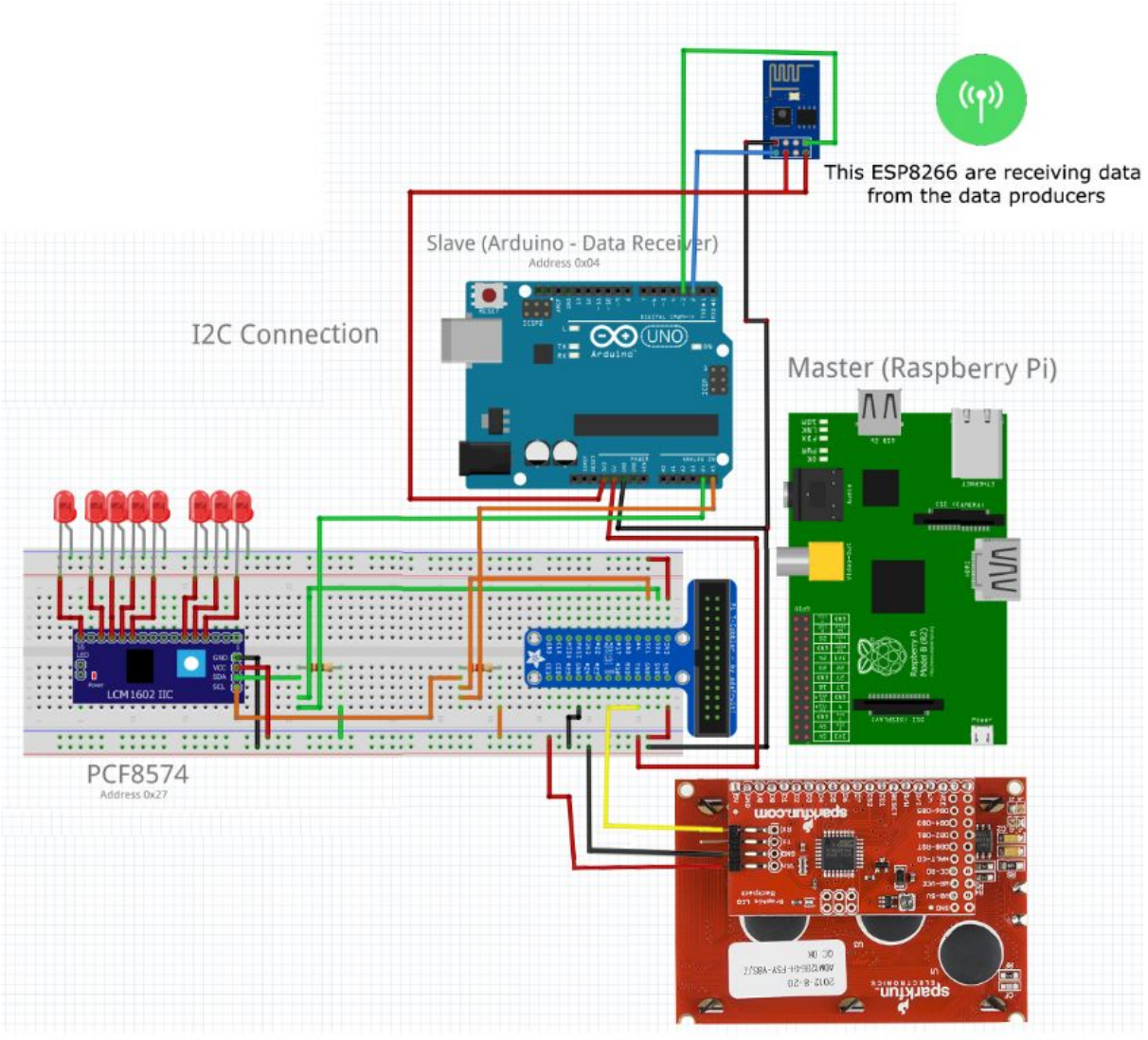
## MINF UDL 20-21

## Ubiquitous and embedded systems

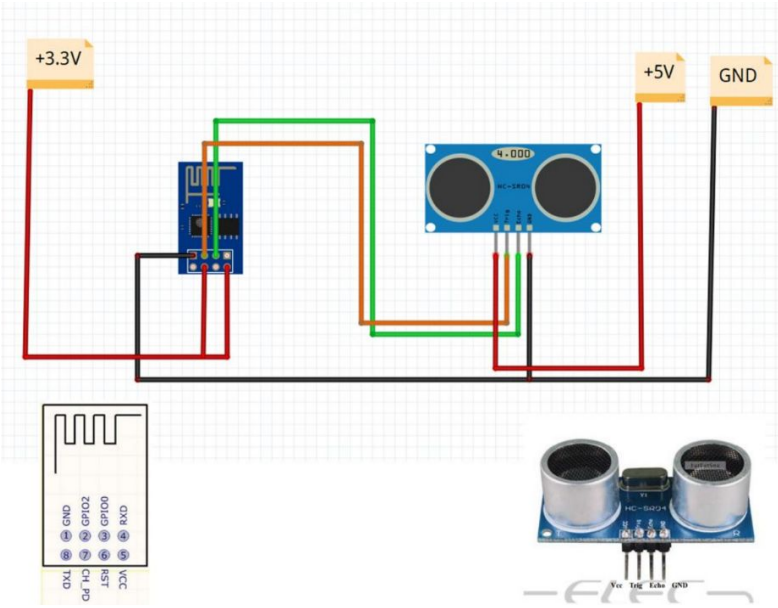
## Team 1

1. **Considerations**
    a. **Power considerations:**
        i. In this case, the data producers 1 and 2 are connected to the Shield of power supply.
        ii. All other components are powered by the Raspberry, as seen in the wiring connection (Point 3).
2. **Preparation**
    a. ChibiOS code
        i. Please refer to /code/ChibiOS_i2c_Raspberry/ for the full code
    b. Arduino (Data Receiver) code
        i. Please refer to /code/Receiver_Arduino/ for full code
    c. Data producers code
        i. Please refer to /code/DataProducer1_ESP_Sender1&2 for full code
3. **Wiring connection**

