FLOOD MONITORING

PHASE – 4 DEVELOPMENT

Creating a real-time flood monitoring Management platform involves a combination of front end and backend technologies. Here's a simplified outline using C and C++ and python programming with wi-fi connection for the front end and Node.js for the back end:

Python:

```
import RPi.GPIO as GPIO

import time

import paho.mqtt.client as mqtt


# GPIO Pins for the ultrasonic sensor

TRIG = 23

ECHO = 24


# MQTT Configuration

mqtt_broker = "mqtt.example.com"

mqtt_topic = "flood_monitoring"


# Initialize GPIO

GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Create MQTT client
client = mqtt.Client()
client.connect(mqtt_broker)

def measure_water_level():
    # Trigger ultrasonic sensor
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    # Measure time for echo
    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()

    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()

    # Calculate distance (cm)
    pulse_duration = pulse_end - pulse_start
```

```python
    distance = pulse_duration * 17150

    return distance

try:
    while True:
        water_level = measure_water_level()
        print(f"Water level: {water_level} cm")

        # Publish data to MQTT
        client.publish(mqtt_topic, f"Water level: {water_level} cm")

        time.sleep(10)  # Read data every 10 seconds

except KeyboardInterrupt:
    GPIO.cleanup()
```

C program:

```c
#include <stdio.h>
#include <wiringPi.h>  // GPIO library for Raspberry Pi
#include <mosquitto.h> // MQTT library
```

```
#define TRIG 23 // GPIO pin for ultrasonic sensor trigger

#define ECHO 24 // GPIO pin for ultrasonic sensor echo


// MQTT Configuration

const char *mqtt_broker = "mqtt.example.com";

const char *mqtt_topic = "flood_monitoring";


// Ultrasonic sensor functions

void setupUltrasonicSensor() {

   wiringPiSetupGpio(); // Initialize wiringPi

   pinMode(TRIG, OUTPUT);

   pinMode(ECHO, INPUT);

}


double measureWaterLevel() {

   // Trigger ultrasonic sensor

   digitalWrite(TRIG, LOW);

   delay(2);

   digitalWrite(TRIG, HIGH);

   delayMicroseconds(10);

   digitalWrite(TRIG, LOW);
```

```
    // Measure time for echo
    while (digitalRead(ECHO) == LOW);
    long startTime = micros();
    while (digitalRead(ECHO) == HIGH);
    long travelTime = micros() - startTime;

    // Calculate distance (cm)
    double distance = travelTime / 58.0;
    return distance;
}

// MQTT functions
void mqtt_message_callback(struct mosquitto *mosq, void *userdata, const
struct mosquitto_message *message) {
    // Handle incoming MQTT messages here if needed
}

int main() {
    // Initialize ultrasonic sensor
    setupUltrasonicSensor();

    // Initialize MQTT
```

```
    mosquitto_lib_init();
    struct mosquitto *mosq = mosquitto_new(NULL, true, NULL);
    if (mosq) {
        mosquitto_connect(mosq, mqtt_broker, 1883, 60);
        mosquitto_subscribe(mosq, NULL, mqtt_topic, 0);
        mosquitto_message_callback_set(mosq, mqtt_message_callback);

        while (1) {
            double waterLevel = measureWaterLevel();
            printf("Water level: %.2f cm\n", waterLevel);

            // Publish data to MQTT
            char message[50];
            sprintf(message, "Water level: %.2f cm", waterLevel);
            mosquitto_publish(mosq, NULL, mqtt_topic, strlen(message), message, 0,
false);

            delay(10000); // Read data every 10 seconds
        }

        mosquitto_disconnect(mosq);
        mosquitto_destroy(mosq);
```

```
        mosquitto_lib_cleanup();
    } else {
        fprintf(stderr, "Error: Unable to initialize MQTT.\n");
    }

    return 0;
}
```

