

Group name & signal names

Participants names  
Nazmul Hasan  
Kasun Akalanka  
Shihab Ahmed

Technical document

April 2023

Engineering Program

Table of contens

[1 Hardware 3](#_Toc130893569)

[2 Software 4](#_Toc130893570)

[2.1 Code 4](#_Toc130893571)

[2.1 Verification 4](#_Toc130893572)

[Attachments 5](#_Toc130893573)

[Whole code 5](#_Toc130893574)

# Hardware

Board, keypad, lcd, explained

# Software The Arduino sketch uses sensors to measure wind direction and speed, and displays the data on an LCD screen. It also connects to an MQTT broker to send the data to the cloud. The code includes libraries for Ethernet, PubSubClient, Keypad, LiquidCrystal, and ArduinoJson. It uses interrupts and a timer to measure wind speed and frequency. The code reads the user input from a keypad and displays the selected data on the LCD screen. Depending on the selected display mode, the code sends MQTT messages with the wind direction, wind speed, or both. The code also attempts to reconnect to the MQTT server if the connection is lost.

## Code

// Include required libraries

#include <Ethernet.h>

#include <PubSubClient.h>

#include <TimerOne.h>

#include <Keypad.h>

#include <LiquidCrystal.h>

#include <ArduinoJson.h>  
  
This code includes several libraries to enable certain functionalities in the Arduino project:

* Ethernet.h: Allows the Arduino to communicate with the Ethernet network.
* PubSubClient.h: Enables the Arduino to subscribe and publish to MQTT brokers.
* TimerOne.h: Provides a library for configuring and running timers in the Arduino.
* Keypad.h: Allows the Arduino to interface with a keypad matrix.
* LiquidCrystal.h: Provides a library to control LCD displays.
* ArduinoJson.h: Allows the Arduino to parse and generate JSON data.

// Initialize the LCD display pins

LiquidCrystal lcd(37, 36, 35, 34, 33, 32);

// Define the number of rows and columns of the keypad matrix

const byte ROWS = 4;

const byte COLS = 4;

// Initialize variables for tracking button presses

volatile boolean buttonPressedA = false;

volatile boolean buttonPressedB = false;

volatile boolean buttonPressedC = false;

// Initialize variables for tracking wind direction and button states

String windDirection;

int buttonState = 0;

int previousButtonState = 0;

// Initialize variables for displaying the data on the LCD screen

bool displayDirection;

bool displaySpeed;

bool displayBoth;

int degree\_Value;

// Initialize variables for measuring wind frequency and speed

volatile unsigned long pulseCount;

volatile unsigned long windFrequency;

float windSpeed;

unsigned long i\_time;

// Define the input pin for the wind direction sensor

const int signalInputPin = 2;

// Define the layout of the keypad matrix

char keys[ROWS][COLS] = {

  {'1','2','3','A'},

  {'4','5','6','B'},

  {'7','8','9','C'},

  {'\*','0','#','D'}

};

//Declare the row and column pins for the keypad

byte rowPins[ROWS] = { 42, 43, 44, 45 };

byte colPins[COLS] = { 46, 47, 48, 49 };

//Create a keypad object using the pins and keys arrays

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

//Create an EthernetClient object to establish an Ethernet connection

EthernetClient ethClient;

//Set the MAC address for the Ethernet shield

static uint8\_t mymac[6] = { 0x44,0x76,0x58,0x10,0x00,0x62 };

//Set the port number and IP address of the MQTT broker

unsigned int Port = 1883;

byte server[] = { 10,6,0,21 };

//Set the device ID, client ID and device secret for the MQTT connection

char\* deviceId     = "esdevice23";

char\* clientId     = "escg223";

char\* deviceSecret = "tamk";

This part of the code initializes various variables and objects, including:

* An LCD object with pins 37, 36, 35, 34, 33, 32
* The number of rows and columns for a 4x4 keypad matrix
* Boolean variables for tracking button presses
* String and integer variables for tracking wind direction and button states
* Boolean variables for displaying wind direction, speed, or both on the LCD screen
* An integer variable for storing the degree value of wind direction
* Variables for measuring wind frequency, speed, and time
* The input pin (2) for the wind direction sensor
* The layout and pins for the keypad matrix
* A Keypad object using the keys and pins arrays
* An EthernetClient object for establishing an Ethernet connection
* The MAC address for the Ethernet shield
* The port number and IP address of the MQTT broker
* The device ID, client ID, and device secret for the MQTT connection.

