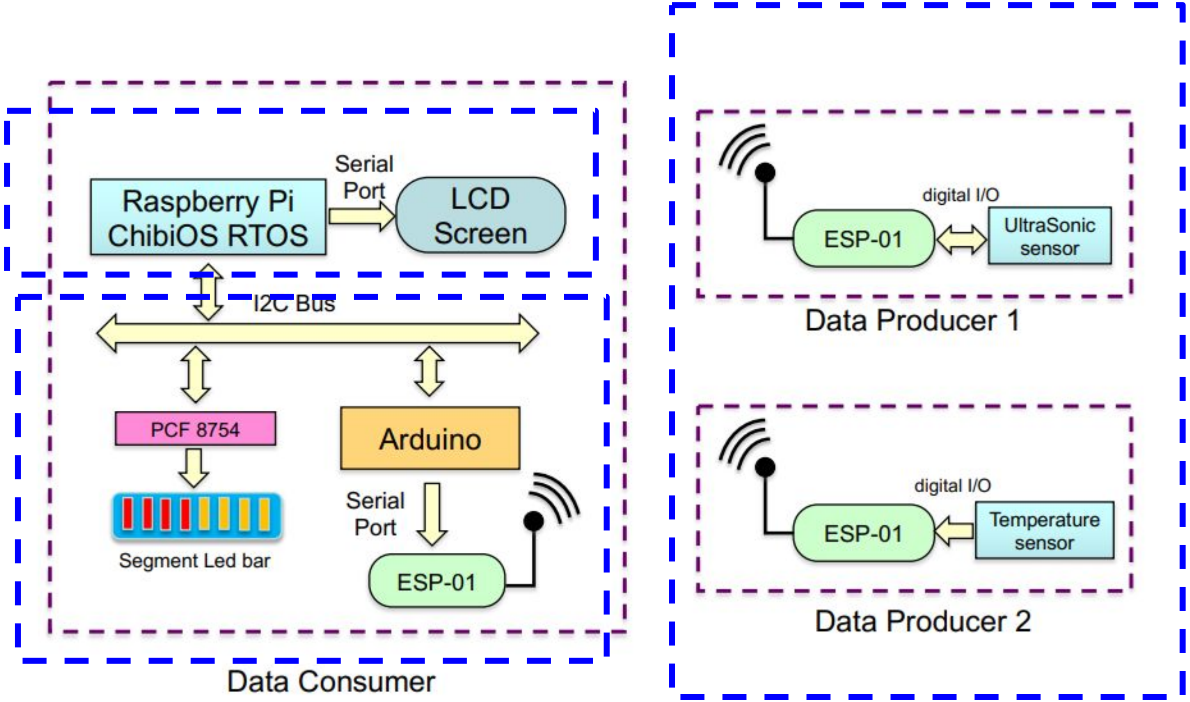
Embedded and Ubiquitous Systems - UDL MINF 20-21

Team 1 - Final Defense

Table of contents

1. **Project overview** - slide 3
2. **Team division** - slide 4
3. **Data Producer 1** - *slide 5*
4. **Data Producer 2** - slide 8
5. **ESP-Receiver and Arduino** - slide 11
6. **ChibiOS** - slide 14
7. **LCD** - slide 17
8. **PCF** - slide 18
9. **Problems encountered** - slide 20

1. Project overview



Sprint 4

Sprint 3

Sprint 2

2. Team Division

Yoon

- Data Producer 1 development
- How to -> I2C protocol
- R.Pi <-> Arduino interaction
- Ultrasonic <-> Led bar representation
- Final assembly

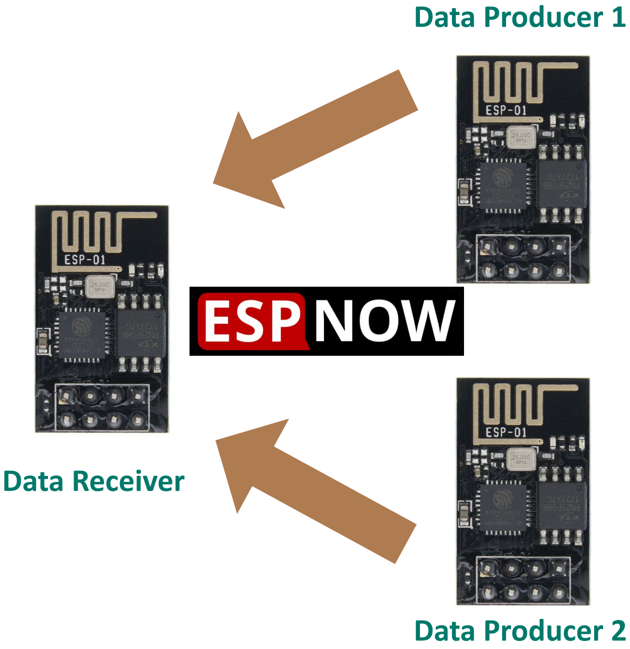
Ron

- Data Producer 2 development
- How to -> LCD Screen
- How to -> PCF8754
- Data log and Screen representation
- Final assembly

Danillo

- ESP-01 <-> Arduino interaction
- How to -> I2C protocol
- R.Pi <-> Arduino interaction
- Data log and Screen representation
- Final assembly

3. Data Producer 1


Data Producer 1

Data Receiver

Data Producer 2

ESPNow is a communication protocol created by Espressif

Many-to-one configuration

Low-power 2.4Ghz

Connection is made through the Mac address of the receiver

Each data producer send to the receiver ESP the following data:

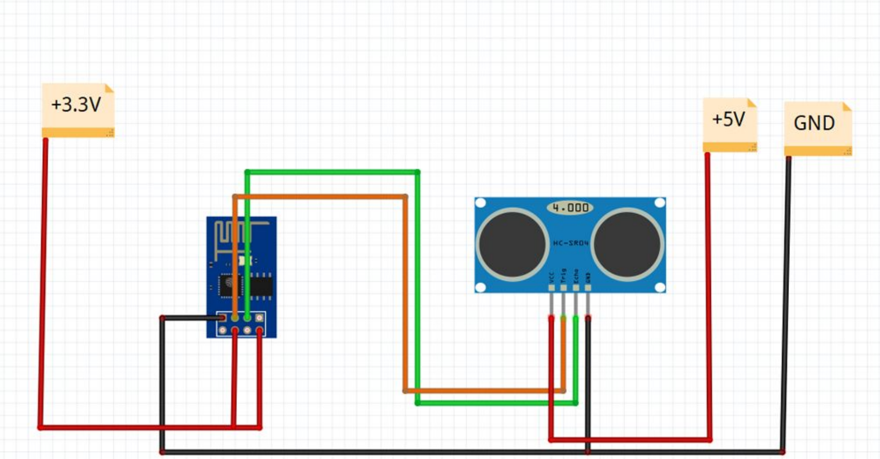
1. id of the sender board (1 or 2)
2. float variable (distance or temperature)
3. float variable (humidity)