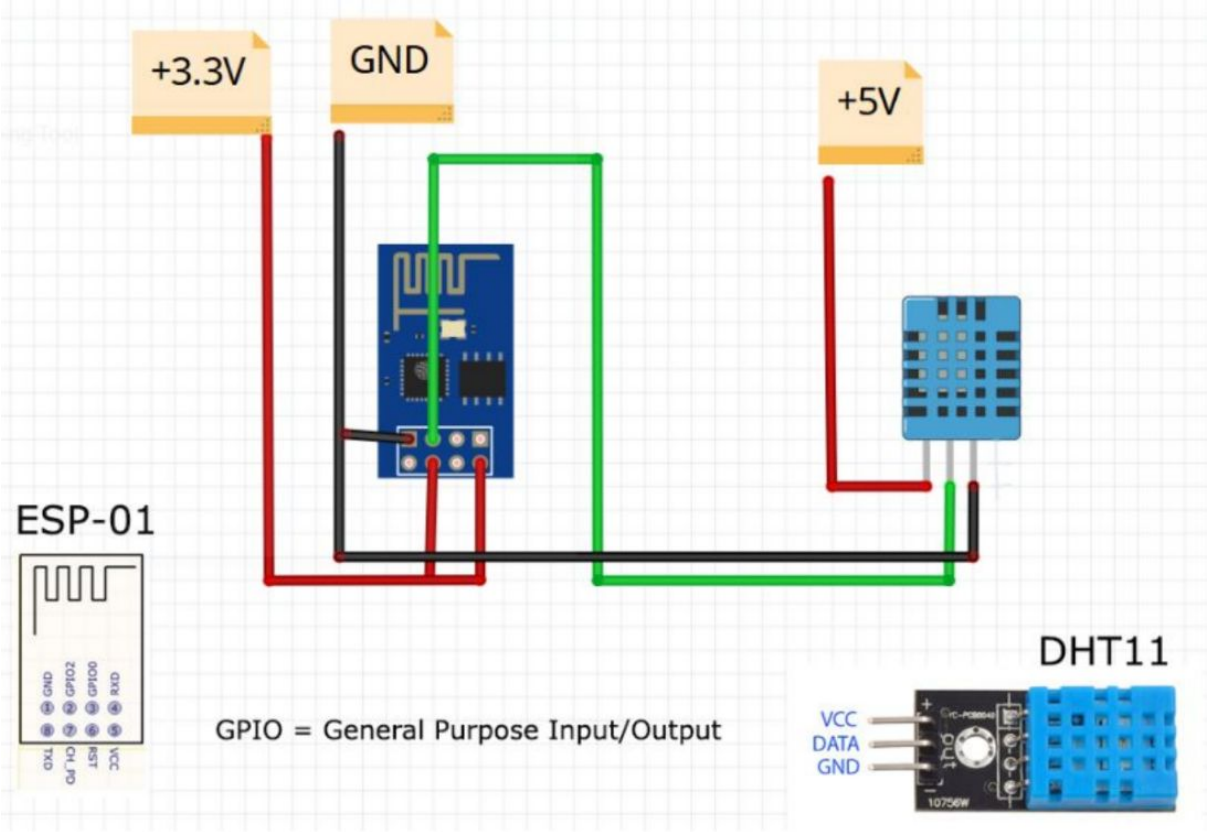
a. **Data receivers schema:**



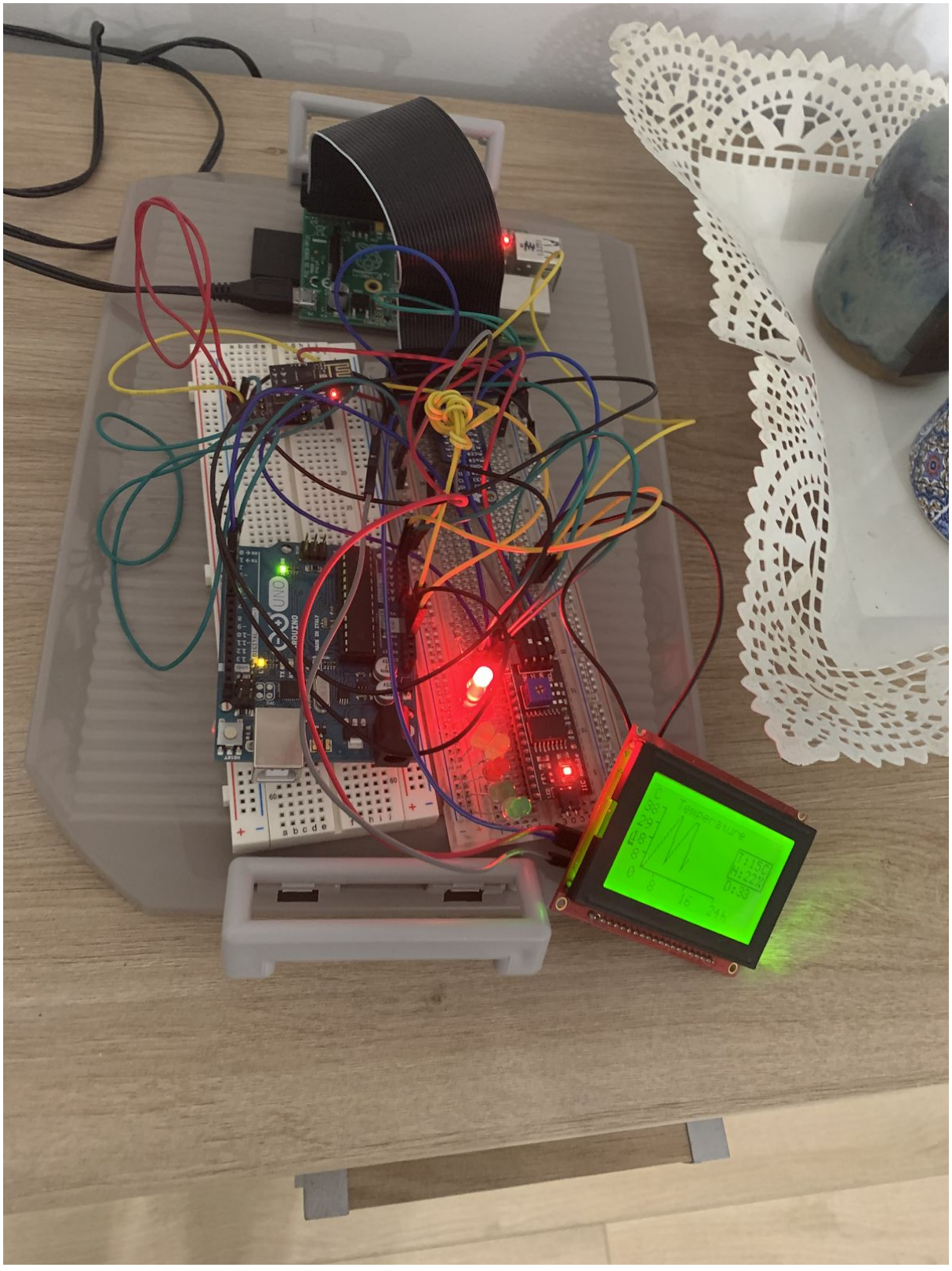