C++:

```
#include <iostream>
#include <wiringPi.h> // GPIO library for Raspberry Pi
#include <mosquitto.h> // MQTT library

#define TRIG 23 // GPIO pin for ultrasonic sensor trigger
#define ECHO 24 // GPIO pin for ultrasonic sensor echo

// MQTT Configuration
const char *mqtt_broker = "mqtt.example.com";
const char *mqtt_topic = "flood_monitoring";

// Ultrasonic sensor functions
void setupUltrasonicSensor() {
```

```
    wiringPiSetupGpio(); // Initialize wiringPi
    pinMode(TRIG, OUTPUT);
    pinMode(ECHO, INPUT);
}

double measureWaterLevel() {
    // Trigger ultrasonic sensor
    digitalWrite(TRIG, LOW);
    delay(2);
    digitalWrite(TRIG, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG, LOW);

    // Measure time for echo
    while (digitalRead(ECHO) == LOW);
    long startTime = micros();
    while (digitalRead(ECHO) == HIGH);
    long travelTime = micros() - startTime;

    // Calculate distance (cm)
    double distance = travelTime / 58.0;
    return distance;
```

```
}

// MQTT functions
void mqtt_message_callback(struct mosquitto *mosq, void *userdata, const struct mosquitto_message *message) {
   // Handle incoming MQTT messages here if needed
}

int main() {
   // Initialize ultrasonic sensor
   setupUltrasonicSensor();

   // Initialize MQTT
   mosquitto_lib_init();
   struct mosquitto *mosq = mosquitto_new(NULL, true, NULL);
   if (mosq) {
      mosquitto_connect(mosq, mqtt_broker, 1883, 60);
      mosquitto_subscribe(mosq, NULL, mqtt_topic, 0);
      mosquitto_message_callback_set(mosq, mqtt_message_callback);

      while (true) {
         double waterLevel = measureWaterLevel();
```

```
        std::cout << "Water level: " << waterLevel << " cm" << std::endl;


        // Publish data to MQTT
        char message[50];
        snprintf(message, sizeof(message), "Water level: %.2f cm", waterLevel);
        mosquitto_publish(mosq, NULL, mqtt_topic, strlen(message), message, 0,
false);


        delay(10000); // Read data every 10 seconds
      }


      mosquitto_disconnect(mosq);
      mosquitto_destroy(mosq);
      mosquitto_lib_cleanup();
   } else {
      std::cerr << "Error: Unable to initialize MQTT." << std::endl;
   }


   return 0;
}
```

Html:

```
<!DOCTYPE html>

<html>

<head>

  <title>Flood Monitoring</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      text-align: center;

    }

    h1 {

      color: #3498db;

    }

    #flood-level {

      font-size: 24px;

    }

  </style>

</head>

<body>

  <h1>Flood Monitoring System</h1>

  <p>Current Water Level:</p>

  <div id="flood-level">Loading...</div>
```

```
    <script>
        // Replace this with code to fetch real-time data from your IoT devices
        function updateFloodLevel() {
            // Simulated data (replace with actual data retrieval logic)
            const waterLevel = Math.random() * 100; // Replace with your IoT data

            // Update the web page with the new data
            const floodLevelElement = document.getElementById("flood-level");
            floodLevelElement.textContent = waterLevel.toFixed(2) + " cm"; //
Display the data with two decimal places

            // Refresh the data every 10 seconds (adjust as needed)
            setTimeout(updateFloodLevel, 10000);
        }

        // Start updating the flood level
        updateFloodLevel();
    </script>
</body>
</html>
```

