Department of Electronic and Telecommunication Engineering

University Of Moratuwa

Final Report of the Project Water Quality Monitoring System.

Index No.	Name
200702H	P.D.G.U.M.B. WEERASINGHE

This assignment is submitted as a partial fulfillment of the module EN2160 – Electronic Design & Realization

2023 July

Contents

1 Project Description

My project idea for Electronic Design Realization is a water quality monitoring system. I named it as “Aguas Claras”. I had a horrible personal experience with consuming water not up to the required quality measures while I was traveling. For a traveler, this could be the worst thing that ever happened since it could ruin the entire journey. Also, I saw continuously bad water conditions in north central province and the water is terribly contaminated with heavy metals due to agrochemicals. Also, in some upcountry areas we are advised not to drink the water out there since that water still contains harmful bacteria because that water cannot be boiled up to 100 degrees Celsius because the atmospheric pressure is lesser in those areas. Existing water quality monitoring systems cannot be used domestically, and they are designed to be used in laboratories, for water utilities, for research authorities etc. So, to bring this matter into more practical aspects and to spread this among people I decided to go with such a project. Here we can use a water quality meter easily to measure the conditions of the water before consuming and since this is portable and efficient this can be taken anywhere. This shows the real time results in an app using IOT, it would be easy to track this by other consumers and get an idea about the water resources before consuming. Aguas Claras is a modular product. It has several numbers of parts in the circuit. Each of these parts will be described hereafter. The system is rechargeable, and it can be charged using an external USB cable. The system takes Total dissolved solid value by the analog TDS sensor module and Takes the Temperature input by DS18B20 temperature sensor. Using those values, the perfect electronic conductivity value is calculated. Then we can check whether it displays the WHO recommended values for the above parameters. TDS value for excellent drinking water is between 50 and 150 ppm and Electronic Conductivity value preferred is below 400 µS cm (0.4 nS cm). Those values will be displayed using an OLED display and at the same time values will be updated in an app immediately. Wi-Fi is used for communication. It enables us to check whether the water sample meets a specific requirement before use for some sort of activities, not only drinking. That is something exceptional. (e.g.: for aquariums, plantations)

The GitHub link to the project: https://github.com/UdithWeerasinghe/EDR

2 Schematic Design of “Aguas Claras”

2.1 Schematic Design

Here is the schematic designed for the project.

