IOT BASED SMART WATER FOUNTAINS…

Date: 24.10.2023

PHASE 4: Development Part 2

To build the IOT smart water fountain systems.

INTRODUCTION:

A water fountain or animated fountain is widely used these days to decorate city parks and squares. It can either spray water into the air or create a waterfall effect.

Water fountains mainly need three parts to operate: water source, water pump and delivery channel . All outdoor water fountains have a water source or reservoir from which they draw their water and to which the water returns, creating a closed circuit. The power to move the water is supplied by a pump. The pump is usually submerged in the water reservoir. A spinning impellor in the pump draws water in and is spinning at such a speed that the water is forced out of the pump by centrifugal force. The water forced from the pump is delivered to the delivery channel or fountain head where it is sprayed into the air through a fine nozzle or where it is allowed to flow down the outside of the structure of the fountain, The water that is sprayed into the air will fall back into the fountain and drain back into the reservoir.

COMPONENTS:

1. Arduino UNO
2. Transistor as switch
3. Water pump
4. 12v power supply
5. Jumper wires
6. Breadboard
7. Solenoid valve

Physical Design:

A pictorial representation of your project that puts your solution in context. Not necessarily restricted to your design. Include other external systems relevant to your project (e.g. if your solution connects to a phone via Bluetooth, draw a dotted line between your device and the phone). Note that this is not a block

diagram and should explain how the solution is used, not a breakdown of inner components.

His excellent build is outlined here, including a surprisingly simple circuit that controls the pump and solenoid valve illumination via a trio of MOSFETs.

Requirements:

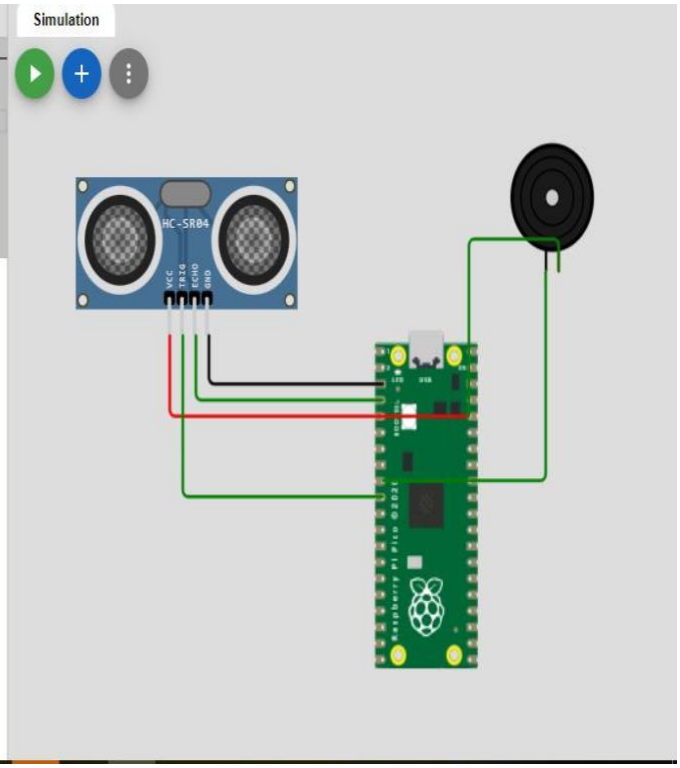
- Wokwi
- ThingSpeak
- ESP32
- Relay
- GND
- VCC
- Resistor
- Ultrasonic sensor
- LED

Wokwi:

Wokwi is an embedded systems and IoT simulator supporting ESP32, Arduino, and the Raspberry Pi Pico. Your code never leaves Your computer – Wokwi runs the simulation inside VS Code, using The firmware binaries from your project.

main.py diagram.json

```
from machine import Pin
import utime

trigger = Pin(7, Pin.OUT)
echo = Pin(2, Pin.IN)

# Define water level thresholds
WATER_LEVEL_LOW = 100  # Adjust this value based on your setup
WATER_LEVEL_HIGH = 400  # Adjust this value based on your setup

# Define buzzer pin and configure it as an output
buzzer_pin = Pin(6, Pin.OUT)

# Define the watering pin and configure it as an output
watering_pin = Pin(8, Pin.OUT)

def ultra():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    return distance

```



ThingSpeak:

ThingSpeak is an IoT analytics platform service that allows You to aggregate, visualize, and analyze live data streams in the Cloud. You can send data to ThingSpeak from your devices, create Instant visualizations of live data, and send alerts using web Services like Twitter and Twilio .

