Smart Water System

Problem Statement:

"Public places like parks and gardens lack real-time water consumption data, hindering water conservation efforts. To address this, our project aims to implement an IoT system with sensors and a data-sharing platform. This will enable accurate monitoring and promote responsible water use in these areas, benefiting both the environment and the community."

Implementing an IOT system to monitor water consumption in public places like parks and gardens is a valuable project that can contribute to water conservation efforts. Here is a step-by-step guide on how to approach this project:

1.Define Objectives and Scope:

Implementing an IOT system to monitor water consumption in public places like parks and gardens is a valuable project that can contribute to water conservation efforts. Here is a step-by-step guide on how to approach this project:

- Clearly define the project's objectives. In this case, it's to monitor water consumption in public places to promote water conservation.
- Specify the locations where sensors will be deployed (parks, gardens, etc.).
- Determine the granularity of data required (e.g., per area, per fountain, per day, etc.).
- Set specific goals for water conservation and resource optimization.

2. Design the IoT Sensor System:
- Choose suitable water flow sensors that are compatible with IoT systems. These could be ultrasonic, electromagnetic, or other types.
- Select microcontrollers or IoT development boards (e.g., Raspberry Pi, Arduino, ESP8266, ESP32) to connect and process data from the sensors.
- Decide on a communication protocol (e.g., MQTT, HTTP, CoAP) for sending

data fromsensors to a central server.
- Consider power sources for the sensors (e.g., batteries, solar panels).
- Plan the sensor network architecture and connectivity (Wi-Fi, cellular, LoRa, etc.).

3. Develop the Data-Sharing Platform:
- Set up a central server or cloud platform to collect and store data from the sensors.
- Design a database schema to store sensor data, including timestamps and locationinformation.
- Implement data security measures (e.g., encryption, authentication) to protect the data.
- Create a user-friendly dashboard or web interface for viewing real-time and historicalwater consumption data.
- Enable data visualization tools to help users understand and analyze the data easily.

4. Integrate Using IoT Technology and Python:
- Develop firmware for the IoT sensors using Python or other suitable programminglanguages.
- Configure the sensors to transmit data at regular intervals to the central server.
- Implement error handling and data synchronization mechanisms to ensure datareliability.
- Use Python libraries and frameworks (e.g., Flask, Django) to build the data-sharingplatform.
- Set up data analytics and reporting features to monitor water consumption trends andanomalies.

5. Testing and Deployment:
- Test the entire system in a controlled environment before deploying it to public places.
- Deploy sensors in selected parks and gardens.
- Monitor the system's performance and make adjustments as needed.
- Train relevant personnel on how to use and maintain the system.

6. Outreach and Public Awareness:
- Promote the availability of real-time water consumption data to the public.
- Educate the public on water conservation and the benefits of the IoT system.

- Encourage user engagement and feedback to improve the system.

7. Maintenance and Scaling:
- Establish a maintenance plan to regularly check and service sensors.
- Consider scaling the system to cover more locations if successful.

8. Data Analysis and Optimization:
- Continuously analyze the data to identify trends and areas for water conservationimprovements.
- Use data-driven insights to optimize water usage in public places.

9. Documentation and Reporting:
- Document the project's progress, design decisions, and outcomes.
- Provide regular reports to stakeholders and the public on water consumption andconservation efforts.

Remember that IOT projects like this one require careful planning, monitoring, and maintenance to achieve their goals effectively. Collaboration with local authorities, environmental organizations, and the community can also enhance the project's success andimpact.

Introduction

General we see a lot of water overflowing from the water tanks, this results in a large scale of water wastage in our household.so to avoid this condition We can use Internet of Things to solve this problem. Our project is IOT based, this project helps us to know the water level in the tank whether it is in the minimum level or the maximum level, the alarm will start for the LOW water level, and you will get a **Blynk notification** on the smartphone through the internet.

Idealogy:

- IoT Based Water Level Indicator using Ultrasonic Sensor
- In this IoT Internet of Things project, I have explained how to make a simple IoT-based water level indicator using an ultrasonic sensor, ESP32, and Blynk IoT platform.
- The alarm will start for the LOW water level, and you will get a **Blynk notification** on the smartphone through the internet. Also, you can monitor the water tank level on OLED.
- The alarm will also start when the tank is full. You press the push button to stop the alarm.

