



Zewail City of Science and Technology
Control Systems
[CIE 405]
4th Year – Spring
2023

Course: Electric Machines

Code:

CIE 405

Report on:

IOT Smart Plant Monitoring and Irrigation System Project – Phase 3

Submitted To:

Dr. Ahmed S. Abd-Rabou

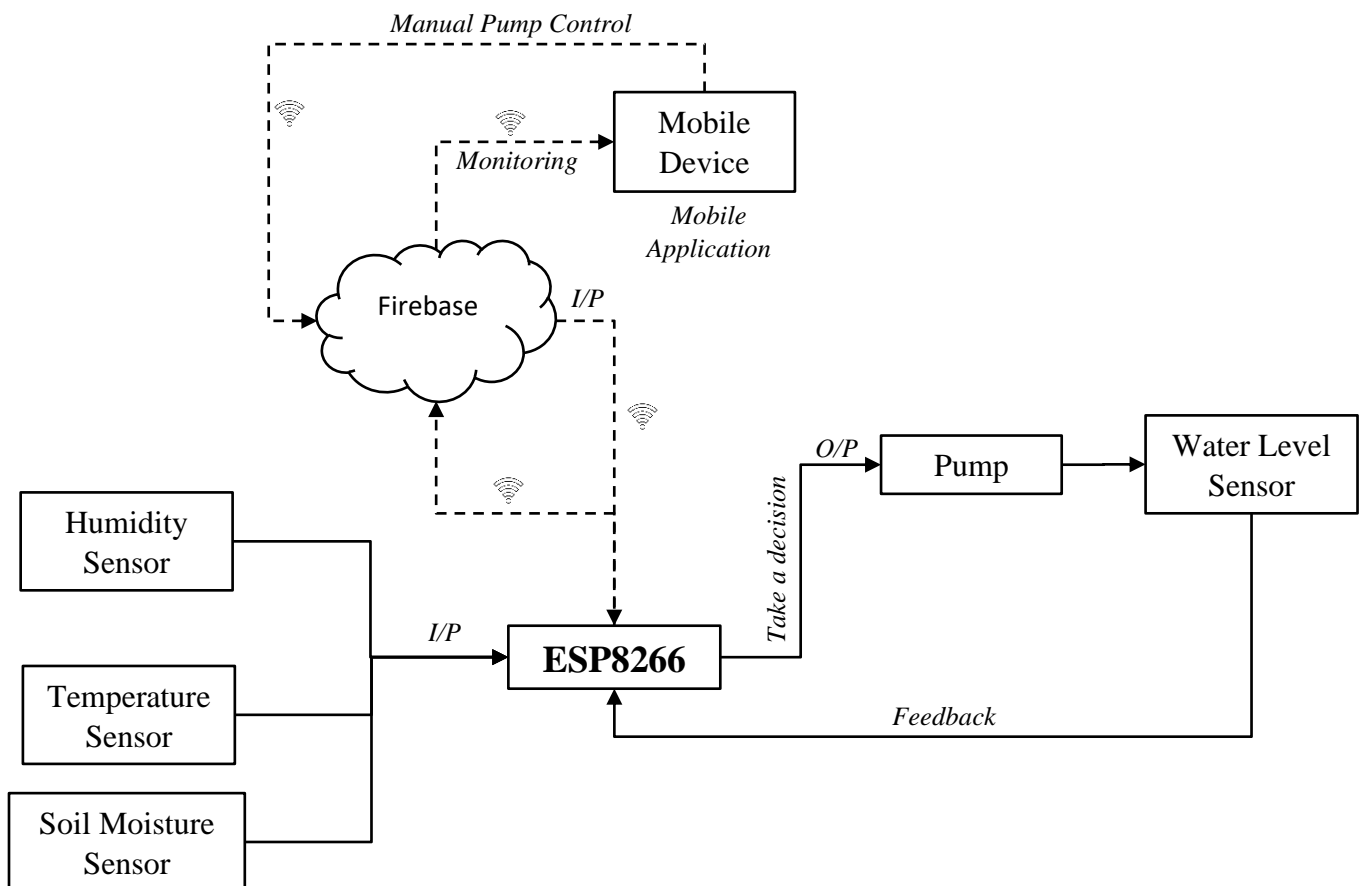
Names:	Mario Youchia, Mahmoud Sheliel, Yahya Mohamed, Ahmed Abdelgawad
IDs:	201902086, 201900697, 202000776, 201901249
Date:	22-May-23

Introduction:

An IoT smart plant monitoring and irrigation system is a technology that helps people monitor and water their plants using the internet. The system includes sensors that measure factors like soil moisture, temperature, and humidity, and then sends that information to the firebase that can be accessed online. Users can then check on their plants remotely, and the system can even water the plants automatically based on the data collected by the sensors.

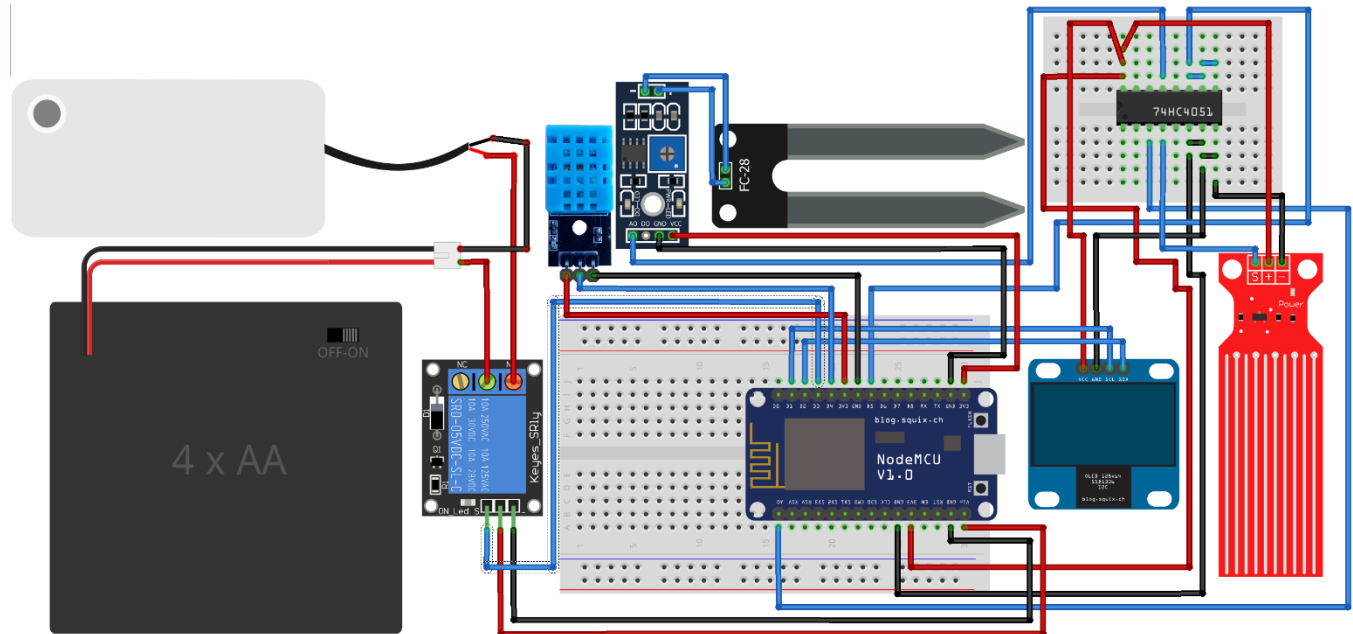
There are many different types of smart plant monitoring and irrigation systems available, ranging from DIY projects like this project that use basic components, to commercial products that are designed specifically for agriculture. These systems can be a great way to keep plants healthy, even if the user is not able to be physically present to care for them.

Functional Block Diagram:


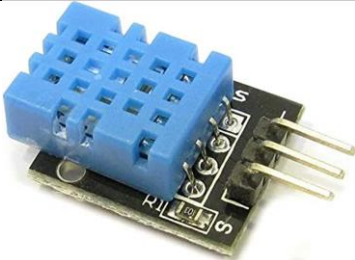


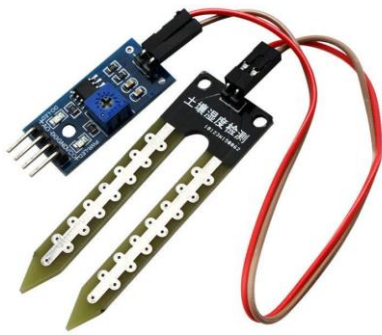
The system begins by measuring the temperature, humidity of the environment, and the soil moisture through the corresponding sensors. These readings are used by the ESP8266 NodeMCU board to take a decision whether to open the pump or not. The water level sensor checks continuously if the water level exceeds some value, if it exceeds this value, the pump will be turned off. Also, the ESP8266 sends the sensors readings to the firebase, so the mobile application can read these values and monitor the irrigation system.

Schematic Diagram:



Components:

Component	Function
 <p>ESP8266 NodeMCU board</p>	It is the microcontroller used to control the system, sends, and receives data to and from the firebase enabling real-time monitoring, and manual control.
 <p>DHT11 Sensor</p>	It is used to measure the humidity and temperature of the environment.



Soil Moisture Sensor

It is used to measure the moisture of the soil.



Relay Module 5V – 1 Channel

It is used to turn ON or OFF the pump.



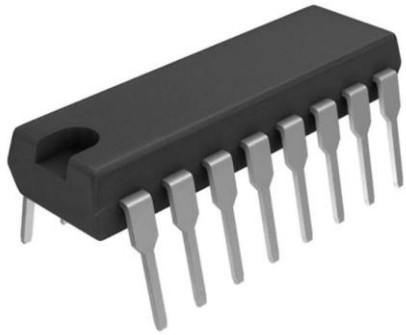
4x AA Battery

It is used to deliver power to the pump when the relay's control signal is high.



DC pump – 6V

It is used to irrigate the plant.



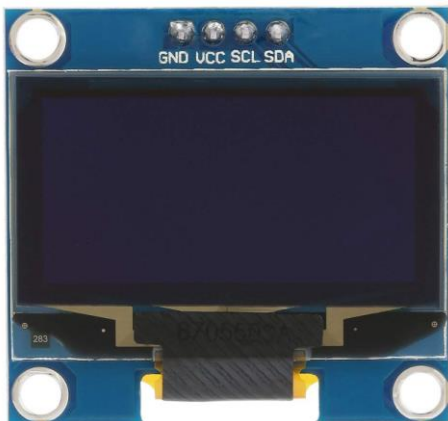
CD74HC4051 – 8 Channels
Multiplexer/Demultiplexer

It is used to enable the multi-use of the single analog pin available in the NodeMCU board.