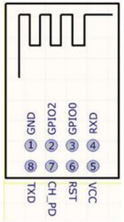
3. Data Producer 1

ESP-01	HC-SR04
2 (GPIO2)	Echo
0 (GPIO)	Trigger

ESP-01	Power
CH_PD, Vcc	+3.3v

HC-SR04	Power
Vcc	+5v





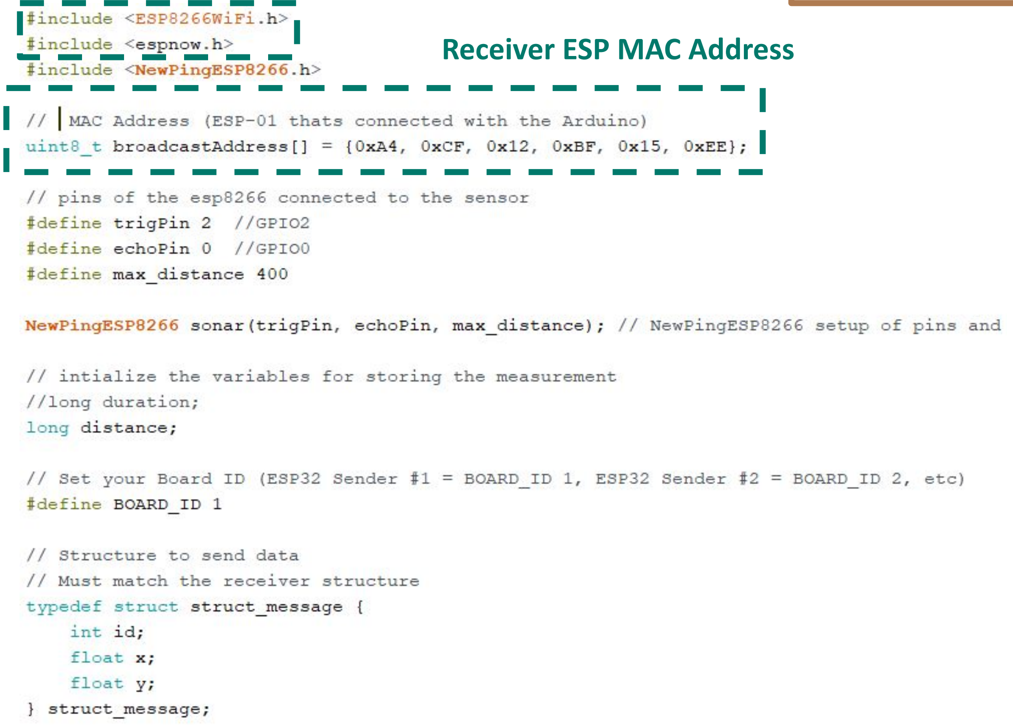
3. Data Producer 1

Sends data every 4 seconds

Library **NewPingESP8266** for reading measures from the sensor

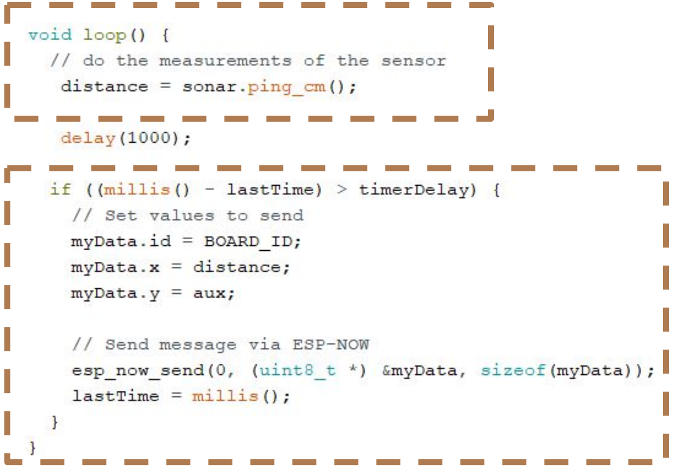
ESPNow library

Receiver ESP MAC Address



```
#include <ESP8266WiFi.h>
#include <espnow.h>
#include <NewPingESP8266.h>

// 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
#define max_distance 400

NewPingESP8266 sonar(trigPin, echoPin, max_distance); // NewPingESP8266 setup of pins and

// intialize the variables for storing the measurement
//long duration;
long 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;
```