Required Components for ESP32 Water Level Sensor:
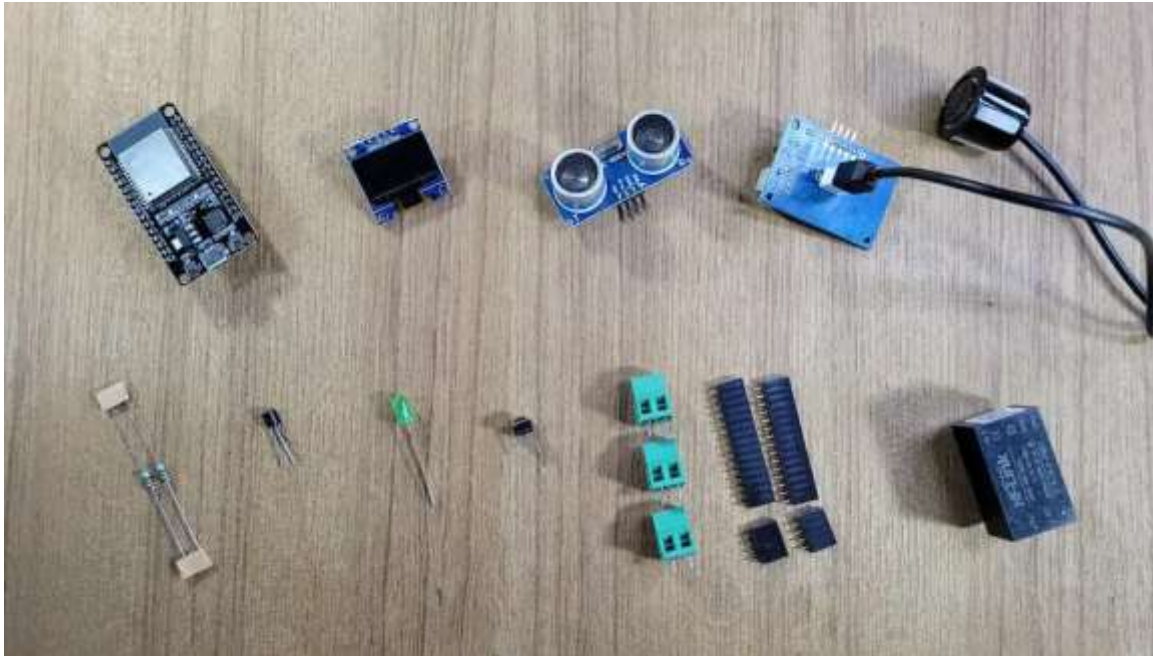
- ESP32 DEV KIT V1
- SR04M waterproof ultrasonic sensor OR HC-SR04 sensor
- 0.96″ I2C OLED Display
- 220-ohm 0.25watt Resistors – 2 no
- BC547 NPN Transistor
- LED 5mm – 1no
- 2-pin Push Button
- 2-pin Terminal connectors (3 no)
- 5V DC Buzzer
- AC to DC converter PM01 5V (Optional)

In the circuit, if you want to use AC voltage, then you have to use a PM01 AC to DC converter. Otherwise, you can directly give 5V DC supply to this circuit.

The GPIO D26 & D27 are connected with the Echo & TREG pins of the SR04M-2 waterproof ultrasonic sensor.

The OLED SDA & SCL pins are connected with the D21 & D22 GPIO of ESP32

GPIO **D12** is connected with the push button to stop the buzzer. I have used the **INPUT_PULLUP** function in Arduino IDE instead of using the pull-up resistors with the push button.

And the indicator LED and Buzzer are connected with GPIO **D14** & **D13** of ESP32.

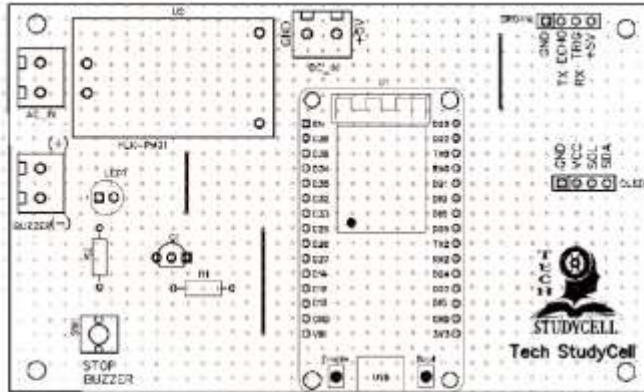You can use any other ultrasonic sensor.