Procedure:

Step 1:Connect the ESP32 microcontroller with the ultrasonic sensor.

Step 2: Connect the LED’s and the relay with the GPIO pins of ESP32.

Step 3: Setup each step in the code and connect Wi-Fi to the ESP32.

Step 4: Declare and initialize each component of the smart water Management. For consumption the water level “Flow sensor” is Not available in the wokwi so, instead of this we used “Ultrasonic Sensor”.

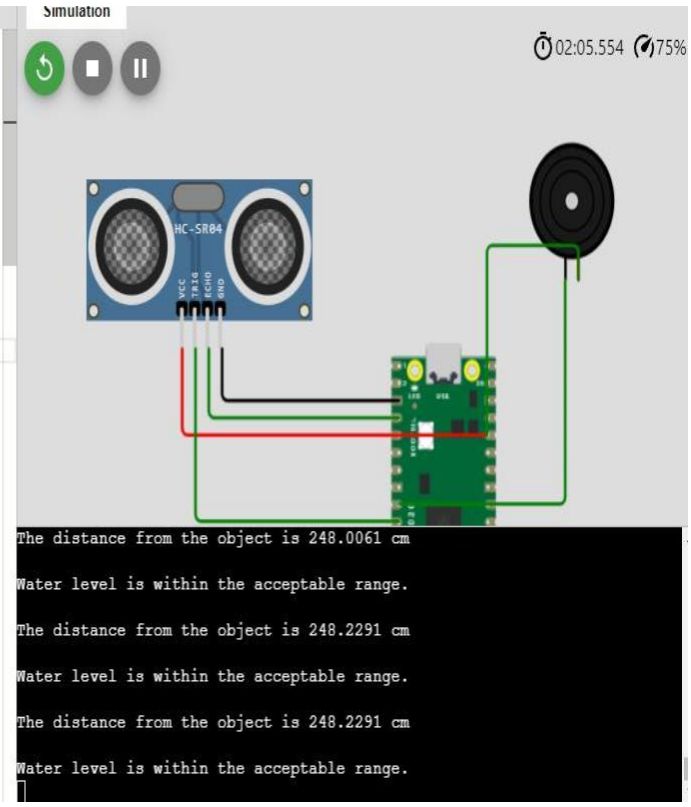
Step 5: The condition for glowing LED’s and motor is based on the level Of the water using the ultrasonic sensor.

Step 6: Run the simulation and check the distance of the water level Through LED’s glow, Motor run and push up messages.

Step 7: View the real-time data transmitting in the sensor through think speak.

main.py diagram.json

```
from machine import Pin
import utime

trigger = Pin(7, Pin.OUT)
echo = Pin(2, Pin.IN)

# Define water level thresholds
WATER_LEVEL_LOW = 100  # Adjust this value based on your setup
WATER_LEVEL_HIGH = 400  # Adjust this value based on your setup

# Define buzzer pin and configure it as an output
buzzer_pin = Pin(6, Pin.OUT)

# Define the watering pin and configure it as an output
watering_pin = Pin(8, Pin.OUT)

def ultra():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    return distance

