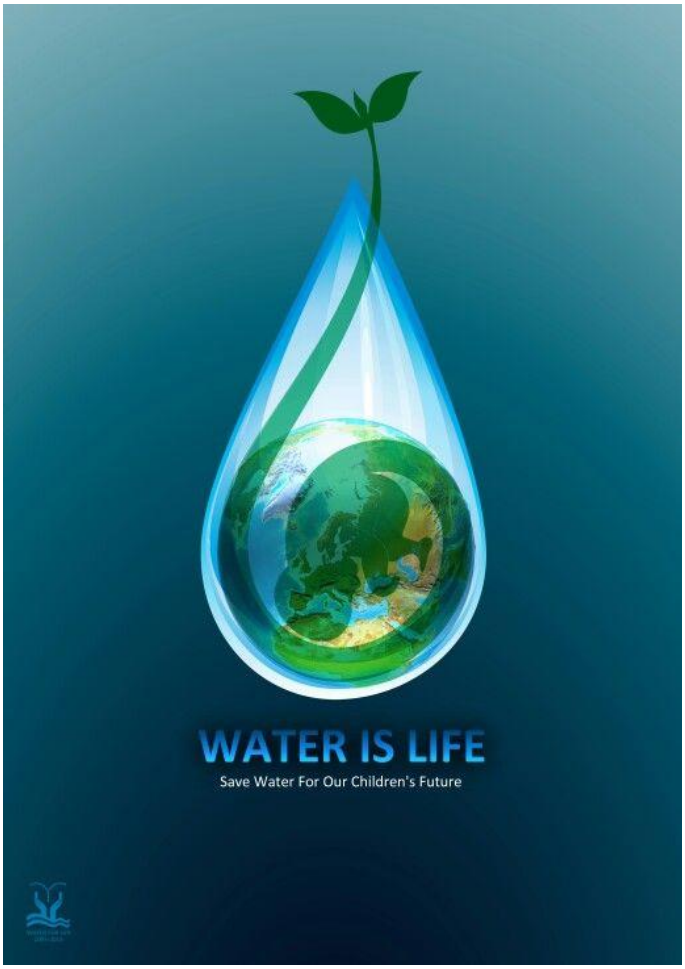
SMART WATER MANAGEMENT

DEVELOPMENT PART 2

IOT PHASE 4



INSTRUCTIONS

- Water monitoring using the ESP32 kit is a cutting-edge solution that leverages the power of IoT (Internet of Things) technology to ensure the efficient and reliable assessment of water quality and quantity. The ESP32, a versatile microcontroller, is at the heart of this system, enabling real-time data collection and transmission from various sensors placed in water bodies. This technology used for monitoring the water level.
- In this discussion, we will delve deeper into the components, program and simulation using wokwi simulator

---

COMPONENTS

- ESP32
- ADAFRUITSSD1306
- ULTRASONIC DISTANCE SENSOR
- ACE BUTTONS
- LED

ESP32:

- The ESP32 kit is a hardware platform that features the ESP32 microcontroller, a powerful and versatile microcontroller unit developed by Espressif Systems.

- It is often used in IoT and embedded systems projects due to its capabilities, which include dual-core processing, built-in Wi-Fi and Bluetooth connectivity, and a wide range of input/output options.

PURPOSE :

- The ESP32 kit serves as the core component in water detection by providing wireless connectivity and data processing, allowing for real-time monitoring and remote alerts, ensuring the efficient management and protection of water resources.

ADAFRUIT SSD1306:

- The Adafruit SSD1306 is a popular OLED (Organic Light-Emitting Diode) display module that can be used in a water monitoring project. In this context, it serves as a visual output interface, displaying important information and data related to water quality, level, or other parameters.
- You can use it to show real-time measurements, alerts, or any relevant information from the monitoring system, making it easier for users or operators to access and understand the data.
- It provides a compact and clear display, which is particularly useful for quick on-site analysis and decision-making in water monitoring applications.

PURPOSE:

- The Adafruit SSD1306 is often used as a display module in water monitoring projects. Its purpose is to visually present important data and information related to water parameters, making it more accessible for users and operators.
- This display can show real-time measurements, alerts, or status information, enhancing the usability and effectiveness of the water monitoring system.

PUSH BUTTON:

- A pushbutton in a water monitoring system typically serves as an input device. It can be used to trigger specific actions or functions within the system, such as starting or stopping data logging, initiating a calibration process, or acknowledging an alarm or alert.

- Operators or users can manually press the pushbutton to interact with the monitoring system, making it a convenient way to have some control over the monitoring process or response to specific events.