Circuit of IoT Based Water Level Indicator:
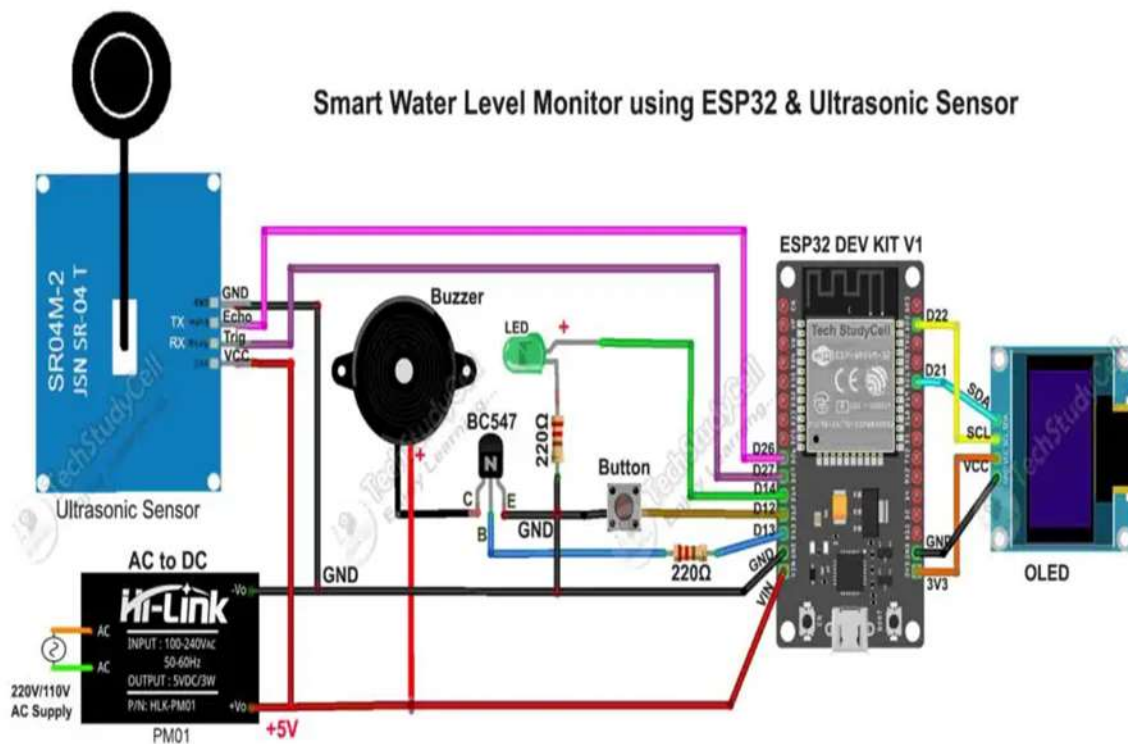
The minimum distance between the sensor and the full tank water level must be greater than 25cm.
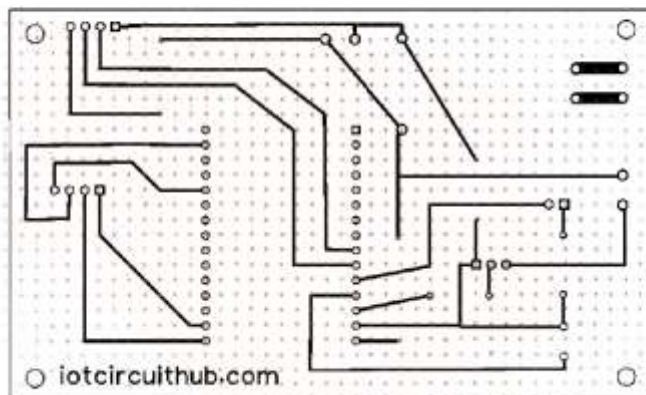
PCB Layout for Water Level Indicator:

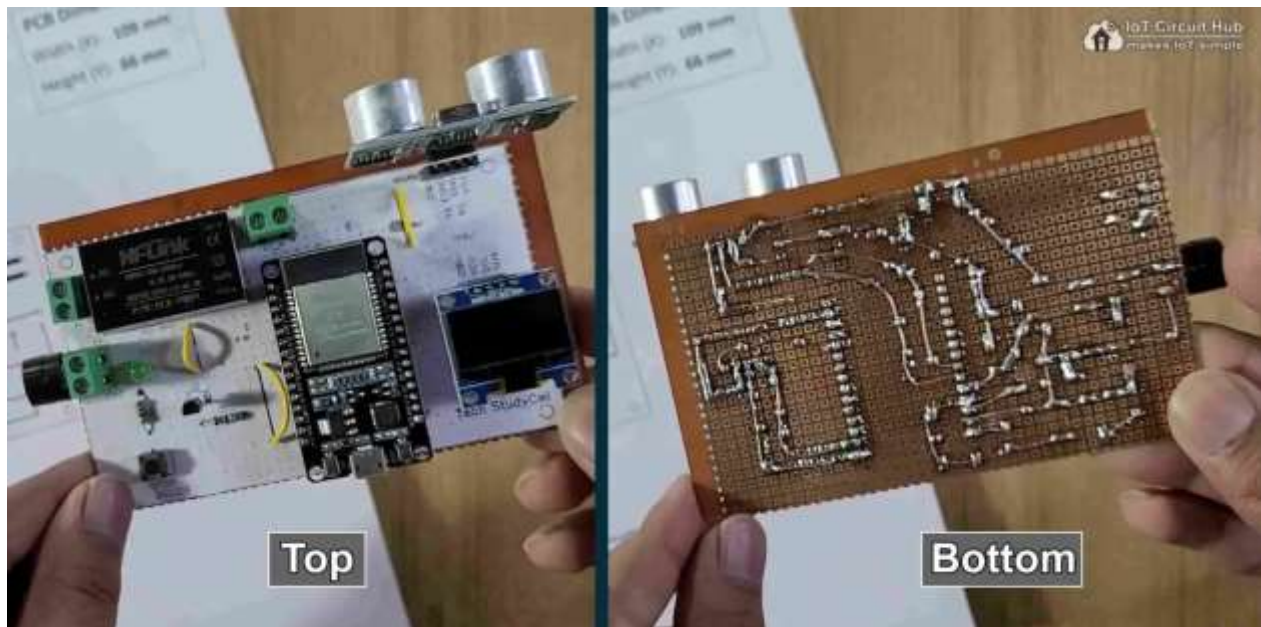**Project:** Water Level Indicator using ESP32 & Ultrasonic Sensor (SR04M)

PCB Dimension:

Width (X): **109 mm**

Height (Y): **66 mm**

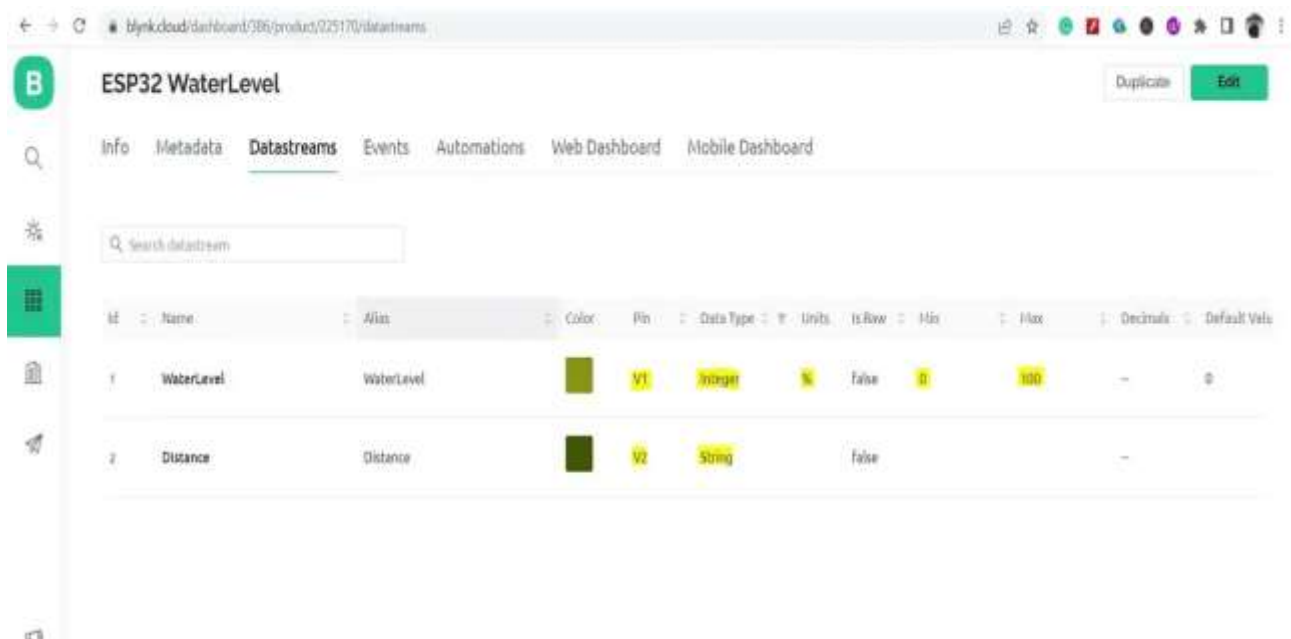## Homemade PCB for the Water Level Indicator:

Set up Blynk IoT Cloud for the IoT-based Project

Create Blynk Template:

During creating the template, I selected ESP32 as the hardware and the connection type as WiFi.

Create Datastreams in Blynk Cloud:

In the template, I have created first Datastreams (Pin: **V1,** Datatype: **Integer**, Min Value: **0**, Max Value: **100**) to show the water level in tank in percentage.

Second Datastream (Pin: **V2,** Datatype: **String**) will show the distance between sensor and water level in cm.
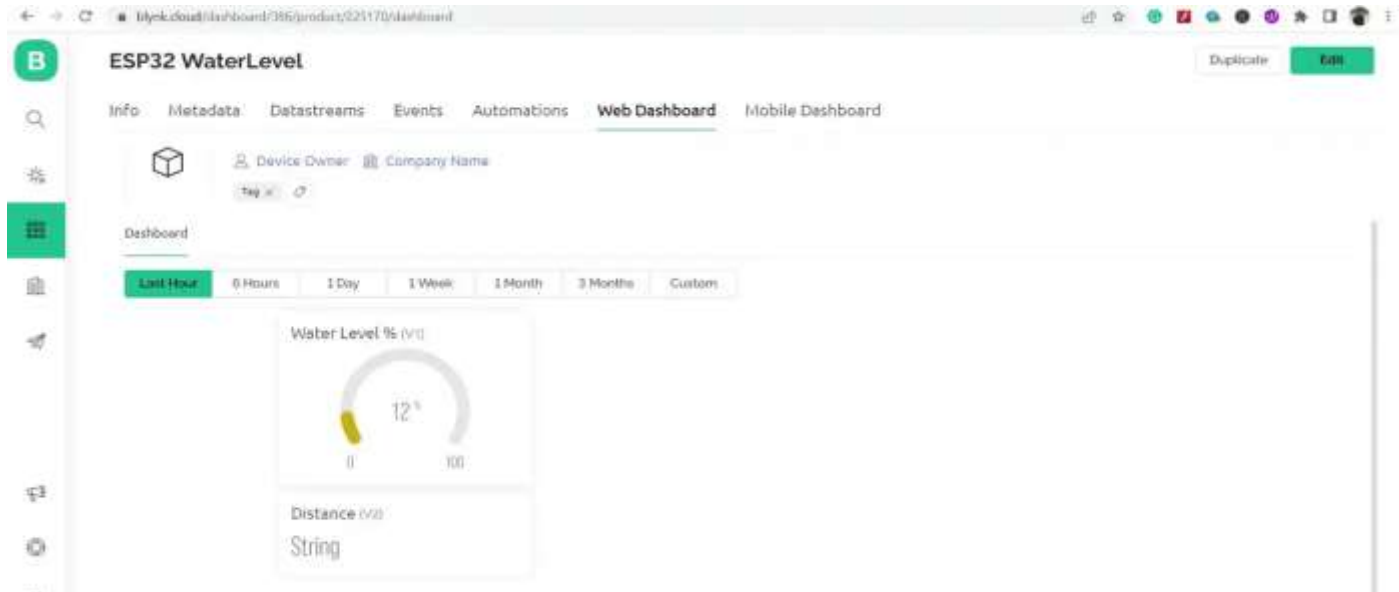
Add Automation in Blynk IoT:

I have added Automation to get the LOW water Level notification in the Blynk IoT app.