Get measures from sensor

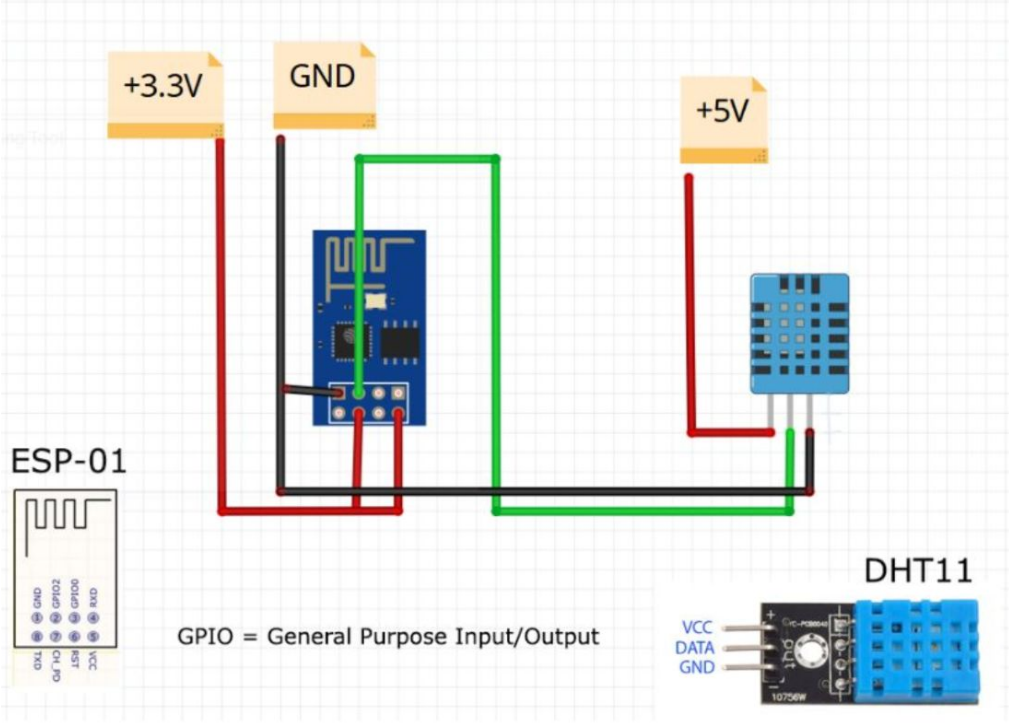


```
void loop() {
  // do the measurements of the sensor
   distance = sonar.ping_cm();

   delay(1000);

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

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

Send data to receiver

4. Data Producer 2

ESP-01	DHT11
2 (GPIO2)	Out



GPIO = General Purpose Input/Output

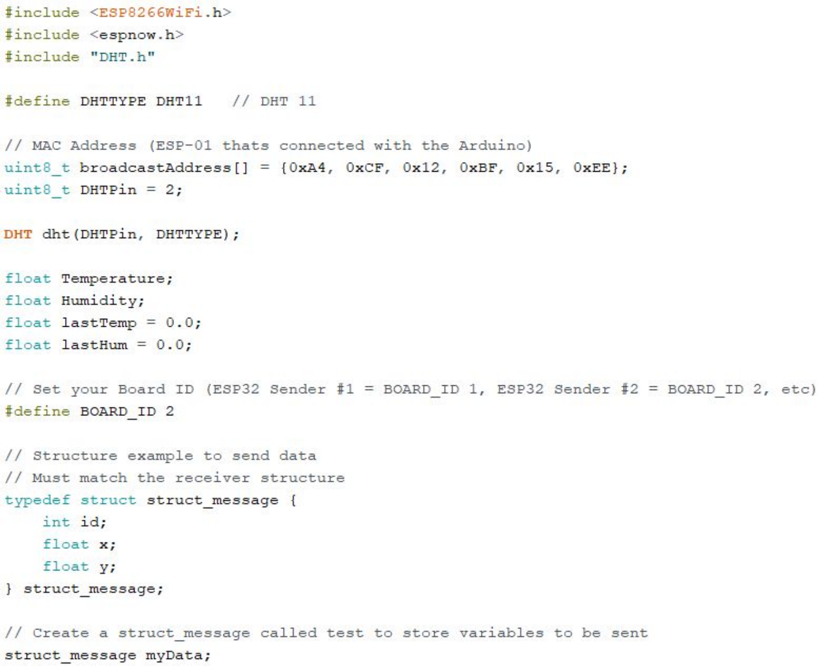
ESP-01	Power
CH_PD, Vcc	+3.3v

DHT11	Power
Vcc	+5v

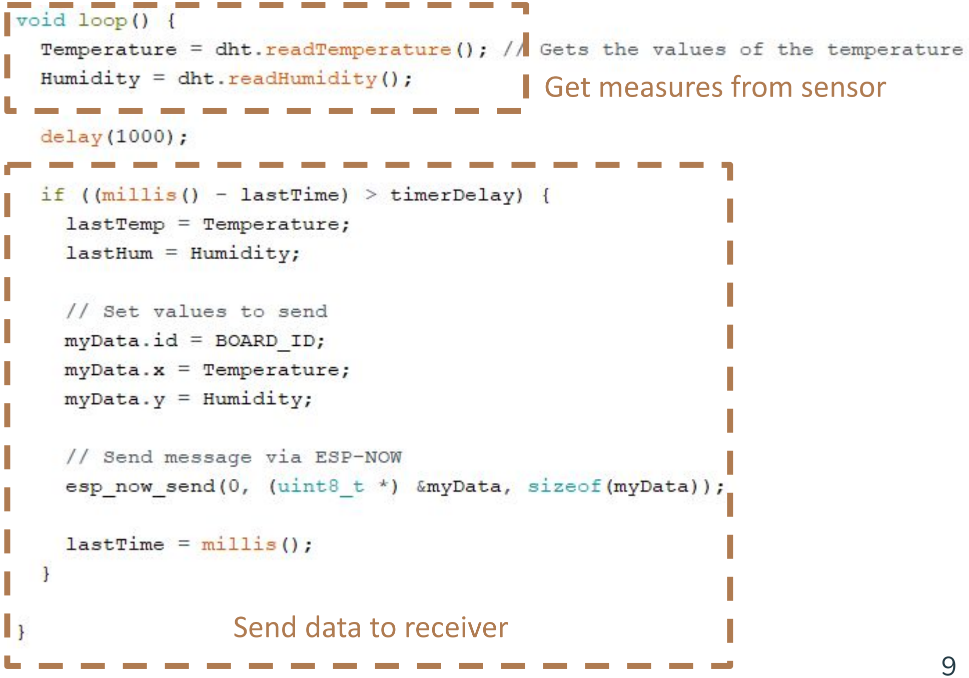
4. Data Producer 2

Sends data every 5 seconds

Library **DHT** for reading measures from the sensor



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

#define DHTTYPE DHT11   // DHT 11

// MAC Address (ESP-01 thats connected with the Arduino)
uint8_t broadcastAddress[] = {0xA4, 0xCF, 0x12, 0xBF, 0x15, 0xEE};
uint8_t DHTPin = 2;

DHT dht(DHTPin, DHTTYPE);

float Temperature;
float Humidity;
float lastTemp = 0.0;
float lastHum = 0.0;

// 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;
```

