**PHASE:5**

**FLOOD MONITORING AND EARLY WARNING**

**Team Member: A. Weis meurial gifta**

**Register Number:210521106059**

**Domain:IOT**

**Abstract**:

Flooding remains a pervasive and destructive natural disaster affecting communities worldwide. This research proposes an innovative solution to address this challenge through the development of a Flood Monitoring and Early Warning System utilizing the ESP32 microcontroller within the Internet of Things (IoT) framework.The system comprises a network of sensors strategically placed in flood-prone areas, including water level sensors and weather monitoring devices. The ESP32 microcontroller serves as the central processing unit, efficiently collecting and processing real-time data from these sensors. The seamless integration of ESP32 with IoT protocols enables the continuous transmission of data to a centralized server.To enhance the system’s predictive capabilities, machine learning algorithms are employed to analyze historical and real-time data patterns. This data-driven approach allows for the identification of potential flood events before they escalate. The ESP32’s computational capabilities and low-power features make it a suitable choice for on-the-edge processing, ensuring timely and accurate flood predictions.In the event of an impending flood, the system triggers early warnings through various communication channels such as Short Message Service (SMS) or push notifications. This multi-channel approach ensures that warnings reach the affected population swiftly, providing them with crucial time to evacuate or take necessary precautions.The prototype implementation of the proposed system demonstrates its effectiveness in real-world scenarios. Performance evaluations indicate the reliability and efficiency of the ESP32-based solution in monitoring floods and issuing early warnings. The system’s low-cost design, scalability, and ease of integration make it a practical and accessible tool for communities prone to flooding.This Flood Monitoring and Early Warning System using ESP32 in IoT not only contributes to the advancement of disaster management but also underscores the importance of leveraging cutting-edge technologies for proactive and community-centric solutions to mitigate the impact of natural disasters.

**CHAPTER 1:**

Introduction to flood monitoring and early warning using esp32

**INTRODUCTION**:

Flood monitoring with ESP32 is a critical application that leverages the capabilities of this versatile microcontroller to detect and respond to potential flooding events. By utilizing various sensors and communication modules, the ESP32 can collect real-time data on water levels, weather conditions, and more. This data can be analyzed and shared to provide early warnings and aid in flood mitigation efforts, making it an invaluable tool for disaster management and public safety. In this context, I can provide more information on how to set up flood monitoring using ESP32 if you’d like.

**Step1**: **Hardware** **Selection**:

Choose appropriate sensors for flood monitoring, such as water level sensors, rain gauges, and weather sensors. Select an ESP32 development board with built-in Wi-Fi or LoRa capabilities for data transmission.Consider a power source, such as a battery or solar panel, to ensure continuous operation.

**Step** 2: **Sensor** **Integration**:

Connect the selected sensors to the ESP32 using the appropriate interfaces (analog, digital, I2C, etc.).Write code to read data from these sensors. Ensure accuracy and reliability.

**Step** 3: **Data** **Collection**:

Program the ESP32 to collect data from the sensors at regular intervals. Implement error handling and data validation to ensure data accuracy.

**Step** 4: **Data** **Transmission**:

Use Wi-Fi or LoRa (Long-Range) communication to transmit data to a central system or the cloud.Implement secure data transmission protocols if necessary.

**Step** 5:**lasso**

LASSO (Locally Adaptive Sparsity prior for Structured Optimization) is primarily a machine learning algorithm used for regression and feature selection. It may not be directly applied in a straightforward manner on a microcontroller like the ESP32 due to computational and memory constraints. However, machine learning models can be trained using more powerful hardware (e.g., servers, cloud services) and then deployed onto the ESP32 for inference.

**Step** 6: **Data** **Storage** **and** **Analysis**:

Set up a central system or cloud platform to receive and store the data. Implement data analysis algorithms to detect flood conditions or trends in the data.

**Step** 7: **Alerting** **System**:

Develop an alerting mechanism that triggers when flood conditions are detected Notifications can be sent through SMS, email, or other means.

**Step** 8: **Power** **Management**:

Ensure efficient power management to extend the ESP32’s battery life or use alternative power sources like solar panels.

**Step** 9: **User** **Interface** :

Create a user interface for monitoring the system and viewing historical data.

**Step** **10**: **Testing** **and** **Calibration**:

Thoroughly test the system under different conditions to ensure its reliability and accuracy.Calibrate the sensors if needed to improve data accuracy.

**Step** **11**: **Deployment**:

Install the flood monitoring system in flood-prone areas, ensuring proper mounting, waterproofing, and protection from environmental factors.