PURPOSE:

- Pushing the button can acknowledge or silence alarms and notifications generated by the system, ensuring they don't go unnoticed.

- In case of critical situations, a pushbutton can be used for an emergency shutdown to prevent further issues or damage

HC-SR04 UITRASONI DISTANCE SENSOR:

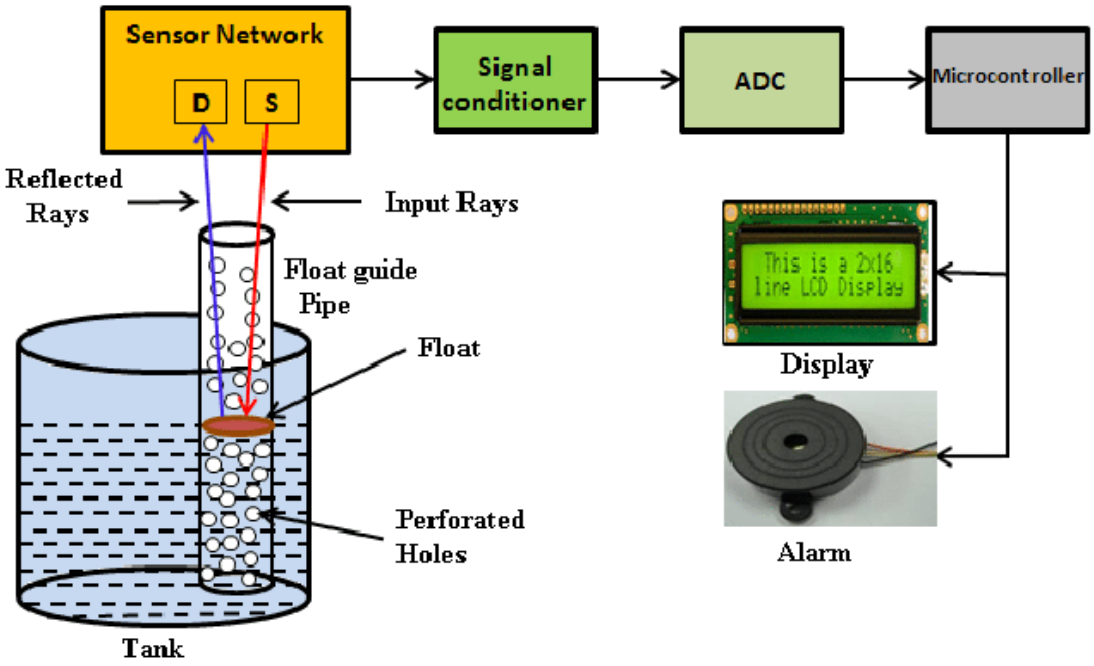
- The HC-SR04 ultrasonic distance sensor is a key component in water monitoring systems. It utilizes ultrasonic sound waves to measure the distance between the sensor and the water surface.
- When deployed above a water body, it calculates the water level by timing the sound wave's round trip.
- This data is crucial for monitoring water levels in reservoirs, rivers, or tanks, aiding in flood prediction, water resource management, and ensuring optimal water usage. Its non-contact nature and accuracy make it a valuable tool for real-time water level measurements, enabling timely responses to water level fluctuations and ensuring efficient water resource utilization.

PURPOSE:

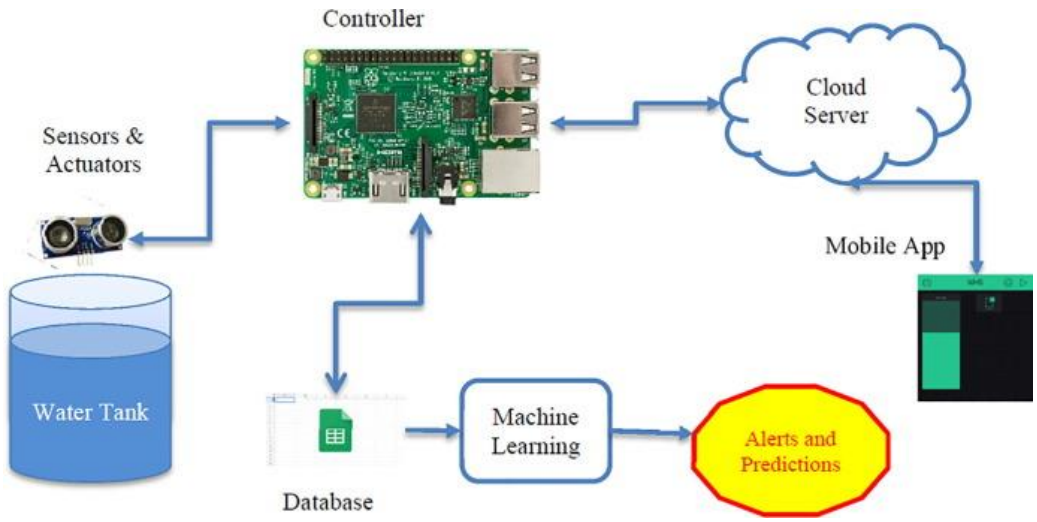
- This sensor emits ultrasonic pulses and calculates the time it takes for the signals to bounce back from the water's surface.