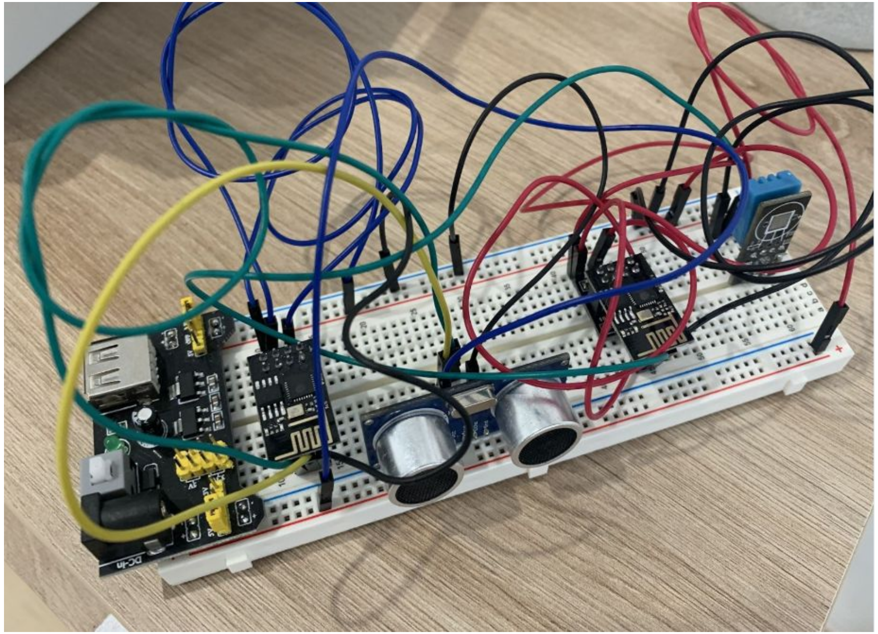


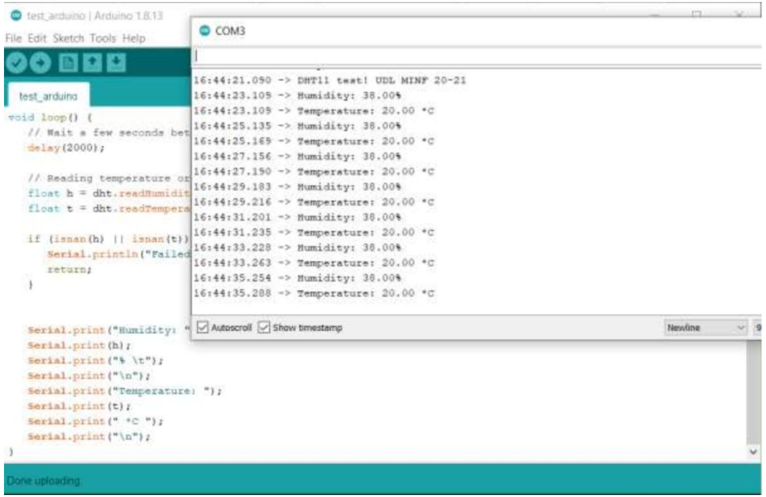
```
void loop() {
  Temperature = dht.readTemperature(); // Gets the values of the temperature
  Humidity = dht.readHumidity();

  delay(1000);

  if ((millis() - lastTime) > timerDelay) {
    lastTemp = Temperature;
    lastHum = 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();
  }

}
```

Get measures from sensor

Send data to receiver

Assembly and testing




test_arduino | Arduino 1.8.13
File Edit Sketch Tools Help
test_arduino
COM3
☑ Autoscroll ☑ Show timestamp
Newline

```
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");
}
```

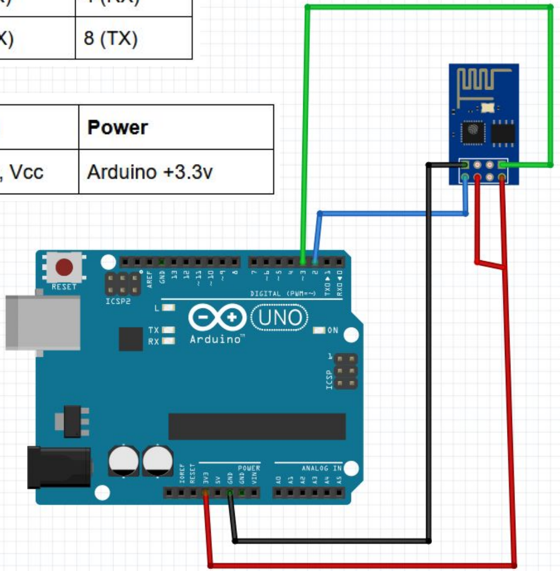
```
16:44:21.050 -> DHT11 test! UDL MINF 20-21
16:44:23.109 -> Humidity: 38.00%
16:44:23.109 -> Temperature: 20.00 *C
16:44:25.135 -> Humidity: 38.00%
16:44:25.169 -> Temperature: 20.00 *C
16:44:27.156 -> Humidity: 38.00%
16:44:27.190 -> Temperature: 20.00 *C
16:44:29.183 -> Humidity: 38.00%
16:44:29.216 -> 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
```

Done uploading.

5. ESP-01 receiver and Arduino

Arduino	ESP-01
3 (TX)	4 (RX)
2 (RX)	8 (TX)

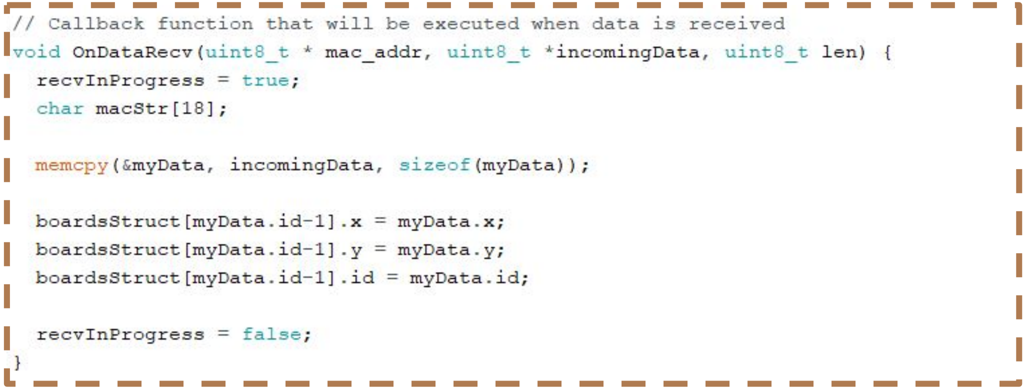
ESP-01	Power
CH_PD, Vcc	Arduino +3.3v



- ESP receive values from senders and store it
- Every 2 seconds, send the data to the Arduino
- Use serial connection (pins RX and TX)
- Send information in format:
  - <String,float,float,float>
  - Message,Temperature,Humidity,Distance