I have turned on the Condition and Action radio button for the first Datastream V1. Type of Automation will be "Sensor".

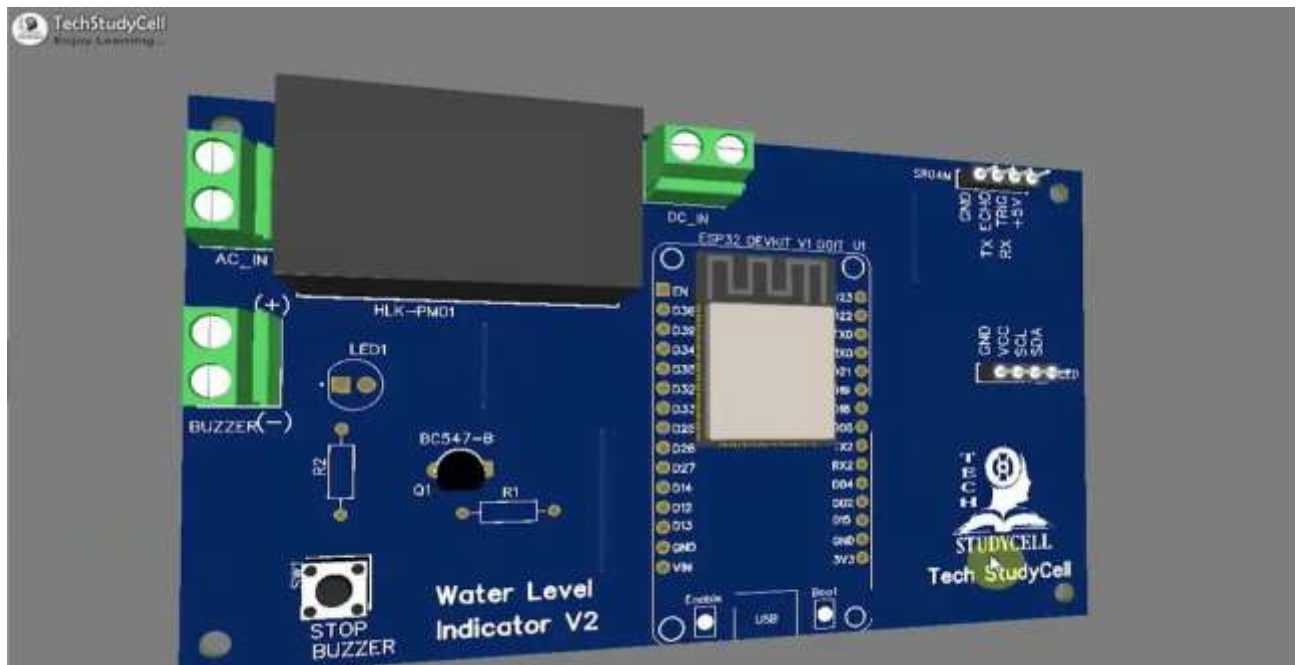## Create Web Dashboard in Blynk Cloud:

After that, click and drag **1 Gauge widget**, and **1 Level widget**, and select the related Datastreams for each widget.

Then click on "**Save**" to save the template.

Add Device in Blynk Cloud using Template:

You can refer to the following article to add a device to the Blynk cloud.

PCB for this ESP32 Water Level Indicator

Program ESP32 with Arduino IDE

For this IoT-based project, I have used the Arduino IDE to program ESP32.

First update the **Preferences** –> **Aditional boards Manager
URLs:** https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

- Then install the **ESP32** board (2.0.5) from the Board manager or **Click Here** to download the **ESP32** board.
- Download the required libraries from the following links:
  - **Blynk** Library **(Version 1.1.0)**
  - **AceButton** Library **(Version 1.9.2)**
  - **Adafruit_SSD1306** Library **(Version 2.5.7)**