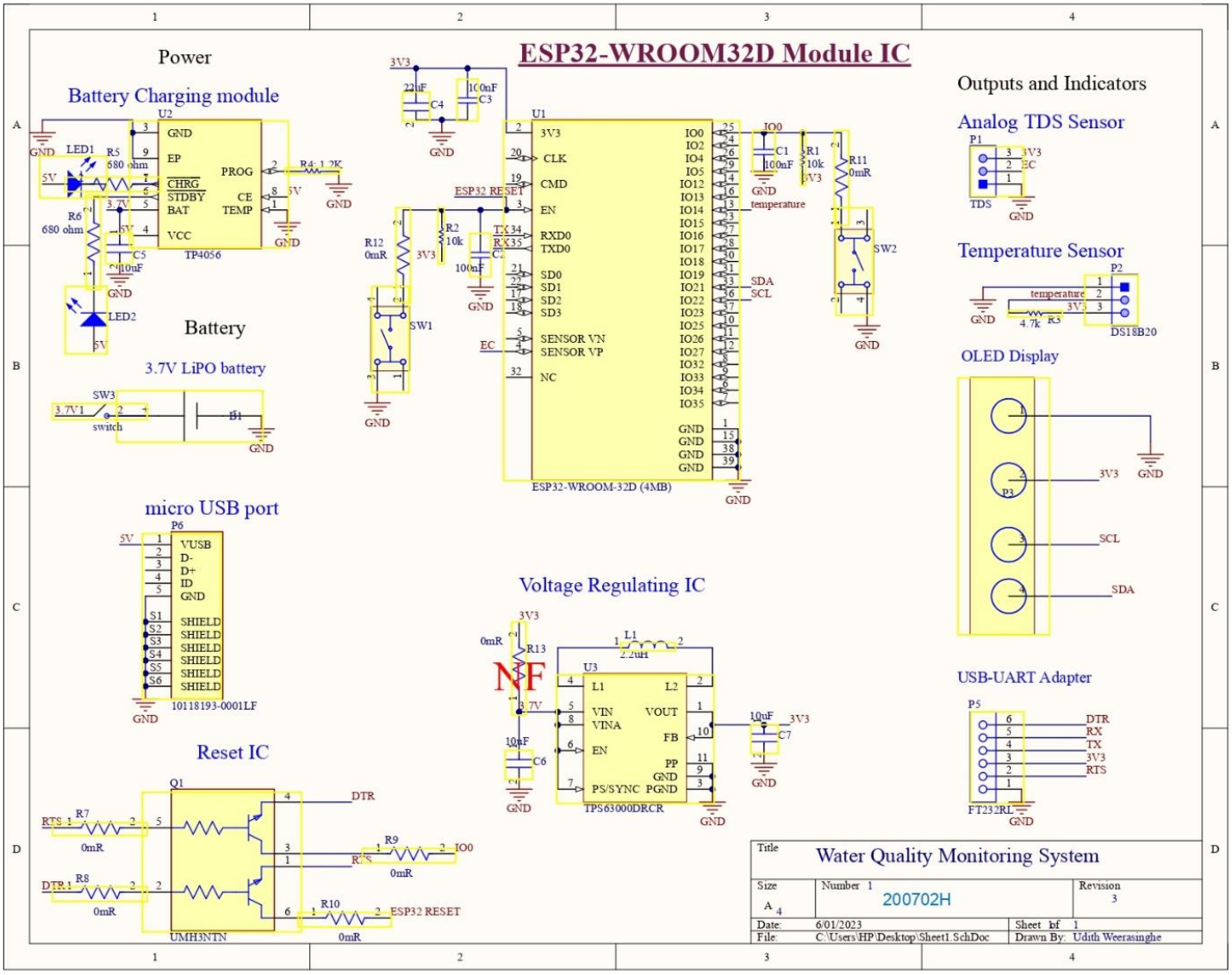


Figure 01: schematic

2.2 BOM of Aguas Claras

No.	Part number	Name	Quantity	Cost ($)
1	SEN0244	Analog TDS sensor	1	15.05
2	DS18B20	Temperature sensor	1	4.43
3	ESP32-WROOM32 Module	MCU chip	1	6.43
4	UM3NTN	Reset IC	1	0.35
5	TPS63000DRCR	Buck-booster IC	1	1.1
6	TP4056	Recharging IC	1	0.07
7	ZOR-25-T-52	0 ohm resistors	7	0.41
8	10118193-0001LF	USB-connector	1	1.06
9	LQM2HPN2R2MG0L	inductor	1	0.42
10	SPC20060	USB cable	1	1
11	FT232RL	USB to UART adapter	1	1.54
12		OLED Display	1	3.06
13		Li-PO Battery	1	4.0
14		LED lights	2	0.1
15		10µF capacitors	2	0.34
16		22µF capacitors	1	0.17
17		100nF capacitors	3	0.51
18		4.7K Resistors	1	0.36
19		680 ohm Resistors	2	0.72
20		10K Resistors	2	0.72
21		1.2K Resistors	1	0.36
22		Push buttons	2	0.38
23		JST 4 pin connectors	2	0.2
24		JST 3 pin connectors	4	0.4
25		Self-tapping screws 20mm	4	0.2
26		Self-tapping screws 10mm	4	0.2
27		Self-tapping screws 5mm	4	0.16
28		Switch	1	0.12
29		Wire tiers	8	0.16

2.3 ESP32-WROOM-32 connections

This subsection of the schematic is focused on the stable power supply to the ESP32 module and other IO connections as recommended in the datasheet. High-frequency noise is filtered using 100nF capacitors. Used an FT232RL module to upload the code. It is needed a USB cable to make the connection with FT232RL and Computer. Since in mass production, the USB interface is used only for charging purposes. Therefore, solder gaps are placed, so once the code is uploaded, the UART interface can be disconnected. Even there is an auto reset IC module used, Also Reset and boot buttons are placed in the circuit as in the development board in case of an emergency using 0 Ω resistors.

2.4 USB to UART Bridge

FT232RL module is used to convert UART protocol to USB communication protocol. A micro-USB port is used to reduce complexity and also due to its popularity.

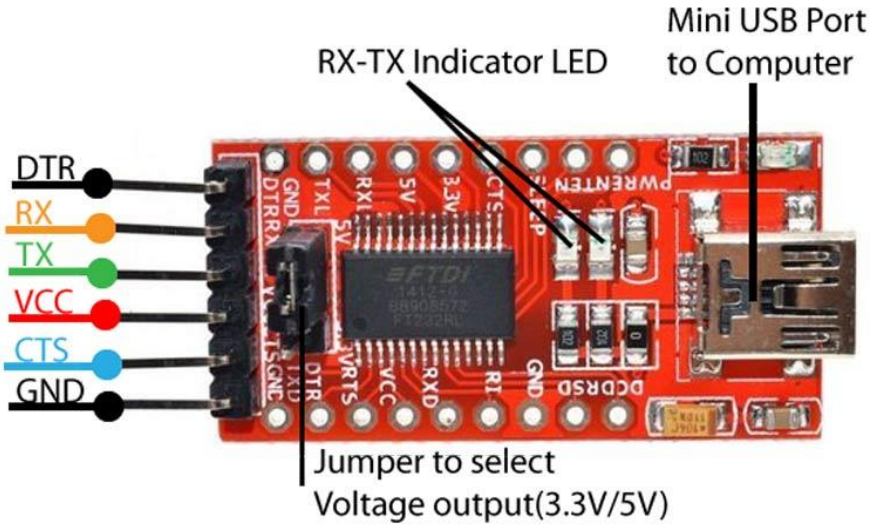


Figure 02: FT232RL

2.5 Power and Voltage regulating ICs.

To supply power, a 3.7V, 1000mAh, Li-PO battery is used. Voltage given by the battery is dropped down to 3.3V(optimal voltage to work ESP32 module) by using a buck-booster IC module (TPS63000DRCR). Related circuitry of that voltage regulating part is designed using two 10Uf capacitors and one 2.2Mh inductor. Which are obtained by calculations and displayed in the datasheet for this particular application. For battery protection it is used an in-built battery protection circuitry with FS8205A Dual N channel power MOSFET and DW01A battery protection IC. For charging, a battery charging circuitry is implemented using TP4056 IC. Required miscellaneous components are added according to the datasheet and calculated results. Also there is a TP4056 module which can be directly plugged instead of this. For the charger it is directly given 5V input & for the microcontroller it is needed 3.3V. That is supplied via buck-booster IC.