Water Level Sensor

It is used to measure the level of the water in the plant.



OLED 128x64 pixels

It is used to monitor the readings of sensors.

List of Components:

Component	Price of One Unit	Quantity	Component Price
4x AA battery	20	1	20
Battery holder	15	1	15
Pump + Hoses	85	1	85
Breadboard	25	1	25
ESP8266	200	1	200
Relay	50	1	50
DHT11	50	1	50
Soil Moisture Sensor	80	1	80
Tie Cables	1	10	10
M3x10 Bolts	1	20	20
M3 Nuts	1	20	20
Plant	25	1	25
Spacers	2.5	4	10
Laser Cut	100	1	100
Jumpers	1	15	15
Multiplexer	45	1	45
Water Level Sensor	25	1	25

Expected Budget: 795 EGP

Multiplexer:

Using a multiplexer with the ESP8266 that has only one analog input pin can provide several benefits, as follow:

- Firstly, it can allow for the expansion of the number of analog inputs available to the ESP8266, which is useful when the project requires multiple analog sensors to be monitored.
- Secondly, it can help to reduce the number of pins required for interfacing with the ESP8266, which is important in space-constrained designs.
- Thirdly, it can simplify the wiring and reduce the complexity of the code required to read from multiple sensors.

Additionally, by using the Arduino UNO with six analog pins and connecting it to the ESP8266, it allows for the flexibility of using the more powerful Arduino board to handle the analog input while still being able to communicate with the ESP8266 wirelessly. However, this requires an additional board and can be more complex to set up.

In contrast, using a multiplexer with the ESP8266 NodeMCU can provide a more compact and cost-effective solution, as well as lower power consumption. It can also be more

suitable for certain applications where space is limited and a smaller number of analog inputs are required.

Using a multiplexer with the ESP8266 to directly read from sensors can eliminate the need for data transfer between the two devices, potentially reducing overall latency and increasing the speed of data acquisition.

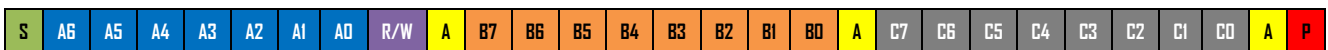
According to the table belowⁱ, the total delay in reading output from multiplexer in worst case after changing the select pins would be $90 + 340 + 340 \text{ ns} = 770 \text{ ns}$.

6.7 Switching Characteristics

over recommended operating free-air temperature range (unless otherwise noted) (see [Figure 9](#))

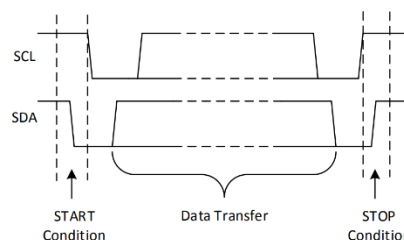
PARAMETER	FROM (INPUT)	TO (OUTPUT)	LOAD CAPACITANCE	V _{EE}	V _{CC}	T _A = 25°C			T _A = -55°C TO 125°C			UNIT
						MIN	TYP	MAX	MIN	TYP	MAX	
t _{pd}	IN	OUT	C _L = 15 pF	0 V	5 V	4						ns
			C _L = 50 pF		2 V	60			90			
					4.5 V	12			18			
					6 V	10			15			
					-4.5 V	4.5 V	8			12		
t _{en}	ADDRESS SEL or E	OUT	C _L = 15 pF	0 V	5 V	19						ns
			C _L = 50 pF		2 V	225			340			
					4.5 V	45			68			
					6 V	38			57			
					-4.5 V	4.5 V	32			48		
t _{dis}	ADDRESS SEL or E	OUT	C _L = 15 pF	0 V	5 V	19						ns
			C _L = 50 pF		2 V	225			340			
					4.5 V	45			68			
					6 V	38			57			
					-4.5 V	4.5 V	32			48		
C _I	Control		C _L = 50 pF				10			10	pF	

In the case of sending data from Arduino UNO to ESP8266 using the I2C protocolⁱⁱ, we would need at least 29 clock cycles.



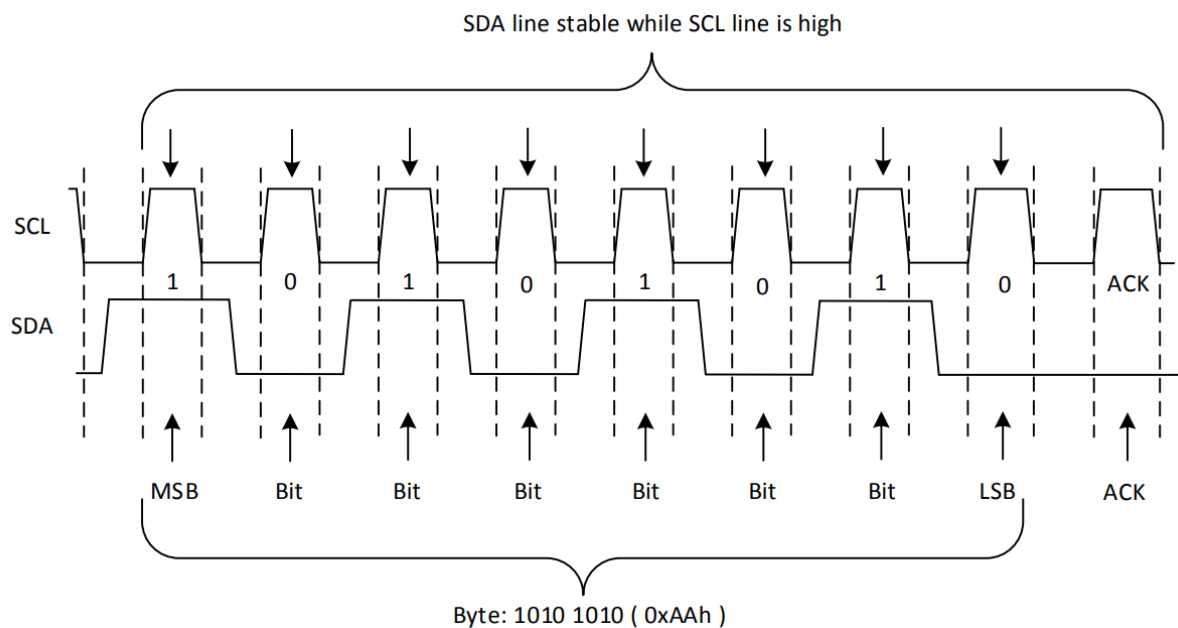
- S: Start Condition
- P: Stop Condition

The start condition is a high-to-low transition on the SDA line while the SCL is high, while the stop condition is a low-to-high transition on the SDA line while the SCL is high.

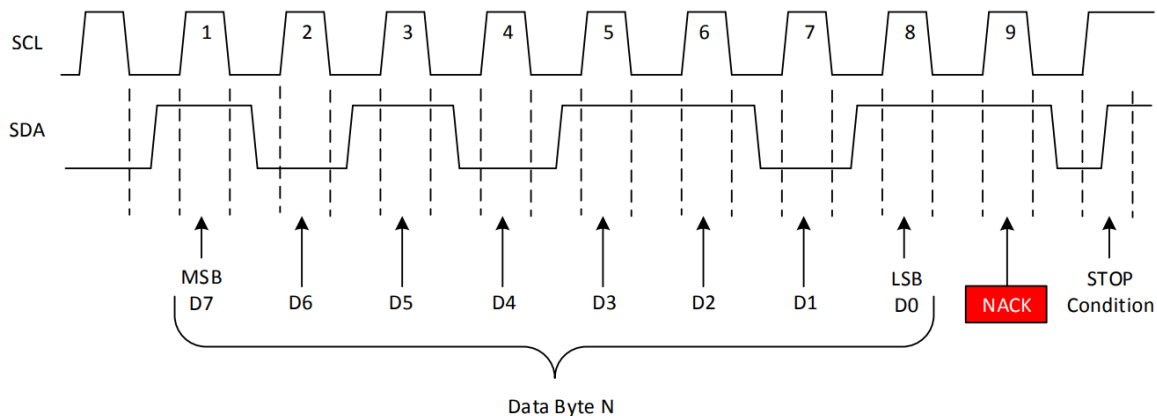


- A6-A0 bits are the address bits of the slave.
- R/W is the read or write bit.
- A is the acknowledgement bit.

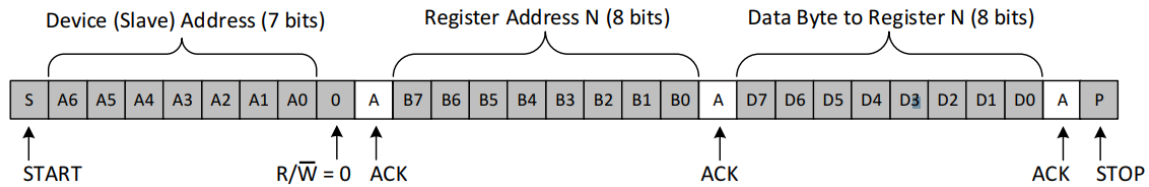
The address of the slave begins with the MSB, and ends with the LSB, each bit is transferred during one clock cycle. The R/W bit indicates whether the master will read from the slave or write to it. If it is high, it means that the master will read from the slave, and if it is low, it means that the master will write to the slave. The A bit is an acknowledgement sent by the slave to indicate that it has received the preceding byte successfully.



- In case of ACK: the SDA line during the corresponding clock cycle will be pulled down, while in NACK it will be pulled high, then the sending process will be stopped.



- B7-B0: are the bits of the register address from which the data will be read, or to which the data will be written.
- C7-C0: are the bits of the read/written data.



Using the default I2C clock speed, which is 100kHz. Then, one clock cycle is equivalent to 10 microseconds. Therefore, the 29 clock cycles take 290 microseconds.

$$290\mu s \gg 770ns$$

This is 376.6 times in the best case greater than the worst-case latency of using a multiplexer with an ESP8266 NodeMCU.