- By converting this time into distance, it provides accurate water level data, essential for monitoring and managing water resources

BLOCK DIAGRAM:



CONNECTING MOBILE APP WITH SMART WATER MANAGEMENT IOT PROJECT:



- 1.Set Up a Server:

✓ Ensure your Smart Water Management Iot project has a server that can receive and process data from IoT devices.

✓ Implement APIs on the server to provide data to the mobileapp.

- ✓ These APIs should handle incoming HTTP requests and return data in a format that the mobile app can understand (usually JSON).

2. Mobile App Development:

- ✓ Use a mobile app development framework like Flutter or React Native to create your mobile app. In this guide, I'll use Flutter.
- ✓ Create the app's user interface and layout, including buttons and widgets to display data.

3. HTTP Requests from Mobile App:

- ✓ Use the http package (or equivalent) in Flutter to send HTTP requests to the server.
- ✓ For example, you can use the http.get method to request data.

PROGRAM

```
/* Fill-in your Template ID (only if using Blynk.Cloud) */
#define BLYNK_TEMPLATE_ID "TMPLlcLQu4bQ"
#define BLYNK_TEMPLATE_NAME "water monitor"
#define BLYNK_AUTH_TOKEN "OgvenxCWu9sG7-9deFGLFCLE4rWCGW7N"
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "Wokwi-GUEST";   //WiFi Name
char pass[] = "";   //WiFi Password
//Set Water Level Distance in CM
int emptyTankDistance = 150 ;  //Distance when tank is empty
int fullTankDistance =  40 ;  //Distance when tank is full (must be greater than 25cm)
//Set trigger value in percentage
int triggerPer =    10 ;  //alarm/pump 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  //D6
#define ECHOPIN    26  //D7
#define wifiLed    2  //D0
#define BuzzerPin  13  //D3
#define RelayPin   14 //D5
#define ButtonPin1 12   //RX   //Mode
#define ButtonPin2 33  //SD3  //Relay
#define ButtonPin3 32  //D4   //STOP Buzzer
#define fullpin    25
//Change the virtual pins according the rooms
#define VPIN_BUTTON_1    V1
#define VPIN_BUTTON_2    V2
#define VPIN_BUTTON_3    V3
#define VPIN_BUTTON_4    V4
#define VPIN_BUTTON_5    V5
#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
bool toggleRelay = false; //Define the toggle state for relay
bool modeFlag = true;
bool conection = true;
String currMode;
char auth[] = BLYNK_AUTH_TOKEN;
ButtonConfig config1;
AceButton button1(&config1);
ButtonConfig config2;
AceButton button2(&config2);
ButtonConfig config3;
AceButton button3(&config3);
void handleEvent1(AceButton*, uint8_t, uint8_t);
void handleEvent2(AceButton*, uint8_t, uint8_t);
void handleEvent3(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);
    conection = true;

  }
  if (isconnected == true) {
    digitalWrite(wifiLed, HIGH);
    //Serial.println("Blynk Connected");
    conection = false;
  }
}
// When App button is pushed - switch the state
BLYNK_WRITE(VPIN_BUTTON_3) {
  modeFlag = param.asInt();
  if(!modeFlag && toggleRelay){
      digitalWrite(RelayPin, LOW);  //turn off the pump
      toggleRelay = false;
    }
    controlBuzzer(500);
    currMode = modeFlag ? "AUTO" : "MANUAL";
}
BLYNK_WRITE(VPIN_BUTTON_4) {
  if(!modeFlag){
    toggleRelay = param.asInt();
    digitalWrite(RelayPin, toggleRelay);
    controlBuzzer(500);
  }
  else{
    Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  }
}
BLYNK_WRITE(VPIN_BUTTON_5) {
  toggleBuzzer = param.asInt();
  digitalWrite(BuzzerPin, toggleBuzzer);
}
BLYNK_CONNECTED() {
  Blynk.syncVirtual(VPIN_BUTTON_1);
  Blynk.syncVirtual(VPIN_BUTTON_2);
  Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
  Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
```

```
}
void displayData(){
  display.clearDisplay();
  display.setTextSize(3);
  display.setCursor(30,0);
  display.print(waterLevelPer);
  display.print(" ");
  display.print("%");
  display.setTextSize(1);
  display.setCursor(0,25);
  display.print(conection ? "OFFLINE" : "ONLINE");
  display.setCursor(60,25);
  display.print(currMode);
  display.setCursor(110,25);
  display.print(toggleRelay ? "! ON" : "OFF");
  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
  // 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);
    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){
      if(modeFlag){
```