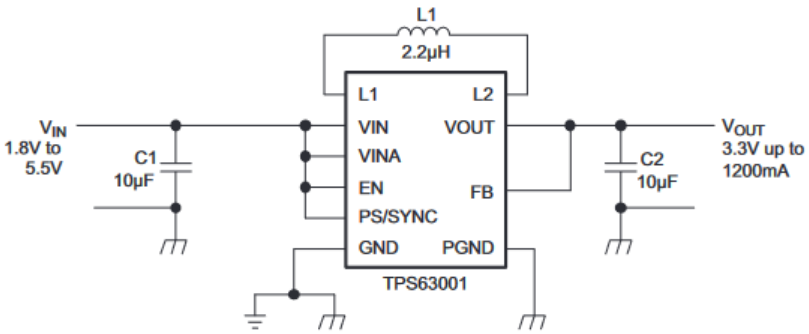


Figure 03 : Voltage regulating part

Calculations:

INDUCTOR SELECTION

To properly configure the TPS6300X devices, an inductor must be connected between pin L1 and pin L2

$$L_1 = \frac{V_{OUT} \times (V_{IN1} - V_{OUT})}{V_{IN1} \times f \times 0.3\ A}$$

$$L_2 = \frac{V_{in2} \times (V_{OUT} - V_{IN2})}{V_{OUT} \times f \times 0.3\ A}$$

f = minimum switching frequency

here V_{OUT} = 3.3 V, V_{IN1} = 3.7V, f is considered as 540kHz then L_1 = 2.2022 µH ≈ 2.2 µH

CAPACITOR SELECTION

To get an estimate of the recommended minimum output capacitance, following equation can be used.

$$C_{OUT} = 5 \times L \times \frac{\mu F}{\mu H}$$

Since L = 2.2µH, C_{OUT} = 11µF ≈10µF capacitor can be used

2.6 Sensors & Indicators

1 Analog TDS sensor

The total dissolved solids (TDS) of a solution, such as salts, minerals, and metals, are measured with a TDS meter. Water from various sources can be compared and its quality can be taken using this measure.

The total dissolved solids value is determined in ppm (mg/L) by multiplying the conductivity of the water by the quantity of dissolved solids in it.

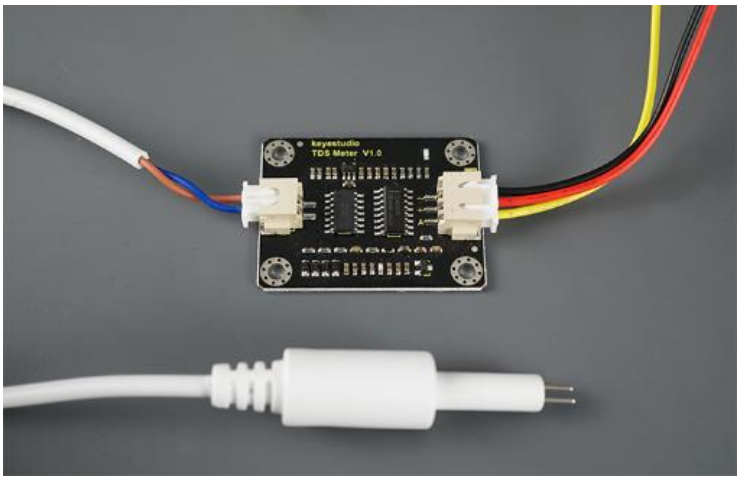


Figure 04: Analog TDS sensor

A TDS sensor is essentially an electrical charge (EC) meter that measures charge in water by inserting two electrodes that are equally spaced apart. The TDS meter interprets the result and converts it to a ppm value.

TDS Sensor working

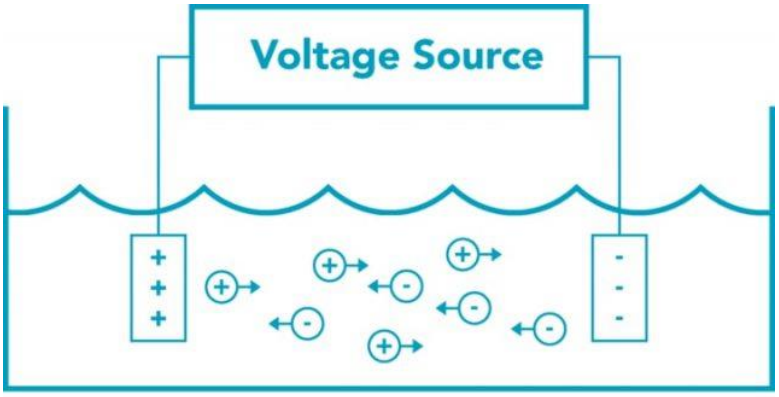


Figure 05: working of a TDS sensor

If the water is pure and free of any soluble compounds, it won't conduct a charge and will have a 0 ppm figure. On the other hand, if the water is heavily dissolved, it will conduct a charge, and the resulting ppm value will be proportionate to the amount of dissolved solids. This is due to the fact that every dissolved solid has an electrical charge, allowing electrical charge to pass across the electrodes.