MICROPROCESSOR PROGRAM:

```
//Early Flood Detection Using IOT ~ A project by Sabyasachi Ghosh
```

```
//<LiquidCrystal.h> is the library for using the LCD 16x2

#include <LiquidCrystal.h>

//"DHT.h" is the library for using the Temperature sensor DHT22

#include "DHT.h"

#define DHTPIN A0                    //here we are initialising a pin for DHT22

#define DHTTYPE DHT22                //We have to declare the type of DHT sensor we are using for its correct functionality

LiquidCrystal lcd(2,3,4,5,6,7);        // Create an instance of the LiquidCrystal library

DHT dht(DHTPIN, DHTTYPE);            // Create an instance of the DHT library for the DHT22 sensor

const int in=8;                      //This is the ECHO pin of The Ultrasonic sensor HC-SR04

const int out=9;                     //This is the TRIG pin of the ultrasonic Sensor HC-SR04

// Define pin numbers for various components

const int green=10;

const int orange=11;

const int red=12;

const int buzz=13;


void setup()

{
```

```
// Start serial communication with a baud rate of 9600
Serial.begin(9600);
// Initialize the LCD with 16 columns and 2 rows
lcd.begin(16, 2);
// Set pin modes for various components
pinMode(in, INPUT);
pinMode(out, OUTPUT);
pinMode(green, OUTPUT);
pinMode(orange, OUTPUT);
pinMode(red, OUTPUT);
pinMode(buzz, OUTPUT);
// Initialize the DHT sensor
dht.begin();
// Set initial states for LEDs and buzzer to LOW (off)
digitalWrite(green,LOW);
digitalWrite(orange,LOW);
digitalWrite(red,LOW);
digitalWrite(buzz,LOW);
// Display a startup message on the LCD
lcd.setCursor(0, 0);
lcd.print("Flood Monitoring");
lcd.setCursor(0,1);
```

```
  lcd.print("Alerting System");
  // Wait for 5 seconds and then clear the LCD
  delay(5000);
  lcd.clear();
}

void loop()
{
  // Read temperature and humidity from the DHT22 sensor
  float T = dht.readTemperature();
  float H = dht.readHumidity();
  // Check if the sensor data is valid
  if (isnan(H) && isnan(T)) {
    lcd.print("ERROR");
    return;
  }
  float f = dht.readTemperature(true);
  // Read distance from the ultrasonic sensor (HC-SR04)
  long dur;
  long dist;
  long per;
  digitalWrite(out,LOW);
```

