Water Quality Monitoring System

A Final Project
By

Dharani Kandhalam
2431389 - dkandhal@cougarnet.uh.edu
Kamal Phaneendra Avula
2447498 - kavula@cougarnet.uh.edu
Hysritha Sai Kasarapu
2442341 - hkasarap@cougarnet.uh.edu
Avadhoot Umesh Likhite
2455727 - aulikhit@cougarnet.uh.edu
Chanakya Teladala
2447520 - cteladal@cougarnet.uh.edu

ECE 6372 Advanced Hardware Design
Fall 2025
University of Houston

ABSTRACT

In an era increasingly focused on public health, environmental sustainability, and smart monitoring technologies, **Water Quality Monitoring using STM32 with Wi-Fi Cloud Connectivity** presents an intelligent and reliable solution for continuous assessment of water safety. The proposed system is designed to monitor critical water quality parameters in real time, enabling early detection of contamination and ensuring safe water conditions. The system integrates a high- performance **STM32 microcontroller** with multiple water quality sensors to measure key parameters such as **pH, turbidity, temperature, and total dissolved solids (TDS)**.

The STM32 microcontroller performs accurate data acquisition, calibration, filtering, and threshold evaluation of sensor readings. For wireless communication, the system utilizes the **ISM43362-M3G-L44 Wi-Fi module**, which is onboard the STM32 Discovery kit. This module enables secure Wi-Fi connectivity using **WPA2 authentication** and allows the system to transmit processed sensor data to the **ThingSpeak IoT cloud platform** through dedicated firmware functions such as wifi_thingspeak_client() and ThingSpeak_Send().

The **ThingSpeak platform** is used exclusively for real-time visualization and trend analysis of the collected sensor data. It provides a reliable framework for periodic data logging and **cloud-based monitoring of water quality parameters.**

Overall, this project delivers a **cost-effective, scalable, and efficient embedded IoT solution** for real-time water quality monitoring. It is well suited for applications in **drinking water safety, environmental monitoring, aquaculture, and industrial discharge surveillance**, while complying with modern public health and environmental standards.

1. INTRODUCTION

1.1. Brief Overview of the Project:

The water quality monitoring with the ESP32 project aims to be a modern, dependable, and real-time method of keeping track of the minimum parameters of water quality. Clean water is of the utmost importance for the health of the community, the well-being of the environment, as well as the operations of the industry. Regular water tests are, in most cases, hard labor and take a lot of time, and the results are not instantly available. This undertaking gets past these inadequacies by the use of embedded systems and IoT technology to provide non-stop and self-active surveillance. The data provided by the sensors measuring the pH, turbidity, TDS, and temperature of the water are grabbed and computed by the STM32 microcontroller of the system. Afterward, the data to the cloud is sent wirelessly through the ESP32 Wi-Fi module. Thus, remote monitoring and alarm become possible.

1.2. Key Objectives:

- ***Accurate Water Parameter Measurement:*** Precise and accurate measurement of pH, turbidity, TDS, and temperature through the use of properly calibrated analog and digital sensors.
- ***Real-Time Monitoring and Alerts:*** Data is acquired in real time and sent to the cloud together with an automatic notification if an abnormal situation is detected.
- ***Hygiene and Safety Assurance:*** The foremost point in the detection of contamination is to be able to use water without worry.
- ***Scalability and Flexibility:*** Additional sensors can be added to the system, and it can be integrated with several IoT platforms without limits, thus, it is flexible.

2. COMPONENTS USED

2.1. Hardware Components:

The hardware design of the Water Quality Monitoring is a finely tuned amalgamation of multiple sensors and a high-performance STM32 microcontroller, which is capable of recording the most vital water parameters, easily and accurately, in real-time. The hardware components of the project are:

2.1.1. STMMicrocontroller Unit (B-L4S5I-IOT1A – STM32L4S5):

- The B-L4S5I-IOT1A (STM32L4S5) microcontroller acts as the central processing unit of the system, handling sensor interfacing, data acquisition, calibration, filtering, and threshold-based evaluation of water quality parameters.
- The microcontroller supports multiple communication protocols such as I2C, UART, and SPI**,** enabling flexible integration with external sensors, ADC modules, and the onboard Wi-Fi interface for real-time cloud communication.