**Step** **12**: **Logistic**

Implementing a logistic regression model on an ESP32 for flood monitoring involves several steps. Logistic regression is commonly used for binary classification problems, making it suitable for predicting whether a flood event is likely or not based on input features. Here’s a general guide:Data Collection: Gather historical data related to flood events and relevant environmental parameters. This dataset should include instances labeled with whether a flood occurred or

**Step** **13**: **Maintenance** **and** **Monitoring**:

Regularly maintain and monitor the system to ensure it continues to operate effectively.

**Step** **14**: **Data** **Visualization**:

Create dashboards or reports to visualize the data, making it easier to interpret and respond to flood conditions.Remember that designing and implementing a flood monitoring system is a complex task. Depending on your specific requirements and environmental conditions, the design may need to be customized. Additionally, ensure compliance with any local regulations and consider working with experts in the field of flood monitoring for a comprehensive and effective system.

**Program**

#include <WiFi.h>

Const char\* ssid = “YourWiFiSSID”;

Const char\* password = “YourWiFiPassword”;

Const char\* serverIP = “YourServerIP”;

Const int serverPort = 80;

Const int sensorPin = 2; // Replace with the GPIO pin connected to your water level sensor

Bool floodDetected = false;

Void setup()

{

Serial.begin(115200);

WiFi.begin(ssid, password);

While (WiFi.status() != WL\_CONNECTED) {

Delay(1000);

Serial.println(“Connecting to WiFi…”);

}

Serial.println(“Connected to WiFi”);

}

Void loop() {

Int waterLevel = digitalRead(sensorPin);

If (waterLevel == HIGH) {

If (!floodDetected) {

floodDetected = true;

sendFloodAlert();

}

} else {

floodDetected = false;

}

Delay(1000); // Check the sensor every second

}

Void sendFloodAlert() {

WiFiClient client;

If (client.connect(serverIP, serverPort)) {

Client.print(“GET /flood\_alert HTTP/1.1\r\n”);

Client.print(“Host: “);

Client.print(serverIP);

Client.print(“\r\n\r\n”);

Delay(10);

Client.stop();

Serial.println(“Flood Alert Sent”);

} else {

Serial.println(“Connection failed”);

}

}

**CHAPTER 2:**

Introducing components and programe

**Definition**:

Natural disasters such as floods and heavy rains pose threats to Lives and property around the world. Without a proper flood monitoring system, These natural events often lead to disasters, severely affecting economic losses, Social chaos, and urban environmental damage. Although IoT solutions cannot Prevent flooding, real-time monitoring of data can help minimize potential Damage by building proactive solutions for the community.