//Declare the callback function for incoming MQTT messages

void callback(char\* topic, byte\* payload, unsigned int length);

//Create a PubSubClient object using the server, port, callback function and Ethernet client

PubSubClient client(server, Port, callback, ethClient);

 //Define the input and output topics for the MQTT messages

 #define inTopic    "ICT4\_in\_2020"

 #define outTopic   "ICT4\_out\_2020"

In this part of the code, the MQTT callback function is declared. This function is used to handle incoming messages from the MQTT broker.

* A PubSubClient object is created using the server, port, callback function, and Ethernet client.
* Two topics are defined for incoming and outgoing MQTT messages. These topics are used to communicate with other devices and send/receive data.
* The "inTopic" is set to "ICT4\_in\_2020" and the "outTopic" is set to "ICT4\_out\_2020".
* The "inTopic" is the topic that the device subscribes to in order to receive messages, and the "outTopic" is the topic that the device publishes to in order to send messages to other devices.

//Setup function to initialize variables and objects

void setup() {

    Serial.begin(9600);

    Serial.println("Start 19.3.2021");

    delay(500);

    //Fetch the IP address using DHCP

    fetch\_IP();

    //Connect to the MQTT server

    Connect\_MQTT\_server();

 //Initialize the LCD display

 lcd.begin(16, 2);

  //Set the analog input pin for wind direction sensor

  pinMode(A2, INPUT);

  pulseCount = 0;

  windFrequency = 0;

  windSpeed = 0.0;

  i\_time = 0;

  //Attach an interrupt to the wind speed sensor input pin

  attachInterrupt(digitalPinToInterrupt(signalInputPin), pulseCountFunction, RISING);

  //Initialize a timer to update wind speed and frequency

  Timer1.initialize(500000);

  Timer1.attachInterrupt(timerInterruptFunction);

}

This code block is the setup() function, which is called once at the beginning of the program. Its main purpose is to initialize variables and objects used in the program.

* Serial.begin(9600) initializes the serial communication with the computer at a baud rate of 9600.
* fetch\_IP() retrieves the IP address of the device using DHCP (Dynamic Host Configuration Protocol).
* Connect\_MQTT\_server() establishes a connection to the MQTT server using the Ethernet client and PubSubClient object created earlier.
* lcd.begin(16, 2) initializes the LCD display with 16 columns and 2 rows.
* pinMode(A2, INPUT) sets analog input pin A2 as an input.
* pulseCount, windFrequency, windSpeed, and i\_time are all initialized to zero.
* attachInterrupt(digitalPinToInterrupt(signalInputPin), pulseCountFunction, RISING) attaches an interrupt to the wind speed sensor input pin to detect a rising edge (when the sensor detects a full revolution).
* Timer1.initialize(500000) sets up a timer that triggers an interrupt every 500 milliseconds (0.5 seconds).
* Timer1.attachInterrupt(timerInterruptFunction) attaches an interrupt function to the timer, which updates the wind speed and frequency variables.

// This function is called repeatedly in a loop

void loop(){

The loop() function is a required function in Arduino programming that is called repeatedly in a loop. It contains the main logic of the program, including reading sensors, updating variables, and sending or receiving data.

// Read the key pressed on the keypad

  char key = keypad.getKey();

  // If key 1 is pressed, set the boolean variables accordingly

  if(key == '1'){

    buttonPressedA = true;

    buttonPressedB = false;

    buttonPressedC = false;

  // If key 2 is pressed, set the boolean variables accordingly

  } else if(key == '2'){

    buttonPressedA = false;

    buttonPressedB = true;

    buttonPressedC = false;

  // If key 3 is pressed, set the boolean variables accordingly

  } else if(key == '3'){

    buttonPressedA = false;

    buttonPressedB = false;

    buttonPressedC = true;

  // If any other key is pressed, set all the boolean variables to false

  } else {

    buttonPressedA = false;

    buttonPressedB = false;

    buttonPressedC = false;

  }

  // Depending on which button was pressed, set the display boolean variables accordingly

  if(buttonPressedA == true){

    displayDirection = true;

    displaySpeed = false;

    displayBoth = false;

  } else if(buttonPressedB == true){

    displayDirection = false;

    displaySpeed = true;

    displayBoth = false;

  } else if(buttonPressedC == true){

    displayDirection = false;

    displaySpeed = false;

    displayBoth = true;

  }

This part of the code reads the key pressed on the keypad and sets boolean variables accordingly. It determines which button was pressed and sets display boolean variables accordingly.

* The first block of code checks which key on the keypad was pressed and sets boolean variables based on the key pressed.
* The second block of code checks which button was pressed and sets display boolean variables accordingly.