2.1.2. Hilitand PH Module Probe Kit, PH0-14 Value Detect Sensor Module + PH Electrode Probe BNC:

- It is used to measure the acidity or alkalinity of water.
- Generates an analog voltage, which is directly proportional to a pH value (0–14 scale).
- Apart from the analog output, it is connected to an external ADC for a detailed measurement.

2.1.3. HiLetgo TDS Sensor Water Conductivity Sensor Liquid Detection Sensor Water Quality Monitoring Sensor Module TDS Monitor Kit Analog Output

- The device which measures the TDS (Total Dissolved Solids) in ppm (parts per million) is called a TDS sensor.
- Investigates the water purification level and is capable of spotting the dissolved contaminants in the water.
- Produces an analog output that is in need of a high-resolution ADC converter for the association.

2.1.4. Gravity: Analog Turbidity Sensor for Arduino / ESP32 / STM32 / Raspberry Pi (ADC Required) | Water Quality Monitor | <500ms Response Time & 0-4000NTU Range

- The sensor that detects the suspended particles in water by using the light scattering method.
- It is used to detect the sediment, algae, or chemical pollution.
- Provides an analog voltage directly proportional to the turbidity level.

2.1.5. BOJACK DS18B20 Temperature Sensor Module Kit with Waterproof Stainless Steel Probe

- Provides the measurement of water temperature is an extremely important factor for the correct pH compensation.
- Communicates via a single-wire digital interface, thereby minimizing noise and wiring complexity.
- Totally waterproof and therefore perfect for immersion applications.

2.1.6. HiLetgo ADS1115 16 Bit 16 Byte 4 Channel I2C IIC Analog-to-Digital ADC PGA Converter with Programmable Gain Amplifier High Precision ADC Converter Development Board

- Provides extremely fine measurement granularity, capturing of the full scale. This is crucial for applications requiring high precision, such as sensitive sensor readings or detailed data logging
- The board can monitor four independent single-ended inputs (A0, A1, A2, A3) or be configured for two differential pairs. Differential mode is excellent for minimizing common-mode noise and ground loops, essential in noisy environments.

- The built-in PGA allows you to select gains from. This feature lets you measure very small signals (down to a few millivolts) with high resolution while still using the full range of the ADC, effectively boosting signal quality without external circuitry.

2.2. Software Components

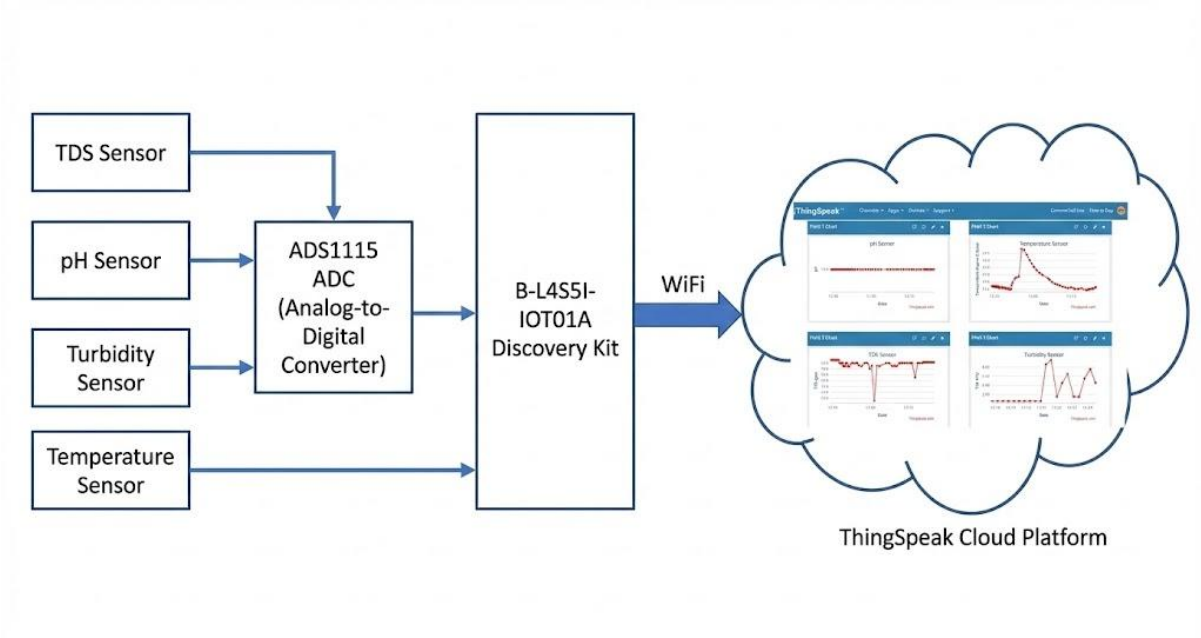
2.2.1. STM32CubeIDE:

- STM32CubeIDE is used as the primary development and debugging environment for the B-L4S5I-IOT01A (STM32L4S5) microcontroller.
- It is used for configuring peripherals such as I²C (ADS1115 ADC), GPIO (DS18B20 and sensor control), UART/SPI (Wi-Fi communication), and system clock settings.
- The IDE supports firmware development, compilation, flashing, and debugging of the embedded application.
- It enables real-time debugging and serial monitoring during sensor data acquisition and processing.

2.2.2. On-Board Wi-Fi Module Firmware and IoT Cloud Platform:

- The B-L4S5I-IOT01A board includes an on-board Wi-Fi module (Inventek ISM43362), which is controlled by the STM32L4S5 microcontroller using standard communication interfaces.
- Processed sensor data from the STM32 is transmitted to the Wi-Fi module and uploaded to the ThingSpeak IoT cloud platform.
- The cloud platform provides real-time visualization, data logging, and monitoring of water quality parameters such as pH, TDS, turbidity, and temperature.

3. SYSTEM WORKING



This project implements a water quality monitoring system using the B-L4S5I-IOT01A (STM32L4S5 IoT Discovery Kit) as the main controller. The system continuously monitors key water quality parameters—pH, Total Dissolved Solids (TDS), Turbidity, and Temperature—and uploads the measured data to the ThingSpeak cloud platform over Wi-Fi for remote monitoring.

3.1. Controller and Communication Architecture

The B-L4S5I-IOT01A board is based on the STM32L4S5 microcontroller, which provides:

- Low-power operation
- Built-in Wi-Fi connectivity
- Multiple digital communication interfaces (I2C, GPIO, UART)

To interface multiple analog sensors accurately, an external ADS1115 16-bit ADC module is used.

3.2. Sensor Interfacing Strategy

3.2.1. ADS1115 External ADC (I2C Interface)

- The ADS1115 is connected to the STM32 controller using the I2C protocol.
- I2C allows multiple sensors to share the same communication bus while providing high-resolution analog measurements.
- The ADS1115 converts analog sensor outputs into digital values, which are then processed by the microcontroller.

Analog sensors connected to ADS1115:

Sensor	ADS1115 Channel	Purpose
pH Sensor	A0	Measures acidity/alkalinity of water
TDS Sensor	A1	Measures dissolved solids concentration
Turbidity Sensor	A2	Measures water cloudiness

Each sensor produces an **analog voltage proportional to the physical parameter**, which is digitized by the ADS1115 and transmitted to the controller via I2C.

3.2.2. pH Sensor

- Purpose: Measures the acidity or alkalinity of water.
- Connection:
  - Analog output → ADS1115 Channel A0
- Working:

  The pH sensor generates a voltage proportional to the hydrogen ion concentration of the water. This voltage is digitized by the ADS1115 and converted into a pH value by the microcontroller.

pH Quality Classification Used:

pH Range	Water Quality
6.5 – 8.5	Good
< 6.5 or > 8.5	Bad

3.2.3. TDS (Total Dissolved Solids) Sensor

- Purpose: Measures the concentration of dissolved salts and minerals in water.
- Connection:
  - Analog output → ADS1115 Channel A1
- Working:

  The TDS sensor outputs a voltage proportional to water conductivity. The STM32 converts this voltage into TDS values (ppm) using a calibration equation.

TDS Quality Classification Used:

TDS (ppm)	Water Quality
0 – 300 ppm	Good
300 – 600 ppm	Medium
> 600 ppm	Bad

3.2.4. Turbidity Sensor

- Purpose: Measures the cloudiness or suspended particles in water.
- Connection:
  - Analog output → ADS1115 Channel A2
- Working:

  The turbidity sensor operates based on light scattering. As turbidity increases, the sensor voltage changes. The system converts this voltage into NTU (Nephelometric Turbidity Units).