**Components**:

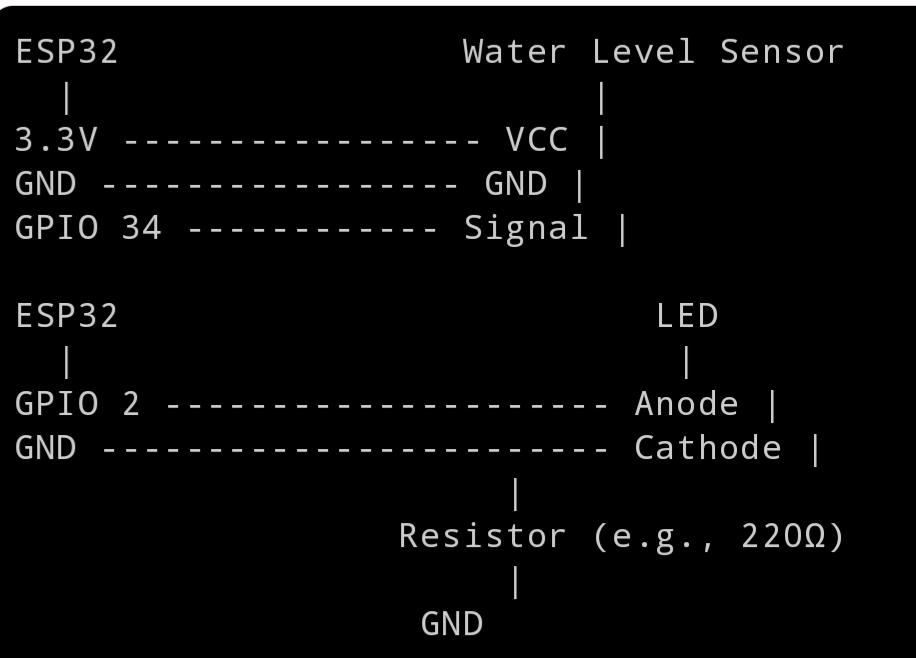
1.ESP32 Development Board

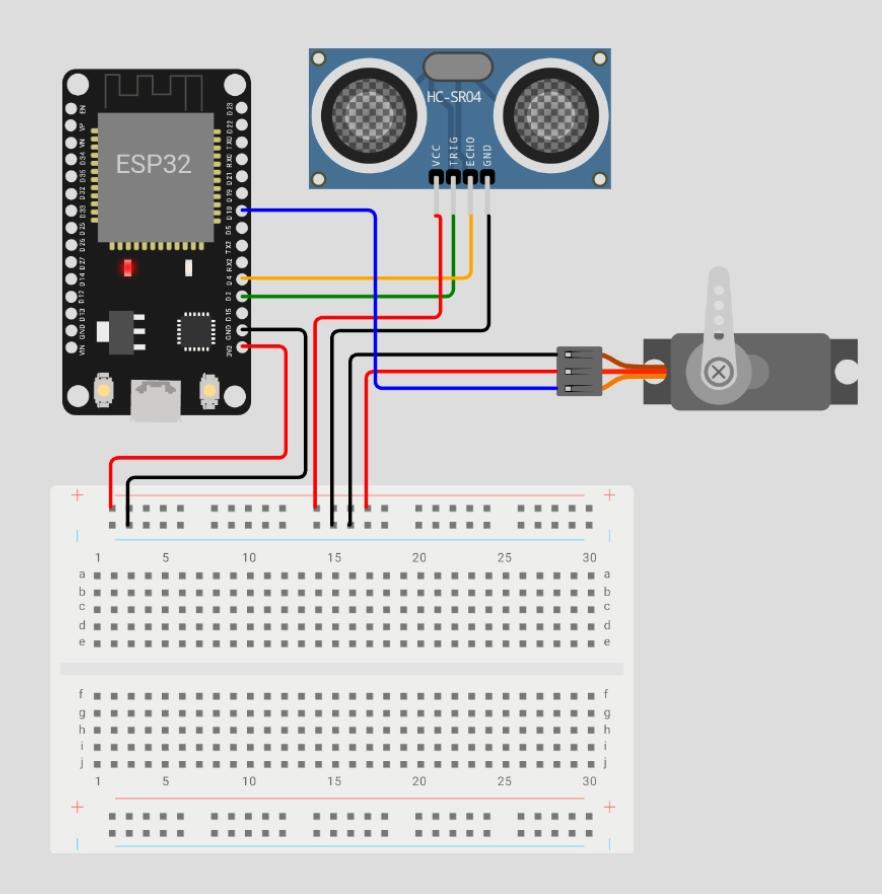
2.Water Level SensorLED (for warning)

3.Resistors, jumper wires, and a breadboard

Circuit Diagram(word Representation):

**Picture**:





**Python** **code**:

Import network

Import machine

Import time

From umqtt.simple import MQTTClient

# WiFi credentials

Ssid = “YourWiFiSSID”

Password = “YourWiFiPassword”

# MQTT broker details

Mqtt\_broker = “mqtt.eclipse.org”

Mqtt\_port = 1883

Mqtt\_topic = b”esp32/flood”

# Pin configuration

Water\_level\_pin = 34 # GPIO pin for water level sensor

Led\_pin = 2 # GPIO pin for warning LED

# Threshold for flood warning

Threshold = 500

Def connect\_to\_wifi():

Sta\_if = network.WLAN(network.STA\_IF)

Sta\_if.active(True)

Sta\_if.connect(ssid, password)

While not sta\_if.isconnected():

Pass

Def send\_mqtt\_message(message):

Client = MQTTClient(“esp32”, mqtt\_broker, port=mqtt\_port)

Client.connect()

Client.publish(mqtt\_topic, message)

Client.disconnect()

Def check\_flood():

Water\_level = adc.read()

If water\_level > threshold:

Print(“Flood warning!”)

Machine.Pin(led\_pin, machine.Pin.OUT).on() # Turn on LED

Send\_mqtt\_message(“Flood Warning!”)

Else:

Machine.Pin(led\_pin, machine.Pin.OUT).off() # Turn off LED

Def main():

Connect\_to\_wifi()

# Set up ADC for water level sensor

Adc = machine.ADC(machine.Pin(water\_level\_pin))

Adc.atten(machine.ADC.ATTN\_11DB)

# Set up LED pin

Machine.Pin(led\_pin, machine.Pin.OUT).off()

While True:

Check\_flood()

Time.sleep(300) # Sleep for 5 minutes before the next reading

If \_\_name\_\_ == “\_\_main\_\_”:

Main()

**CHAPTER 3:**

Processing data using code

**Required web development platform:**

➢ Html/css:Design the dashboard’s layout and style using HTML and CSS.