Processing Power of Arduino UNO vs Processing Power of ESP8266 NodeMCU:

The processing power of the ESP8266ⁱⁱⁱ is higher than that of the Arduino Uno^{iv}. The ESP8266 is equipped with a 32-bit RISC processor, running at 80 MHz, while the Arduino Uno is equipped with an 8-bit AVR processor, running at 16 MHz. Additionally, the ESP8266 has more memory than the Arduino Uno, with up to 96 KB of RAM and 4 MB of flash memory available. This means that the ESP8266 can handle more complex tasks and programs than the Arduino Uno.

In summary, the primary difference between a 32-bit RISC (Reduced Instruction Set Computing) processor and an 8-bit AVR processor is their word size and instruction set architecture, which can affect their speed and performance in different types of applications.

■ **ATMega328P** Processor

■ **Memory**

- AVR CPU at up to 16 MHz
- 32KB Flash
- 2KB SRAM
- 1KB EEPROM

3.1. CPU, Memory, and Flash

3.1.1. CPU

The ESP8266EX integrates a Tensilica L106 32-bit RISC processor, which achieves extra-low power consumption and reaches a maximum clock speed of 160 MHz. The Real-Time Operating System (RTOS) and Wi-Fi stack allow 80% of the processing power to be available for user application programming and development. The CPU includes the interfaces as below:

Ram:

According to our current version of SDK, SRAM space available to users is assigned as below.

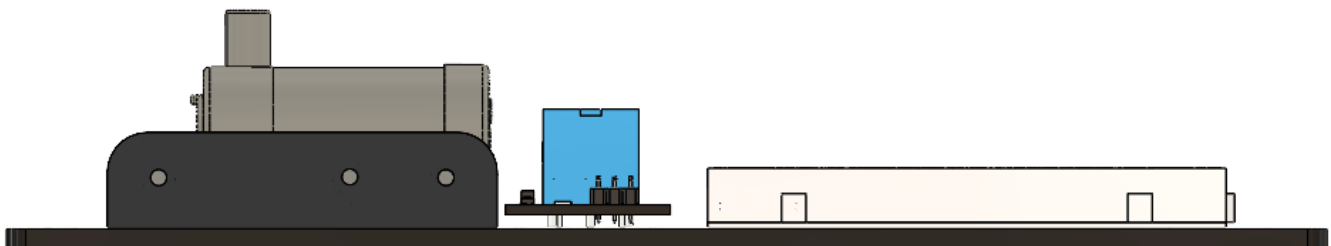
- RAM size < 50 kB, that is, when ESP8266EX is working under the Station mode and connects to the router, the maximum programmable space accessible in Heap + Data section is around 50 kB.

∴ The processing power of ESP8266 is higher than that of Arduino UNO, and it takes much more time to send data from Arduino UNO to ESP8266 using the I2C communication protocol.

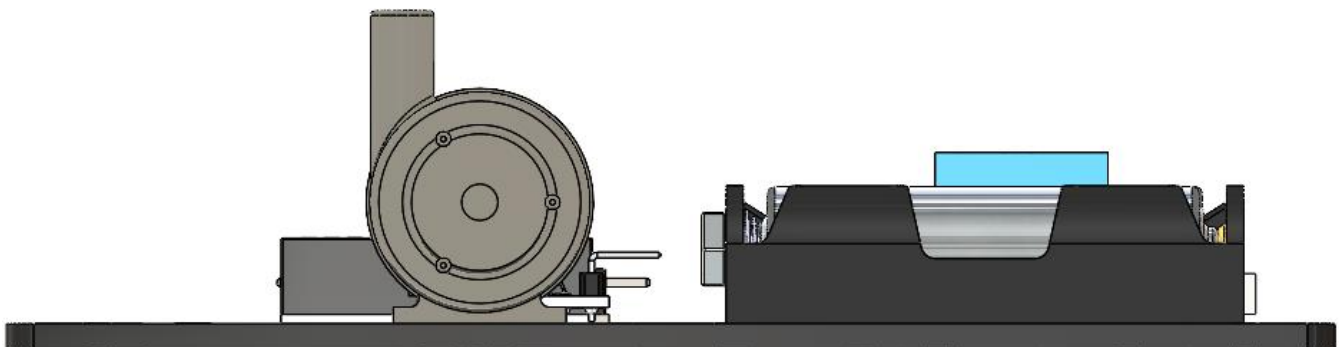
∴ It is much wiser to use a multiplexer with ESP8266, which is much cheaper, and smaller than Arduino UNO board.

CAD Design:

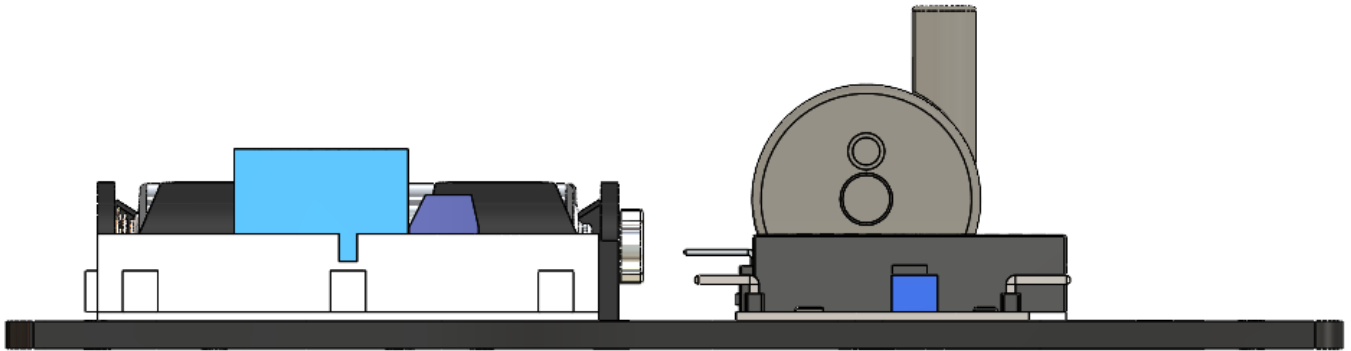
Elevation View:



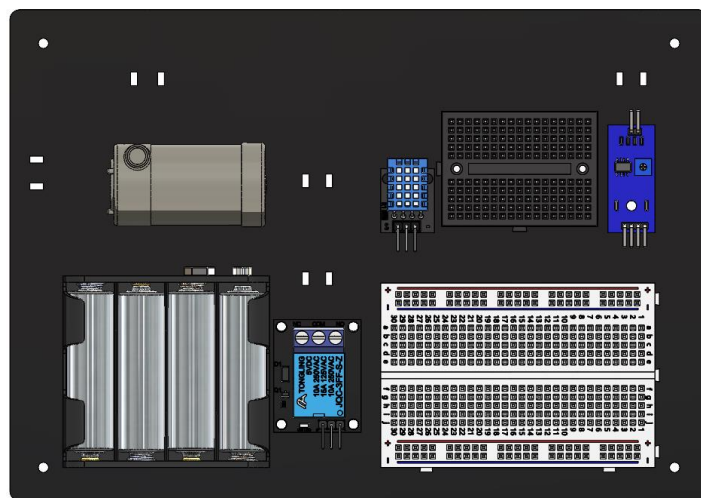
Left Side View:



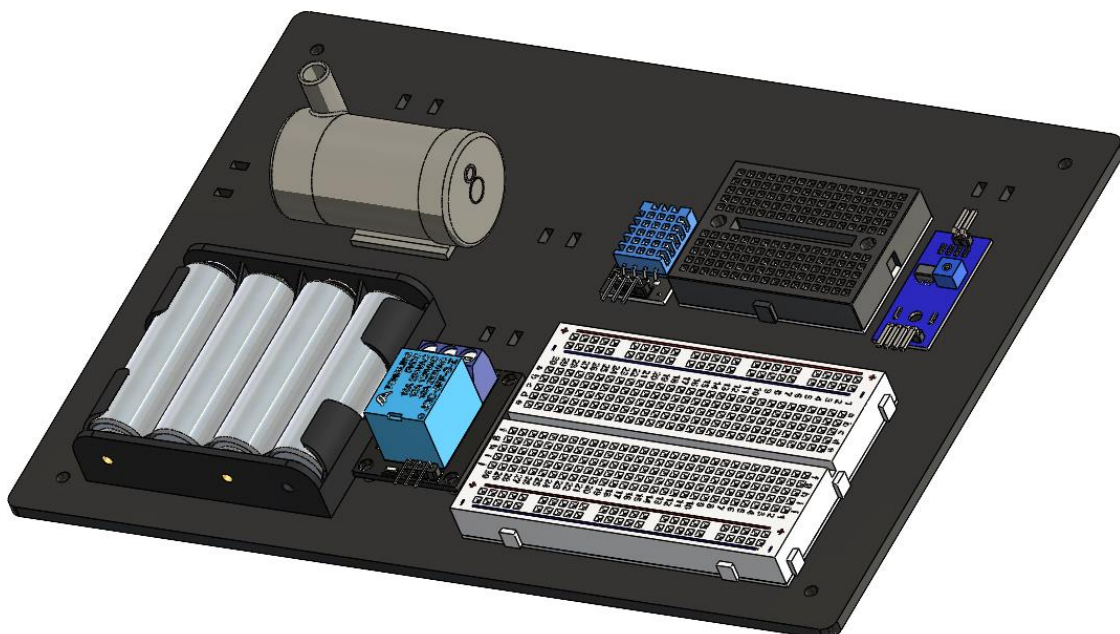
Right Side View:



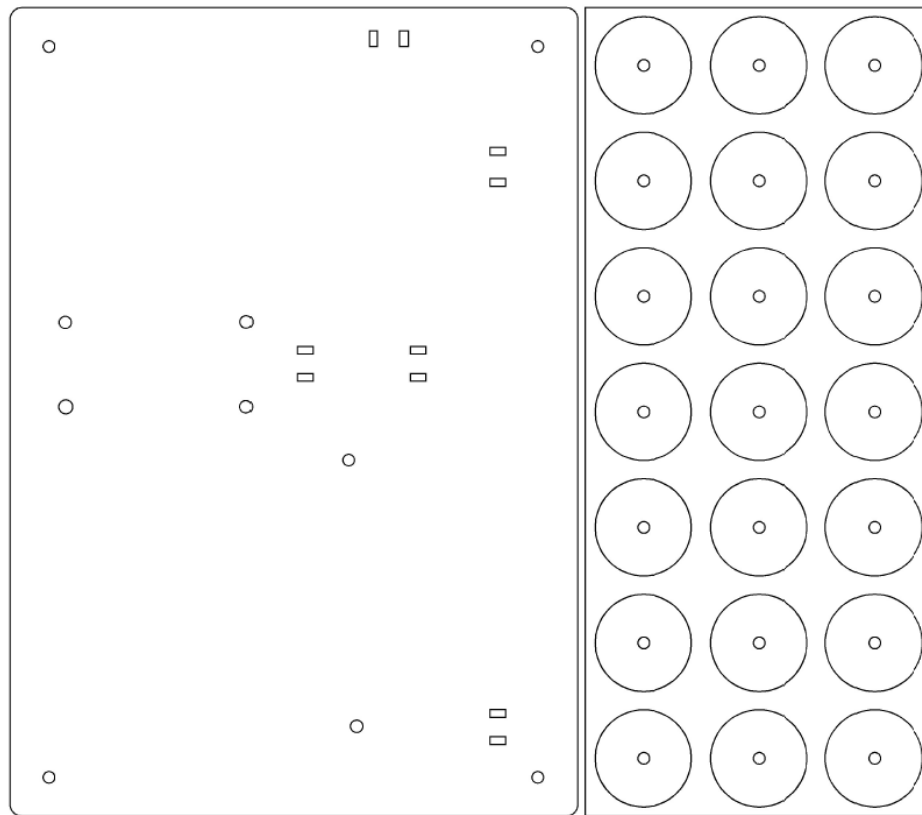
Plan View:



Inclined View:

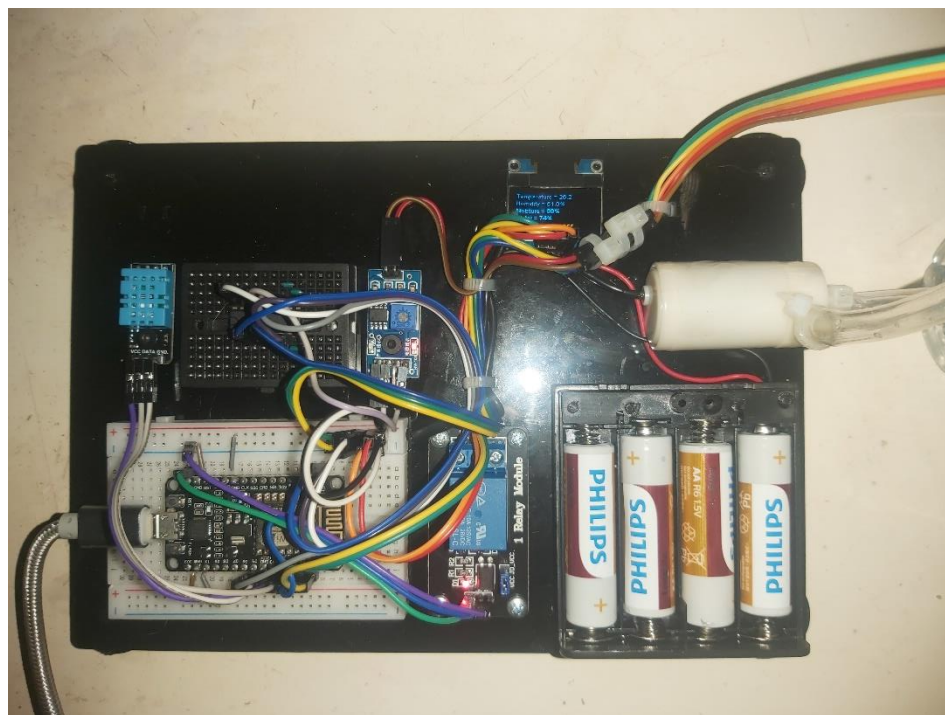


Laser Cut File:



Assembly of Complete System:

Setup:



Plant:



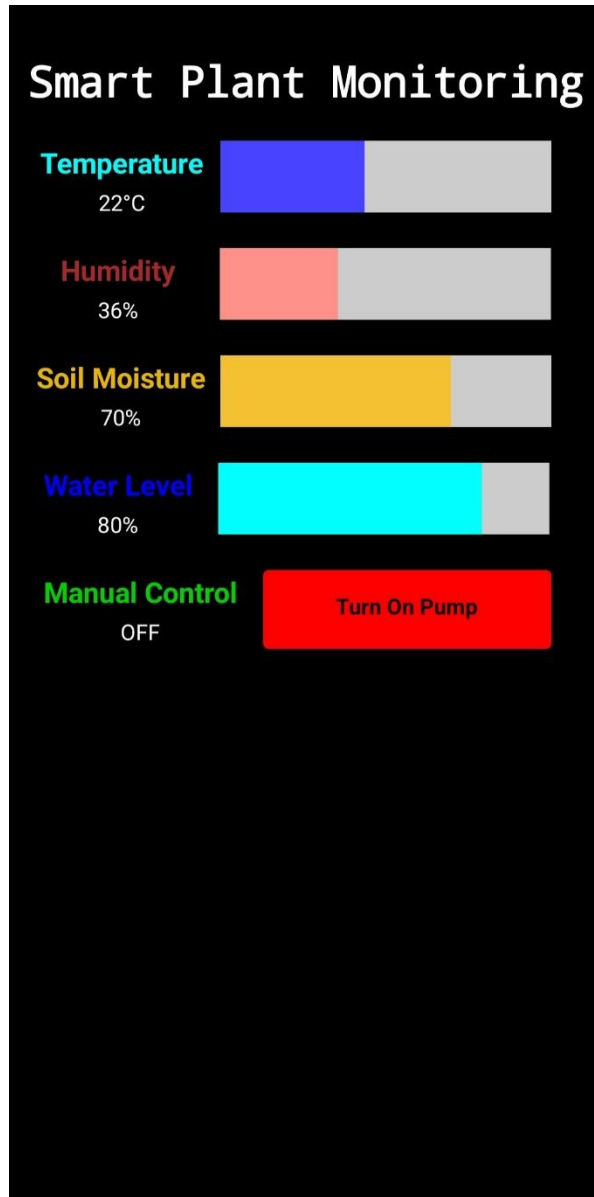
Whole System:



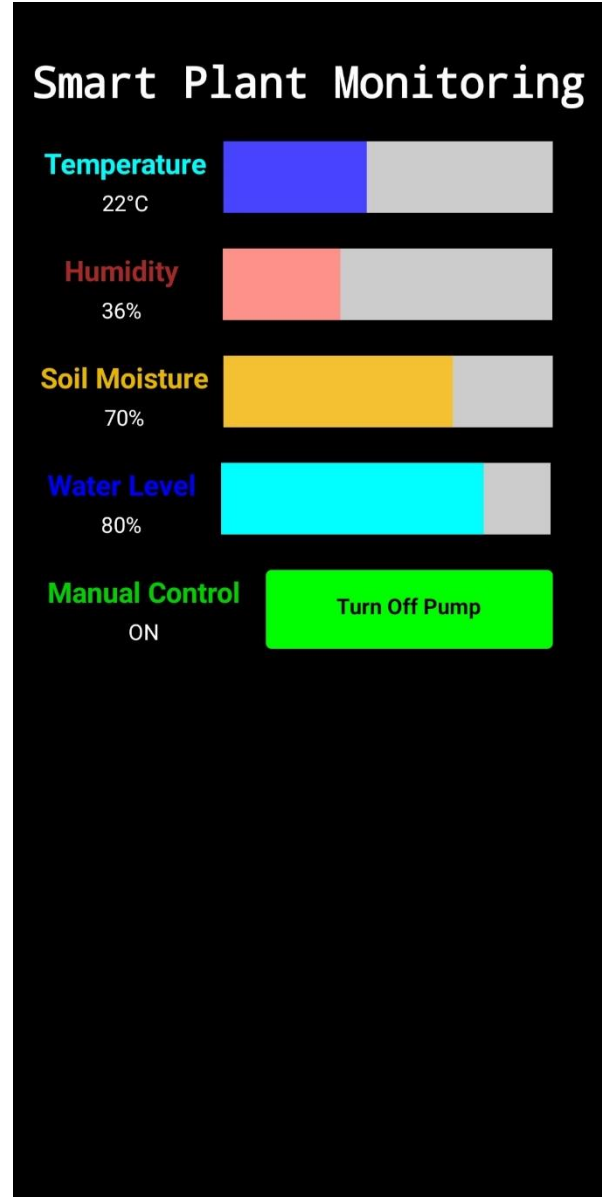
Mobile Application:

GUI:

Manual Control is OFF:

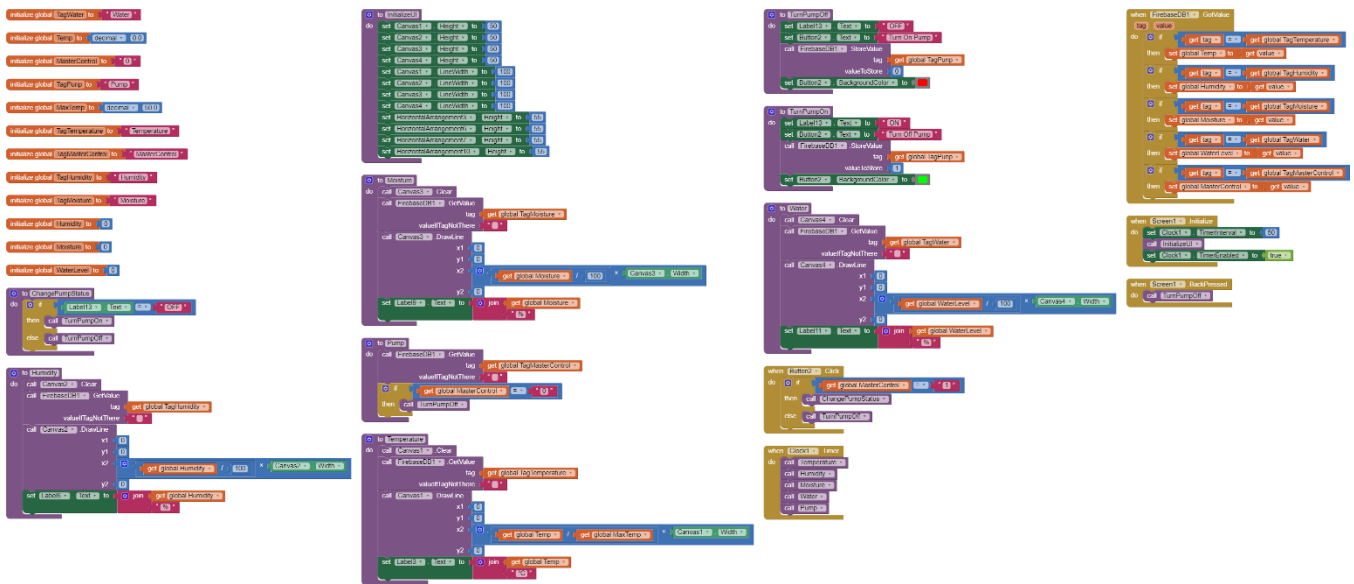


Manual Control is ON:



This mobile application is created by MIT App Inventor^v

Blocks:



This mobile application reads the temperature, humidity, soil moisture, and water level sensor from Firebase. Then, update the progress bars depending on the values retrieved from Firebase.