- Arduino receives the data over SoftwareSerial
- Pins 2 and 3
- Parse the data
- Store the data
- Send data over i2c

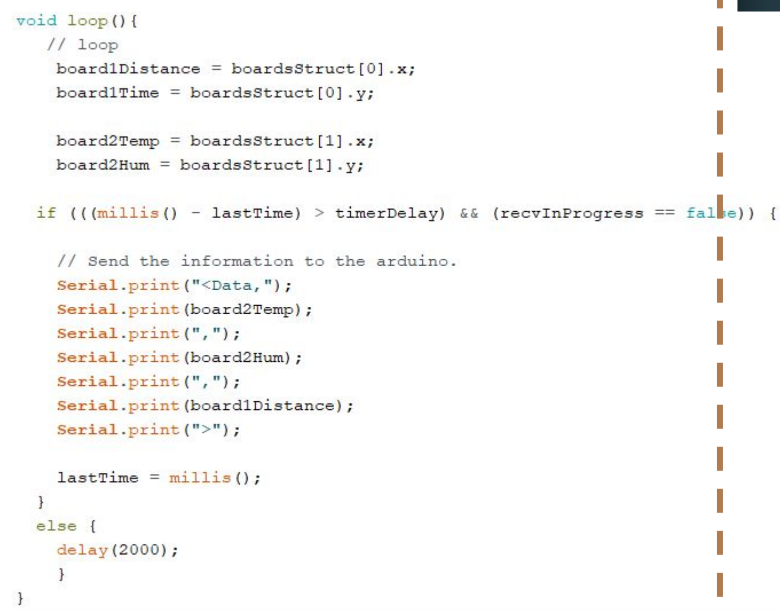
5. ESP-01 receiver code



```
// Callback function that will be executed when data is received
void OnDataRecv(uint8_t * mac_addr, uint8_t *incomingData, uint8_t len) {
  recvInProgress = true;
  char macStr[18];

  memcpy(&myData, incomingData, sizeof(myData));

  boardsStruct[myData.id-1].x = myData.x;
  boardsStruct[myData.id-1].y = myData.y;
  boardsStruct[myData.id-1].id = myData.id;

  recvInProgress = false;
}
```

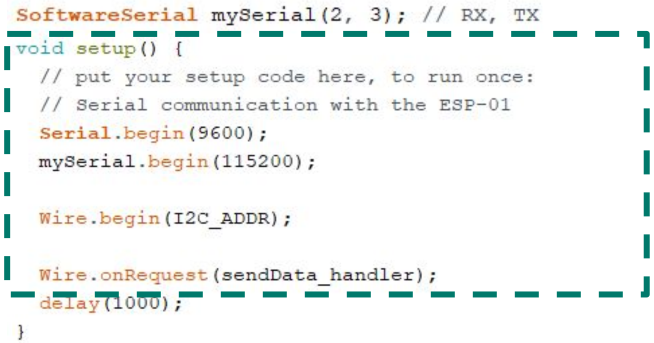
- Callback function to when data is received
- Store data into structs

- Prepare and then send data over serial



```
void loop(){
   // loop
    board1Distance = boardsStruct[0].x;
    board1Time = boardsStruct[0].y;

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

  if (((millis() - lastTime) > timerDelay) && (recvInProgress == false)) {

    // Send the information to the arduino.
    Serial.print("<Data,");
    Serial.print(board2Temp);
    Serial.print(",");
    Serial.print(board2Hum);
    Serial.print(",");
    Serial.print(board1Distance);
    Serial.print(">");

    lastTime = millis();
  }
  else {
    delay(2000);
    }
}
```

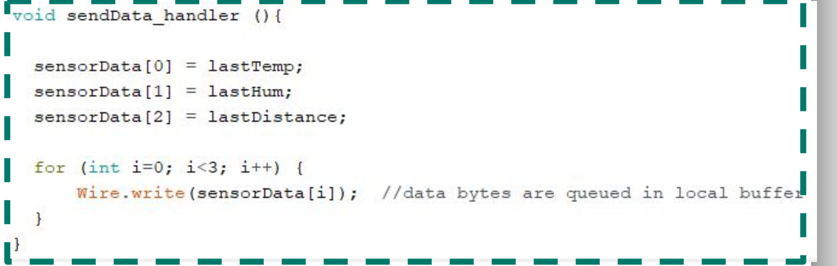
5. Arduino code



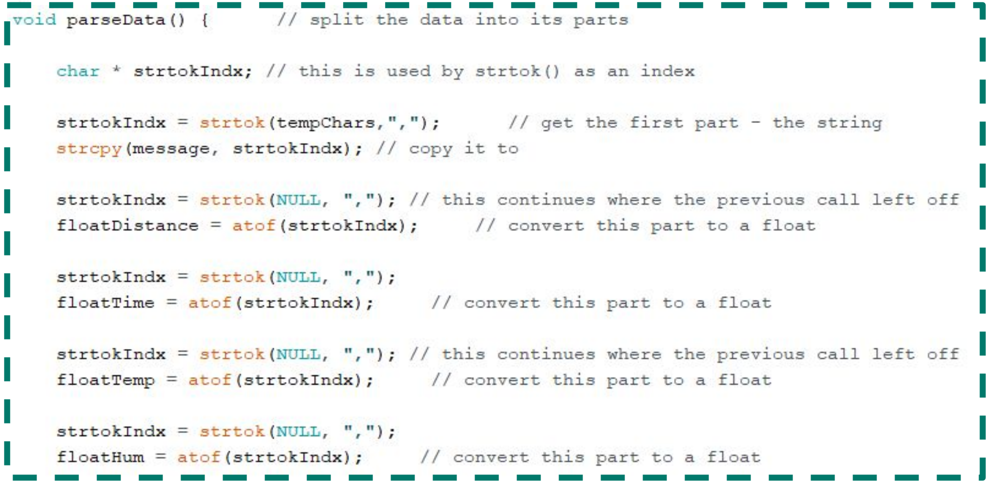
```
SoftwareSerial mySerial(2, 3); // RX, TX
void setup() {
  // put your setup code here, to run once:
  // Serial communication with the ESP-01
  Serial.begin(9600);
  mySerial.begin(115200);

  Wire.begin(I2C_ADDR);

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

- Initiate Serial connection
- Initiate i2c as slave and register i2c callback function
- i2c Address: 0x04

- Send data over i2c when requested by master (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
  }
}
```



```
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
```