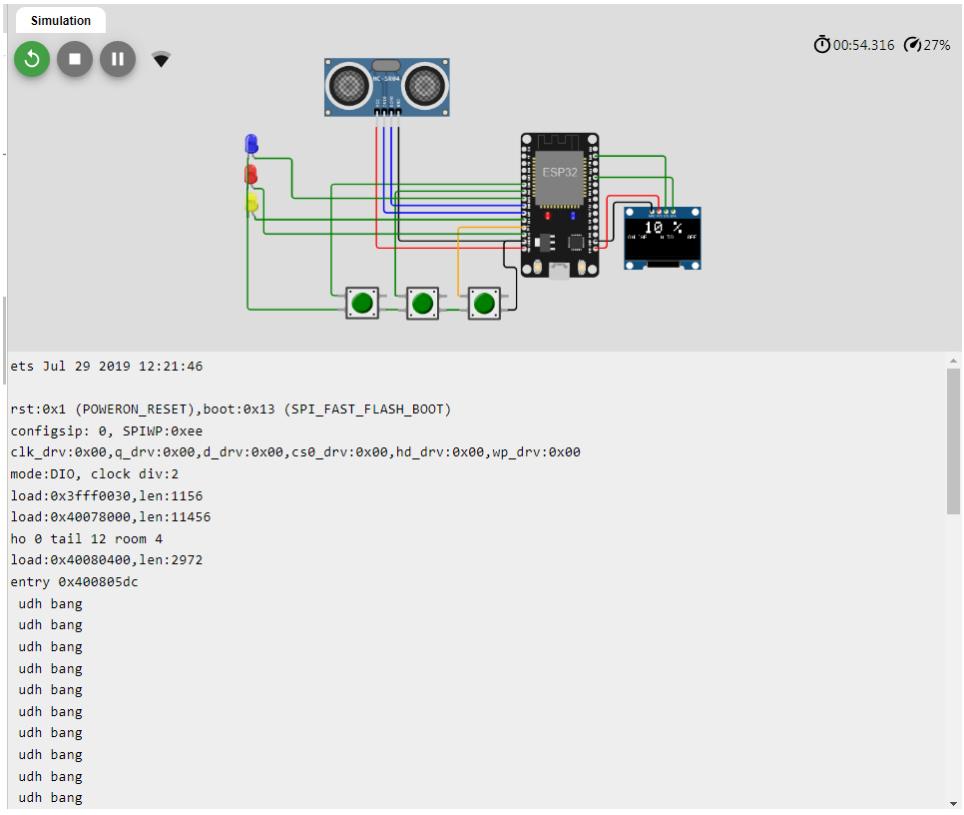
```
      if(!toggleRelay){
        controlBuzzer(500);
        digitalWrite(RelayPin, HIGH); //turn on relay
        toggleRelay = true;
        Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
      }
    }
    else{
      if (toggleBuzzer == HIGH){
        digitalWrite(BuzzerPin, HIGH);
        Serial.println(" BuzzerPin high");
      }
    }
  }
  if (distance < fullTankDistance){
    digitalWrite(fullpin, HIGH);
    if(modeFlag){
      if(toggleRelay){
        digitalWrite(RelayPin, LOW); //turn off relay
        toggleRelay = false;
        Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
        controlBuzzer(500);
      }
    }
    else{
      if (toggleBuzzer == HIGH){
      digitalWrite(BuzzerPin, HIGH);
      }
    }
  }
  if (distance > (fullTankDistance + 5) && waterLevelPer > (triggerPer + 5)){
    toggleBuzzer = HIGH;
    Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
    digitalWrite(BuzzerPin, LOW);
  }
  if (distance = fullTankDistance){
  Serial.println(" udh bang ");
  }
 }
 displayData();
 delay(100);
}
```

```
void controlBuzzer(int duration){
  digitalWrite(BuzzerPin, HIGH);
  Serial.println(" BuzzerPin HIT");
  delay(duration);
  digitalWrite(BuzzerPin, LOW);
}
void setup() {
  // Set up serial monitor
  Serial.begin(9600);
  // Set pinmodes for sensor connections
  pinMode(ECHOPIN, INPUT);
  pinMode(TRIGPIN, OUTPUT);
  pinMode(wifiLed, OUTPUT);
  pinMode(RelayPin, OUTPUT);
  pinMode(BuzzerPin, OUTPUT);
  pinMode(fullpin, OUTPUT);
  pinMode(ButtonPin1, INPUT_PULLUP);
  pinMode(ButtonPin2, INPUT_PULLUP);
  pinMode(ButtonPin3, INPUT_PULLUP);
  digitalWrite(wifiLed, HIGH);
  digitalWrite(RelayPin, LOW);
  digitalWrite(BuzzerPin, LOW);
  config1.setEventHandler(button1Handler);
  config2.setEventHandler(button2Handler);
  config3.setEventHandler(button3Handler);
  button1.init(ButtonPin1);
  button2.init(ButtonPin2);
  button3.init(ButtonPin3);
  currMode = modeFlag ? "AUTO" : "MANUAL";
  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
  timer.setInterval(1000L,  measureDistance); // measure water level every 1 seconds
  Blynk.config(auth);
```