2 DS18B20 TEMPERATURE SENSOR

The DS18B20 temperature sensor is a one-wire digital temperature sensor. This means that it just requires one data line (and GND) to communicate with Arduino.

It can be powered by an external power supply, or it can derive power from the data line (called "parasite mode"), which eliminates the need for an external power supply.

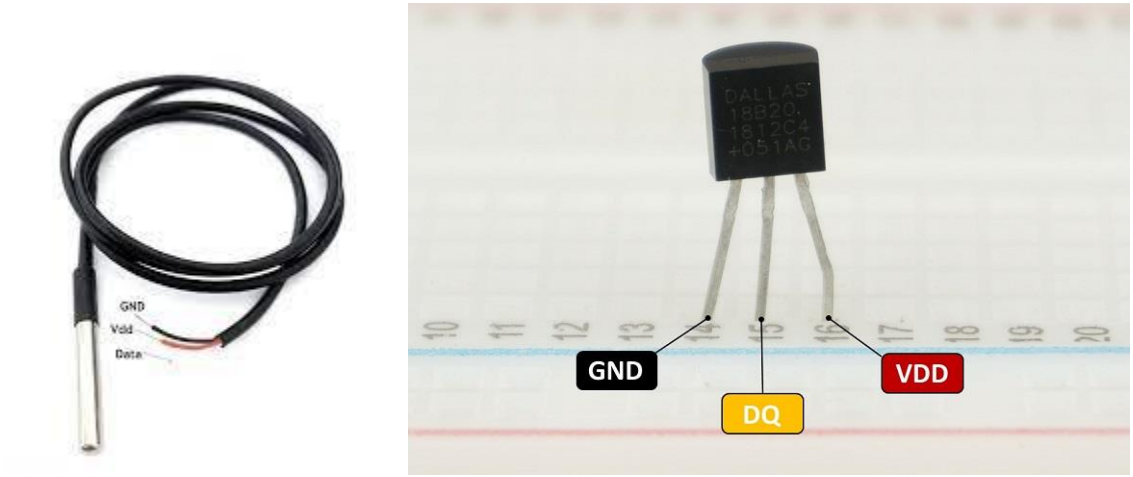


Figure 06 : DS18B20 temperature sensor

3 OLED display

An OLED display is used to indicate sensor outputs and calculated EC value.

Figure 07 : OLED display

3 PCB Design

3.1 PCB Design properties

PCB design properties play a major role in Aguas Claras. Therefore, we used default settings in JLC 2-layer PCB order page. And the design rules of Altium were updated with the manufacturing capabilities of JLC PCB company.