➢ Java script: Implement interactivity for real-time updates, charts, and user management.

➢ Web framework:can use popular frameworks like React, Angular, or Vue.js for a organized and responsive interface.

**UPDATES** **AND** **REPORTING**:

❖ **Develop** **Backend** **APIs**:

Create a set of API endpoints on your server to handle various functionalities of the early smart flood monitoring such as user authentication, parking spot availability, reservations, and payments. You can use a web framework like Express.js (Node.js) or Django (Python) to develop these APIs.

❖ **User** **data** **analysis**:

Use web development technologies to ensure real-time updates on parking spot availability, reservation confirmation, and payment status. You can achieve this with technologies like WebSocket for real-time communication between the server and clients. WebSocket: Implement WebSocket communication to push realtime updates to the web and mobile clients when a parking spot’s status changes.

❖ **Parking** **Spot** **Availability**:

Develop an API endpoint to provide real-time information about flood updatespot availability.

❖ **Reservations**:

Create APIs for reserving flood monitoring. When a user selects a spot and reserves it, the mobile app should send a request to the reservation API. Implement logic to check spot availability and confirm the reservation. Return a response .

❖ **API** **Integration**:

Use HTTP requests (e.g., GET, POST, PUT, DELETE) in the mobile app to communicate with the backend APIs. Handle API responses in the app to update the user interface and provide feedback to the user.

❖ **Testing** **and** **Debugging**:

Test the authenticating functionality by creating test scenarios and debugging any issues that arise. Verify that the app can interact seamlessly with the backend APIs.

❖ **Deployment**:

Deploy the mobile app to app stores (Google Play Store and Apple App Store) for public use.

❖ **User** **Support** **and** **Updates**:

Provide ongoing support and maintenance for the mit app Implement updates as needed, addressing user feedback and making improvements.

**PROGRAM**:

Creating a complete mobile app for an IoT early flood monitoringSystem is a complex task that requires a significant amount of code and development effort. I can provide you with a simplified example of a Python program using the Kivy framework to create a basic user interface for a mobile app. Please note that this example is a basic starting point, and it would need to extend it significantly to implement the full functionality of the Smart flood watering System.

**CODE**:

<!DOCTYPE HTML>

<

Hielo by TEMPLATED templated.co @templatedco

Released for free under the Creative Commons Attribution 3.0 license (templated.co/license)

<html>

<head>

<title>DISASTER MANAGEMENT</title>

<meta charset=”utf-8” />

<meta name=”viewport” content=”width=device-width, initialscale=1” />

<link rel=”stylesheet” href=”assets/css/main.css” />

</head>

<body class=”subpage”>

<!—Header 🡪

<header id=”header”>

<div class=”logo”><a href=”index.html”>DISASTER

MANAGEMENT</a></div>

<a href=”#menu”>Menu</a>

</header>

<!—Nav 🡪

<nav id=”menu”>

<ul class=”links”>

<li><a href=”index.html”>Home</a></li>

<li><a href=https://www.accuweather.com/>Weather Report</a></li>

<li><a

Href=https://www.theweathernetwork.com/maps/currentweather>current news</a></li>

</ul>

</nav>

<!—One 🡪

<section id=”One” class=”wrapper style3”>

<div class=”inner”>

<header class=”align-center”>

<h2>FLOOD PREPARATION</h2>

</header>

</div>

</section>

<!—Two 🡪

<section id=”two” class=”wrapper style2”>

<div class=”inner”>

<div class=”box”>

<div class=”content”>

<!--<header class=”align-center”>

<p>maecenas sapien feugiat ex purus</p>

<h2>Lorem ipsum dolor</h2>

</header>🡪

<img src=”images/flood.jpg” width=”100%” height=”500px”></br></br>

Floods

</br>

▪ Failing to evacuate flooded areas, entering flood waters, or remaining after a flood has passed can result in injury or death. Flooding is a temporary overflow of water onto land that is normally dry. Floods are the most common natural disaster in the United States. Floods may:

<ul>

<li>Result from rain, snow, coastal storms, storm surges, and overflows of dams and other water systems.</li>

<li>Develop slowly or quickly – Flash floods can come with no warning.</li>

<li>Cause outages, disrupt transportation, damage buildings, and create landslides.</li>

</ul>

This code provides a very basic user interface for the flood monitoring System.

