L&T EduTech

Industrial Applications of Microcontrollers - A Practice Based Approach

PROJECT

DAM WATER LEVEL ALERT SYSTEM

VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Name: NANDANA.M.K

Reg.no. 21BEC2027

DAM WATER LEVEL ALERT SYSTEM

1. Problem statement

The primary objective is to develop a Dam Water Level Alert System that provides real-time monitoring and timely alerts to relevant authorities and communities in the event of abnormal water level fluctuations. This system aims to enhance dam safety, minimize the risk of dam failures, and facilitate proactive responses to changing water conditions.

2. Scope of the solution

- Develop a functional prototype of a dam water level monitoring system, including water level sensors, data acquisition, data transmission, and a central monitoring dashboard.
- Test and validate the prototype under both simulated and real-world conditions to ensure data accuracy, reliability, and performance.
- Provide comprehensive documentation, including system architecture and acceptance criteria for the successful implementation of the prototype.

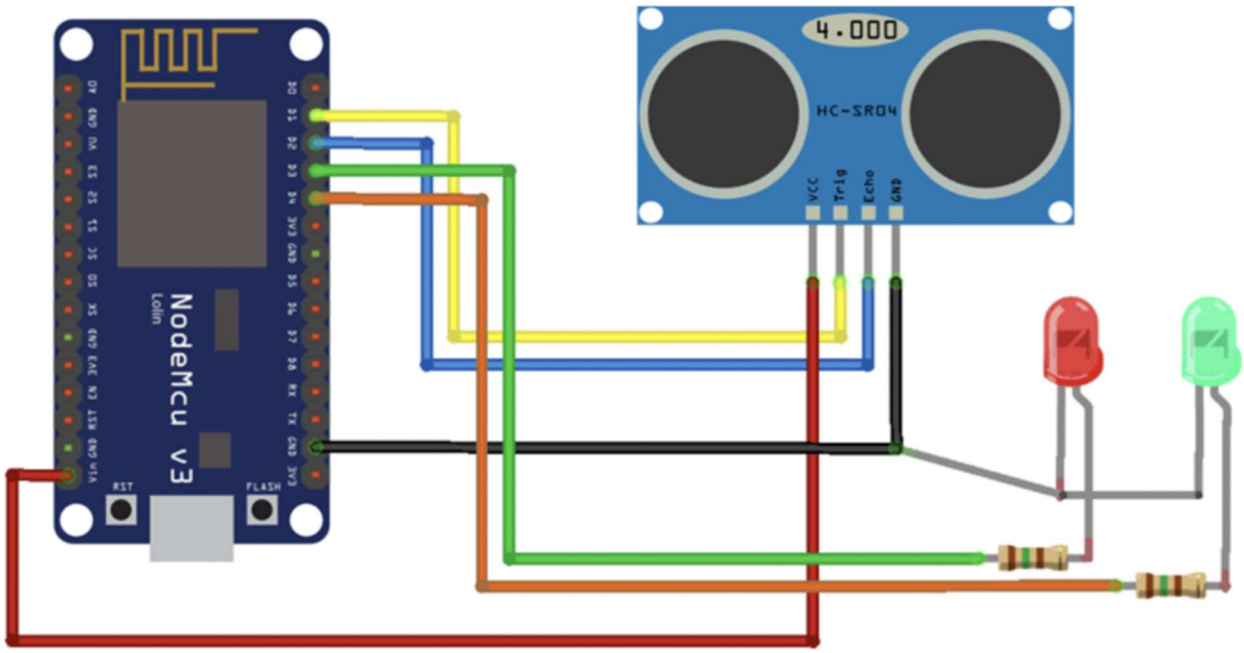
3. Required components to develop solutions

- Arduino IDE
- ThingSpeak

Hardware

- ESP8266 NodeMCU
- Ultrasonic sensor
- Power supply
- LEDs (Red and Green)
- Breadboard
- Jumper wires

4. Simulated Circuit



5. Video of the demo

https://github.com/nandana-mk/dam-monitoring/blob/main/VIT_21BEC2027/Dam%20Water%20Level%20Project%20Video.mp4

6. Gerber file

https://github.com/nandana-mk/dam-monitoring/blob/main/VIT_21BEC2027/DAM_gerber_file.zip

7. Code for the solution

Begin the code by incorporating essential library files, such as ESP8266WiFi.h for ESP8266 boards, among others. In this context, the ThingSpeak.h library is employed for the ThingSpeak platform, and it can be integrated into the Arduino IDE through the following procedure:

```
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>
```

Specify the pins utilized for both the ultrasonic sensors and LEDs.

```
const int trigPin1 = D1;
const int echoPin1 = D2;
#define redled D3
#define greenled D4
```

Next, we establish the network credentials, encompassing the SSID and password necessary for the NodeMCU's internet connection. Following that, it's imperative to modify these variables with the appropriate ThingSpeak account credentials.

```
unsigned long ch_no = 102xxxx;
const char * write_api = "SXSSSSS";
char auth[] = "fuEERRXXXXXXXXXXXXXX";
char ssid[] = "admin";
char pass[] = "";
```