Turbidity Quality Classification Used:

Turbidity (NTU)	Water Quality
0 – 5 NTU	Good
5 – 50 NTU	Medium
> 50 NTU	Bad

3.2.5. Temperature Sensor (DS18B20)

- Purpose: Measures water temperature.
- Connection:
  - Digital 1-Wire → PB2 (Arduino D8)

- Working:

  The DS18B20 is a digital temperature sensor that communicates via a single-wire protocol. It provides accurate temperature measurements without requiring ADC resources.

Temperature Range Used (Informational):

Temperature (°C)	**Interpretation**
20 – 30 °C	Normal
< 20 or > 35 °C	Abnormal

3.3. Data Processing and Monitoring

- The STM32 controller periodically reads all sensor values.
- The raw digital values are converted into engineering units:
  - pH
  - TDS (ppm)
  - Turbidity (NTU)
  - Temperature (°C)
- Each parameter is compared against predefined thresholds to classify water quality as Good or Bad.
- The processed values are displayed on the serial terminal (PuTTY) for local monitoring.

3.4. Cloud Connectivity and Visualization

- Using the **on-board Wi-Fi module**, the STM32 connects to the internet.
- Sensor data is transmitted to **ThingSpeak** using HTTP requests.
- Each parameter is mapped to a dedicated ThingSpeak field, enabling:
  - Real-time visualization
  - Historical data analysis
  - Remote access from anywhere

    pH : VALUE – Good/Bad

    TDS : VALUE – Good/Bad

    TUB : VALUE – Good/Bad

    TEMP: VALUE °C

4. Control Flow of the Water Quality Monitoring Sytem

1) Program entry and system setup:

main()

Overall role: initialize MCU + peripherals → connect Wi-Fi → run continuous sensing + cloud upload loop.

Control flow:

1. **HAL + clock**
   - HAL_Init()
   - SystemClock_Config() (sets PLL system clock and ADC kernel clock selection).

2. **UART terminal (debug prints) — only if TERMINAL_USE**
   - Configures hDiscoUart and calls BSP_COM_Init(...)
   - Initializes board temperature sensor BSP_TSENSOR_Init() (note: not used later in the sensing loop).

3. **Sensor interface init**
   - MX_I2C1_Init() → initializes I2C1 used to talk to ADS1115.
   - HAL_I2C_IsDeviceReady(&hi2c1, ADS1115_ADDR, ...) → prints ADS1115 presence check.
   - DWT_Delay_Init() → enables cycle-counter based microsecond delay used by 1-Wire timing.
   - DS18B20_GPIO_Init() → configures PB2 as open-drain with pull-up for DS18B20.
   - BSP_LED_Init(LED2)
   - BSP_PB_Init(BUTTON_USER, BUTTON_MODE_EXTI) (button interrupt is wired, but not used in the main loop logic).

4. **Application start**
   - Calls wifi_thingspeak_client(); and never returns.

2) Wi-Fi connection flow

wifi_thingspeak_client()

Role: **connect Wi-Fi once → then loop forever reading sensors and pushing to ThingSpeak.**

1. Calls wifi_connect()
   - If fail → prints error and returns -1 (but main() doesn't handle it; function returns to main() end).
2. If connected → enters while(1) infinite loop.

wifi_connect()

Role: **initialize Wi-Fi module + connect using hardcoded credentials + fetch IP.**

Flow:

1. wifi_start()
2. Loads SSID/PASSWORD/SECURITY from:

- MY_WIFI_SSID, MY_WIFI_PASSWORD, MY_WIFI_SECURITY (hardcoded; flash config not used).