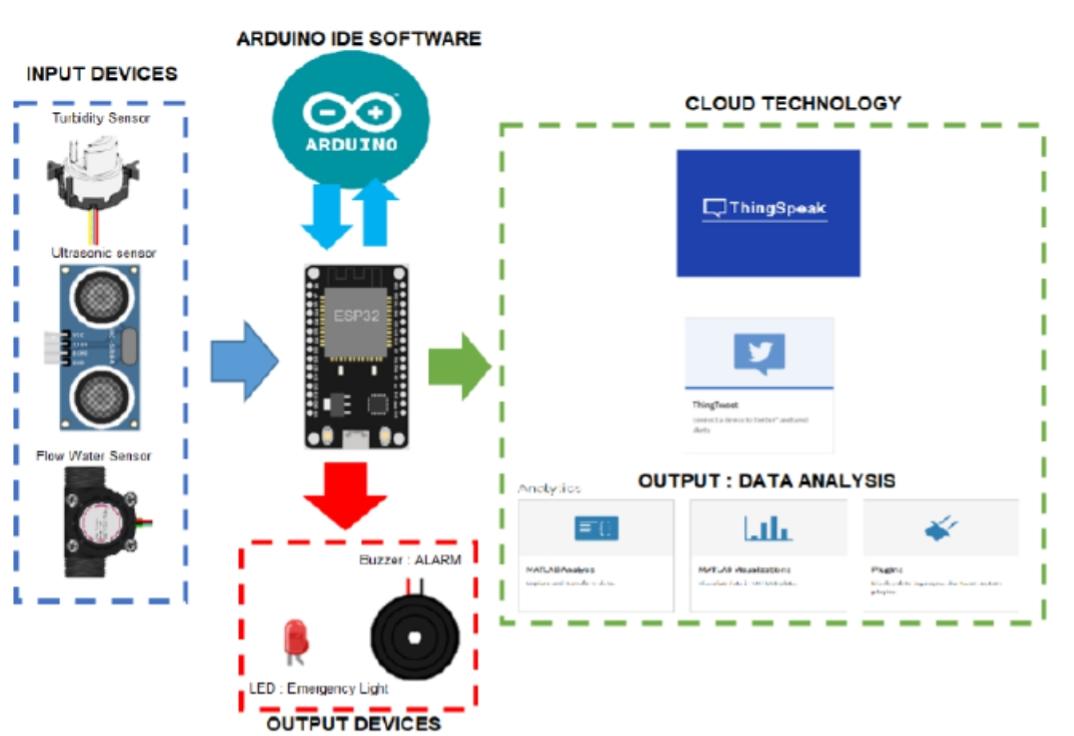
**CHAPTER 4:**

RealTimeprocessingthe data

**Definition :**

Creating a complete flood monitoring system using an ESP32 microcontroller involves various components, including sensors, connectivity, and code. Below is a simplified example of source code for an ESP32-based flood monitoring system using a water level sensor. This code assumes that you have already set up the necessary hardware, including an ESP32 and a water level sensor.

**Diagram:**

****

**Program 1:**

#include <Wire.h>

#include <WiFi.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BME280.h>

// Replace with your network credentials

Const char\* ssid = “your\_SSID”;

Const char\* password = “your\_PASSWORD”;

// Replace with your ThingSpeak channel information

Const char\* server = “api.thingspeak.com”;

Const char\* apiKey = “your\_API\_KEY”;

// BME280 sensor setup

Adafruit\_BME280 bme;

Void setup() {

Serial.begin(115200);

// Connect to Wi-Fi

WiFi.begin(ssid, password);

While (WiFi.status() != WL\_CONNECTED) {

Delay(1000);

Serial.println(“Connecting to WiFi…”);

}

Serial.println(“Connected to WiFi”);

// Initialize BME280 sensor

If (!bme.begin()) {

Serial.println(“Could not find a valid BME280 sensor, check wiring!”);

While (1);

}

}

Void loop() {

// Read sensor data

Float temperature = bme.readTemperature();

Float humidity = bme.readHumidity();

Float pressure = bme.readPressure() / 100.0F;

// Print sensor data

Serial.print(“Temperature: “);

Serial.print(temperature);

Serial.println(“ \*C”);

Serial.print(“Humidity: “);

Serial.print(humidity);

Serial.println(“ %”);

Serial.print(“Pressure: “);

Serial.print(pressure);

Serial.println(“ hPa”);

// Send data to ThingSpeak

If (WiFi.status() == WL\_CONNECTED) {

WiFiClient client;

If (client.connect(server, 80)) {

String postStr = “api\_key=” + String(apiKey) + “&field1=” + String(temperature) +