**b. Data Producer 1 schema:**



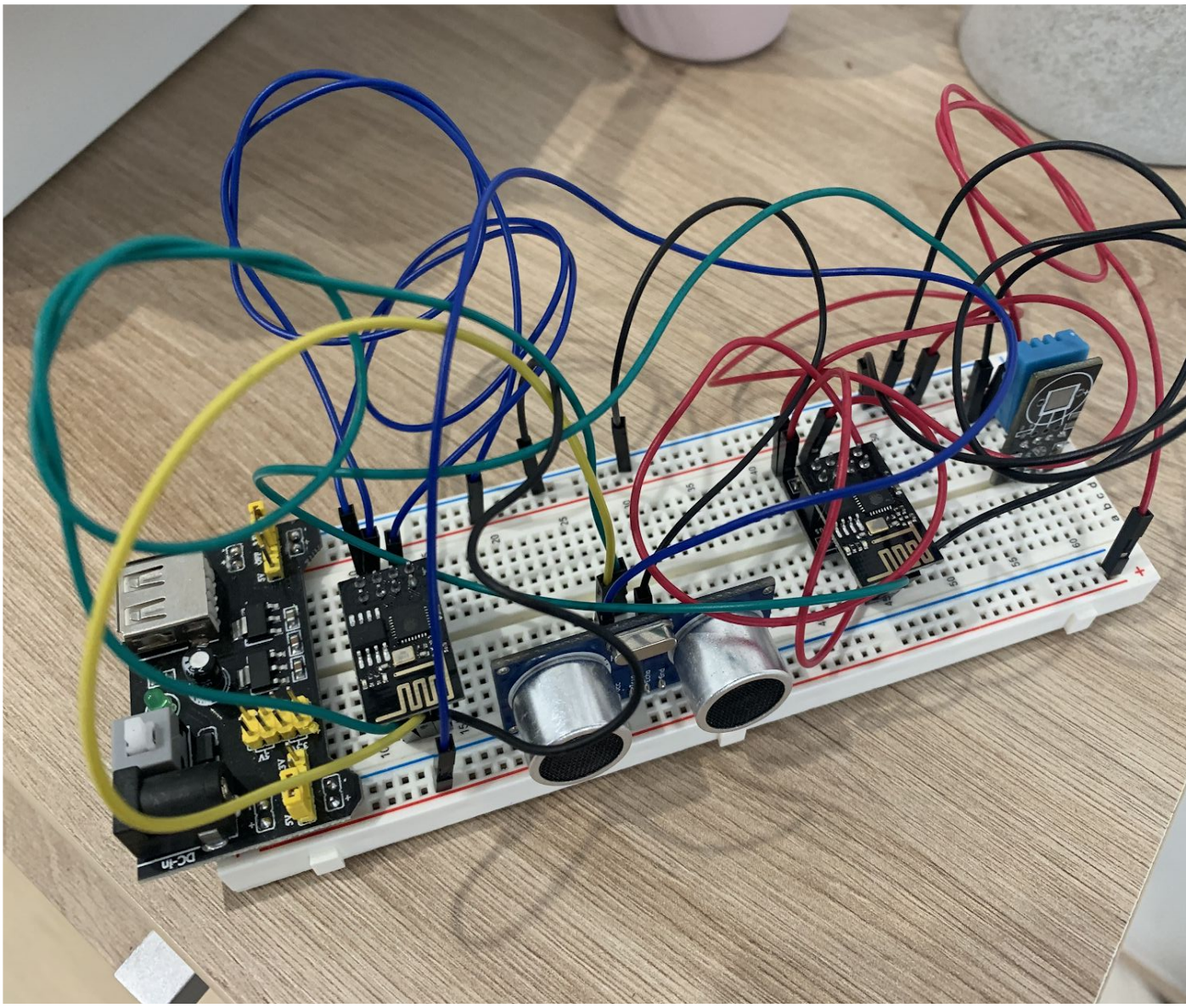
c. **Data Producer 2 schema:**




4. **Final results:**




**Data receivers(using 2 breadboards)**




**Data producers 1 & 2 (using 1 breadboard)**