Problem Statement:

In hydroelectric power plant dams, there is a constant danger of flooding by extreme weather events like heavy rainfall. Flooding can be predicted by monitoring the water level in the dam. If the water level in a dam has risen above a certain limit the authorities may be alerted so that steps may be taken to reduce the volume of water in the dam thereby preventing flooding. The problem this project aims to solve is developing a method which monitors water level in a dam and alert the authorities when the water level has risen above a certain threshold. The project should alert the authorities locally as well as remotely and wirelessly. This alert should be flashy and loud so as to attract attention of the on-duty personnel.

Creation of hydroelectric power plants and related infrastructures like dams are important to fulfill the electric power requirement of a developing country like India. But construction of dams on rivers also contribute to huge ecological losses and can result in the surrounding areas becoming more prone to flooding. Moreover, due to climate change, the world is bound to face extreme weather changes and the rivers are predicted to behave more erratically in the future. It is predicted that the hydroelectric power infrastructure around the world is not ready for such sudden change and this makes the areas around dams and near river banks more prone to the danger of catastrophic flooding. Therefore, it is important that technology is utilized to develop a smart solution which can help us predict flooding so that steps may be taken to prevent flooding.

In theory, dams are designed such that they may control floods by storing the water. However, in practice, dams are only helpful for flood containment purpose if they are operated with this purpose in mind. This is not how most dams operate as they fill up completely as soon as water is available and since then they do not have space to store more water, they release all the water downstream as soon as there is more inflow. This exacerbates floods and results in much heavier losses to life and property than it would have been were the dam never constructed. Clearly, the current solutions are insufficient and efforts must be focused to tackle the problem from a different angle.

Scope of solution:

The solution has a few boundaries and objectives-

- Designing and building a prototype for monitoring water level in a dam and warning authorities—locally and remotely—if the water level rises above a certain threshold
- To develop a working small scale prototype which demonstrates the principle, and not necessarily a full scale, production ready product
- To keep costs low
- To keep parts and components—both hardware and software—to a minimum to avoid complexity and expense

The solution aims to create a system which detects the level of water in a container. It displays the level of water on a display in realtime and also sends it to some IoT analytics platform service which displays a visualization of the data so that the water level can also be monitored remotely. If the water level in the container rises above a set threshold in the container, then the system must warn the users both locally and remotely as this situation is similar to water level rising in a dam before flood. To warn the users locally, the device will flash the word ‘Warning!’

in bold letters on the display while displaying the water level, also in bold font. Also, the system will sound an alarm so as to attract attention to the emergency. To warn people remotely, a solution must be sought from the IoT data analytics platform. If the water level then falls back to normal level, the system must cease the warning and alarm and go back to the state it was in before the water level reached dangerous levels, i.e., silently displaying the water level on the display.

This project is limited by the availability and affordability of parts. As mentioned above, the objectives of this project is to demonstrate the working principle of a water level monitoring system. The reason for not making a full scale production ready product is that the components for such a system are not readily available or affordable. For, example, the HC-SR04 sensor used here can only measure lengths upto 4 metres, which is nowhere near enough to monitor water level in a real dam.

Required components to build the solution:

We list the hardware and software components to build the project -

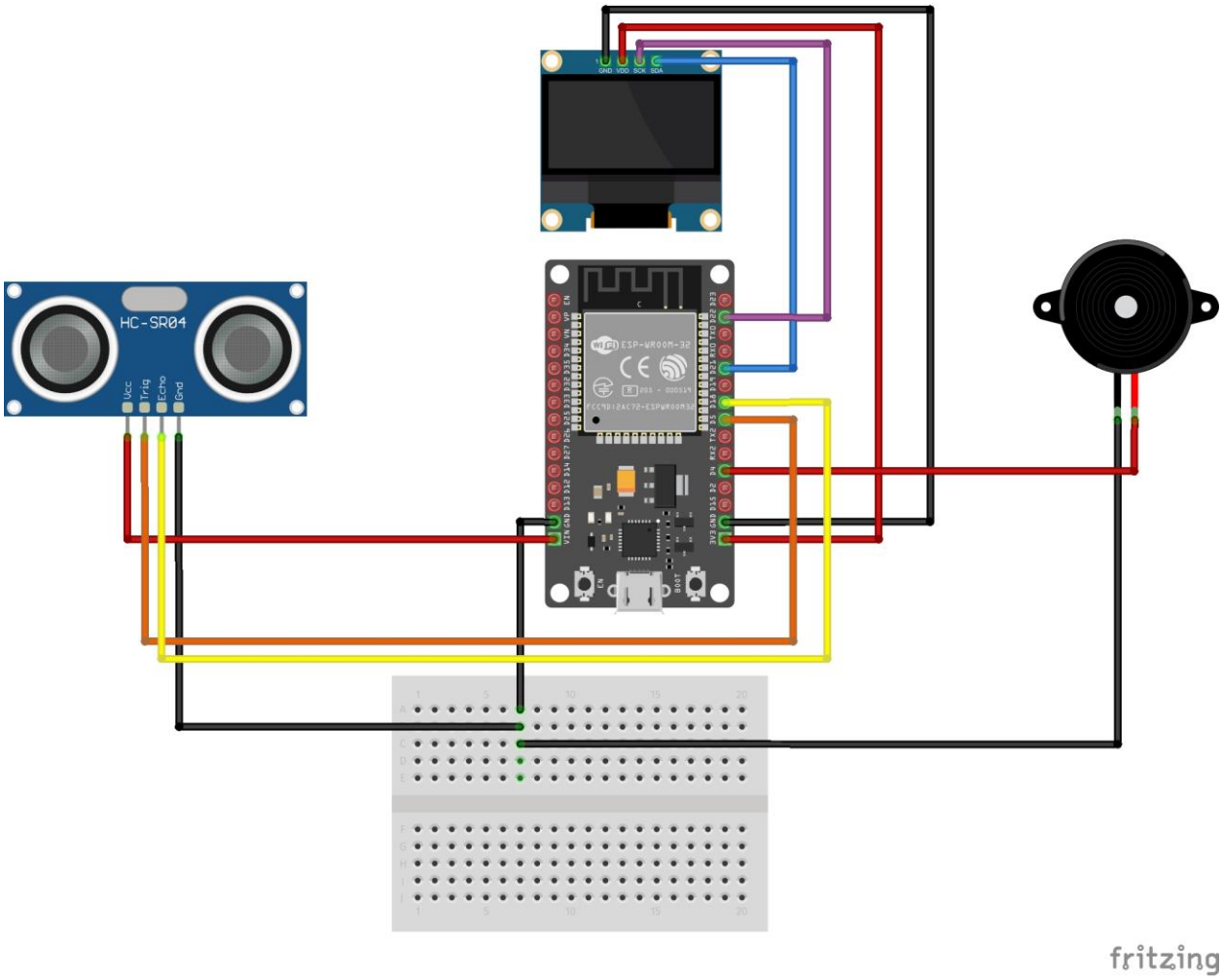
- Hardware components -
  - ESP 32 microcontroller with WiFi and Bluetooth. It connects to WiFi and sends the sensor readings to the ThingSpeak IoT data analytics platform.
  - HC-SR04 ultrasonic sensor
  - SSD1306 128x64 monochromatic OLED display
  - Buzzer
  - Jumper Cables and micro USB cable
  - A plastic container to hold water
  - Some fittings to fix the components on the container
- Software components -
  - Embedded C
  - Arduino IDE 2.0 for writing the Embedded C code and uploading the code to the microcontroller
  - ThingSpeak IoT data analytics platform to record realtime data and display the visualization over the internet
- Libraries Used -