// Print the option for the user to select which information to display on the LCD

  lcd.setCursor(20, 1);

  lcd.print("1-Dir 2-Spd 3-Both");

  // Read the analog value from A2 pin, and convert it to a voltage value

  float analog\_Value = analogRead(A2);

  float voltage\_Value = 5 \* analog\_Value / 1023;

  // Calculate the degree value based on the voltage value

  if (voltage\_Value < 1.2) {

    degree\_Value = 0;

  } else{

    degree\_Value = ((voltage\_Value - 1.2) / 3.8) \* 360;

  }

This part of the code does the following:

* Sets the cursor position on the LCD to print the options for the user to select what information to display.
* Reads the analog value from pin A2 and converts it to a voltage value.
* Calculates the degree value based on the voltage value. This value is used to determine the direction of the wind. If the voltage value is less than 1.2, the degree value is set to 0. Otherwise, the degree value is calculated based on a formula that converts the voltage value to degrees.

// This part checks which button is pressed and sets displayDirection, displaySpeed, or displayBoth to true or false accordingly

  // It then displays the corresponding information on the LCD screen

  if (displayDirection == true) {

                          lcd.setCursor(0, 0);

                          if (voltage\_Value <= 1.19) {

                            windDirection = "Wait....";

                          } else if (degree\_Value >= 337.5 || (degree\_Value > -1 && degree\_Value <= 22.4)) {

                            windDirection = "North";

                          } else if (degree\_Value > 22.4 && degree\_Value <= 67.5) {

                            windDirection = "North East";

                          } else if (degree\_Value > 67.5 && degree\_Value <= 112.5) {

                            windDirection = "East";

                          } else if (degree\_Value > 112.5 && degree\_Value <= 157.5) {

                            windDirection = "South East";

                          } else if (degree\_Value > 157.5 && degree\_Value <= 202.5) {

                            windDirection = "South";

                          } else if (degree\_Value > 202.5 && degree\_Value <= 247.5) {

                            windDirection = "South West";

                          } else if (degree\_Value > 247.5 && degree\_Value <= 292.5) {

                            windDirection = "West";

                          } else {

                            windDirection = "North West";

                          }

                          lcd.clear();

                          lcd.setCursor(0, 0);

                          lcd.print("Dir : ");

                          lcd.print(windDirection);

                          lcd.setCursor(0, 1);

                          lcd.print(degree\_Value);

                          lcd.print(" degrees");

                          lcd.setCursor(20, 1);

                          lcd.print("1-Dir  2-Spd  3-Both");

  } else if (displaySpeed == true){

                          lcd.clear();

                          lcd.setCursor(0, 0);

                          lcd.print("Wind Speed: ");

                          lcd.setCursor(0, 1);

                          lcd.print("Freq: ");

                          lcd.setCursor(7, 1);

                          lcd.print(windFrequency);

                          lcd.setCursor(13, 1);

                          lcd.print("Hz");

                          lcd.setCursor(13, 0);

                          lcd.print(windSpeed);

                          lcd.setCursor(20, 1);

                          lcd.print("1-Dir  2-Spd  3-Both");

  } else if (displayBoth == true){

                          lcd.clear();

                          lcd.setCursor(0, 0);

                          lcd.print("Dir : ");

                          lcd.print(degree\_Value);

                          lcd.setCursor(0, 1);

                          lcd.print("Wind Speed: ");

                          lcd.setCursor(13, 1);

                          lcd.print(windSpeed);

                          lcd.setCursor(20, 1);

                          lcd.print("1-Dir  2-Spd  3-Both");

  }

This part of the code displays information on the LCD screen based on which button is pressed. The code checks if the displayDirection, displaySpeed, or displayBoth variables are true and displays the corresponding information on the LCD screen. Here's a breakdown of what the code does:

* If displayDirection is true, the code reads the analog value from A2 pin and converts it to a voltage value. Based on the voltage value, it calculates the degree value, which represents the wind direction. It then displays the wind direction and degree value on the LCD screen.
* If displaySpeed is true, the code clears the LCD screen and displays the wind speed and frequency on the screen.
* If displayBoth is true, the code displays both the wind direction and wind speed on the LCD screen.

 // If the client is connected, the wind direction and wind speed values are converted to strings and then passed to the send\_MQTT\_message function to be sent to the MQTT broker. If the client is not connected, it tries to reconnect to the broker.

  if (client.connected()){

    const char\* windDirections = windDirection.c\_str();

    send\_MQTT\_message(degree\_Value,windSpeed);

  } else {

    Serial.println("No, re-connecting" );

    client.connect(clientId, deviceId, deviceSecret);

  }

  delay(1000);

}

This is the end of void loop().  
  