- Parse received data from ESP

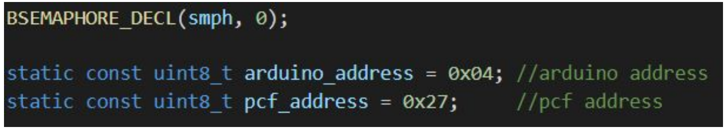
6. ChibiOS (Raspberry Pi)

- LCD Thread size 512
- Arduino Thread (receive data) size 512
- PCF Thread (send data) size 1024

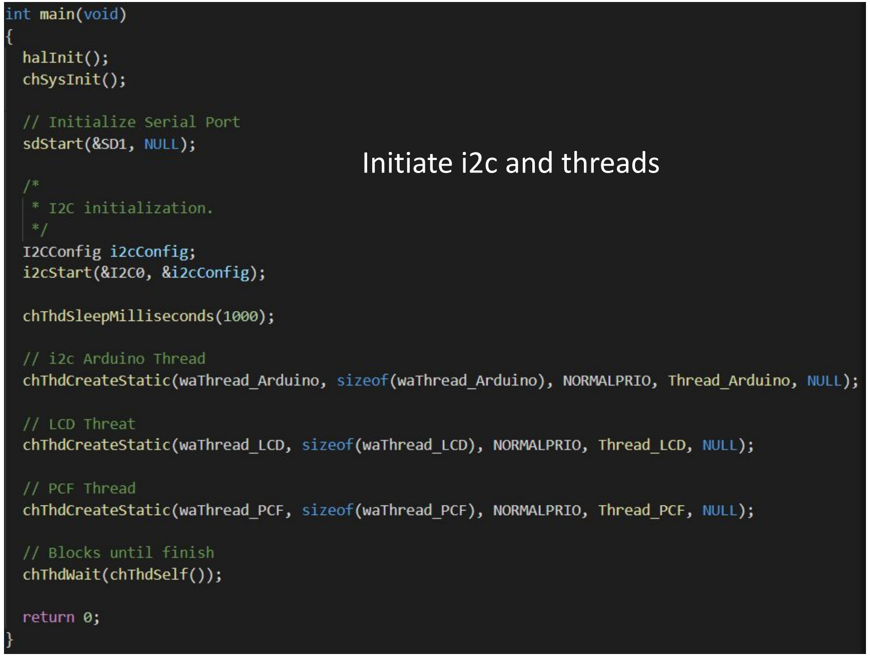
- Threads managed by **Binary Semaphore** (Arduino and PCF threads)

- LCD Thread is executed when **new data has arrived**
- Arduino Thread runs **every 6 seconds**
- PCF Thread (send data) runs **only when** the led level needs to be **updated**

6. ChibiOS code

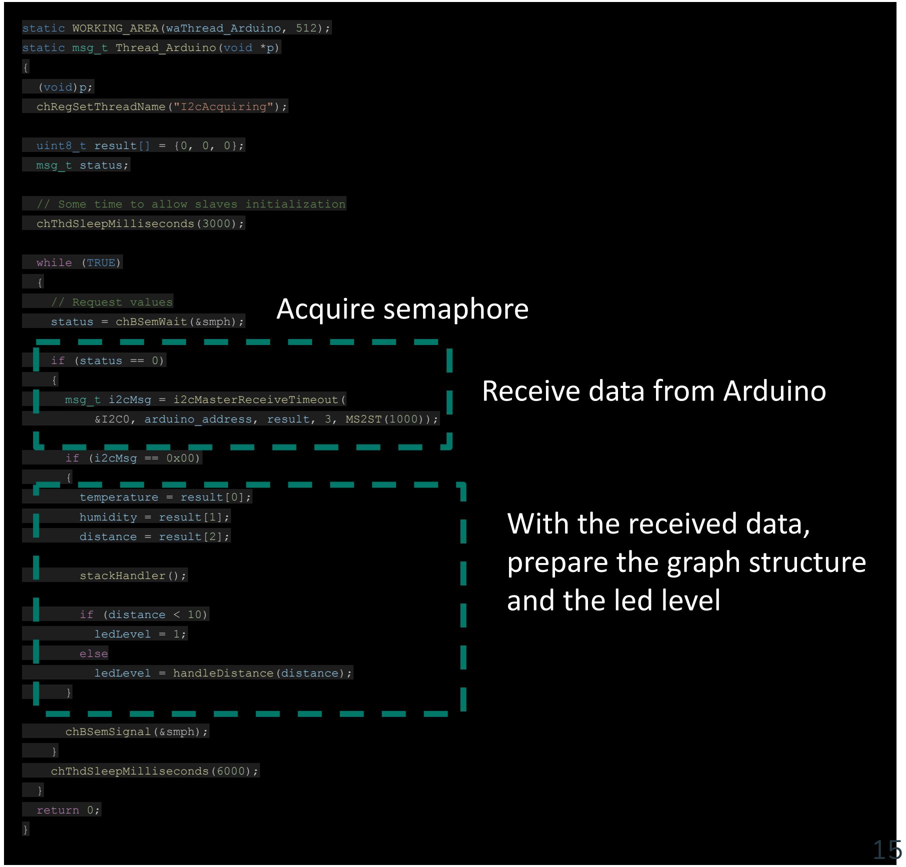


```
BSEMAPHORE_DECL(smph, 0);

static const uint8_t arduino_address = 0x04; //arduino address
static const uint8_t pcf_address = 0x27;     //pcf address
```



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

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

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

  chThdSleepMilliseconds(1000);

  // i2c Arduino Thread
  chThdCreateStatic(waThread_Arduino, sizeof(waThread_Arduino), NORMALPRIO, Thread_Arduino, NULL);