Here is a screenshot from the firebase:

<https://smart-plant-monitoring-769a7-default-rtdb.firebaseio.com>

<https://smart-plant-monitoring-769a7-default-rtdb.firebaseio.com/>

Smart_Plant_Monitoring

Humidity: 63

MasterControl: "1"

Moisture: 65

Pump: "0"

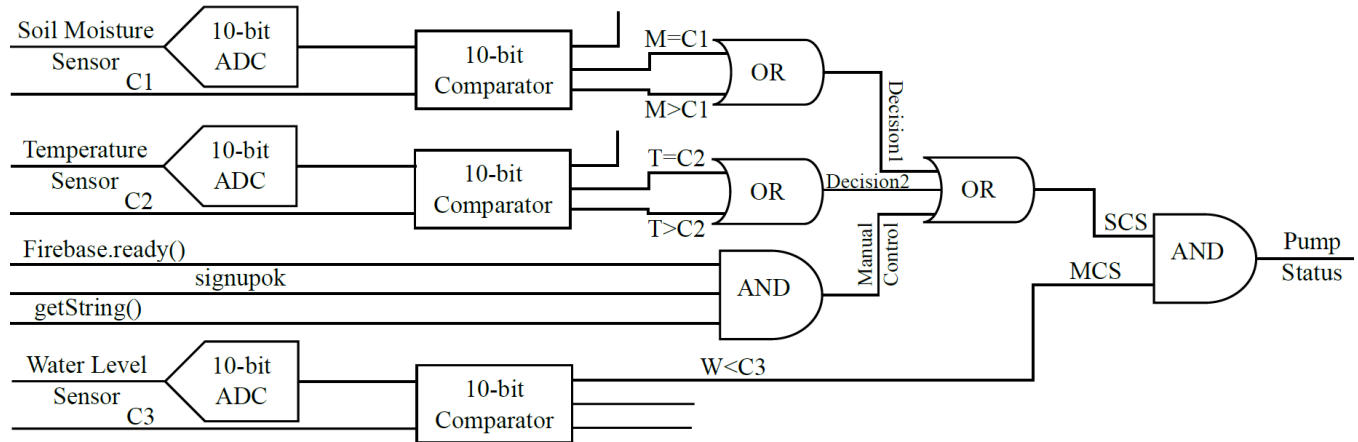
Temperature: 26.1

Water: 45

Also, the mobile application, can turn ON the pump by clicking on the button “turn on pump” if the MasterControl is high. The MasterControl is high when the water level is not high. If the water level is high, the MasterControl will be low, and this will turn OFF the

pump even if it is turned ON by the user, and the status of button in mobile application will be changed accordingly.

Logic Circuit Diagram:



- SCS: Slave Control Signal
- MCS: Master Control Signal
- M: Moisture Level
- C1: Moisture_Dry = $750_{10} = 10\ 1110\ 1110_2$
- T: Temperature in degree Celsius
- C2: TempHigh = $30^{\circ}\text{C} \approx 614_{10} = 10\ 0110\ 0110_2$, this is because the temperature range that DHT11 can measure is between 0 and 50°C .
- W: Water Level
- C3: Water_High = $600_{10} = 10\ 0101\ 1000_2$

Note: All the ADCs' non-connected pins are floating.

Code:

The program consists of 3 files:

- Code.ino (main file)
- helper_functions.h (contains the declarations of functions and variables)
- helper_functions.cpp (contains the implementation of functions).

Code.ino:

```
#include "helper_functions.h"
#include "addons/RTDBHelper.h"
#include "addons/TokenHelper.h"
```



```

void setup() {
  Serial.begin(SerialBaudRate);
  initWiFi();
  initFirebase();
  initDHTSensor();
  initAnalogSensors();
  initPump();
  initDisplay();
  updateScreen();
}

void loop() {
  Serial.println("***** New Loop *****");
  Decision2 = getDHTData();
  Decision1 = getMoistureData();
  delay(100);
  MasterControl = getWaterData();
  delay(1000);
  controlPump();
  updateFirebase();
  updateScreen();
}

```

Explanation:

```

#include "helper_functions.h"
#include "addons/RTDBHelper.h"
#include "addons/TokenHelper.h"

```

The code begins by including header files for some helper functions and libraries that will be used in the sketch.

```

void setup() {
  Serial.begin(SerialBaudRate);
  initWiFi();
  initFirebase();
  initDHTSensor();
  initAnalogSensors();
  initPump();

```

```
initDisplay();  
updateScreen();  
}
```

The *setup()* function starts by initializing the serial communication at the baud rate specified by the *SerialBaudRate* macro in the *helper_functions.h* file. Then it initializes various components of the system, including the WiFi connection, Firebase database, DHT11 temperature and humidity sensor, soil moisture sensor, water level sensor, pump, and OLED display. Finally, it calls the *updateScreen()* function to display initial system properties on the OLED screen.

```
void loop() {  
    Serial.println("***** New Loop *****");  
    Decision2 = getDHTData();  
    Decision1 = getMoistureData();  
    delay(100);  
    MasterControl = getWaterData();  
    delay(1000);  
    controlPump();  
    updateFirebase();  
    updateScreen();  
}
```

The *loop()* function starts by printing a message to the serial monitor. Then, it calls the *getDHTData()* function to get the temperature and humidity readings from the DHT11 sensor and store them in the *Decision2* boolean variable. Next, it calls the *getMoistureData()* function to get the moisture readings from the analog sensors and store them in the *Decision1* boolean variable. After setting the values of *Decision2* and *Decision1*, it waits for a 100 msec delay before calling another function. Then, it calls the *getWaterData()* function to get the water level reading and store it in the *MasterControl* Boolean variable, this will be the main control signal based on which the significance of other control signals will be determined. As we did before, a 1 second delay is inserted before making other calls. Next, it calls the *controlPump()* function to determine whether or not the pump should be turned on based on the readings from the sensors, the manual control signal, and the *MasterControl* signal. At the end, it calls the *updateFirebase()* function to update the Firebase database with the sensor readings, and calls the *updateScreen()* function again to update the display with the latest sensor readings.

helper_functions.h

```
#ifndef HELPER_FUNCTIONS_H
#define HELPER_FUNCTIONS_H

#include <ESP8266WiFi.h>
#include <Firebase_ESP_Client.h>
#include "DHTesp.h"
#include "SSD1306Wire.h"
#include "Wire.h"

#define ssid "AndroidAP5C22"
#define pass "12345678"

#define API_KEY "AIzaSyCM_3eziXBsBALqacktNZ3dMetAFyTNBvw"
#define DATABASE_URL "https://smart-plant-monitoring-769a7-default-rtdb.firebaseio.com/"
#define PUMP_URL "/Smart_Plant_Monitoring/Pump"
#define TEMPERATURE_URL "/Smart_Plant_Monitoring/Temperature"
#define WATER_URL "/Smart_Plant_Monitoring/Water"
#define MOISTURE_URL "/Smart_Plant_Monitoring/Moisture"
#define HUMIDITY_URL "/Smart_Plant_Monitoring/Humidity"
#define MASTERCONTROL_URL "/Smart_Plant_Monitoring/MasterControl"

#define PunpPin D3
#define DHTpin D4
#define MuxSelectPin D5
#define AnalogPin A0

#define SerialBaudRate 115200
#define TempHigh 30
#define Moisture_Wet 500
#define Moisture_Dry 750
#define Water_Low 450
#define Water_High 600

extern bool signupOk;
extern bool Decision1;
extern bool Decision2;
```

```

extern bool MasterControl; // true means that other control signals are
significant.
extern bool ManualControl;
extern uint16_t Water;
extern uint16_t Moisture;
extern float Temperature;
extern float Humidity;
extern String PumpStatus;

extern FirebaseData cloud;
extern FirebaseConfig config;
extern FirebaseAuth auth;
extern DHTesp dht;
extern SSD1306Wire display;

void initWiFi();
void onWiFiConnect();
void initFirebase();
void onFirebaseSignUp();
void onFirebaseSignUpError();
void initDHTSensor();
void initAnalogSensors();
void initPump();
void initDisplay();
bool getDHTData();
bool getMoistureData();
bool getWaterData();
void controlPunp();
void updateFirebase();
void updateScreen();

#endif

```

Explanation:

The "helper_functions.h" header file contains set of declarations of helper functions and constants used in our project. It includes constants for network credentials, Firebase Realtime Database URLs, and pin assignments for the sensors, relay, analog pin, and display. It also defines global variables for storing data and status flags. It also includes the

necessary libraries used in implementing the functions declared in the same file. The functions declared in this file include initialization functions for WiFi, Firebase, sensors, pump, and display, as well as functions for collecting sensor data, controlling the pump, updating Firebase, and updating the OLED display.