  For the Embedded C code, the following libraries have been used -

  - Arduino.h - contains many commonly used definitions
  - WiFi.h – This library is used to connect the ESP 32 to WiFi
  - Wire.h – This library enables I2C communication between the microcontroller and sensors
  - Adafruit_GFX.h – This is a low level library which is not used directly. Rather, it is used by the Adafruit_SSD1306.h library to interface with the SSD1306 OLED display
  - Adafruit_SSD1306.h - Contains all the functions and definitions to interface with the SSD1306 OLED display conveniently

- NewPing.h – Contains functions to interface with the HC-SR04 ultrasonic sensor conveniently. It makes taking readings from the ultrasonic sensor easy and hassle free

Simulated Circuit:



In the above circuit diagram, there are three devices connected to the ESP 32 microcontroller. One is a sensor—The HC-SR04—while the other two, the SSD1306 OLED display and the piezo buzzer are output devices. The shown connections are as follows:

The HC-SR04 has four pins VCC, Trig, Echo and GND which are connected to VIN, D5, D18 and GND on the ESP 32 microcontroller. The GND pin is connected indirectly, through a breadboard, to the ESP 32.

The SSD1306 also has four pins GND, VDD, SCK and SDA which are connected to GND, 3V3, D22 and D21 pins on the ESP 32. The SSD1306 OLED display communicates with the ESP 32 via the I2C protocol.

The Buzzer only has two pins—positive and negative—which are connected to D4 and GND on the ESP 32 microcontroller respectively. The negative pin is connected indirectly, through a breadboard, to the ESP 32.

As seen in the schematic, the GND pin of the ESP 32 microcontroller is connected to a breadboard and the HC-SR04 and buzzer connect to the ESP 32 GND through it.

Code for the solution:

- **Programming language-** Embedded C
- **External libraries-** ThingSpeak.h, Adafruit_GFX.h, Adafruit_SSD1306.h, NewPing.h
- **Build instructions –** Install the Arduino IDE 2.0, then install the above mentioned external libraries. Enable support for ESP 32 microcontroller by going to File>Preferences and paste this URL in the Additional Boards Manager's URL field -