3. Maps numeric security (0–3) → WIFI_Ecn_t
4. Calls WIFI_Connect(...)
5. Calls WIFI_GetIP_Address(IP_Addr, ...) and prints IP.

wifi_start()

Role: **WIFI_Init + read module MAC.**

3) Main sensing + reporting + upload loop

Inside while(1) in wifi_thingspeak_client() (runs every ~20 seconds):

Step A — Read pH

- float ph = PH_ReadPH();

PH_ReadPH() flow:

1. v_mV = ADS1115_Read_mV(0) (ADS channel A0)
2. sanity check low voltage
3. converts 0..Vref to 0..14 pH using linear map ph = 14*(v_mV/(PH_VREF*1000)) and clamps to [0,14].

pH Sensor Data Acquisition and Conversion (PH_ReadPH):

```
static float PH_ReadPH(void)
{

  float v_mV = ADS1115_Read_mV(0);  // A0

 // Basic sanity check
 if (v_mV < 1.0f) {
            int v10 = (int)(v_mV * 10.0f + 0.5f);  // e.g. 3293.4 -> 32934

               printf("pH: ADC too low (%d.%01d mV), returning 0\r\n",
                    v10 / 10,      // integer part
                    v10 % 10);     // 1 decimal digit

               return 0.0f;
 }

 float fullscale_mV = PH_VREF * 1000.0f;

 // Simple linear map: 0..Vref  ->  0..14
 float ph = 14.0f * (v_mV / fullscale_mV);
```

```
 if (ph < 0.0f)  ph = 0.0f;
 if (ph > 14.0f) ph = 14.0f;

 int ph_x100  = (int)(ph * 100.0f + 0.5f);   // pH * 100

//  printf("Computed pH = %d.%02d from %ld mV\r\n",
//        ph_x100 / 100,       // integer part of pH
//        ph_x100 % 100,       // two decimal places
//        (long)v_mV);

 return ph;
}
```

Step B — Read TDS

- int tds_ppm = TDS_ReadPPM();

main

TDS_ReadPPM() flow:

v_mV = ADS1115_Read_mV(1) (ADS channel A1)

if v_mV < 50 → return 0

uses DFRobot polynomial to compute raw TDS

multiplies by calibration constant TDS_CALIB_K

returns integer ppm.

TDS Sensor Data Acquisition and PPM Computation (TDS_ReadPPM):

```
/* === TDS Meter helper (using same ADC on A0) === */
#define TDS_CALIB_K  0.08f   // <-- we will tune this
//#define TDS_CALIB_K  (100.0f / 800.0f)   // 0.125f



static int TDS_ReadPPM(void)
{
 float v_mV = ADS1115_Read_mV(1);   // A1

 if (v_mV < 50.0f)
 {
  return 0;
 }

 float v = v_mV / 1000.0f;   // volts
```

```
 /* Raw DFRobot TDS formula */
 float raw_tds = (133.42f * v * v * v
               - 255.86f * v * v
               + 857.39f * v) * 0.5f;

 if (raw_tds < 0.0f)
   raw_tds = 0.0f;

 /* Apply calibration factor */
 float tds = raw_tds * TDS_CALIB_K;

 int tds_int = (int)(tds + 0.5f);

//  printf("TDS raw = %ld ppm, calibrated = %d ppm from %ld mV\r\n",
//         (long)raw_tds,
//         tds_int,
//         (long)v_mV);

 return tds_int;
}
```

Step C — Read temperature (DS18B20 on PB2)

- float tempC = DS18B20_ReadTempC();

main

DS18B20_ReadTempC() flow:

OneWire_Reset() presence detect

Send SKIP ROM (0xCC) + CONVERT T (0x44)

HAL_Delay(750) wait conversion

reset again

Send SKIP ROM + READ SCRATCHPAD (0xBE)

read 9 bytes via OneWire_ReadByte()

compute rawTemp → tempC = rawTemp / 16.0 (0.0625°C resolution).

```
/* ===== Temperature sensor ===== */
static void DWT_Delay_Init(void)
{
 /* Enable trace and debug block DEMCR (TRCENA) */
 CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
```

```
 /* Reset cycle counter */
 DWT->CYCCNT = 0;

 /* Enable cycle counter */
 DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;
}

static void delay_us(uint32_t us)
{
 uint32_t cycles = (SystemCoreClock / 1000000U) * us;
 uint32_t start  = DWT->CYCCNT;

 while ((DWT->CYCCNT - start) < cycles)
 {
   __NOP();
 }
}
```