helper_functions.cpp

```
#include "helper_functions.h"

bool signupOk = false;
bool Decision1 = false;
bool Decision2 = false;
bool MasterControl = false; // true means that other control signals are
significant.
bool ManualControl = false;
uint16_t Water = 0;
uint16_t Moisture = 0;
float Temperature = 0;
float Humidity = 0;
String PumpStatus = "";

FirebaseData cloud;
FirebaseConfig config;
FirebaseAuth auth;
DHTesp dht;
SSD1306Wire display(0x3c, D2, D1);

void initWiFi() {
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, pass);
    while(WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    onWiFiConnect();
}

void onWiFiConnect() {
    Serial.println("");
    Serial.print("Connected: ");
```

```

    Serial.print(ssid);
    Serial.println("");
    Serial.print(WiFi.localIP());
    Serial.println("");
}

void initFirebase() {
    config.api_key = API_KEY;
    config.database_url = DATABASE_URL;
    Firebase.signUp(&config,&auth,"","") ? onFirebaseSignUp() :
onFirebaseSignUpError();
    Firebase.begin(&config,&auth);
    Firebase.reconnectWiFi(true);
}

void onFirebaseSignUp() {
    signupOk = true;
    Serial.println("OK");
}

void onFirebaseSignUpError() {
    Serial.printf("%s\n", config.signer.signupError.message.c_str());
    Serial.println("Error");
}

void initDHTSensor() {
    dht.setup(DHTpin, DHTesp::DHT11);
}

void initAnalogSensors() {
    pinMode(MuxSelectPin, OUTPUT);
}

void initPump() {
    digitalWrite(PunpPin, LOW);
    pinMode(PunpPin, OUTPUT);
}

void initDisplay() {

```

```

display.init();
display.clear();
}

bool getDHTData() {
    delay(dht.getMinimumSamplingPeriod());
    Humidity = dht.getHumidity();
    Temperature = dht.getTemperature();
    Serial.printf("DHT11 Status String: %s\nHumidity = %.1f%%\nTemperature = %.1f\u2103\n", dht.getStatusString(), Humidity, Temperature);
    return (Temperature >= TempHigh);
}

bool getMoistureData() {
    digitalWrite(MuxSelectPin, 0); // Moisture level will be measured
    Moisture = analogRead(AnalogPin);
    Serial.printf("Moisture Level: %.2f%%\n%s\n",
        Moisture * 100.0 / 1024.0, (Moisture < Moisture_Wet) ? "Status: Soil is too wet" :
        (Moisture < Moisture_Dry) ? "Status: Soil moisture is perfect" :
        "Status: Soil is too dry - time to water!");
    return (Moisture >= Moisture_Dry);
}

bool getWaterData() {
    digitalWrite(MuxSelectPin, 1); // Water level will be measured
    Water = analogRead(AnalogPin);
    Serial.printf("Water Level: %.2f%%\nStatus: Water level is %s\n",
        Water * 100.0 / 1024.0, (Water < Water_Low) ? "LOW" : (Water <
Water_High) ? "MEDIUM" : "HIGH");
    return (Water < Water_High);
}

void controlPump() {
    if (MasterControl) { // if water level is not high
        if(Firebase.ready() && signupOk) {
            Firebase.RTDB.setString(&cloud, MASTERCONTROL_URL, "1");
            PumpStatus = Firebase.RTDB.getString(&cloud, PUMP_URL) ?
cloud.stringData() : "";

```

```

    }
    ManualControl = (PumpStatus == "1");
    digitalWrite(PunpPin, (PumpStatus == "1" || Decision1 || Decision2) ?
HIGH : LOW); // Manual Control
    } else { // if water level is high, Close the pump if it is open
        Firebase.RTDB.setString(&cloud, MASTERCONTROL_URL, "0");
        Firebase.RTDB.setString(&cloud, PUMP_URL, "0");
        digitalWrite(PunpPin, 0);
        ManualControl = false;
        PumpStatus = "0";
    }
}

void updateFirebase() {
    Firebase.RTDB.setFloat(&cloud, TEMPERATURE_URL, Temperature);
    Firebase.RTDB.setFloat(&cloud, HUMIDITY_URL, Humidity);
    Firebase.RTDB.setInt(&cloud, WATER_URL, Water * 100 / 1024);
    Firebase.RTDB.setInt(&cloud, MOISTURE_URL, Moisture * 100 / 1024);
}

void updateScreen() {
    display.clear();
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.setFont(ArialMT_Plain_10);
    display.drawString(0, 0, "Temperature = " + String(Temperature, 1) +
String("\u2103"));
    display.drawString(0, 12, "Humidity = " + String(Humidity, 1) + "%");
    display.drawString(0, 24, "Moisture = " + String(Moisture * 100 / 1024) +
"%");
    display.drawString(0, 36, "Water = " + String(Water * 100 / 1024) + "%");
    ManualControl ? display.drawString(0, 48, "Manual Control: ON") :
display.drawString(0, 48, "Manual Control: OFF");
    display.display();
}

```

The *helper_functions.cpp* file contains a set of functions used in the project. This file implements important functions for the ESP8266 to send data to Firebase. First, it connects to a Firebase Realtime Database for data storage and retrieval and is controlled via the

internet. The file initializes the various components of the system, such as the Wi-Fi connection, Firebase configuration, sensors, pump, and OLED display. It also defines global variables used by the system, such as the sensor readings, pump status, and control signals.

The functions *getDHTData()*, *getMoistureData()*, and *getWaterData()* retrieve sensor readings and return true or false based on whether the condition meets the threshold defined in the header file or not. The *controlPunp()* function turns the water pump on or off based on control signals, which are determined by the water level, user input through mobile application, or the sensor readings..

The *updateFirebase()* and *updateScreen()* functions update the Firebase database and OLED display, respectively, with the latest sensor readings.

Generating all test cases scenarios:

```
import itertools
import pandas as pd

WaterLevel = {"0-599", "600-1024"}
C1 = "0-599" #<600
Firebase_ready = {"0", "1"}
signupok = {"0", "1"}
getString = {"0", "1"}
SoilMoistureSensor = {"0-749", "750-1024"}
C2 = "0-749" #<750
TemperatureSensor = {"0-29", "30-50"}
C3 = "0-29" #<30
userInput = {"0", "1"}

# define the ranges for each variable
ranges = [WaterLevel, Firebase_ready, signupok, getString, SoilMoistureSensor,
TemperatureSensor, userInput]

# generate all possible test cases
TestCases = list(itertools.product(*ranges))

# adding output variables to test cases
TestCases = [{"WaterLevel":testCase[0],
               "Firebase_ready":testCase[1],
               "signupok":testCase[2],
               "getString":testCase[3],
               "SoilMoistureSensor":testCase[4],
               "TemperatureSensor":testCase[5],
```

```

        "userInput":testCase[6],
        "MasterControlSignal":testCase[0] == C1,
        "Decision1":testCase[4] != C2,
        "Decision2":testCase[5] != C3,
        "ManualControl":testCase[1] == "1" and testCase[2] == "1" and
testCase[3] == "1" and testCase[6] == "1" and testCase[0] == C1,
        "SlaveControlSignal":testCase[4] != C2 or testCase[5] != C3 or
        (testCase[1] == "1" and testCase[2] == "1" and testCase[3] == "1" and testCase
[6] == "1" and testCase[0] == C1),
        "PumpStatus":(testCase[0] == C1) and ((testCase[4] != C2 or te
stCase[5] != C3 or (testCase[1] == "1" and testCase[2] == "1" and testCase[3] =
= "1" and testCase[6] == "1")) == True)} for testCase in TestCases]
for i, testCase in enumerate(TestCases):
    for key in testCase:
        if type(testCase[key]) == bool:
            testCase[key] = int(testCase[key])
    print(i+1, testCase)
df = pd.DataFrame(TestCases)

df.to_csv("output.csv", index = False)

```

Explanation:

The code uses *itertools* and *pandas* to generate all possible test cases for the project based on predefined ranges for some variables that represent different sensor readings, Firebase status, and user input through the mobile application. After generating the test cases, the code adds output variables to each test case based on certain conditions defined in the logic diagram. Finally, the code converts boolean values to integers in order to facilitate the logical operations done in the verification phase and saves the test cases to a CSV file used in the verification phase.

Output:

	Water Level	Firebase_ready	signupok	getString	SoilMoistureSensor	TemperatureSensor	userinput	MasterControlSignal	Decision1	Decision2	ManualControl	SlaveControlSignal	PumpStatus
0	0-599	1	1	1	0-749	30-50	1	1	0	1	1	1	1
1	0-599	1	1	1	0-749	30-50	0	1	0	1	0	1	1
2	0-599	1	1	1	0-749	0-29	1	1	0	0	1	1	1
3	0-599	1	1	1	0-749	0-29	0	1	0	0	0	0	0
4	0-599	1	1	1	750-1024	30-50	1	1	1	1	1	1	1
5	0-599	1	1	1	750-1024	30-50	0	1	1	1	0	1	1
6	0-599	1	1	1	750-1024	0-29	1	1	1	0	1	1	1

	Water Level	Firebase _ready	signu pok	getSt ring	SoilMoistur eSensor	Temperatur eSensor	userI nput	MasterCont rolSignal	Decis ion1	Decis ion2	ManualC ontrol	SlaveContr olSignal	PumpS tatus
7	0-599	1	1	1	750-1024	0-29	0	1	1	0	0	1	1
8	0-599	1	1	0	0-749	30-50	1	1	0	1	0	1	1
9	0-599	1	1	0	0-749	30-50	0	1	0	1	0	1	1
10	0-599	1	1	0	0-749	0-29	1	1	0	0	0	0	0
11	0-599	1	1	0	0-749	0-29	0	1	0	0	0	0	0
12	0-599	1	1	0	750-1024	30-50	1	1	1	1	0	1	1
13	0-599	1	1	0	750-1024	30-50	0	1	1	1	0	1	1
14	0-599	1	1	0	750-1024	0-29	1	1	1	0	0	1	1
15	0-599	1	1	0	750-1024	0-29	0	1	1	0	0	1	1
16	0-599	1	0	1	0-749	30-50	1	1	0	1	0	1	1
17	0-599	1	0	1	0-749	30-50	0	1	0	1	0	1	1
18	0-599	1	0	1	0-749	0-29	1	1	0	0	0	0	0
19	0-599	1	0	1	0-749	0-29	0	1	0	0	0	0	0
20	0-599	1	0	1	750-1024	30-50	1	1	1	1	0	1	1
21	0-599	1	0	1	750-1024	30-50	0	1	1	1	0	1	1
22	0-599	1	0	1	750-1024	0-29	1	1	1	0	0	1	1
23	0-599	1	0	1	750-1024	0-29	0	1	1	0	0	1	1
24	0-599	1	0	0	0-749	30-50	1	1	0	1	0	1	1
25	0-599	1	0	0	0-749	30-50	0	1	0	1	0	1	1
26	0-599	1	0	0	0-749	0-29	1	1	0	0	0	0	0
27	0-599	1	0	0	0-749	0-29	0	1	0	0	0	0	0
28	0-599	1	0	0	750-1024	30-50	1	1	1	1	0	1	1
29	0-599	1	0	0	750-1024	30-50	0	1	1	1	0	1	1
30	0-599	1	0	0	750-1024	0-29	1	1	1	0	0	1	1
31	0-599	1	0	0	750-1024	0-29	0	1	1	0	0	1	1
32	0-599	0	1	1	0-749	30-50	1	1	0	1	0	1	1
33	0-599	0	1	1	0-749	30-50	0	1	0	1	0	1	1
34	0-599	0	1	1	0-749	0-29	1	1	0	0	0	0	0
35	0-599	0	1	1	0-749	0-29	0	1	0	0	0	0	0
36	0-599	0	1	1	750-1024	30-50	1	1	1	1	0	1	1
37	0-599	0	1	1	750-1024	30-50	0	1	1	1	0	1	1