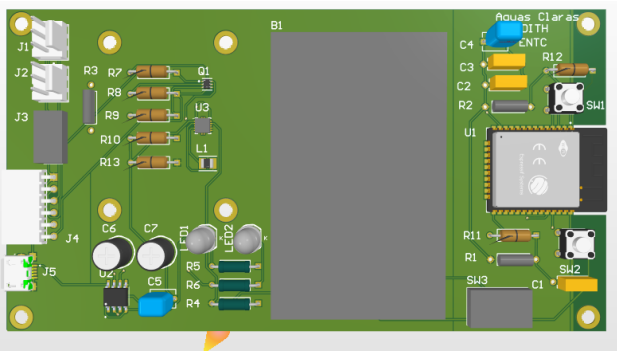


Figure 08 :Top view

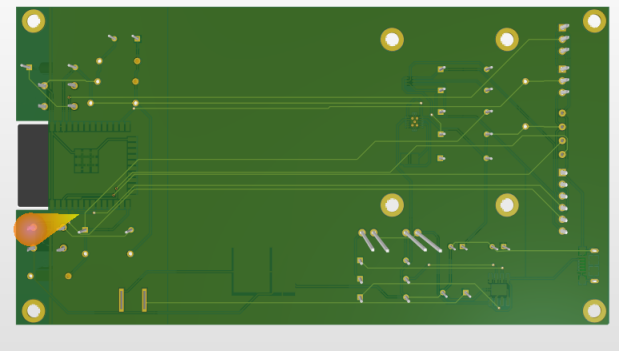


Figure 09 :bottom view

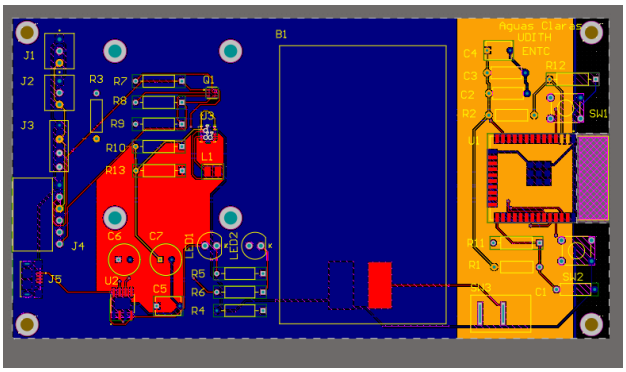
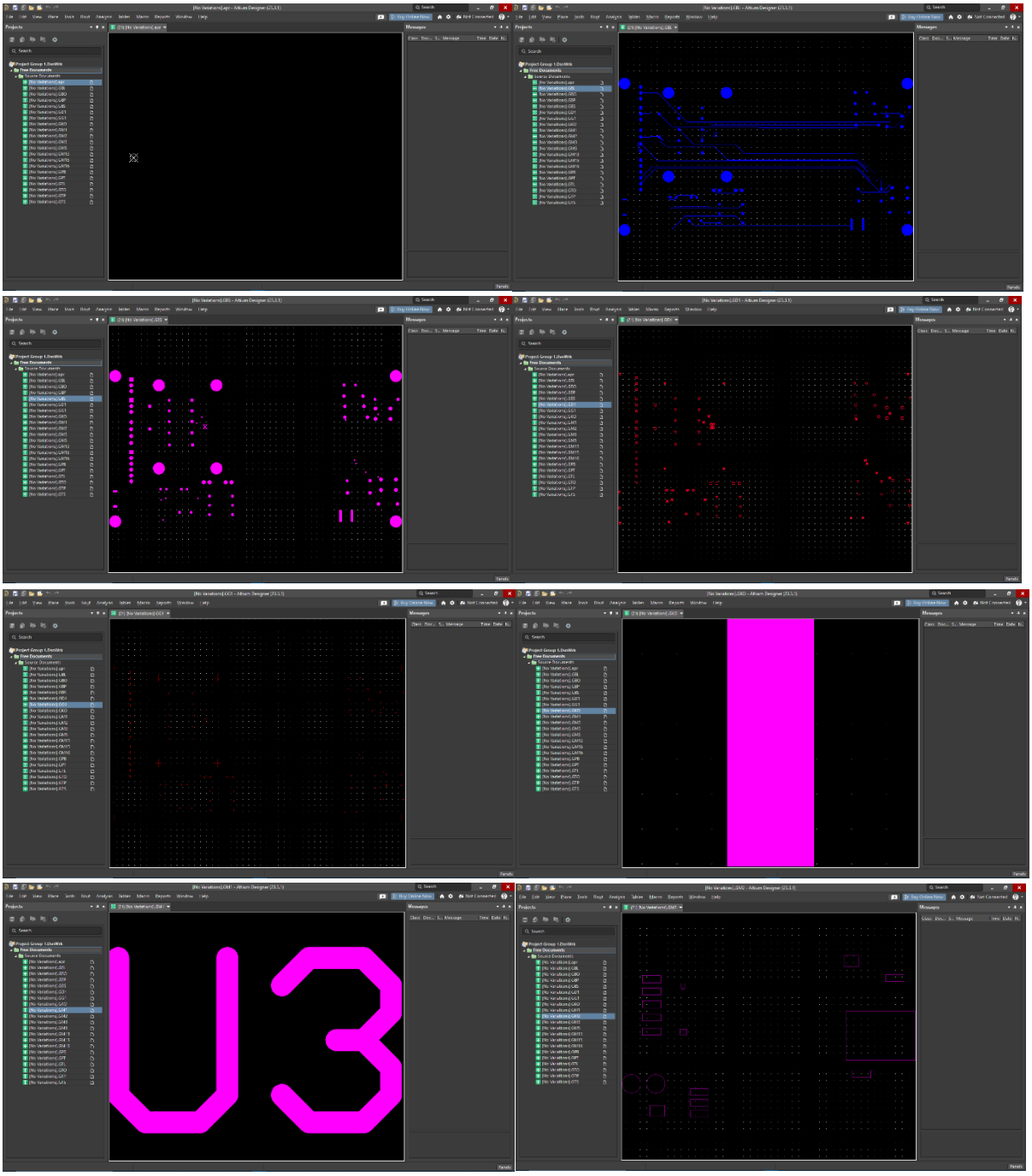
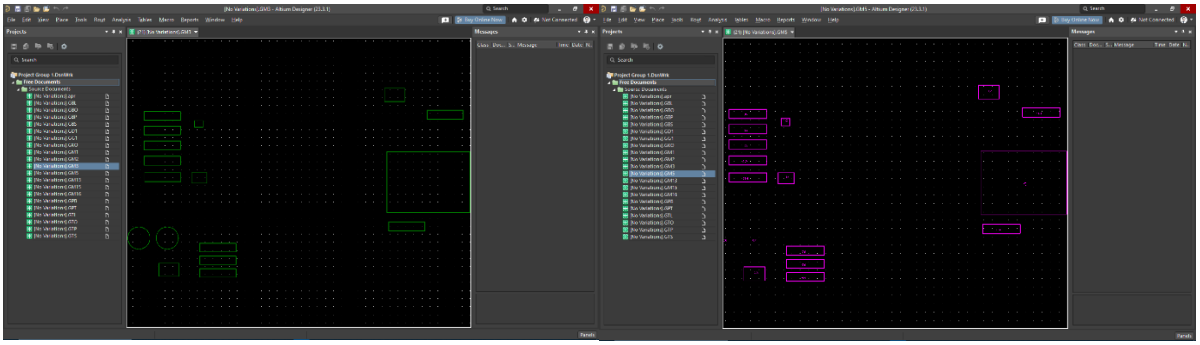
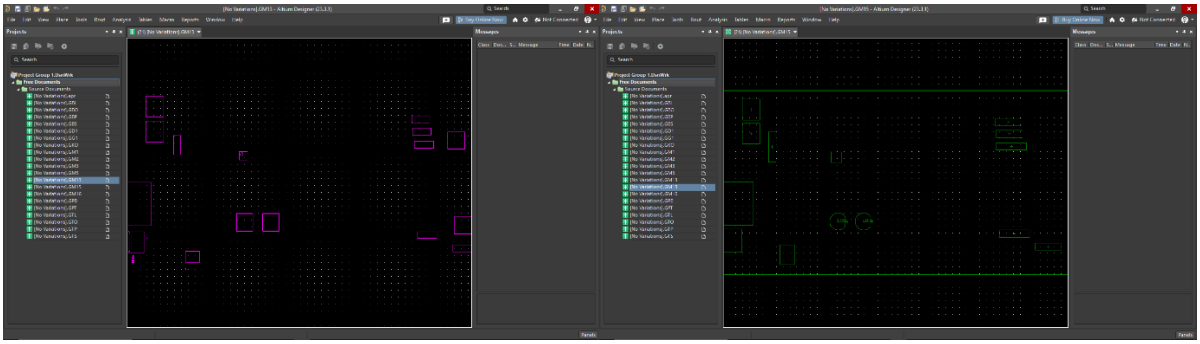