This part of the code checks if the MQTT client is connected to the broker. If it is connected, it converts the wind direction and wind speed values to strings and sends them to the MQTT broker using the send\_MQTT\_message function. If the client is not connected, it tries to reconnect to the broker. It then adds a 1 second delay before starting the loop again.

// The fetch\_IP() function is called to get the IP address of the Ethernet controller and it prints the IP address on the Serial Monitor.

void fetch\_IP(void){

  byte rev=1;

  rev=Ethernet.begin(mymac);

  Serial.print( F("\nW5100 Revision ") );

  if (rev == 0){

    Serial.println( F( "Failed to access Ethernet controller" ) );

  }

  Serial.println( F( "Setting up DHCP" ));

  Serial.print("Connected with IP: ");

  Serial.println(Ethernet.localIP());

}

This function uses the Ethernet library to begin communication with the Ethernet controller and fetches the IP address of the device using Dynamic Host Configuration Protocol (DHCP). Once the IP address is obtained, it is printed on the Serial Monitor for the user to see.

// This function is used to send MQTT messages based on the selected display mode

void send\_MQTT\_message(int dir, float spd){

        // Set the delay between messages to 1 second

        int timeDelay = 1000;

        // Check if the display mode is set to wind direction

        if (displayDirection == true){

          // Create a JSON document with two fields: "S\_name1" and "S\_value1" representing the sensor name and the sensor value

          const size\_t bufferSize = JSON\_OBJECT\_SIZE(2) + 30;

        StaticJsonDocument<bufferSize> jsonDoc;

        jsonDoc["S\_name1"] = "ES Wind Direction";

        jsonDoc["S\_value1"] = dir;

        // Serialize the JSON document to a string and format the MQTT message

        String jsonStr;

        serializeJson(jsonDoc, jsonStr);

        char message[jsonStr.length() + 10];

        sprintf(message, "IOTJS=%s", jsonStr.c\_str());

          // Print the message to the Serial Monitor and publish the message to the MQTT broker

          Serial.println( message );

          client.publish(outTopic,message);

          delay(timeDelay);

        // Check if the display mode is set to wind speed

        } else if (displaySpeed == true){

          // Create a JSON document with two fields: "S\_name1" and "S\_value1" representing the sensor name and the sensor value

          const size\_t bufferSize = JSON\_OBJECT\_SIZE(2) + 30;

        StaticJsonDocument<bufferSize> jsonDoc;

        jsonDoc["S\_name1"] = "ES Wind Speed";

        jsonDoc["S\_value1"] = spd;

        // Serialize the JSON document to a string and format the MQTT message

        String jsonStr;

        serializeJson(jsonDoc, jsonStr);

        char message[jsonStr.length() + 10];

        sprintf(message, "IOTJS=%s", jsonStr.c\_str());

        // Print the message to the Serial Monitor and publish the message to the MQTT broker

          Serial.println( message );

          client.publish(outTopic,message);

          delay(timeDelay);

        // Check if the display mode is set to both wind directin & speed

        } else if (displayBoth == true) {

    // Create a JSON document with two fields: "S\_name1" and "S\_value1" representing the sensor name and the sensor value

    const size\_t bufferSize = JSON\_OBJECT\_SIZE(2) + 30;

    StaticJsonDocument<bufferSize> jsonDoc;

    jsonDoc["S\_name1"] = "ES Wind Direction";

    jsonDoc["S\_value1"] = dir;

    // Serialize the JSON document to a string and format the MQTT message

    String jsonStr;

    serializeJson(jsonDoc, jsonStr);

    char message[jsonStr.length() + 10];

    sprintf(message, "IOTJS=%s", jsonStr.c\_str());

    // Print the message to the Serial Monitor and publish the message to the MQTT broker

    Serial.println(message);

    client.publish(outTopic, message);

    jsonDoc.clear();

    delay(500);

        StaticJsonDocument<bufferSize> jsonDoc2;

        jsonDoc2["S\_name1"] = "ES Wind Speed";

        jsonDoc2["S\_value1"] = spd;

        // Serialize the JSON document to a string and format the MQTT message

        String jsonStr2;

        serializeJson(jsonDoc2, jsonStr2);

        char message2[jsonStr2.length() + 10];

        sprintf(message2, "IOTJS=%s", jsonStr2.c\_str());

          // Print the message to the Serial Monitor and publish the message to the MQTT broker

          Serial.println( message2 );

          client.publish(outTopic,message2);

          delay(timeDelay);

}

}

The send\_MQTT\_message function is responsible for sending MQTT messages based on the selected display mode. Here's what it does:

* The function takes in two parameters, dir and spd, representing the wind direction and wind speed values respectively.
* It first sets a time delay of 1 second between messages.
* It then checks the selected display mode using the if statements.
* If the display mode is set to wind direction, it creates a JSON document with two fields representing the sensor name and value, serializes the document to a string, and formats the MQTT message with the string. It then prints the message to the Serial Monitor and publishes it to the MQTT broker using the client.publish method.
* If the display mode is set to wind speed, it does the same as above but with the wind speed value instead.
* If the display mode is set to both wind direction and speed, it creates two separate JSON documents and publishes each one as an MQTT message. It first creates a JSON document with the wind direction values, formats it to an MQTT message, publishes it to the broker, clears the document, waits for 500ms, and then creates another JSON document with the wind speed values, formats it to an MQTT message, and publishes it to the broker.

// The following function attempts to connect to an MQTT server.

void Connect\_MQTT\_server(){

  // Display a message indicating that the connection to the MQTT server is being attempted.

  Serial.println(" Connecting to MQTT" );

  // Display the IP address of the MQTT server.

  Serial.print(server[0]); Serial.print(".");

  Serial.print(server[1]); Serial.print(".");

  Serial.print(server[2]); Serial.print(".");

  Serial.println(server[3]);

  // Check if the client is not already connected to the MQTT server.

  if (!client.connected()){

    // If the client is not already connected, attempt to connect using the specified client ID, device ID, and device secret.

    if (client.connect(clientId, deviceId, deviceSecret)){

      // Display a message indicating that the connection was successful.

      Serial.println(" Connected OK " );

      // Subscribe to the specified MQTT topic.

      client.subscribe(inTopic);

    }

    // If the connection attempt fails, display the current state of the client.

    else{

       Serial.println(client.state());

    }

  }

}

The Connect\_MQTT\_server function attempts to connect to an MQTT server by performing the following steps:

* Display a message indicating that the connection to the MQTT server is being attempted.
* Display the IP address of the MQTT server.
* Check if the client is not already connected to the MQTT server.
* If the client is not already connected, attempt to connect using the specified client ID, device ID, and device secret.
* If the connection is successful, display a message indicating that the connection was successful.
* Subscribe to the specified MQTT topic.
* If the connection attempt fails, display the current state of the client.

// The following function is called whenever a new MQTT message is received.

void callback(char\* topic, byte\* payload, unsigned int length){

  // Allocate memory for a string to hold the payload of the message.

  char\* receiv\_string;

  receiv\_string = (char\*) malloc(length + 1);

  // Copy the payload into the string.

  memcpy(receiv\_string, payload, length);

  receiv\_string[length] = '\0';

  // Print the payload to the Serial Monitor.

  Serial.println( receiv\_string );

  // Free the memory used by the string.

  free(receiv\_string);

}

This function is a callback function that is called whenever a new message is received over MQTT. It performs the following actions:

* It takes in three parameters: the topic of the message, a pointer to the payload of the message, and the length of the payload.
* It allocates memory for a new string to hold the payload of the message.
* It copies the payload into the string and adds a null character at the end to terminate the string.
* It prints the payload to the Serial Monitor.
* It frees the memory used by the string to avoid memory leaks.

// The following function is called whenever the pulse count changes.

void pulseCountFunction() {

  pulseCount++;

}

This function is an interrupt service routine that gets called whenever the signal on the wind speed sensor input pin changes from low to high. It increments the value of a global variable pulseCount by one. This variable is used to calculate the wind speed and frequency later in the code.

// The following function is called whenever the timer interrupt occurs.

void timerInterruptFunction() {

  i\_time++;

  // If 20 interrupts have occurred (500ms), calculate the wind speed and frequency.

  if (i\_time > 19) {

    windFrequency = pulseCount / 10;

    pulseCount = 0;

    windSpeed = -0.24 + windFrequency \* 0.699;

    i\_time = 0;

  }

}

This function updates the wind speed and frequency values based on the number of interrupts from the wind speed sensor that have occurred within a certain time period (500ms in this case). Here are the steps it takes:

* Increments the i\_time variable by 1.
* Checks if i\_time is greater than 19, which means that 20 interrupts have occurred (since the interrupt frequency is 2 interrupts per second).
* If 20 interrupts have occurred, it calculates the wind frequency by dividing the pulseCount (which represents the number of interrupts) by 10.
* It resets the pulseCount to 0.
* It calculates the wind speed using a linear equation, where the wind speed is equal to -0.24 + wind frequency \* 0.699.
* It resets i\_time to 0.