  https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

  Then, copy paste the following code in a new file, connect your ESP 32 to the PC/Laptop and click on the dropdown menu and click 'select another board and port' .Then, select 'DOIT ESP32 DEVKIT V1' from the boards menu and the visible COM port from the menu on right. Finally, click upload and wait for the file to upload. Before uploading, though, the WiFi credentials must be changed to connect to the local network and a ThingSpeak write API key and channel number must be changed so that it can connect to a ThingSpeak channel.

```
/*

floodDetecter.ino

Author - Hashir Sibtain

This program reads water level in a container, displays the reading on an
OLED display and sends it to a ThingSpeak channel.

If the water level rises above a certain thershold a warning is displayed
on the OLED screen in bold font and the buzzer is

sounded to attract attention.

*/



#include <Arduino.h> // Arduino.h contains many commonly used definitions

#include "WiFi.h" // WiFi.h is included to connect to WiFi

#include <ThingSpeak.h> // ThingSpeak library is the official library to
connect to a ThingSpeak channel
```

```
// Below libraries are required for interfacing SSD1306 OLED display with the microcontroller

#include <Wire.h> // The Wire library is used for I2C communication with the OLED display

#include <Adafruit_GFX.h> // Adafruit_GFX library is a low level library which is used as a base by the Adafruit_SSD1306 library for the OLED display

#include <Adafruit_SSD1306.h> //Contains all the functions and definitions to interface with the SSD1306 OLED display conveniently



// NewPing is a HC-SR04 Ultrasonic Sensor library which makes taking readings easy and convenient as it handles all the

// calculations and formulae.

#include <NewPing.h>



// We define the trigger and echo pins for the ultrasonic sensor and also define a maximum distance which it can measure

#define TRIG_PIN 5

#define ECHO_PIN 18

#define MAX_DIST 2000



// In the below two lines, we define the width and height of the OLED display in pixels

#define SCREEN_WIDTH 128

#define SCREEN_HEIGHT 64



// We define the WiFi network name, password and timeout in millisecond

#define WIFI_NETWORK "Your WiFi Network" // For privacy and security, this particular WiFi network is only a placeholder and should be replaced with the user's WiFi newtork name

#define WIFI_PASSWORD "Your WiFi Password" // For privacy and security, this particular WiFi network password is only a placeholder and should be replaced with the user's WiFi network password
```

```
#define WIFI_TIMEOUT 20000 // The ESP 32 will try to connect to the WiFi network for 20 seconds before moving on



const char* writeAPIKey = "0123456789ABCDEF"; // The ThingSpeak Write API Key is required to write to a ThingSpeak Channel. For privacy and security, this is a dummy key and should be replaced with an actual Write API Key

unsigned long channelNumber = 1234567; // The ThingSpeak Channel ID identifies a unique ThingSpeak Channel so the data reaches the correct channel. For privacy and security, this is a dummy channel ID and should be replaced with a real ThingSpeak channel ID



unsigned long lastTime = 0; // long variable to keep time

int upDelay = 5000; // Every 5 seconds, the ESP32 will try uploading to the ThingSpeak channel

NewPing levelSensor(TRIG_PIN, ECHO_PIN, MAX_DIST); // We create a NewPing variable which represents the HC-SR04 Ultrasonic Sensor



#define OLED_RESET -1 // Reset pin related to OLED Display

#define SCREEN_ADDRESS 0x3C // OLED Display's I2C Slave address

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET); // We create a new display variable which represents the SSD1306 OLED Display



const int buzzer = 4; // This constant integer denotes the pin the buzzer is connected to



WiFiClient client; // We create a WiFi Client for use with the ThingSpeak library



void connectToWiFi()

{
```

```
  /* This is a user defined function which connects the ESP32 to WiFi and
initializes the ThingSpeak library.

    It returns nothing and takes no arguments

  */

  Serial.print("Connecting to WiFi"); // Print a line denoting the start
of an attempt to connect to WiFi

  WiFi.mode(WIFI_STA); // Set ESP32 WiFi mode to station mode to connect
to a WiFi hotspot

  ThingSpeak.begin(client); // Initialize the ThingSpeak library and
network setting with the WiFi client

  WiFi.begin(WIFI_NETWORK, WIFI_PASSWORD); // Connect the ESP32 to a WiFi
network using the provided credentials



  unsigned long startAttemptTime = millis(); // This variable keeps track
of time while the ESP attempts connecting to the WiFi network

  while(WiFi.status() != WL_CONNECTED && millis() - startAttemptTime <
WIFI_TIMEOUT) // This while loop prints a '.' while the ESP is attempting
to connect to WiFi.

  {
// It ends when the connection is successful or when WIFI_TIMEOUT
milliseconds (in this case 20 seconds) have passed

    Serial.print("."); // prints the '.;

    delay(100); // delay for 100 ms

  }



  if(WiFi.status() != WL_CONNECTED) // Print 'Failed if WiFi not
connected

  {

    Serial.println("Failed.");

  }

  else // else print connected successfully and print local IP

  {

    Serial.println("Connected successfully");
```

```