Step D — Read turbidity

- float turb_ntu = TURB_ReadNTU();

main

TURB_ReadNTU() flow:

v = TURB_ReadVoltage_V()

reads ADS channel A2 via ADS1115_Read_mV(2)

multiplies by 2 because of divider, returns volts.

main

converts voltage → NTU using TURB_VoltageToNTU(v)

prints turb debug line and returns NTU.

```
/*======== Turbidity sensor =====*/
static float TURB_ReadVoltage_V(void)
{
   // ADS A2 reads the divided voltage (Vin/2)
   float v_adc_mV = ADS1115_Read_mV(2);

   // Convert back to sensor output voltage (because divider = 0.5)
   float v_sensor_mV = v_adc_mV * 2.0f;
   return v_sensor_mV / 1000.0f;   // return volts
}
```

```
static float TURB_VoltageToNTU(float v_sensor_V)
{
   float ntu;

   if (v_sensor_V < 2.5f)
      ntu = 3000.0f;
   else
      ntu = -1120.4f*v_sensor_V*v_sensor_V + 5742.3f*v_sensor_V - 4352.9f;

   if (ntu < 0) ntu = 0;
   return ntu;
}



static float TURB_ReadNTU(void)
{
   float v = TURB_ReadVoltage_V();          // volts
   float ntu = TURB_VoltageToNTU(v);

   int32_t v_mV = (int32_t)(v * 1000.0f + 0.5f);
   int32_t ntu_x10 = (int32_t)(ntu * 10.0f + 0.5f);

   printf("[TURB] V=%ld mV, NTU=%ld.%01ld\r\n",
         (long)v_mV,
         (long)(ntu_x10/10), (long)(ntu_x10%10));

   return ntu;
}
```

Step E — Categorize quality + print summary

Uses these label helpers:

- PH_Label(ph)
- TDS_Label(tds_ppm)
- TURB_Label(turb_ntu)

  Each returns "Good" / "Medium" / "Bad" based on thresholds.

Step F — Upload to ThingSpeak

- ThingSpeak_Send(ph, tempC, (float)tds_ppm, turb_ntu);

ThingSpeak_Send() flow:

1. DNS: WIFI_GetHostAddress("api.thingspeak.com", ...)
2. TCP connect: WIFI_OpenClientConnection(socket0, WIFI_TCP_PROTOCOL, ...)
3. Builds HTTP GET request:
    - field1 = pH, field2 = temp, field3 = TDS, field4 = turbidity
4. Sends via WIFI_SendData(...)
5. Reads small response via WIFI_ReceiveData(...) (ignored)
6. WIFI_CloseClientConnection(0)

Step G — Delay

- HAL_Delay(20000); (20 seconds)

4) ADS1115 acquisition control flow (shared by pH/TDS/Turbidity)

All analog measurements use this pattern:

main

ADS1115_Read_mV(channel):

1. takes 8 samples in a loop
2. each sample calls ADS1115_ReadSingleEnded(channel)
3. skips samples that fail (returns INT16_MIN)
4. averages valid raw counts
5. converts to volts via ADS1115_RawToVoltage(), then to mV
6. prints [ADS] ch=X raw=... mv=... (ok=.../8) and returns mV.

ADS1115_ReadSingleEnded(channel):

- writes config register (start single-shot, AINx-GND, PGA ±4.096V, 128 SPS)
- waits 10 ms
- reads conversion register.

5) **Interrupt / button flow (implemented but not part of sensing logic)**

- HAL_GPIO_EXTI_Callback() routes:
    - user button pin → Button_ISR() → increments button_flag
    - Wi-Fi SPI IRQ pin → SPI_WIFI_ISR()

main

Also provided:

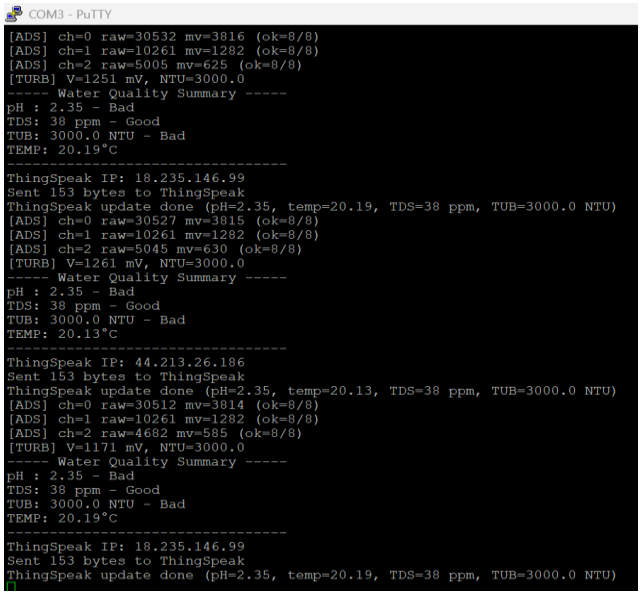
- Button_Reset(), Button_WaitForPush() (polls button_flag with timeout) — not called by main app loop.

5. Results

The results of the water quality monitoring system were verified using two observation methods: local serial output through PuTTY and cloud-based visualization using the ThingSpeak dashboard. Among these, the ThingSpeak dashboard serves as the primary platform for monitoring and result analysis.

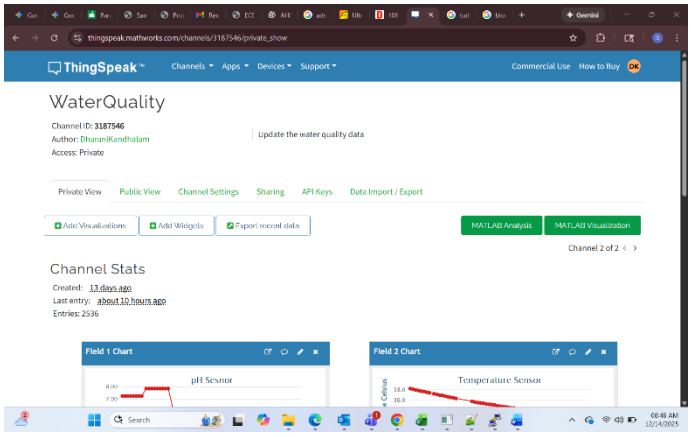
- **Results Observed on PuTTY (Serial Terminal) :** During system operation, all sensor readings were printed on the PuTTY serial terminal for local monitoring and debugging purposes. The terminal displayed real-time values of pH, temperature, TDS, and turbidity after sensor data acquisition and processing by the STM32 microcontroller.

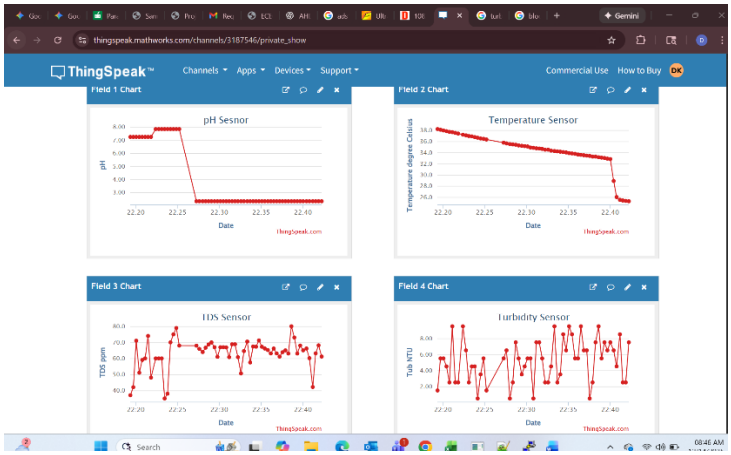
  The PuTTY output:



- **Results Observed on the ThingSpeak Dashboard (Primary Results):** The primary results of the project were obtained from the ThingSpeak IoT cloud dashboard. Sensor data was uploaded periodically at fixed intervals and mapped to individual fields on the dashboard.