```



Program (html):

```
<!DOCTYPE.html>

<html>

<head>

    <title>Firebase Data Retrieval</title>

    <!—Include the Firebase JavaScript SDK →

    <script src=p ></script>

   <script src= https://smart-water-fountains-default-rtdb.firebaseio.com/><script>

  <script src=https://www.gstatic.com/firebasejs/8.2.10/firebase-app.js></script>

</head>

<body>

    <h1>Firebase Data Retrieval</h1>

    <ul id=”data-list”>

        <!—Data will be displayed here →

    </ul>


    <!—Initialize Firebase →

    <script>

        Var firebaseConfig = {

            apiKey: “AIzaSyCZC3oIwQqjDh4Mk1SV4PdPzxmm2xQjrrM”,
```

```
authDomain: “ibmgroup5.firebaseapp.com”,
projectId: “ibmgroup5”,
storageBucket: “ibmgroup5.appspot.com”,
messagingSenderId: “624649784140”,
appId: “1:624649784140:web:52c7d72343628bbdd6dde9”
    };
    // Initialize Firebase
    Firebase.initializeApp(firebaseConfig);

    // Reference to your Firebase Realtime Database
    Var database = firebase.database();

    // Function to retrieve and display data
    Function retrieveData() {
        Var dataList = document.getElementById(‘data-list’);
        Var rootRef = database.ref();

        rootRef.on(‘value’, function(snapshot) {
            // Clear the existing data
            dataList.innerHTML = ‘’;
```

```
        snapshot.forEach(function(childSnapshot) {
            var childData = childSnapshot.val();
            // Display the retrieved data
            Var listItem = document.createElement('li');

            // Format the timestamp field to human-readable date and time
            If (childData.hasOwnProperty('timestamp')) {
                Var timestamp = new Date(childData.timestamp);
                childData.timestamp = timestamp.toLocaleString();
            }

            Var dataString = JSON.stringify(childData);
            listItem.innerHTML = dataString;
            dataList.appendChild(listItem);
        });
    });
}

// Initial data retrieval
```

```
        retrieveData();


        // Set up auto-refresh every 10 seconds
        setInterval(retrieveData, 10000); // 10000 milliseconds = 10 seconds
    </script>
</body>
</html>
```

PROGRAM:

```
 from machine import Pin
import utime


trigger = Pin(7, Pin.OUT)
echo = Pin(2, Pin.IN)


# Define water level thresholds
WATER_LEVEL_LOW = 100  # Adjust this value based on your setup
WATER_LEVEL_HIGH = 400  # Adjust this value based on your
setup
```

```
# Define buzzer pin and configure it as an output
buzzer_pin = Pin(6, Pin.OUT)

# Define the watering pin and configure it as an output
watering_pin = Pin(8, Pin.OUT)

def ultra():
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
        signaloff = utime.ticks_us()
    while echo.value() == 1:
        signalon = utime.ticks_us()
    timepassed = signalon - signaloff
    distance = (timepassed * 0.0343) / 2
    return distance

def activate_buzzer():
    # Function to activate the buzzer
    buzzer_pin.on()
```

```
    utime.sleep(1)  # Buzzer on for 1 second
    buzzer_pin.off()

def start_watering():
    # Function to start watering
    watering_pin.on()
    print("\nWatering started.")
    activate_buzzer()  # Activate the buzzer when watering
starts

def stop_watering():
    # Function to stop watering
    watering_pin.off()
    print("\nWatering stopped.")
    activate_buzzer()  # Activate the buzzer when watering
stops

def check_water_level():
    distance = ultra()
    print("\nThe distance from the object is", distance, "cm")

    # Check the water level
    if distance < WATER_LEVEL_LOW:
        print("\nWater level is low. Refill the tank.")
```

```
        start_watering()  # Start watering when water level is
low


    elif distance > WATER_LEVEL_HIGH:
        print("\nWater level is high. Tank is full.")
        stop_watering()  # Stop watering when water level is
high


    else:
        print("\nWater level is within the acceptable range.")


while True:
    check_water_level()
    utime.sleep(1)


 ------------------smart water fountain ------------------


import time  # Import the time module for time delays


# Define GPIO pin numbers
TRIG_PIN = 2  # GPIO pin number for the ultrasonic sensor's
trigger
ECHO_PIN = 3  # GPIO pin number for the ultrasonic sensor's
echo
PUMP_PIN = 4  # GPIO pin number for the water pump
```