Program ESP32 with Arduino IDE

Source Codes for Blynk ESP32 Water Level Sensor

```
/****************************************************************************
**
 *   TITLE: IoT-based Water Level Indicator using ESP32, Ultrasonic Sensor & Blynk
with 0.96" OLED
 *   Click on the following links to learn more.
 *   YouTube Video: https://youtu.be/9geREeE13jc
 *   Related Blog : https://iotcircuithub.com/esp32-projects/
 *
 *   This code is provided free for project purpose and fair use only.
 *   Please do mail us to techstudycell@gmail.com if you want to use it
commercially.
 *   Copyrighted © by Tech StudyCell
 *
 *   Preferences--> Aditional boards Manager URLs :
 *   https://raw.githubusercontent.com/espressif/arduino-esp32/gh-
pages/package_esp32_dev_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json
 *
 *   Download Board ESP32 (2.0.5) : https://github.com/espressif/arduino-esp32
 *
 *   Download the libraries
 *   Blynk Library (1.1.0):  https://github.com/blynkkk/blynk-library
 *   Adafruit_SSD1306 Library (2.5.7):
https://github.com/adafruit/Adafruit_SSD1306
 *   AceButton Library (1.9.2): https://github.com/bxparks/AceButton
 ****************************************************************************
**/

/* Fill-in your Template ID (only if using Blynk.Cloud) */
#define BLYNK_TEMPLATE_ID ""
#define BLYNK_DEVICE_NAME ""
#define BLYNK_AUTH_TOKEN ""

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "";
char pass[] = "";

//Set Water Level Distance in CM
int emptyTankDistance = 70 ;  //Distance when tank is empty
int fullTankDistance =  30 ;  //Distance when tank is full
```

```
//Set trigger value in percentage
int triggerPer =   10 ;  //alarm will start when water level drop below
triggerPer

#include <Adafruit_SSD1306.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <AceButton.h>
using namespace ace_button;

// Define connections to sensor
#define TRIGPIN    27  //D27
#define ECHOPIN    26  //D26
#define wifiLed    2   //D2
#define ButtonPin1 12  //D12
#define BuzzerPin  13  //D13
#define GreenLed   14  //D14

//Change the virtual pins according the rooms
#define VPIN_BUTTON_1    V1
#define VPIN_BUTTON_2    V2

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


float duration;
float distance;
int   waterLevelPer;
bool  toggleBuzzer = HIGH; //Define to remember the toggle state

char auth[] = BLYNK_AUTH_TOKEN;

ButtonConfig config1;
AceButton button1(&config1);

void handleEvent1(AceButton*, uint8_t, uint8_t);

BlynkTimer timer;
```

```
void checkBlynkStatus() { // called every 3 seconds by SimpleTimer

  bool isconnected = Blynk.connected();
  if (isconnected == false) {
    //Serial.println("Blynk Not Connected");
    digitalWrite(wifiLed, LOW);
  }
  if (isconnected == true) {
    digitalWrite(wifiLed, HIGH);
    //Serial.println("Blynk Connected");
  }
}

BLYNK_CONNECTED() {
  Blynk.syncVirtual(VPIN_BUTTON_1);
  Blynk.syncVirtual(VPIN_BUTTON_2);
}

void displayData(int value){
  display.clearDisplay();
  display.setTextSize(4);
  display.setCursor(8,2);
  display.print(value);
  display.print(" ");
  display.print("%");
  display.display();
}

void measureDistance(){
  // Set the trigger pin LOW for 2uS
  digitalWrite(TRIGPIN, LOW);
  delayMicroseconds(2);

  // Set the trigger pin HIGH for 20us to send pulse
  digitalWrite(TRIGPIN, HIGH);
  delayMicroseconds(20);

  // Return the trigger pin to LOW
  digitalWrite(TRIGPIN, LOW);

  // Measure the width of the incoming pulse
  duration = pulseIn(ECHOPIN, HIGH);

  // Determine distance from duration
  // Use 343 metres per second as speed of sound
```

```arduino
  // Divide by 1000 as we want millimeters

  distance = ((duration / 2) * 0.343)/10;

  if (distance > (fullTankDistance - 10)  && distance < emptyTankDistance ){
    waterLevelPer = map((int)distance ,emptyTankDistance, fullTankDistance, 0,
100);
    displayData(waterLevelPer);
    Blynk.virtualWrite(VPIN_BUTTON_1, waterLevelPer);
    Blynk.virtualWrite(VPIN_BUTTON_2, (String(distance) + " cm"));

    // Print result to serial monitor
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");

    if (waterLevelPer < triggerPer){
      digitalWrite(GreenLed, HIGH);
      if (toggleBuzzer == HIGH){
        digitalWrite(BuzzerPin, HIGH);
      }
    }
    if (distance < fullTankDistance){
      digitalWrite(GreenLed, LOW);
      if (toggleBuzzer == HIGH){
        digitalWrite(BuzzerPin, HIGH);
      }
    }

    if (distance > (fullTankDistance + 5) && waterLevelPer > (triggerPer + 5)){
      toggleBuzzer = HIGH;
      digitalWrite(BuzzerPin, LOW);
    }
  }

  // Delay before repeating measurement
  delay(100);
}


void setup() {
  // Set up serial monitor
  Serial.begin(115200);

  // Set pinmodes for sensor connections
```

```
  pinMode(ECHOPIN, INPUT);
  pinMode(TRIGPIN, OUTPUT);
  pinMode(wifiLed, OUTPUT);
  pinMode(GreenLed, OUTPUT);
  pinMode(BuzzerPin, OUTPUT);

  pinMode(ButtonPin1, INPUT_PULLUP);

  digitalWrite(wifiLed, LOW);
  digitalWrite(GreenLed, LOW);
  digitalWrite(BuzzerPin, LOW);

  config1.setEventHandler(button1Handler);

  button1.init(ButtonPin1);

  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  delay(1000);
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.clearDisplay();

  WiFi.begin(ssid, pass);
  timer.setInterval(2000L, checkBlynkStatus); // check if Blynk server is
connected every 2 seconds
  Blynk.config(auth);
  delay(1000);

}
 void loop() {

  measureDistance();

  Blynk.run();
  timer.run(); // Initiates SimpleTimer

  button1.check();

}

void button1Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT1");
```

```
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      digitalWrite(BuzzerPin, LOW);
      toggleBuzzer = LOW;
      break;
  }
}
```