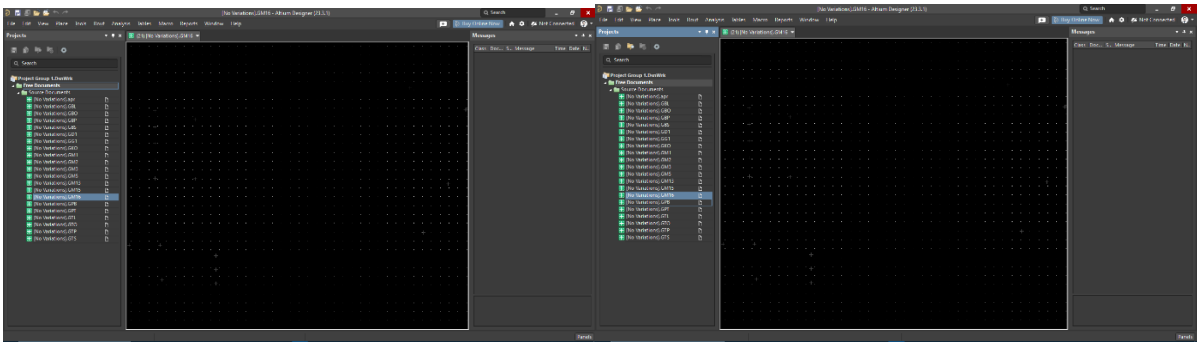
Figure 10 : 2D view

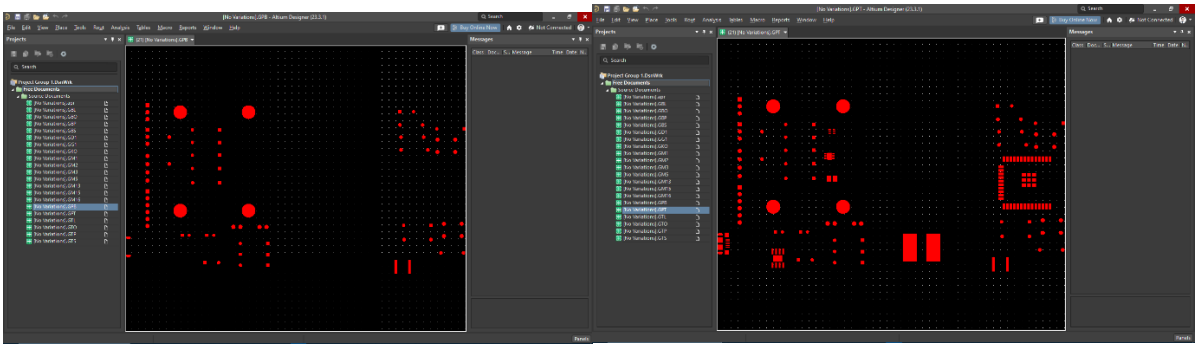
3.2 Gerber Files

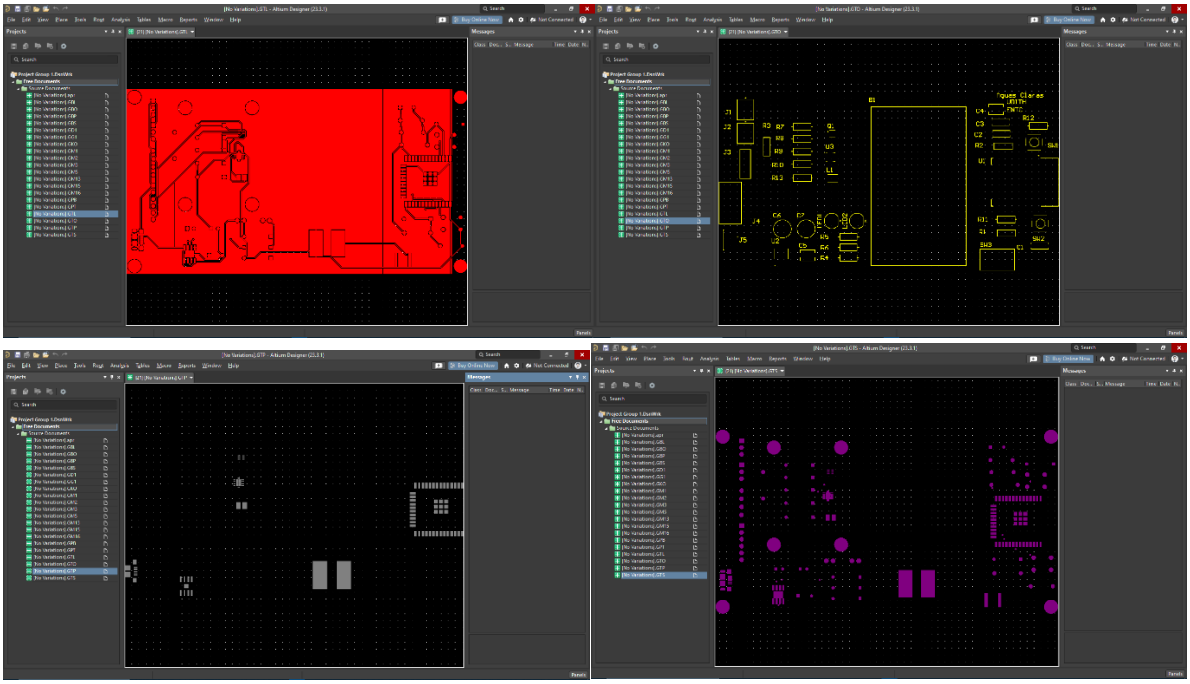






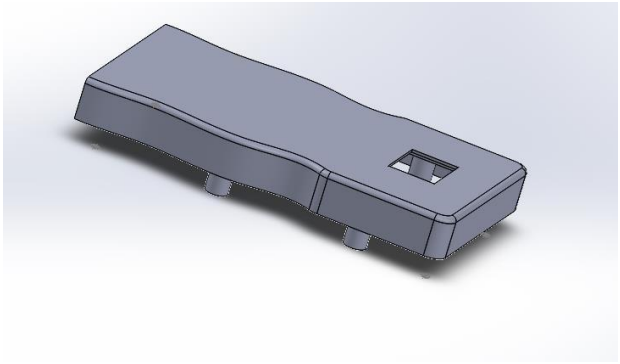


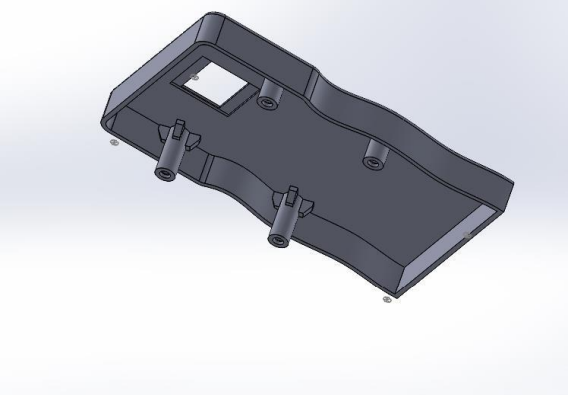


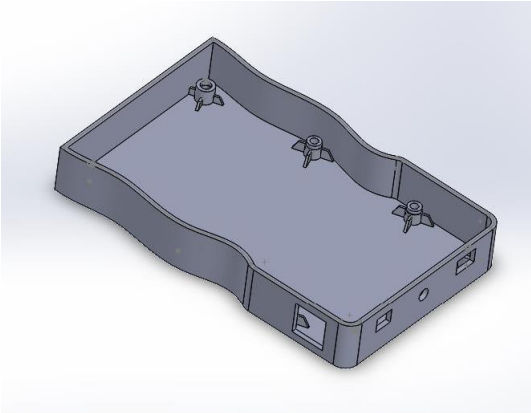


Gerber files link : https://drive.google.com/drive/folders/1vNwvf-gWp29YcGmn42f9EwAKeTcb5edD?usp=drive_link

5 Enclosure Design







Assembly file link :
https://drive.google.com/file/d/1Ru9n3p7hyRnCZq8Vuj8enB1ereABx-x3/view?usp=sharing

5.1 Enclosure Design for Aguas Claras

These factors were the main considerations when designing the enclosure.

1) Must occupy the least amount of space

2) UI should be easy to use.

3) Should appear appealing

As a result, it is choosing its own shape rather than creating an enclosure that is entirely box shaped. Four mounting bosses are used to install the PCB in the enclosure and four mounting bosses are used to attach the enclosure's box and lid parts. Additionally, there are holes necessary for taking out the sensor probes and inserting the charger pin. Also, there is a space left to fit the OLED display It is strictly fit for the PCB because it was designed in SolidWorks using the PCB's step file. The water quality monitoring device is also intended to remain stable on a level surface.

6 Testing & debugging

6.1 Testing for functionality

To check the functionality, it needs to be done a simple procedure. First it needed to be switched on and then put the temperature probe and TDS probe into the liquid/water sample that needs to check for the quality. It is better, if possible, to place them in the center of the container without touching container walls. Then the TDS, EC, temperature values will be displayed in OLED display and moreover the same results will be updated in the app in real time. For drinking water TDS value should be between 50-150 ppm , EC value should be less than 0.4 nScm and the temperature is what the temperature in the sample we are using.

6.2 Errors occurred during uploading codes.

Even though there are not any errors in the code, it took a long time to upload the code. As a solution, the only thing left is to repeatedly check the circuit and code. After a long time, it was uploaded, the reason could be the routing. With that lengths of paths signals reachability to the ESP32 chip can be disturbed.