“&field2=” + String(humidity) + “&field3=” + String(pressure);

Client.println(“POST /update HTTP/1.1”);

Client.println(“Host: “ + String(server));

Client.println(“Connection: close”);

Client.println(“Content-Type: application/x-www-form-urlencoded”);

Client.println(“Content-Length: “ + String(postStr.length()));

Client.println();

Client.print(postStr);

Delay(500); // Wait for server response

Serial.println(“Data sent to ThingSpeak”);

Client.stop(); // Close connection

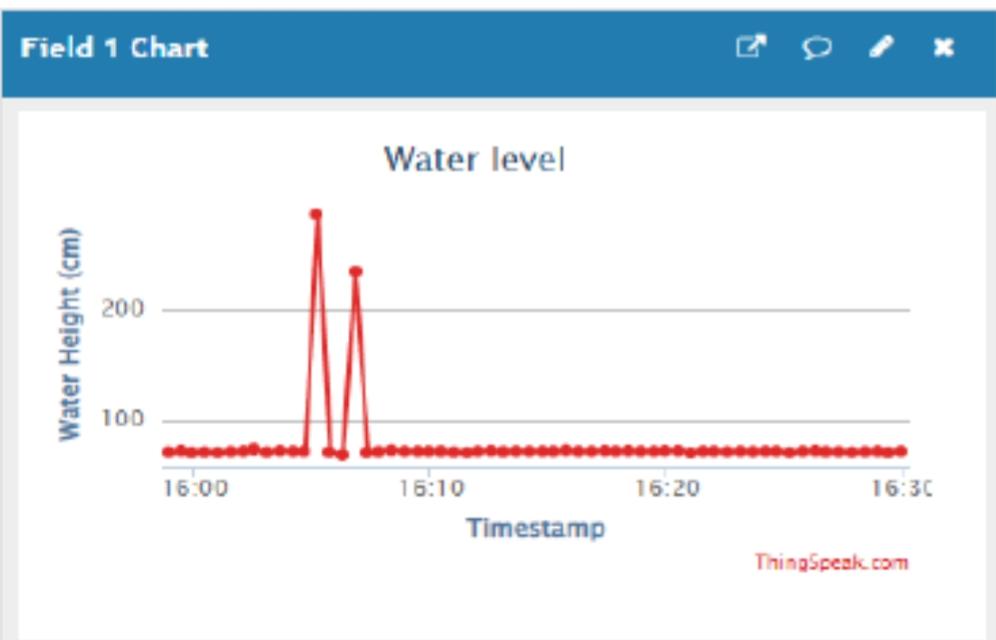
}

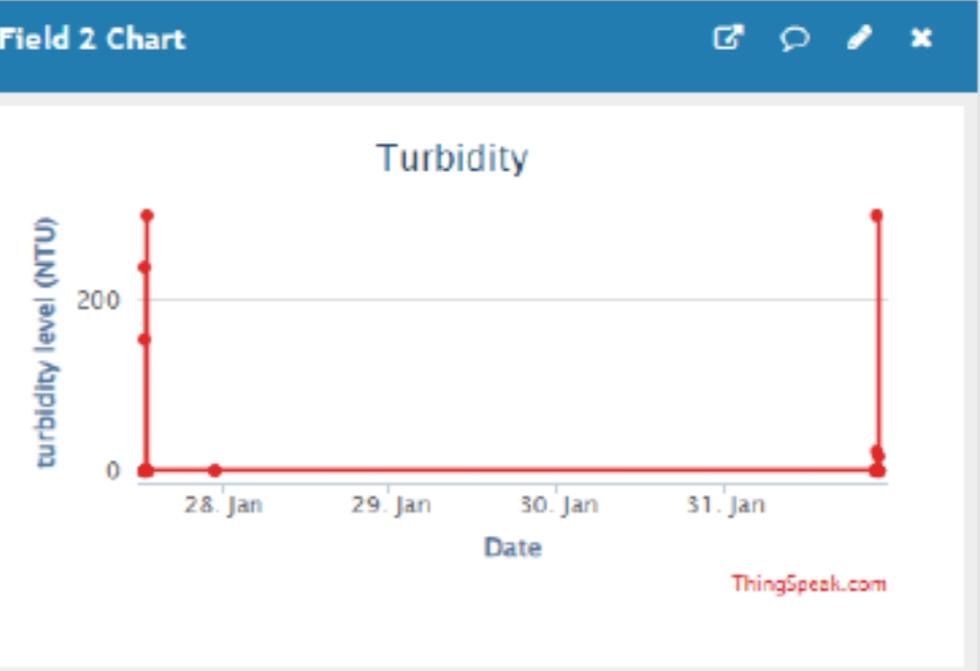
}

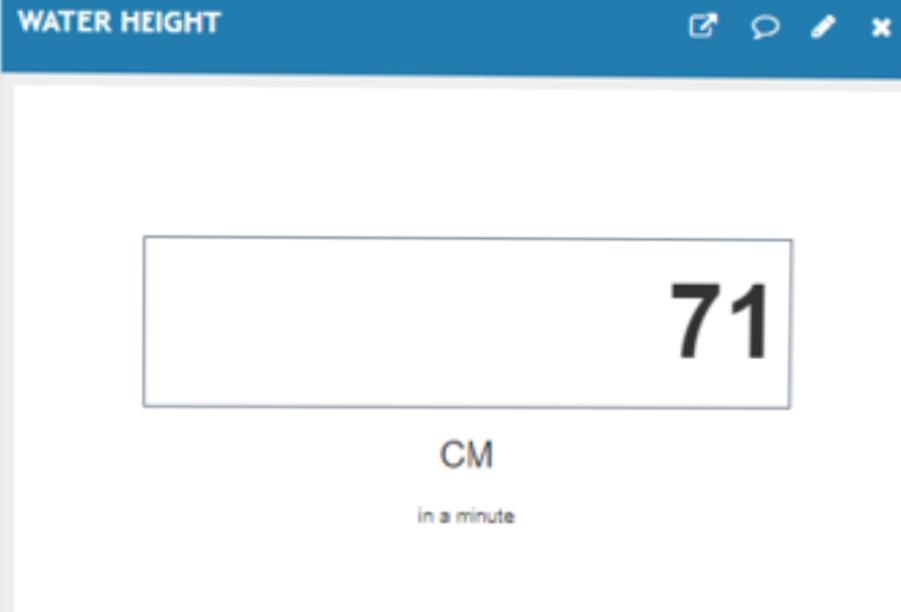
Delay(60000); // Delay for 1 minute before the next reading