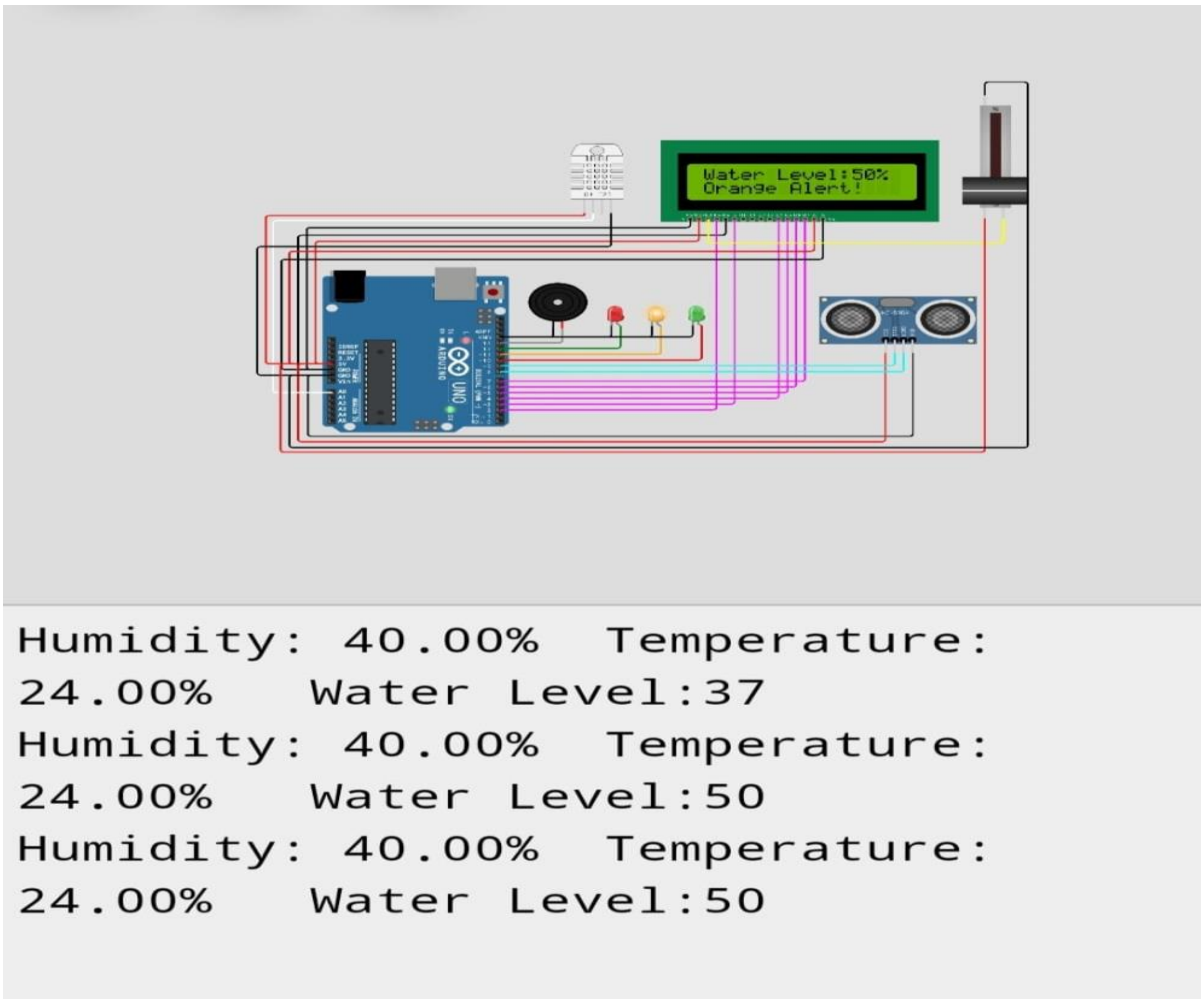
```
delayMicroseconds(2);
digitalWrite(out,HIGH);
delayMicroseconds(10);
digitalWrite(out,LOW);
dur=pulseIn(in,HIGH);
dist=(dur*0.034)/2;
// Map the distance value to a percentage value
per=map(dist,10.5,2,0,100);
// Ensure that the percentage value is within bounds
if(per<0)
{
  per=0;
}
if(per>100)
{
  per=100;
}
// Print sensor data and percentage value to serial
Serial.print(("Humidity: "));
Serial.print(H);
Serial.print(("%  Temperature: "));
Serial.print(T);
```

```
Serial.print("%   Water Level:");
Serial.println(String(per));
lcd.setCursor(0,0);
lcd.print("Temperature:");
lcd.setCursor(0,1);
lcd.print("Humidity   :");
lcd.setCursor(12,0);
lcd.print(T);
lcd.setCursor(12,1);
lcd.print(H);
delay(1000);
lcd.clear();
lcd.print("Water Level:");
lcd.print(String(per));
lcd.print("%  ");
// Check water level and set alert levels
if(dist<=3)
{
   lcd.setCursor(0,1);
   lcd.print("Red Alert!   ");
   digitalWrite(red,HIGH);
   digitalWrite(green,LOW);
```

```
    digitalWrite(orange,LOW);

    digitalWrite(buzz,HIGH);

    delay(2000);

    digitalWrite(buzz,LOW);

    delay(2000);

    digitalWrite(buzz,HIGH);

    delay(2000);

    digitalWrite(buzz,LOW);

    delay(2000);

}

else if(dist<=10)

{

    lcd.setCursor(0,1);

  lcd.print("Orange Alert!  ");

  digitalWrite(orange,HIGH);

  digitalWrite(red,LOW);

  digitalWrite(green,LOW);

  digitalWrite(buzz,HIGH);

  delay(3000);

  digitalWrite(buzz,LOW);

    delay(3000);
  }else
  {
```

```
     lcd.setCursor(0,1);
    lcd.print("Green Alert!  ");
    digitalWrite(green,HIGH);
    digitalWrite(orange,LOW);
    digitalWrite(red,LOW);
    digitalWrite(buzz,LOW);
  }
}
```

RESULT:



Conclusion:

In conclusion, the use of IoT for flood monitoring is a highly effective and promising solution. It enables real-time data collection and analysis, enhancing early warning systems and disaster management. By deploying sensors and connected devices in flood-prone areas, we can significantly improve our ability to monitor, predict, and respond to flood events, ultimately saving lives and reducing the impact of floods on communities and infrastructure. This technology has the potential to revolutionize flood management and improve resilience in the face of climate change and increasing flood risks.