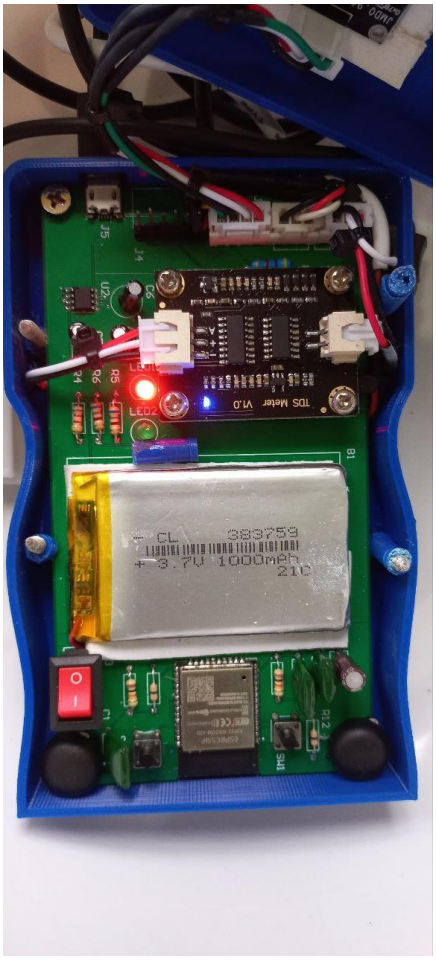
6.3 Errors occurred during work.

While checking for functionality it displayed unreal values for the parameters. Several reasons can affect those. One could be the sensors are not properly tuned. After continuously working with sensors that problem vanished. Also, the memory in the ESP32 might not be enough. So, reducing lines of the code and logically shortening the code leads to reducing those errors.

6.4 Errors occurred in online cloud service.

Sometimes, it didn't show correct values on the dashboard in real time. The reason is the wi-fi connection. After installing a proper wi-fi connection, results are displayed in the dashboard as soon as probes are inserted to the testing sample.

7.Final product



8.Appendix

```
#include <Arduino.h>
#include <Wire.h>
#include <EEPROM.h>
#include <WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Adafruit_ADS1X15.h>
#include <DFRobot_ESP_EC.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ThingSpeak.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define ONE_WIRE_BUS 4                // this is the gpio pin 13 on esp32.
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

DFRobot_ESP_EC ec;
Adafruit_ADS1115 ads;

float voltage, ecValue, temperature = 25;

String apiKey = "Q8R5I05D4N08XJMH";     //  Enter your Write API key from
ThingSpeak

const char *ssid =  "Dialog 4G 106";     // replace with your wifi ssid and wpa2
key
const char *pass =  "5793C7ED";
const char* server = "api.thingspeak.com";
unsigned long channelID = 2101201;

WiFiClient client;


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

```
  EEPROM.begin(32);//needed EEPROM.begin to store calibration k in eeprom
  ec.begin();
  sensors.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }
  delay(2000);
  display.clearDisplay();

  Serial.println("Connecting to ");
  Serial.println(ssid);


  WiFi.begin(ssid, pass);
  ThingSpeak.begin(client);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  //ThingSpeak.begin(client);
}


void loop()
{
  voltage = analogRead(A0); // A0 is the gpio 36
  sensors.requestTemperatures();
  temperature = sensors.getTempCByIndex(0);  // read your temperature sensor to
execute temperature compensation
  ecValue = ec.readEC(voltage, temperature); // convert voltage to EC with
temperature compensation

  Serial.print("Temperature:");
  Serial.print(temperature, 2);
  Serial.println("ºC");

  Serial.print("EC:");
  Serial.println(ecValue*1000, 2);
```

```
  display.setTextSize(2);
  display.setTextColor(WHITE);

  display.setCursor(0, 10);
  display.print("T:");
  display.print(temperature, 2);
  display.drawCircle(85, 10, 2, WHITE); // put degree symbol ( ° )
  display.setCursor(90, 10);
  display.print("C");

  display.setCursor(0, 40);
  display.print("EC:");
  display.print(ecValue *1000 , 2);
  display.display();
  delay(1500);
  display.clearDisplay();

  ec.calibration(voltage, temperature); // calibration process by Serail CMD

  if (client.connect(server, 80))  //   "184.106.153.149" or api.thingspeak.com
  {

    String postStr = apiKey;
    postStr += "&field1=";
    postStr += String(temperature, 2);
    postStr += "&field2=";
    postStr += String(ecValue*1000, 2);
    postStr += "\r\n\r\n";
    delay(500);

    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
    client.print(postStr.length());
    client.print("\n\n");
    client.print(postStr);
    delay(500);

  }
  client.stop();
}
```

References

1. https://wiki.dfrobot.com/Gravity__Analog_TDS_Sensor___Meter_For_Arduino_SKU__SEN0244
2. https://www.apogeeweb.net/circuitry/ds18b20-working-principle.html
3. all datasheets : https://drive.google.com/drive/folders/1_chYkdKy_TTjzD2sAsCUnzpHw-HOj6el?usp=drive_link
4. https://youtu.be/x9GfxgkEpXg
5. https://youtu.be/41L3wW0PhGo
6. https://youtu.be/9nzlOQkxXgU