## 2.1 Verification

Picture of parsed message from the server

Attachments

## Whole code

//Final\_Assignment\_Rev\_2.0

// Include required libraries

#include <Ethernet.h>

#include <PubSubClient.h>

#include <TimerOne.h>

#include <Keypad.h>

#include <LiquidCrystal.h>

#include <ArduinoJson.h>

// Initialize the LCD display pins

LiquidCrystal lcd(37, 36, 35, 34, 33, 32);

// Define the number of rows and columns of the keypad matrix

const byte ROWS = 4;

const byte COLS = 4;

// Initialize variables for tracking button presses

volatile boolean buttonPressedA = false;

volatile boolean buttonPressedB = false;

volatile boolean buttonPressedC = false;

// Initialize variables for tracking wind direction and button states

String windDirection;

int buttonState = 0;

int previousButtonState = 0;

// Initialize variables for displaying the data on the LCD screen

bool displayDirection;

bool displaySpeed;

bool displayBoth;

int degree\_Value;

// Initialize variables for measuring wind frequency and speed

volatile unsigned long pulseCount;

volatile unsigned long windFrequency;

float windSpeed;

unsigned long i\_time;

// Define the input pin for the wind direction sensor

const int signalInputPin = 2;

// Define the layout of the keypad matrix

char keys[ROWS][COLS] = {

  {'1','2','3','A'},

  {'4','5','6','B'},

  {'7','8','9','C'},

  {'\*','0','#','D'}

};

//Declare the row and column pins for the keypad

byte rowPins[ROWS] = { 42, 43, 44, 45 };

byte colPins[COLS] = { 46, 47, 48, 49 };

//Create a keypad object using the pins and keys arrays

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

//Create an EthernetClient object to establish an Ethernet connection

EthernetClient ethClient;

//Set the MAC address for the Ethernet shield

static uint8\_t mymac[6] = { 0x44,0x76,0x58,0x10,0x00,0x62 };

//Set the port number and IP address of the MQTT broker

unsigned int Port = 1883;

byte server[] = { 10,6,0,21 };

//Set the device ID, client ID and device secret for the MQTT connection

char\* deviceId     = "esdevice23";

char\* clientId     = "escg223";

char\* deviceSecret = "tamk";

//Declare the callback function for incoming MQTT messages

void callback(char\* topic, byte\* payload, unsigned int length);

//Create a PubSubClient object using the server, port, callback function and Ethernet client

PubSubClient client(server, Port, callback, ethClient);

 //Define the input and output topics for the MQTT messages

 #define inTopic    "ICT4\_in\_2020"

 #define outTopic   "ICT4\_out\_2020"

//Setup function to initialize variables and objects

void setup() {

    Serial.begin(9600);

    Serial.println("Start 19.3.2021");

    delay(500);

    //Fetch the IP address using DHCP

    fetch\_IP();

    //Connect to the MQTT server

    Connect\_MQTT\_server();

 //Initialize the LCD display

 lcd.begin(16, 2);

  //Set the analog input pin for wind direction sensor

  pinMode(A2, INPUT);

  pulseCount = 0;

  windFrequency = 0;

  windSpeed = 0.0;

  i\_time = 0;

  //Attach an interrupt to the wind speed sensor input pin

  attachInterrupt(digitalPinToInterrupt(signalInputPin), pulseCountFunction, RISING);

  //Initialize a timer to update wind speed and frequency

  Timer1.initialize(500000);

  Timer1.attachInterrupt(timerInterruptFunction);

}

// This function is called repeatedly in a loop

void loop(){

  // Read the key pressed on the keypad

  char key = keypad.getKey();

  // If key 1 is pressed, set the boolean variables accordingly

  if(key == '1'){

    buttonPressedA = true;

    buttonPressedB = false;

    buttonPressedC = false;

  // If key 2 is pressed, set the boolean variables accordingly

  } else if(key == '2'){

    buttonPressedA = false;

    buttonPressedB = true;

    buttonPressedC = false;

  // If key 3 is pressed, set the boolean variables accordingly

  } else if(key == '3'){

    buttonPressedA = false;

    buttonPressedB = false;

    buttonPressedC = true;

  // If any other key is pressed, set all the boolean variables to false

  } else {

    buttonPressedA = false;

    buttonPressedB = false;

    buttonPressedC = false;

  }

  // Depending on which button was pressed, set the display boolean variables accordingly

  if(buttonPressedA == true){

    displayDirection = true;

    displaySpeed = false;

    displayBoth = false;

  } else if(buttonPressedB == true){

    displayDirection = false;

    displaySpeed = true;

    displayBoth = false;

  } else if(buttonPressedC == true){

    displayDirection = false;

    displaySpeed = false;

    displayBoth = true;

  }

  // Print the option for the user to select which information to display on the LCD

  lcd.setCursor(20, 1);

  lcd.print("1-Dir 2-Spd 3-Both");