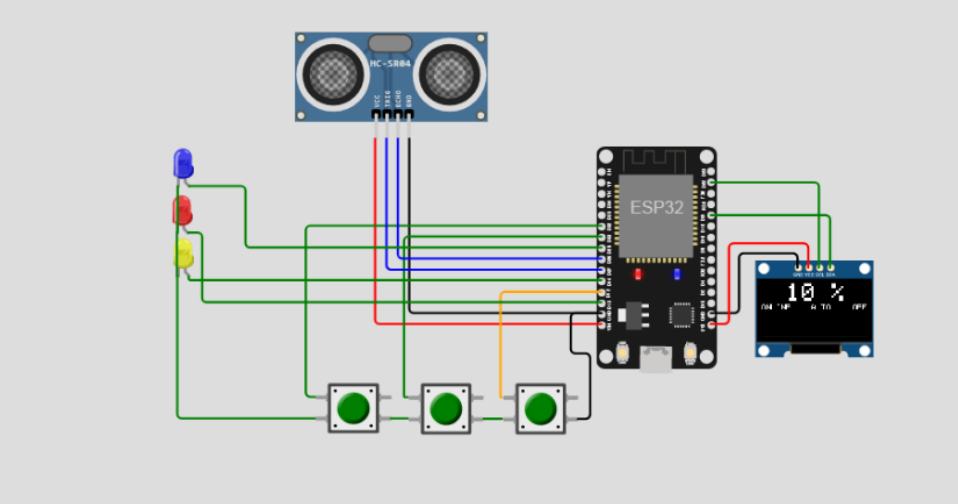
```
  delay(1000);
  Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
  Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
  delay(500);
}
void loop() {
  Blynk.run();
  timer.run(); // Initiates SimpleTimer
  button1.check(); //mode change
  button3.check(); //buzzer reset
  if(!modeFlag){  //if in manual mode
    button2.check();
  }
}
void button1Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT1");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      if(modeFlag && toggleRelay){
        digitalWrite(RelayPin, LOW);  //turn off the pump
        toggleRelay = false;
        controlBuzzer(500);
      }
      modeFlag = !modeFlag;
      currMode = modeFlag ? "AUTO" : "MANUAL";
      Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
      controlBuzzer(200);
      break;
  }
}
void button2Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT2");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      if(toggleRelay){
        digitalWrite(RelayPin, LOW);  //turn off the pump
        toggleRelay = false;
      }
      else{
```

```
      digitalWrite(RelayPin, HIGH);  //turn on the pump
      toggleRelay = true;
    }
    Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
    controlBuzzer(500);
    delay(1000);
    break;
  }
}
void button3Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT3");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      digitalWrite(BuzzerPin, LOW);
      toggleBuzzer = LOW;
      Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
      break;
  }
}
```

OUTPUT:



CIRCUIT DIAGRAM:



CONCLUSION:

- By leveraging IoT technology, this project has provided a cost-effective and efficient means of collecting and analyzing water quality data in real-time.
- This has wide-ranging implications for safeguarding our water resources, ensuring public health, and preserving the environment.As we continue to face challenges related to water scarcity and pollution, the insights and data generated by this project will play a vital role in making informed decisions and implementing proactive measures.
- In upcoming phase, we see html program to publish the iot collected informations to the mobile phones.

THANK YOU