	Water Level	Firebase _ready	signu pok	getSt ring	SoilMoistur eSensor	Temperatur eSensor	userI nput	MasterCont rolSignal	Decis ion1	Decis ion2	ManualC ontrol	SlaveContr olSignal	PumpS tatus
38	0-599	0	1	1	750-1024	0-29	1	1	1	0	0	1	1
39	0-599	0	1	1	750-1024	0-29	0	1	1	0	0	1	1
40	0-599	0	1	0	0-749	30-50	1	1	0	1	0	1	1
41	0-599	0	1	0	0-749	30-50	0	1	0	1	0	1	1
42	0-599	0	1	0	0-749	0-29	1	1	0	0	0	0	0
43	0-599	0	1	0	0-749	0-29	0	1	0	0	0	0	0
44	0-599	0	1	0	750-1024	30-50	1	1	1	1	0	1	1
45	0-599	0	1	0	750-1024	30-50	0	1	1	1	0	1	1
46	0-599	0	1	0	750-1024	0-29	1	1	1	0	0	1	1
47	0-599	0	1	0	750-1024	0-29	0	1	1	0	0	1	1
48	0-599	0	0	1	0-749	30-50	1	1	0	1	0	1	1
49	0-599	0	0	1	0-749	30-50	0	1	0	1	0	1	1
50	0-599	0	0	1	0-749	0-29	1	1	0	0	0	0	0
51	0-599	0	0	1	0-749	0-29	0	1	0	0	0	0	0
52	0-599	0	0	1	750-1024	30-50	1	1	1	1	0	1	1
53	0-599	0	0	1	750-1024	30-50	0	1	1	1	0	1	1
54	0-599	0	0	1	750-1024	0-29	1	1	1	0	0	1	1
55	0-599	0	0	1	750-1024	0-29	0	1	1	0	0	1	1
56	0-599	0	0	0	0-749	30-50	1	1	0	1	0	1	1
57	0-599	0	0	0	0-749	30-50	0	1	0	1	0	1	1
58	0-599	0	0	0	0-749	0-29	1	1	0	0	0	0	0
59	0-599	0	0	0	0-749	0-29	0	1	0	0	0	0	0
60	0-599	0	0	0	750-1024	30-50	1	1	1	1	0	1	1
61	0-599	0	0	0	750-1024	30-50	0	1	1	1	0	1	1
62	0-599	0	0	0	750-1024	0-29	1	1	1	0	0	1	1
63	0-599	0	0	0	750-1024	0-29	0	1	1	0	0	1	1
64	600- 1024	1	1	1	0-749	30-50	1	0	0	1	0	1	0
65	600- 1024	1	1	1	0-749	30-50	0	0	0	1	0	1	0
66	600- 1024	1	1	1	0-749	0-29	1	0	0	0	0	0	0

	Water Level	Firebase _ready	signu pok	getSt ring	SoilMoistur eSensor	Temperatur eSensor	userI nput	MasterCont rolSignal	Decis ion1	Decis ion2	ManualC ontrol	SlaveContr olSignal	PumpS tatus
67	600-1024	1	1	1	0-749	0-29	0	0	0	0	0	0	0
68	600-1024	1	1	1	750-1024	30-50	1	0	1	1	0	1	0
69	600-1024	1	1	1	750-1024	30-50	0	0	1	1	0	1	0
70	600-1024	1	1	1	750-1024	0-29	1	0	1	0	0	1	0
71	600-1024	1	1	1	750-1024	0-29	0	0	1	0	0	1	0
72	600-1024	1	1	0	0-749	30-50	1	0	0	1	0	1	0
73	600-1024	1	1	0	0-749	30-50	0	0	0	1	0	1	0
74	600-1024	1	1	0	0-749	0-29	1	0	0	0	0	0	0
75	600-1024	1	1	0	0-749	0-29	0	0	0	0	0	0	0
76	600-1024	1	1	0	750-1024	30-50	1	0	1	1	0	1	0
77	600-1024	1	1	0	750-1024	30-50	0	0	1	1	0	1	0
78	600-1024	1	1	0	750-1024	0-29	1	0	1	0	0	1	0
79	600-1024	1	1	0	750-1024	0-29	0	0	1	0	0	1	0
80	600-1024	1	0	1	0-749	30-50	1	0	0	1	0	1	0
81	600-1024	1	0	1	0-749	30-50	0	0	0	1	0	1	0
82	600-1024	1	0	1	0-749	0-29	1	0	0	0	0	0	0
83	600-1024	1	0	1	0-749	0-29	0	0	0	0	0	0	0
84	600-1024	1	0	1	750-1024	30-50	1	0	1	1	0	1	0
85	600-1024	1	0	1	750-1024	30-50	0	0	1	1	0	1	0
86	600-1024	1	0	1	750-1024	0-29	1	0	1	0	0	1	0
87	600-1024	1	0	1	750-1024	0-29	0	0	1	0	0	1	0
88	600-1024	1	0	0	0-749	30-50	1	0	0	1	0	1	0

	Water Level	Firebase_ready	signu pok	getSt ring	SoilMoistur eSensor	Temperatur eSensor	userI nput	MasterCont rolSignal	Decis ion1	Decis ion2	ManualC ontrol	SlaveContr olSignal	PumpS tatus
89	600-1024	1	0	0	0-749	30-50	0	0	0	1	0	1	0
90	600-1024	1	0	0	0-749	0-29	1	0	0	0	0	0	0
91	600-1024	1	0	0	0-749	0-29	0	0	0	0	0	0	0
92	600-1024	1	0	0	750-1024	30-50	1	0	1	1	0	1	0
93	600-1024	1	0	0	750-1024	30-50	0	0	1	1	0	1	0
94	600-1024	1	0	0	750-1024	0-29	1	0	1	0	0	1	0
95	600-1024	1	0	0	750-1024	0-29	0	0	1	0	0	1	0
96	600-1024	0	1	1	0-749	30-50	1	0	0	1	0	1	0
97	600-1024	0	1	1	0-749	30-50	0	0	0	1	0	1	0
98	600-1024	0	1	1	0-749	0-29	1	0	0	0	0	0	0
99	600-1024	0	1	1	0-749	0-29	0	0	0	0	0	0	0
100	600-1024	0	1	1	750-1024	30-50	1	0	1	1	0	1	0
101	600-1024	0	1	1	750-1024	30-50	0	0	1	1	0	1	0
102	600-1024	0	1	1	750-1024	0-29	1	0	1	0	0	1	0
103	600-1024	0	1	1	750-1024	0-29	0	0	1	0	0	1	0
104	600-1024	0	1	0	0-749	30-50	1	0	0	1	0	1	0
105	600-1024	0	1	0	0-749	30-50	0	0	0	1	0	1	0
106	600-1024	0	1	0	0-749	0-29	1	0	0	0	0	0	0
107	600-1024	0	1	0	0-749	0-29	0	0	0	0	0	0	0
108	600-1024	0	1	0	750-1024	30-50	1	0	1	1	0	1	0
109	600-1024	0	1	0	750-1024	30-50	0	0	1	1	0	1	0
110	600-1024	0	1	0	750-1024	0-29	1	0	1	0	0	1	0

	Water Level	Firebase_ready	signu pok	getSt ring	SoilMoistur eSensor	Temperatur eSensor	userI nput	MasterCont rolSignal	Decis ion1	Decis ion2	ManualC ontrol	SlaveContr olSignal	PumpS tatus
111	600-1024	0	1	0	750-1024	0-29	0	0	1	0	0	1	0
112	600-1024	0	0	1	0-749	30-50	1	0	0	1	0	1	0
113	600-1024	0	0	1	0-749	30-50	0	0	0	1	0	1	0
114	600-1024	0	0	1	0-749	0-29	1	0	0	0	0	0	0
115	600-1024	0	0	1	0-749	0-29	0	0	0	0	0	0	0
116	600-1024	0	0	1	750-1024	30-50	1	0	1	1	0	1	0
117	600-1024	0	0	1	750-1024	30-50	0	0	1	1	0	1	0
118	600-1024	0	0	1	750-1024	0-29	1	0	1	0	0	1	0
119	600-1024	0	0	1	750-1024	0-29	0	0	1	0	0	1	0
120	600-1024	0	0	0	0-749	30-50	1	0	0	1	0	1	0
121	600-1024	0	0	0	0-749	30-50	0	0	0	1	0	1	0
122	600-1024	0	0	0	0-749	0-29	1	0	0	0	0	0	0
123	600-1024	0	0	0	0-749	0-29	0	0	0	0	0	0	0
124	600-1024	0	0	0	750-1024	30-50	1	0	1	1	0	1	0
125	600-1024	0	0	0	750-1024	30-50	0	0	1	1	0	1	0
126	600-1024	0	0	0	750-1024	0-29	1	0	1	0	0	1	0
127	600-1024	0	0	0	750-1024	0-29	0	0	1	0	0	1	0

Verifying test cases:

Code:

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <windows.h>
```

```

#define TempHigh 30
#define Moisture_Dry 750
#define Water_High 600

using namespace std;

float random_number(const string& range) {
    int lowerBoundary = stoi(range.substr(0, range.find('-')));
    int upperBoundary = stoi(range.substr(range.find('-') + 1));
    return lowerBoundary + (upperBoundary - lowerBoundary) *
static_cast<float>(rand()) / RAND_MAX;
}

int main() {
    bool* passedTest = new bool[128] {};
    ifstream file("output.csv");

    if (!file) {
        std::cout << "Error opening file." << endl;
        return 1;
    }

    string line;
    int TestCase = 0;
    getline(file, line);
    string WaterLevel, Firebase_ready, signupok, getString,
SoilMoistureSensor,
    TemperatureSensor, userInput, MasterControlSignal, Decision1,
Decision2, ManualControl,
    SlaveControlSignal, PumpStatus;
    int PassedTestCases = 0;
    srand(time(nullptr));
    while (getline(file, line)) {
        std::cout << "Test Case " << ++TestCase << endl;
        stringstream ss(line);
        string reading;
        int col = 0;

```