  // Read the analog value from A2 pin, and convert it to a voltage value

  float analog\_Value = analogRead(A2);

  float voltage\_Value = 5 \* analog\_Value / 1023;

  // Calculate the degree value based on the voltage value

  if (voltage\_Value < 1.2) {

    degree\_Value = 0;

  } else{

    degree\_Value = ((voltage\_Value - 1.2) / 3.8) \* 360;

  }

  // This part checks which button is pressed and sets displayDirection, displaySpeed, or displayBoth to true or false accordingly

  // It then displays the corresponding information on the LCD screen

  if (displayDirection == true) {

                          lcd.setCursor(0, 0);

                          if (voltage\_Value <= 1.19) {

                            windDirection = "Wait....";

                          } else if (degree\_Value >= 337.5 || (degree\_Value > -1 && degree\_Value <= 22.4)) {

                            windDirection = "North";

                          } else if (degree\_Value > 22.4 && degree\_Value <= 67.5) {

                            windDirection = "North East";

                          } else if (degree\_Value > 67.5 && degree\_Value <= 112.5) {

                            windDirection = "East";

                          } else if (degree\_Value > 112.5 && degree\_Value <= 157.5) {

                            windDirection = "South East";

                          } else if (degree\_Value > 157.5 && degree\_Value <= 202.5) {

                            windDirection = "South";

                          } else if (degree\_Value > 202.5 && degree\_Value <= 247.5) {

                            windDirection = "South West";

                          } else if (degree\_Value > 247.5 && degree\_Value <= 292.5) {

                            windDirection = "West";

                          } else {

                            windDirection = "North West";

                          }

                          lcd.clear();

                          lcd.setCursor(0, 0);

                          lcd.print("Dir : ");

                          lcd.print(windDirection);

                          lcd.setCursor(0, 1);

                          lcd.print(degree\_Value);

                          lcd.print(" degrees");

                          lcd.setCursor(20, 1);

                          lcd.print("1-Dir  2-Spd  3-Both");

  } else if (displaySpeed == true){

                          lcd.clear();

                          lcd.setCursor(0, 0);

                          lcd.print("Wind Speed: ");

                          lcd.setCursor(0, 1);

                          lcd.print("Freq: ");

                          lcd.setCursor(7, 1);

                          lcd.print(windFrequency);

                          lcd.setCursor(13, 1);

                          lcd.print("Hz");

                          lcd.setCursor(13, 0);

                          lcd.print(windSpeed);

                          lcd.setCursor(20, 1);

                          lcd.print("1-Dir  2-Spd  3-Both");

  } else if (displayBoth == true){

                          lcd.clear();

                          lcd.setCursor(0, 0);

                          lcd.print("Dir : ");

                          lcd.print(degree\_Value);

                          lcd.setCursor(0, 1);

                          lcd.print("Wind Speed: ");

                          lcd.setCursor(13, 1);

                          lcd.print(windSpeed);

                          lcd.setCursor(20, 1);

                          lcd.print("1-Dir  2-Spd  3-Both");

  }

  // If the client is connected, the wind direction and wind speed values are converted to strings and then passed to the send\_MQTT\_message function to be sent to the MQTT broker. If the client is not connected, it tries to reconnect to the broker.

  if (client.connected()){

    const char\* windDirections = windDirection.c\_str();

    send\_MQTT\_message(degree\_Value,windSpeed);

  } else {

    Serial.println("No, re-connecting" );

    client.connect(clientId, deviceId, deviceSecret);

  }

  delay(1000);

}

// The fetch\_IP() function is called to get the IP address of the Ethernet controller and it prints the IP address on the Serial Monitor.

void fetch\_IP(void){

  byte rev=1;

  rev=Ethernet.begin(mymac);

  Serial.print( F("\nW5100 Revision ") );

  if (rev == 0){

    Serial.println( F( "Failed to access Ethernet controller" ) );

  }

  Serial.println( F( "Setting up DHCP" ));

  Serial.print("Connected with IP: ");

  Serial.println(Ethernet.localIP());

}

// This function is used to send MQTT messages based on the selected display mode

void send\_MQTT\_message(int dir, float spd){

        // Set the delay between messages to 1 second

        int timeDelay = 1000;

        // Check if the display mode is set to wind direction

        if (displayDirection == true){

          // Create a JSON document with two fields: "S\_name1" and "S\_value1" representing the sensor name and the sensor value

          const size\_t bufferSize = JSON\_OBJECT\_SIZE(2) + 30;

        StaticJsonDocument<bufferSize> jsonDoc;

        jsonDoc["S\_name1"] = "ES Wind Direction";

        jsonDoc["S\_value1"] = dir;