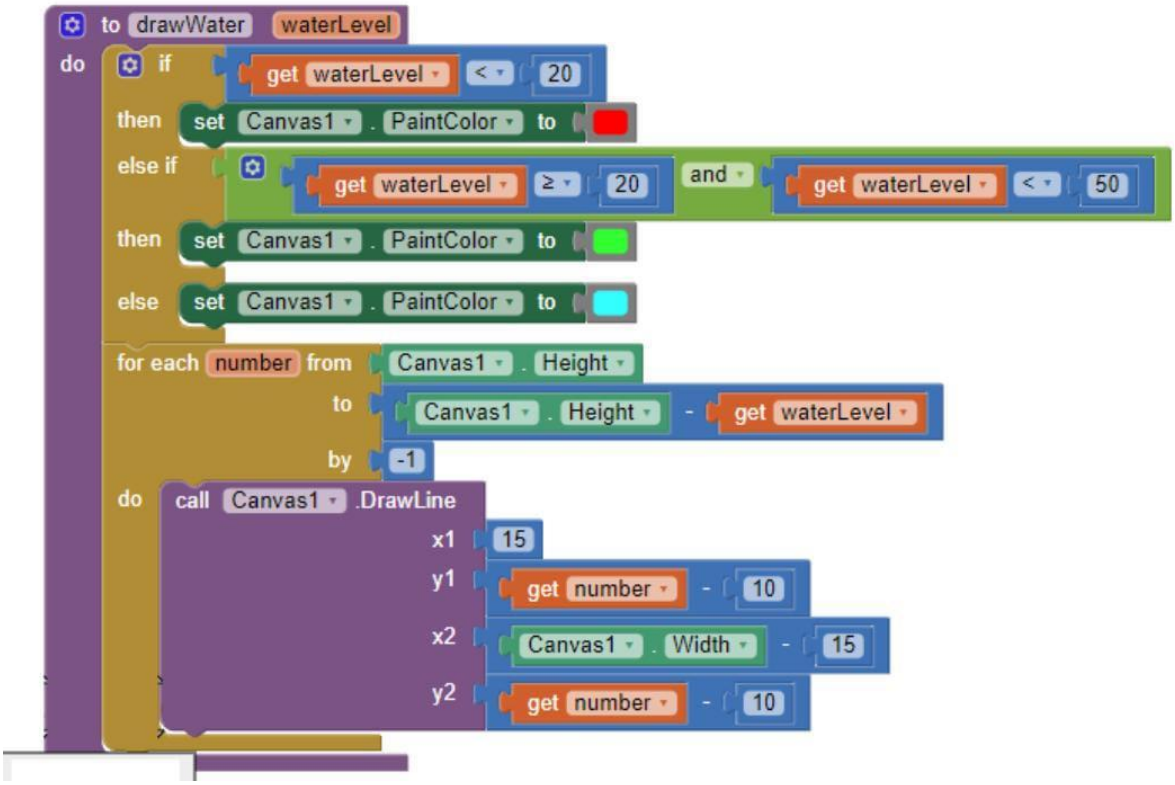
```
LED_PIN = 5   # GPIO pin number for the LED


# Initialize components (virtual components for Wokwi)
ultrasonic_sensor = Ultrasonic(TRIG_PIN, ECHO_PIN)  # Create an ultrasonic sensor
pump = Motor(PUMP_PIN)  # Create a water pump
led = LED(LED_PIN)  # Create an LED


while True:
    # Measure distance
    distance = ultrasonic_sensor.distance_cm  # Measure distance in centimeters


    if distance > 200:  # Water level is above 200 cm
        # Make the LED blink
        led.blink(on_time=0.5, off_time=0.5)  # LED blinks with 0.5 seconds on and off time
        pump.on()  # Water pump is turned on
    else:
        # Water level is below 200 cm
        # Turn off the LED and the pump
        led.off()
        pump.off()
```

```
    # Introduce a small delay to control the loop rate
time.sleep(0.1)  # Sleep for 0.1 second.
```

MID APP INVENTOR :

Mid app inventor link : https://bit.ly/3Hxu4PT

We are created this app.



This is a produce that draws a colored rectangle on Canvas1,Centered 10 pixels above the bottom, thin line by thin line, from the bottomTo the top. For height of water level, the input parameter.

The color chosen is red for low waterlevel (below 20), green for between 20 and 50, light blue over 50.

The loop could have been eliminated by setting the Canvas1 Line Width to the water level, and drawing it in 1 stroke. (A little research would reveal the proper y value).

There is a bug in this procedure. It fails to clear the Canvas, so the highest water level remains painted onto the canvas, like the inside of a commode tank (lift the lid if you dare.)