In the sketch, you have to update only the **BLYNK_TEMPLATE_ID**, **BLYNK_DEVICE NAME**, **Auth Token.**

```
/* Fill-in your Template ID (only if using Blynk.Cloud) */
#define BLYNK_TEMPLATE_ID "" #define BLYNK_DEVICE_NAME ""
#define BLYNK_AUTH_TOKEN ""
```

Enter the **WiFi Credentials**.

```
// Set password to "" for open networks.
```
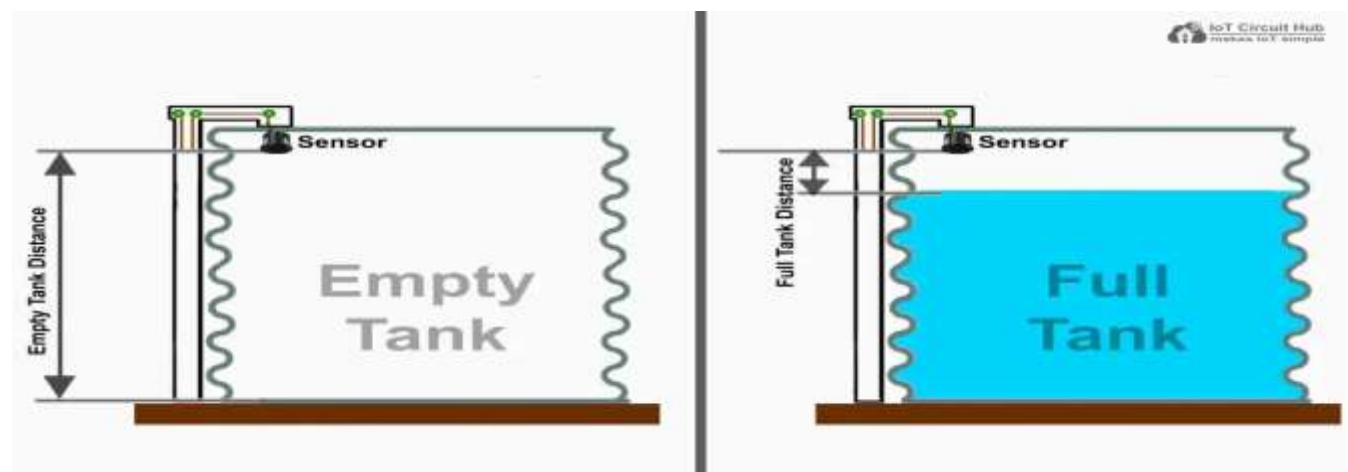```
char ssid[] = "";
```
```
char pass[] = "";
```

Define Water Level Distance for empty tank and full tank in **CM**.

```
int emptyTankDistance = 70 ; //Distance when tank is empty
int fullTankDistance = 30 ; //Distance when tank is full
```
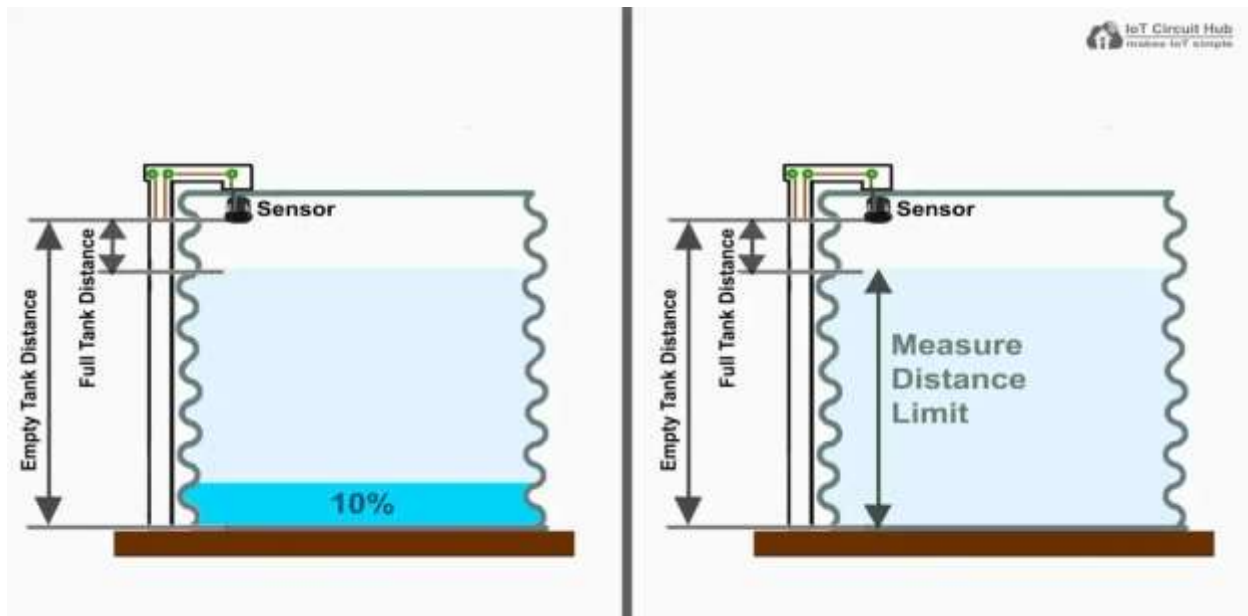
Define trigger value in percentage, alarm will start when water level drop below triggerPer.