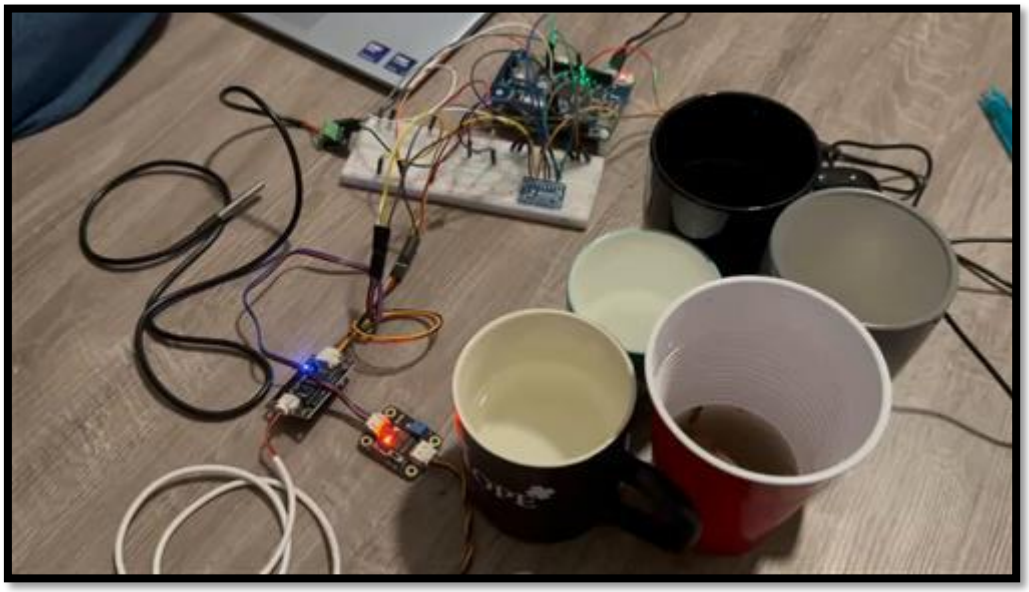
  The ThingSpeak Dashboard:





Link To Dashboard: https://thingspeak.mathworks.com/channels/3187546

Project Photo:



6. Applications of the Project

- The water quality monitoring system with ESP32 has enabled the usage of the system in a broad scope of domains of nature, industry, and public health:
- Environmental Monitoring: Monitoring rivers, lakes, and reservoirs through non-stop data collection to be able to tell pollution and sediment changes apart.
- Drinking Water Management: Assuring the conformity of the requirements in the water quality of the municipal water supply systems.
Aquaculture: The condition of water such as pH, temperature, and TDS is kept at the best level for fish growth and health.
- Industrial Discharge Monitoring: Letting go of the pollutants and wastewater, which might cause contamination in the natural water bodies, only after they are detected.
- Educational and Research Use: A feasible and affordable platform for academic projects and environmentals.

7. FUTURE SCOPE

- **Advanced Hardware:** Design a compact, custom PCB that brings all components together on a single board , improving system reliability while making the device easier to deploy in real-world field conditions.
- **Power Autonomy:** Use the STM32 ’s lowpower operating modes to enable efficient battery usage, allowing the system to run for long periods without maintenance in remote environmental monitoring locations.
- **Expanded Sensing:** Add Dissolved Oxygen (DO) and ORP sensors to provide a more complete picture of water quality, which is especially valuable for aquaculture and ecosystem health monitoring.
- **Automated Calibration:** Implement automatic , infield calibration routines and store calibration data in flash memory to ensure consistent and accurate sensor readings over time.
- **Smart Alerts:** Go beyond basic threshold alerts by analyzing historical sensor data to detect anomalies early and predict potential pollution events before they become critical.
- **Scalability:** Enable LoRaWAN or Sub-GHz communication so the system can be scaled into a large network of sensor nodes supported by a centralized gateway architecture.
- **Two-Way Control:** Allow commands to be sent from the cloud back to the STM32, enabling remote control of actuators such as pumps and valves for active system management.

8. CONCLUSION

This project successfully implements a water quality monitoring system using the STM32L4S5 IoT Discovery Kit and multiple sensors to measure pH, TDS, turbidity, and temperature. High-resolution data acquisition is achieved using the ADS1115 ADC along with the DS18B20 temperature sensor.

The STM32 microcontroller reliably processes sensor data and uploads it to the ThingSpeak cloud platform at fixed intervals using the on-board ISM43362 Wi-Fi module. The ThingSpeak dashboard provides real-time visualization and historical tracking of water quality parameters.

Overall, the system offers a cost-effective and scalable embedded IoT solution for periodic water quality data collection and cloud-based visualization, making it suitable for educational, environmental, and laboratory applications.

Thanks,
Cougar Circuitry