Then, the variables for timing purposes are defined and to connect NodeMCU to the internet, call **WiFi.begin**

```
unsigned long startMillis;
unsigned long currentMillis;
const unsigned long period = 10000;

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED)
   {
    delay(500);
    Serial.print(".");
   }
  Serial.println("WiFi connected");
  Serial.println(WiFi.localIP());
```

Next, establish a connection to the ThingSpeak platform utilizing:

```
 ThingSpeak.begin(client);
```

To compute the distance, an initial pulse is transmitted to the sensor via the trig pin of the Ultrasonic sensor. Following the specifications outlined in the HC-SR04 datasheet, a 2-microsecond pulse is initiated. Subsequently, the sensor's output pulse is captured from the echo pin, enabling the calculation of the distance in centimeters.

```
digitalWrite(trigPin1, LOW);
delayMicroseconds(2);
digitalWrite(trigPin1, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin1, LOW);
duration1 = pulseIn(echoPin1, HIGH);
distance1 = duration1 * 0.034 / 2;
```

Next, an if-else statement is crafted to control the LED indicators for both regular and rise in water level beyond the threshold conditions. In the model used, 4 centimeters as the reference point.

```
if (distance1 <= 4)
  {
   digitalWrite(D3, HIGH);
   digitalWrite(D4, LOW);
  }
 else
  {
   digitalWrite(D4, HIGH);
   digitalWrite(D3, LOW);
  }
```

Ultimately, the dam water level data is transmitted to the ThingSpeak channel at regular 15-second intervals as selected.

```
if (currentMillis - startMillis >= period)
  {
   ThingSpeak.setField(1, distance1);
   ThingSpeak.writeFields(ch_no, write_api);
   startMillis = currentMillis;
  }
```

Complete Code:

```
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>
const int trigPin1 = D1;
const int echoPin1 = D2;
#define redled D3
#define greenled D4
unsigned long ch_no = 1026389;
const char * write_api =
"XK88XXXXXXX";
char auth[] =
"fu0o5JaLXXXXXXXXXXXXXXXX";
char ssid[] = "admin";
char pass[] = "";
unsigned long startMillis;
unsigned long currentMillis;
const unsigned long period =
10000;
WiFiClient  client;
long duration1;
int distance1;
void setup()
{

  digitalWrite(trigPin1, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin1, LOW);
  duration1 = pulseIn(echoPin1,
HIGH);
  distance1 = duration1 * 0.034 /
2;
  Serial.println(distance1);
  if (distance1 <= 4)
  {
    digitalWrite(D3, HIGH);
    digitalWrite(D4, LOW);
  }
  else
{
 pinMode(trigPin1, OUTPUT);
  pinMode(echoPin1, INPUT);
  pinMode(redled, OUTPUT);
  pinMode(greenled, OUTPUT);
  digitalWrite(redled, LOW);
  digitalWrite(greenled, LOW);
  Serial.begin(9600);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");
  Serial.println(WiFi.localIP());
  ThingSpeak.begin(client);
  startMillis = millis();  //initial
start time
}
void loop()
{
  digitalWrite(trigPin1, LOW);
  delayMicroseconds(2);
    digitalWrite(D4, HIGH);
    digitalWrite(D3, LOW);
  }
  currentMillis = millis();
  if (currentMillis - startMillis >=
period)
  {
    ThingSpeak.setField(1, distance1);
    ThingSpeak.writeFields(ch_no,
write_api);
    startMillis = currentMillis;
  }
```

8. Working

The setup includes the ThingSpeak platform and the Arduino IDE in addition to necessary hardware like the ESP8266 NodeMCU, ultrasonic sensors, and LEDs to bring this creative project to life. The sensor carefully measures the distance between itself and the water level of the dam when the device is turned on. The sensor diligently monitors and records these oscillations in real-time as the water level progressively rises. When the water level rises above a set point, our system kicks off and starts sending out notifications right away, enabling us to react quickly and proactively to any changes in the water level. When it comes to guaranteeing dam safety and community well-being, this degree of real-time monitoring and reaction capabilities is revolutionary.

Moreover, the versatility of this technology makes it a good fit for widespread use. Because of its scalability, it may be widely used across a range of dam architecture, improving dam safety and facilitating effective water level fluctuation management. A proactive approach to addressing concerns connected to dam water levels is made possible by the real-time monitoring and prompt alerting mechanism, which lowers the risk of failures and minimizes potential damages to downstream communities and infrastructure.

9. Empathy Inference:

An important invention that aims to protect not only infrastructure but also the lives and livelihoods of communities at risk is the Dam Water Level Alert System. It demonstrates an awareness of the possible destruction brought about by dam failures and the need of reacting to changes in water level by offering proactive alarms and real-time monitoring. This approach acknowledges the necessity for prompt information and action to lessen the impact of such disasters and demonstrates a profound empathy for the safety and well-being of those who live downstream of these dams. The project team's dedication to improving catastrophe resilience and the welfare of vulnerable populations is evident in the way they developed this solution.

This Dam Water Level Alert System can be implemented on a large scale, revolutionizing dam safety and disaster prevention. By providing real-time monitoring and timely alerts, it's not just a technological advancement – it's a safeguard for communities and infrastructure downstream.