SMART WATER MANAGEMENT SYSTEM

DEVELOPEMENTAL PHASE

PHASE-IV

Name : MOHAMED AHIL A

Nm.no: au813821106069

Mail.id:mohamedahil.a2004@gmail.com

Code Implementation for raspberrypi pico:

from machine import Pin, ADC

import utime

# Ultrasonic sensor

trigger = Pin(2, Pin.OUT)

echo = Pin(3, Pin.IN)

distance = 0 # Initialize distance as a global variable

# Simulate the turbidity sensor with a potentiometer

potentiometer\_pin = 26 # Use Pin 26 for the ADC input

turbidity\_adc = ADC(Pin(potentiometer\_pin))

# Simulate the pH sensor with another potentiometer

ph\_potentiometer\_pin = 27 # Use Pin 27 for the ADC input of the simulated pH sensor

ph\_adc = ADC(Pin(ph\_potentiometer\_pin))

# Simulate the water flow sensor with a push button

water\_flow\_pin = 14 # Use Pin 14 for the push button input

water\_flow\_sensor = Pin(water\_flow\_pin, Pin.IN, Pin.PULL\_UP)

water\_flow\_count = 0

water\_flow\_pressed = False # Flag to track button press

# Function to perform filtering process

def perform\_filtering\_process():

# Simulated filtering process

print("Performing filtering process...")

def ultra():

global distance # Declare distance as a global variable

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

while True:

ultra()

utime.sleep(1)

if 70 < distance < 90:

print("Water tank is going to be full")

if distance < 100 and not (70 < distance < 90):

print("Water tank is full")

break

# Simulate turbidity level with the potentiometer

turbidity\_value = turbidity\_adc.read\_u16()

print("Turbidity Level:", turbidity\_value)

if turbidity\_value > 50000:

print("Water is dirty")

while turbidity\_value > 50000: # Keep filtering until water is pure

perform\_filtering\_process() # Function to perform filtering process

turbidity\_value = turbidity\_adc.read\_u16() # Update turbidity value

print("Turbidity Level:", turbidity\_value)

utime.sleep(1)

print("Water is now pure")

else:

print("Water is pure")

# Simulate pH level with the potentiometer

simulated\_ph\_value = ph\_adc.read\_u16()

print("Simulated pH Level:", simulated\_ph\_value)

if simulated\_ph\_value > 30000: # Adjust the threshold value as needed

print("Water pH is outside the acceptable range")

perform\_filtering\_process() # Function to perform the filtering process

else:

print("Water pH is within the acceptable range")

# Simulate water flow sensor with a push button

if not water\_flow\_sensor.value() and not water\_flow\_pressed:

water\_flow\_count += 1

water\_flow\_pressed = True # Set flag to True when button is pressed

print("Water flow count:", water\_flow\_count)

elif water\_flow\_sensor.value() and water\_flow\_pressed:

water\_flow\_pressed = False # Reset flag when button is released

utime.sleep(1)

DESCRIPTION:

1.Implements an IoT system using Raspberry Pi Pico microcontroller and various simulated sensors.

2.Simulates an ultrasonic sensor, turbidity sensor, pH sensor, and water flow sensor using potentiometers and push buttons.

3.Includes functions for ultrasonic distance measurement and filtering process simulation.

4.Monitors sensor values and performs actions based on the predefined thresholds.

HTML CODE :

<!DOCTYPE html>

<html>

<head>

<title>Data Display</title>

<script>

function displayData() {

// Simulated variables

let distance = 80;

let turbidity\_value = 60000; // Simulated turbidity value

let ph\_value = 20000; // Simulated pH value

let water\_flow\_count = 0;

let water\_flow\_pressed = false;

// Simulate function

function ultra() {

// Simulate the ultrasonic sensor

}

function perform\_filtering\_process() {

// Simulated filtering process

console.log("Performing filtering process...");

}

// Simulate data reading loop

setInterval(function () {

ultra();

if (70 < distance && distance < 90) {

console.log("Water tank is going to be full");

}

if (distance < 100 && !(70 < distance && distance < 90)) {

console.log("Water tank is full");

}

if (turbidity\_value > 50000) {

console.log("Water is dirty");

while (turbidity\_value > 50000) {

perform\_filtering\_process();

turbidity\_value -= 1000; // Simulate filtering process

console.log("Turbidity Level:", turbidity\_value);

}

console.log("Water is now pure");

} else {

console.log("Water is pure");

if (ph\_value > 30000) {

console.log("Water pH is outside the acceptable range");

perform\_filtering\_process(); // Simulate filtering process for pH adjustment

} else {

console.log("Water pH is within the acceptable range");

}

if (!water\_flow\_pressed) {

water\_flow\_count++;

water\_flow\_pressed = true;

console.log("Water flow count:", water\_flow\_count);

} else {

water\_flow\_pressed = false;

}

}

}, 1000);

// Update the HTML element with the fetched data

document.getElementById('data-container').innerHTML = 'Data displayed here';

}

</script>

</head>

<body onload="displayData()">

<h1>Data Display</h1>

<div id="data-container"></div>

</body>

</html>

DESCRIPTION:

1.Displays simulated sensor data in a web browser using JavaScript.

2.Simulates the behavior of various sensor readings, such as distance, turbidity, pH, and water flow.

3.Updates the displayed data in real-time using the setInterval function to mimic sensor data changes.

CODE TO FETCH DATA FROM THE WOWKI SIMULATION:

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Data Display</title>**

**<script>**

**function displayData() {**

**function fetchData() {**

**fetch('/data')**

**.then(response => response.json())**

**.then(data => {**

**document.getElementById('data-container').innerHTML = JSON.stringify(data);**

**})**

**.catch(error => {**

**console.error('Error fetching data:', error);**

**});**

**}**

**// Fetch data every 3 seconds**

**setInterval(fetchData, 3000);**

**}**

**</script>**

**</head>**

**<body onload="displayData()">**

**<h1>Data Display</h1>**

**<div id="data-container"></div>**

**</body>**

**</html>**

**Fetch Data from Wokwi Simulation:**

**1.Retrieves data from a Wokwi simulation using the fetch API in JavaScript.**

**2.Periodically fetches data every 3 seconds to ensure up-to-date information is displayed in the browser.**

**3.Handles any errors that might occur during the data retrieval process to maintain smooth operation.**

**CODE FOR SETTING UP SERVER:**

**import http.server**

**import socketserver**

**PORT = 8000**

**Handler = http.server.SimpleHTTPRequestHandler**

**with socketserver.TCPServer(("", PORT), Handler) as httpd:**

**print("Serving at port", PORT)**

**httpd.serve\_forever()**

**SERVER SETUP:**

**Sets up a basic HTTP server in Python using the http.server and socketserver modules.**

**Serves the specified HTML files at the designated port to allow access through a web browser.**

**Continuously runs the server to ensure the HTML files are accessible to clients and updates are reflected in real-time.**

**OUTPUT FROM THE BROWSER:**