```
//Set trigger value in percentage
int triggerPer = 10 ;
```
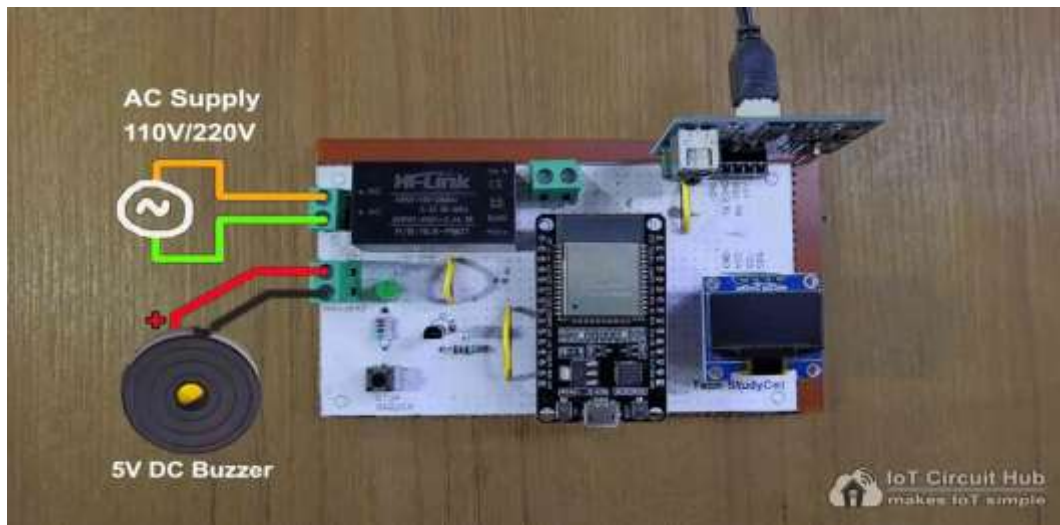
Here I have defined the triggerPer 10%, but you can define any value like 5%, 20% etc.



The ESP32 will only calculate the water level if the measured distance is between empty tank distance and the full tank distance.

After doing these changes, please upload the code to ESP32.

# Install the Water Level Sensor circuit



On the PCB, either you can give AC supply as per the above circuit or you can give 5V DC supply as per the following circuit.
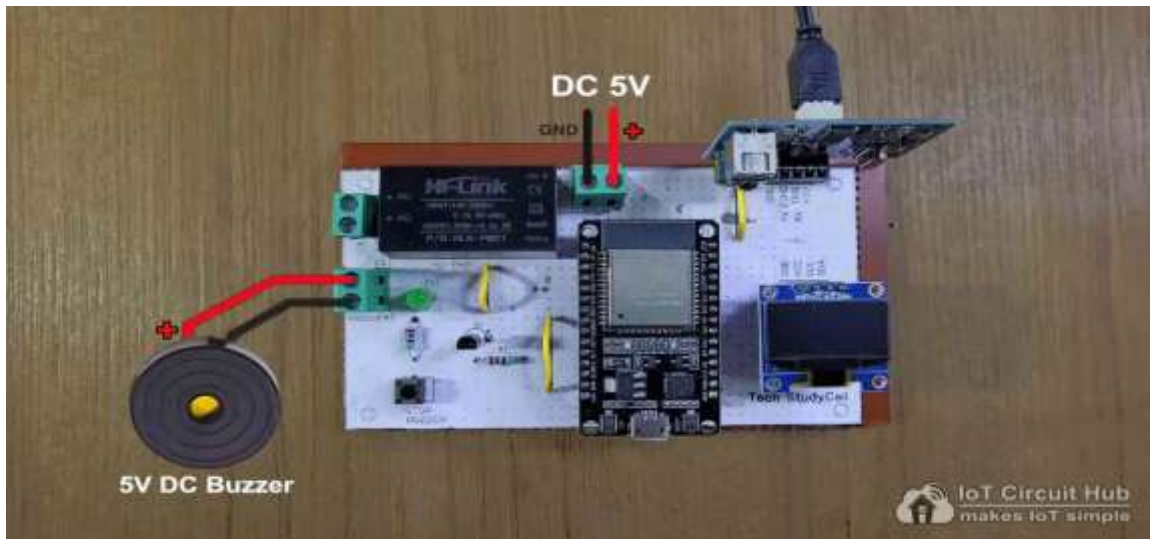


***While fitting the ultrasonic sensor in the water tank, please make sure the minimum distance between the **sensor** and the **full tank water level must be greater than 25cm.**