  // LCD Thread
  chThdCreateStatic(waThread_LCD, sizeof(waThread_LCD), NORMALPRIO, Thread_LCD, NULL);

  // PCF Thread
  chThdCreateStatic(waThread_PCF, sizeof(waThread_PCF), NORMALPRIO, Thread_PCF, NULL);

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

  return 0;
}
```

Initiate i2c and threads



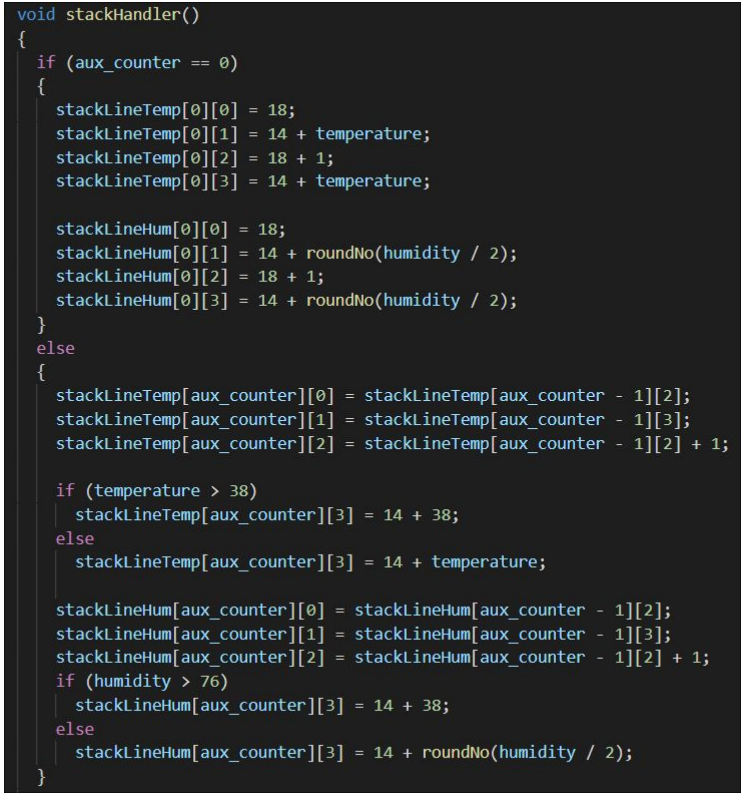
```
static WORKING_AREA(waThread_Arduino, 512);
static msg_t Thread_Arduino(void *p)
{
  (void)p;
  chRegSetThreadName("I2cAcquiring");

  uint8_t result[] = {0, 0, 0};
  msg_t status;

  // Some time to allow slaves initialization
  chThdSleepMilliseconds(3000);

  while (TRUE)
  {
    // 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();

        if (distance < 10)
          ledLevel = 1;
        else
          ledLevel = handleDistance(distance);
      }

      chBSemSignal(&smph);
    }
    chThdSleepMilliseconds(6000);
  }
  return 0;
}
```

Acquire semaphore

Receive data from Arduino

With the received data, prepare the graph structure and the led level

6. ChibiOS Code (Graphics lines)

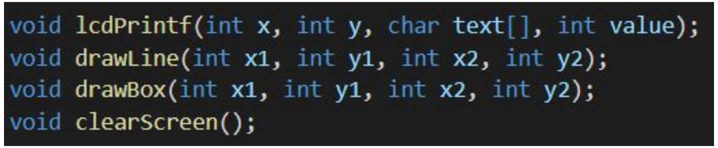


```
void stackHandler()
{
  if (aux_counter == 0)
  {
    stackLineTemp[0][0] = 18;
    stackLineTemp[0][1] = 14 + temperature;
    stackLineTemp[0][2] = 18 + 1;
    stackLineTemp[0][3] = 14 + temperature;

    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);
  }
```

- When receives data from arduino, start stackHandler function
- This function handle the received values and store it in form as a 2-dimensional array to plot the lines
- For testing, the size of the array is 64x4, but we can adapt it to fit the data from the last 24 hours
- This function also controls the LCD behaviour
- For stability reasons, is the only function in the code which handles shared variables and it is inside the semaphore call

7. LCD

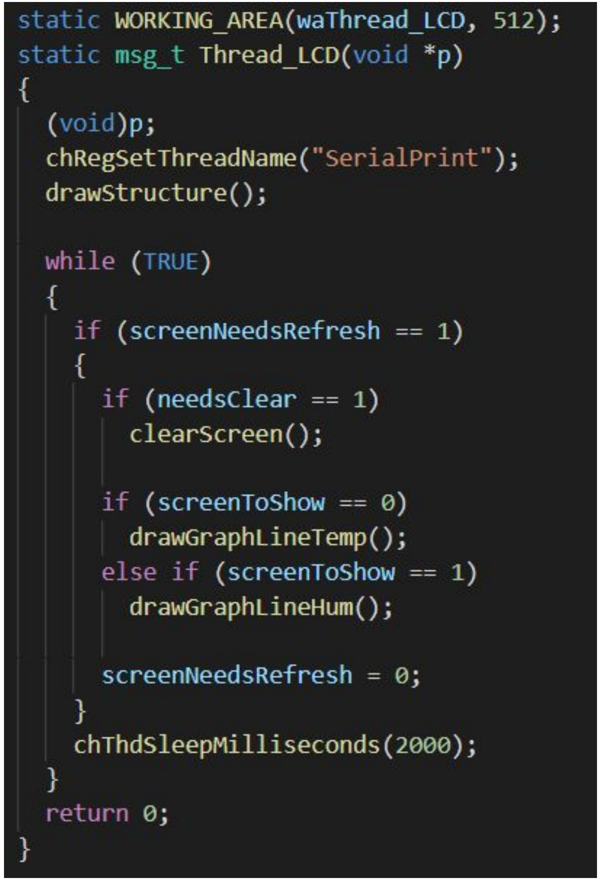
Graphics are made through functions:



```
void lcdPrintf(int x, int y, char text[], int value);
void drawLine(int x1, int y1, int x2, int y2);
void drawBox(int x1, int y1, int x2, int y2);
void clearScreen();
```

This thread does not require semaphore access

LCD is connected to Raspberry through serial connection (Pins TX -> RX)

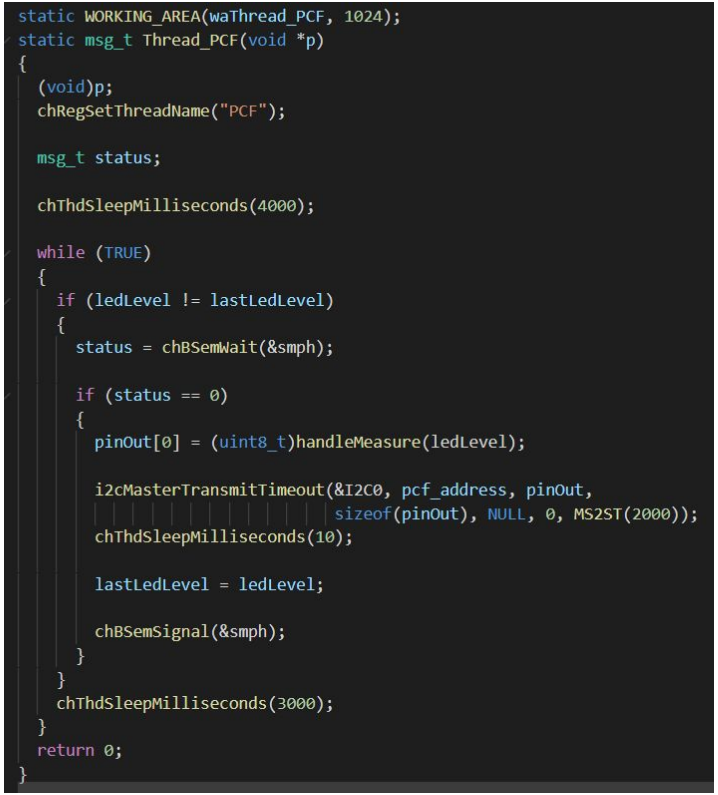


```
static WORKING_AREA(waThread_LCD, 512);
static msg_t Thread_LCD(void *p)
{
  (void)p;
  chRegSetThreadName("SerialPrint");
  drawStructure();

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

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

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

Variables screenToShow controls which graphic to show (Humidity or Temperature) this value changes evey 5 iterations

Auxiliar functions plot the line data which is already stored inside a 2-dimensional array prepared before

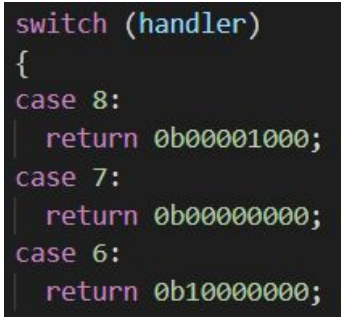
8. PCF 8574



```c
static WORKING_AREA(waThread_PCF, 1024);
static msg_t Thread_PCF(void *p)
{
  (void)p;
  chRegSetThreadName("PCF");

  msg_t status;

  chThdSleepMilliseconds(4000);

  while (TRUE)
  {
    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);
  }
  return 0;
}
```

Accessible through i2c protocol (as a slave with address 0x27)

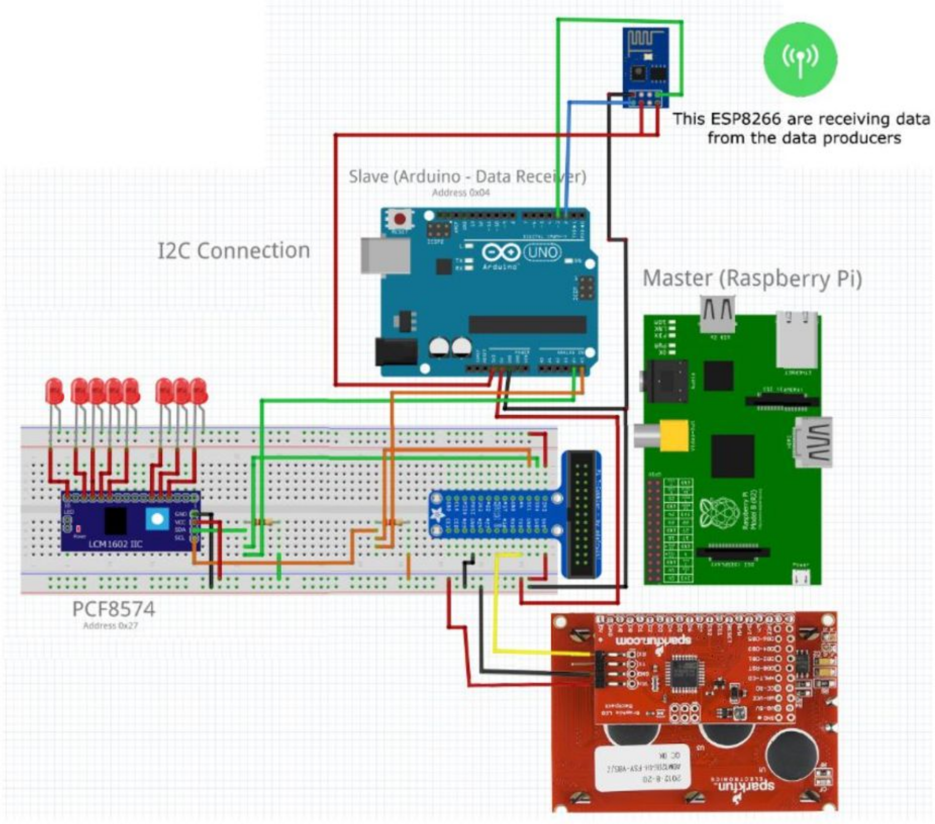
If led level needs to be changed, chibiOS sends a 8bit int to the PCF Indicating which LED's to turn on



```c
switch (handler)
{
case 8:
  return 0b00001000;
case 7:
  return 0b00000000;
case 6:
  return 0b10000000;
```

Example of data sent to PCF8574

Final wiring scheme


This ESP8266 are receiving data from the data producers
Slave (Arduino - Data Receiver)
Address 0x04
I2C Connection
Master (Raspberry Pi)
LCM1602 IIC
PCF8574
Address 0x27

9. Problems encountered

- **Difficult to debug chibiOS**
- **Connections, wires and breadboard problems**
- **Unexpected behaviors when connecting everything together**