    Serial.println(WiFi.localIP());
  }
}


void warn(int waterLevel)
{
  /* This function is user-defined and deals with the display of the high water-level warning on the
  SSD1306 OLED display. It does various things like set cursor to a correct position, set an appropriate
  font size and an appropriate colour and finally it displays the warning which consists of the word 'WARNING'
  written in large font with white background and black text on it and the water level on the next line
  in bold font */
  display.setCursor(11, 16); // Set the cursor position to 11 pixels from the right (x co-ordinate) and 16 pixels from the top (y co-ordinate)
  display.clearDisplay(); // Clear the display to get a fresh display and space to write on
  display.setTextSize(2); // Set text size to 2
  display.setTextColor(BLACK, WHITE); // Set text colour to black and background colour to white where 'BLACK' and 'WHITE' are macros defined in Adafruit_SSD1306.h file
  display.println(" WARNING "); // Display the text 'WARNING' as specified above
  display.setTextColor(WHITE); // Set text colour to white and display colour to black
  display.setCursor(50, 40); // Set the cursor position to 50 pixels from the right (x co-ordinate) and 40 pixels from the top (y co-ordinate)
  display.print(waterLevel); // Display the water level as specified by the above two lines
  display.print("cm"); // Display the unit 'cm' with the water level
  display.display(); // Actually draw on the OLED display
```

```
}



void buzz()

{

  /* This function is concerned with sounding the buzzer and returns
nothing. It sounds the alarm

     according to a particular pattern */

  tone(buzzer, 1000); // the tone() function plays a particular takes two
parameters, the buzzer pin and the frequency. It plays a tone of that
frequency on the buzzer. Here, the frequency is 1 KHz

  delay(500); // The program waits for 0.5 seconds. So, for these 0.5
seconds the buzzer keeps playing the 1 KHz tone

  noTone(buzzer); // the noTone() function takes only one argument, the
buzzer pin and stops any tone that is playing on that buzzer. So, the
1KHz tone stops

}

void setup() {

  // In this section we setup Serial communication, WiFi, and OLED
display

  Serial.begin(115200); // we begin serial communication at 115200 bps

  connectToWiFi(); // we call the connectToWiFi() function



  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) // we try
initializing the SSD1306 OLED display module. if an error occurs we print
'SSD1306 allocation failed' and loop indefinitely

  {

    Serial.println(F("SSD1306 allocation failed"));

    while(true);

  }

  // The following four lines are related to OLED display module

  display.clearDisplay(); // clear the display

  display.setTextSize(1); // set font size to 1
```

```
  display.setTextColor(WHITE); // set font colour to White

  display.setCursor(0,28); // set cursor position to co ordinate (0, 28)



}



void loop() {

  // This is the main loop of the program where the code repeatedly runs. It takes readings, sends them to ThingSpeak, display them on OLED screen and sound the buzzer if necessary

  delay(50); // small delay for callibration


  int lev = 22 - levelSensor.ping_cm(); // lev keeps track of the water level. The levelSensor.ping_cm() function is from the NewPing library and returns reading

                                        // from the HC-SR04 sensor in cm. Since, the initial distance of the sensor from the bottom of the container is 22cm, we

                                        // subtract the reading from 22 to get the height of the water.



  if((millis() - lastTime) > upDelay)   // If, upDelay amount of time or more has passed, then send the data to the ThingSpeak channel.

  {

    lastTime = millis(); // update the time in tracking variable

    Serial.println("Uploading to ThingSpeak"); // print "Uploading to ThingSpeak" on serial monitor

    int x = ThingSpeak.writeField(channelNumber, 1, lev, writeAPIKey); // Actually attempt uploading the data to the ThingSpeak channel using given credentials. It returns an int we save in the variable x

    if (x == 200) // If x is 200, upload was successfull so print a success message on serial monitor

      Serial.println("Successfully uploaded to ThingSpeak!");

    else Serial.println(x); // else print out the error code
```

```
  }

  if(lev > 10) // If lev is greater than 10 cm then display the warning
on OLED display by calling the warn() function and sound the buzzer by
calling the buzz() function

  {

    warn(lev);

    buzz();

  }

  else // Else, print the water level normally on OLED display

  {

    display.clearDisplay(); // clear display

    display.setTextSize(1); // set font size to 1

    display.setTextColor(WHITE); // set font colour to White

    display.setCursor(5,28); // set cursor position to co-ordinate (5,28)

    display.clearDisplay(); // clear display

    display.print("Water Level: "); // print "Water Level: " at the
configured co-ordinate and font size and style on the OLED display

    display.print(lev); // print the actual water level on the OLED
display

    display.print("cm"); // print "cm" on the OLED display

    display.display(); // finally, display the text on the OLED display
so that everything shows up

  }




}
```