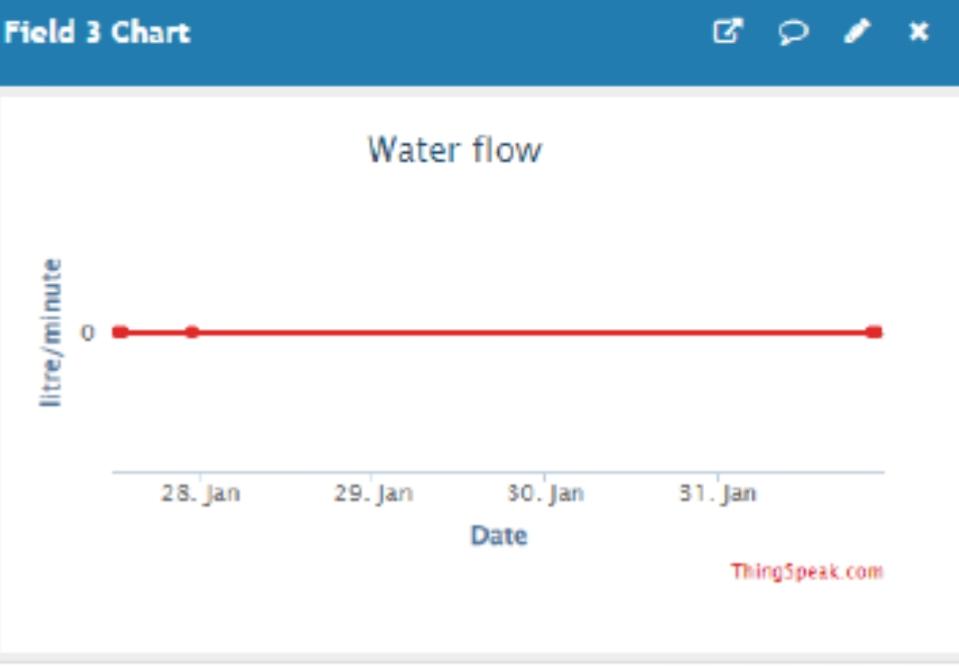
}

**Output in images:**

****







**Program 2 :**

```cpp

#include <WiFi.h>

#include <Wire.h>

Const char\* ssid = “Your\_WiFi\_SSID”;

Const char\* password = “Your\_WiFi\_Password”;

Int waterLevelPin = 34; // The pin connected to the water level sensor

Int alertPin = 2; // The pin to trigger an alert (e.g., a buzzer or LED)

Void setup() {

Serial.begin(115200);

pinMode(waterLevelPin, INPUT);

pinMode(alertPin, OUTPUT);

// Connect to Wi-Fi

WiFi.begin(ssid, password);

While (WiFi.status() != WL\_CONNECTED) {

Delay(1000);

Serial.println(“Connecting to WiFi…”);

}

Serial.println(“Connected to WiFi”);

}

Void loop() {

Int waterLevel = digitalRead(waterLevelPin);

If (waterLevel == HIGH) {

// Water level is high, indicating a flood

digitalWrite(alertPin, HIGH); // Activate alert (buzzer/LED)

Serial.println(“Flood detected!”);

} else {

digitalWrite(alertPin, LOW); // Deactivate alert

Serial.println(“No flood”);

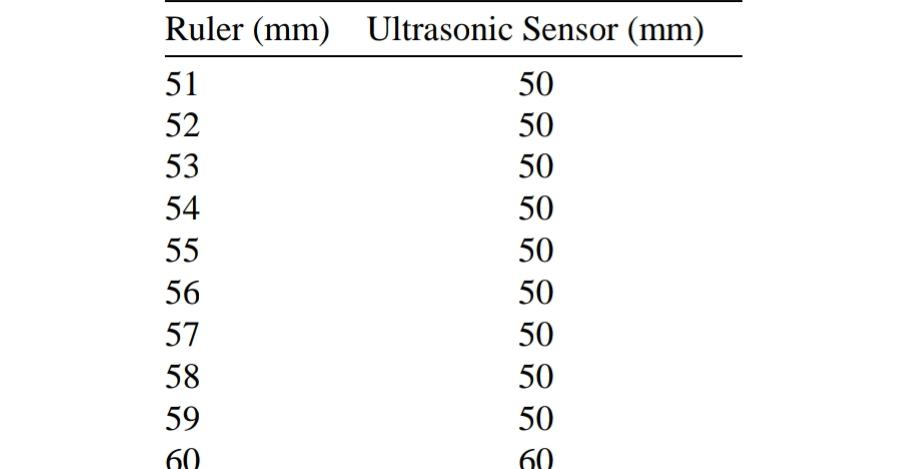
}

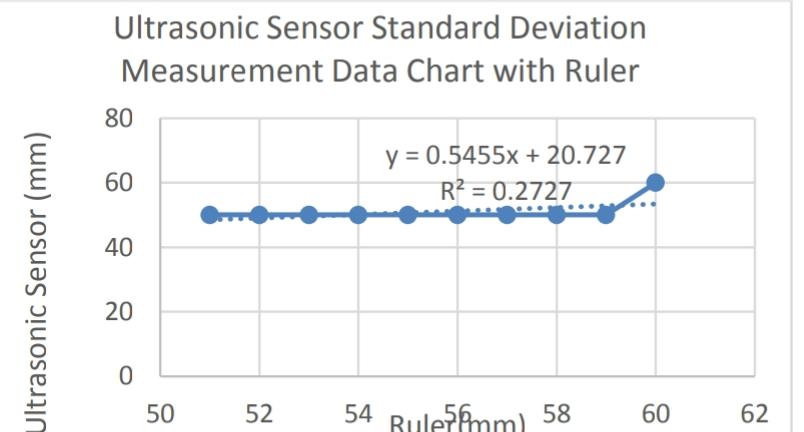
Delay(10000); // Check water level every 10 seconds

}

```

**Output:**





**Conclusion**:

In conclusion, the provided Arduino code demonstrates a simple implementation of flood monitoring and early warning using the ESP32 in IoT. The code utilizes a BME280 sensor to measure temperature, humidity, and pressure, and it sends this data to a ThingSpeak channel for monitoring.Sensor Network: Incorporating a network of sensors strategically placed in flood-prone areas to gather comprehensive data.Data Analysis: Implementing advanced algorithms for real-time data analysis to detect anomalies or potential flood conditions.Communication Redundancy: Ensuring reliable communication methods, such as GSM, Wi-Fi, or LoRa, and incorporating redundancy for data transmission.Integration with External Systems: Connecting the ESP32 with external systems or services for more sophisticated early warning capabilities.Power Management: Implementing efficient power management strategies to ensure the longevity of the deployed devices, especially in remote locations.User Interface: Creating a user interface, possibly a web or mobile application, for users and authorities to access real-time data and receive alerts.Remember, the provided code serves as a starting point, and the development of a comprehensive flood monitoring system requires careful consideration of environmental conditions, system requirements, and the integration of various technologies for an effective early warning mechanism.