```

while (getline(ss, reading, ',')) {
    col++;
    switch (col) {
        case 1: WaterLevel = reading; break;
        case 2: Firebase_ready = reading; break;
        case 3: signupok = reading; break;
        case 4: getString = reading; break;
        case 5: SoilMoistureSensor = reading; break;
        case 6: TemperatureSensor = reading; break;
        case 7: userInput = reading; break;
        case 8: MasterControlSignal = reading; break;
        case 9: Decision1 = reading; break;
        case 10: Decision2 = reading; break;
        case 11: ManualControl = reading; break;
        case 12: SlaveControlSignal = reading; break;
        case 13: PumpStatus = reading; break;

    }
}

std::cout << "WaterLevel: " + WaterLevel +
"\n";
// Input
std::cout << "Firebase_ready: " + Firebase_ready +
"\n";
// Input
std::cout << "signupok: " + signupok +
"\n";
// Input
std::cout << "getString: " + getString +
"\n";
// Input
std::cout << "SoilMoistureSensor: " + SoilMoistureSensor +
"\n";
// Input
std::cout << "TemperatureSensor: " + TemperatureSensor +
"\n";
// Input
std::cout << "userInput: " + userInput +
"\n";
// Input
std::cout << "MasterControlSignal: " + MasterControlSignal +
"\n";
// Depends on WaterLevel
std::cout << "Decision1: " + Decision1 +
"\n";
// Depends on Moisture
Level
DONE

```

```

        std::cout << "Decision2: " + Decision2 +
"\n";
        // Depends on
Temperature DONE
        std::cout << "ManualControl: " + ManualControl +
"\n";
        // Depends on Firebase_ready,
signupok, getString DONE
        std::cout << "SlaveControlSignal: " + SlaveControlSignal +
"\n";
        // Depends on Decision1, Decision2,
ManualControl DONE
        cout << "PumpStatus: " + PumpStatus +
"\n";
        // Depend on SlaveControlSignal and
MasterControlSignal DONE

float _Temperature = random_number(TemperatureSensor);
bool _Decision2 = (_Temperature >= TempHigh);
int _Moisture = (int)random_number(SoilMoistureSensor);
bool _Decision1 = (_Moisture >= Moisture_Dry);
int _Water = (int)random_number(WaterLevel);
bool _MasterControl = (_Water < Water_High);
string _Pump, _PumpStatus, _SlaveControlSignal;
bool _ManualControl;

if (_MasterControl) { // if water level is not high
    if (stoi(Firebase_ready) && stoi(signupok)) {
        _Pump = stoi(getString) ? userInput : "";
    }
    _ManualControl = (_Pump == "1");
    _PumpStatus = (_Pump == "1" || _Decision1 || _Decision2) ? "1" :
"0"; // Manual Control
}
else { // if water level is high, Close the pump if it is open
    _ManualControl = false;
    _PumpStatus = "0";
}
_SlaveControlSignal = (_Pump == "1" || _Decision1 || _Decision2) ?
"1" : "0"; // Manual Control

```

```

        if (stoi(MasterControlSignal) == _MasterControl && stoi(Decision1)
== _Decision1 && stoi(Decision2) == _Decision2
        && stoi(ManualControl) == _ManualControl && _PumpStatus ==
PumpStatus && _SlaveControlSignal == SlaveControlSignal) {
            passedTest[PassedTestCases++] = 1;
        }
        else cout << "<><><><><><><><><><><><><><><><><><><>\n";
        cout << "*****\n";

        std::cout << "_MasterControlSignal: " + to_string(_MasterControl) +
"\n";
        // Depends on WaterLevel DONE
        std::cout << "_Decision1: " + to_string(_Decision1) +
"\n";
        // Depends on Moisture
Level DONE
        std::cout << "_Decision2: " + to_string(_Decision2) +
"\n";
        // Depends on
Temperature DONE
        std::cout << "_ManualControl: " + to_string(_ManualControl) +
"\n";
        // Depends on Firebase_ready,
signupok, getString
        std::cout << "SlaveControlSignal: " + _SlaveControlSignal +
"\n";
        // Depends on Decision1, Decision2, ManualControl
        cout << "_PumpStatus: " + _PumpStatus +
"\n";
        // Depend on SlaveControlSignal and
MasterControlSignal
        cout << "Passed Test Cases = " + to_string(PassedTestCases) + "\n";
        std::cout << endl;
    }
    cout << "Passed Test Cases = " + to_string(PassedTestCases) + "\n\n\n";
    int x = 1;
    for (int i = 0; i < 128; i++) {
        if (passedTest[i]) cout << "Test Case " + to_string(i + 1) + ": " <<
"\033[32m" << "Passed" "\033[0m" << "\t";
        else cout << "Test Case " + to_string(i + 1) + ": " << "\033[31m" <<
"Failed" "\033[0m" << "\t";
        if (x++ % 4 == 0) cout << endl;
    }
    cout << "\n\t\t\t\t\tPassed Test Cases = " << "\033[33m" +
to_string(PassedTestCases) + "\033[0m" + "\n";

```

```
file.close();  
cout << "\n\nThe end";  
return 0;  
}
```

Explanation:

This code uses the data read from the file generated from last phase as input to test the cases and decide whether the test case passes the test or not. The code defines three constants *TempHigh*, *Moisture_Dry*, and *Water_High* as done in the main code that runs on the ESP8266. First, it defines a function *random_number* that takes a string range as an argument and returns a random float number within that range. The main function reads each line of the file and extracts data from each column by splitting it with commas as it reads from a .csv file in which the columns are separated by commas. The extracted data is assigned to respective variables. The program then generates random float values for *TemperatureSensor* and *SoilMoistureSensor* using the define function. Then, some decisions are made based on the extracted data and the generated random numbers. Finally, the program checks whether the expected output matches the actual output and stores the results in an array named *passedTest*.

Output:

```
Test Case 1: Passed    Test Case 2: Passed    Test Case 3: Passed    Test Case 4: Passed
Test Case 5: Passed    Test Case 6: Passed    Test Case 7: Passed    Test Case 8: Passed
Test Case 9: Passed    Test Case 10: Passed   Test Case 11: Passed   Test Case 12: Passed
Test Case 13: Passed   Test Case 14: Passed   Test Case 15: Passed   Test Case 16: Passed
Test Case 17: Passed   Test Case 18: Passed   Test Case 19: Passed   Test Case 20: Passed
Test Case 21: Passed   Test Case 22: Passed   Test Case 23: Passed   Test Case 24: Passed
Test Case 25: Passed   Test Case 26: Passed   Test Case 27: Passed   Test Case 28: Passed
Test Case 29: Passed   Test Case 30: Passed   Test Case 31: Passed   Test Case 32: Passed
Test Case 33: Passed   Test Case 34: Passed   Test Case 35: Passed   Test Case 36: Passed
Test Case 37: Passed   Test Case 38: Passed   Test Case 39: Passed   Test Case 40: Passed
Test Case 41: Passed   Test Case 42: Passed   Test Case 43: Passed   Test Case 44: Passed
Test Case 45: Passed   Test Case 46: Passed   Test Case 47: Passed   Test Case 48: Passed
Test Case 49: Passed   Test Case 50: Passed   Test Case 51: Passed   Test Case 52: Passed
Test Case 53: Passed   Test Case 54: Passed   Test Case 55: Passed   Test Case 56: Passed
Test Case 57: Passed   Test Case 58: Passed   Test Case 59: Passed   Test Case 60: Passed
Test Case 61: Passed   Test Case 62: Passed   Test Case 63: Passed   Test Case 64: Passed
Test Case 65: Passed   Test Case 66: Passed   Test Case 67: Passed   Test Case 68: Passed
Test Case 69: Passed   Test Case 70: Passed   Test Case 71: Passed   Test Case 72: Passed
Test Case 73: Passed   Test Case 74: Passed   Test Case 75: Passed   Test Case 76: Passed
Test Case 77: Passed   Test Case 78: Passed   Test Case 79: Passed   Test Case 80: Passed
Test Case 81: Passed   Test Case 82: Passed   Test Case 83: Passed   Test Case 84: Passed
Test Case 85: Passed   Test Case 86: Passed   Test Case 87: Passed   Test Case 88: Passed
Test Case 89: Passed   Test Case 90: Passed   Test Case 91: Passed   Test Case 92: Passed
Test Case 93: Passed   Test Case 94: Passed   Test Case 95: Passed   Test Case 96: Passed
Test Case 97: Passed   Test Case 98: Passed   Test Case 99: Passed   Test Case 100: Passed
Test Case 101: Passed  Test Case 102: Passed  Test Case 103: Passed  Test Case 104: Passed
Test Case 105: Passed  Test Case 106: Passed  Test Case 107: Passed  Test Case 108: Passed
Test Case 109: Passed  Test Case 110: Passed  Test Case 111: Passed  Test Case 112: Passed
Test Case 113: Passed  Test Case 114: Passed  Test Case 115: Passed  Test Case 116: Passed
Test Case 117: Passed  Test Case 118: Passed  Test Case 119: Passed  Test Case 120: Passed
Test Case 121: Passed  Test Case 122: Passed  Test Case 123: Passed  Test Case 124: Passed
Test Case 125: Passed  Test Case 126: Passed  Test Case 127: Passed  Test Case 128: Passed

Passed Test Cases = 128
```

ⁱ Texas Instruments. (2021). CD74HC4051-EP Analog Multiplexer and Demultiplexer. Retrieved from <https://www.ti.com/lit/ds/symlink/cd74hc4051-ep.pdf>

ⁱⁱ Texas Instruments. (2015). *Understanding the I2C Bus*. Retrieved from <https://www.ti.com/lit/an/slva704/slva704.pdf>

ⁱⁱⁱ Espressif Systems. (2023). ESP8266EX Datasheet. Retrieved from https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

^{iv} Arduino. (2009). Arduino Uno Datasheet. Retrieved from <https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>

^v <https://appinventor.mit.edu/>