        // Serialize the JSON document to a string and format the MQTT message

        String jsonStr;

        serializeJson(jsonDoc, jsonStr);

        char message[jsonStr.length() + 10];

        sprintf(message, "IOTJS=%s", jsonStr.c\_str());

          // Print the message to the Serial Monitor and publish the message to the MQTT broker

          Serial.println( message );

          client.publish(outTopic,message);

          delay(timeDelay);

        // Check if the display mode is set to wind speed

        } else if (displaySpeed == true){

          // Create a JSON document with two fields: "S\_name1" and "S\_value1" representing the sensor name and the sensor value

          const size\_t bufferSize = JSON\_OBJECT\_SIZE(2) + 30;

        StaticJsonDocument<bufferSize> jsonDoc;

        jsonDoc["S\_name1"] = "ES Wind Speed";

        jsonDoc["S\_value1"] = spd;

        // Serialize the JSON document to a string and format the MQTT message

        String jsonStr;

        serializeJson(jsonDoc, jsonStr);

        char message[jsonStr.length() + 10];

        sprintf(message, "IOTJS=%s", jsonStr.c\_str());

        // Print the message to the Serial Monitor and publish the message to the MQTT broker

          Serial.println( message );

          client.publish(outTopic,message);

          delay(timeDelay);

        // Check if the display mode is set to both wind directin & speed

        } else if (displayBoth == true) {

    // Create a JSON document with two fields: "S\_name1" and "S\_value1" representing the sensor name and the sensor value

    const size\_t bufferSize = JSON\_OBJECT\_SIZE(2) + 30;

    StaticJsonDocument<bufferSize> jsonDoc;

    jsonDoc["S\_name1"] = "ES Wind Direction";

    jsonDoc["S\_value1"] = dir;

    // Serialize the JSON document to a string and format the MQTT message

    String jsonStr;

    serializeJson(jsonDoc, jsonStr);

    char message[jsonStr.length() + 10];

    sprintf(message, "IOTJS=%s", jsonStr.c\_str());

    // Print the message to the Serial Monitor and publish the message to the MQTT broker

    Serial.println(message);

    client.publish(outTopic, message);

    jsonDoc.clear();

    delay(500);

        StaticJsonDocument<bufferSize> jsonDoc2;

        jsonDoc2["S\_name1"] = "ES Wind Speed";

        jsonDoc2["S\_value1"] = spd;

        // Serialize the JSON document to a string and format the MQTT message

        String jsonStr2;

        serializeJson(jsonDoc2, jsonStr2);

        char message2[jsonStr2.length() + 10];

        sprintf(message2, "IOTJS=%s", jsonStr2.c\_str());

          // Print the message to the Serial Monitor and publish the message to the MQTT broker

          Serial.println( message2 );

          client.publish(outTopic,message2);

          delay(timeDelay);

}

}

// The following function attempts to connect to an MQTT server.

void Connect\_MQTT\_server(){

  // Display a message indicating that the connection to the MQTT server is being attempted.

  Serial.println(" Connecting to MQTT" );

  // Display the IP address of the MQTT server.

  Serial.print(server[0]); Serial.print(".");

  Serial.print(server[1]); Serial.print(".");

  Serial.print(server[2]); Serial.print(".");

  Serial.println(server[3]);

  // Check if the client is not already connected to the MQTT server.

  if (!client.connected()){

    // If the client is not already connected, attempt to connect using the specified client ID, device ID, and device secret.

    if (client.connect(clientId, deviceId, deviceSecret)){

      // Display a message indicating that the connection was successful.

      Serial.println(" Connected OK " );

      // Subscribe to the specified MQTT topic.

      client.subscribe(inTopic);

    }

    // If the connection attempt fails, display the current state of the client.

    else{

       Serial.println(client.state());

    }

  }

}

// The following function is called whenever a new MQTT message is received.

void callback(char\* topic, byte\* payload, unsigned int length){

  // Allocate memory for a string to hold the payload of the message.

  char\* receiv\_string;

  receiv\_string = (char\*) malloc(length + 1);

  // Copy the payload into the string.

  memcpy(receiv\_string, payload, length);

  receiv\_string[length] = '\0';

  // Print the payload to the Serial Monitor.

  Serial.println( receiv\_string );

  // Free the memory used by the string.

  free(receiv\_string);

}

// The following function is called whenever the pulse count changes.

void pulseCountFunction() {

  pulseCount++;

}

// The following function is called whenever the timer interrupt occurs.

void timerInterruptFunction() {

  i\_time++;

  // If 20 interrupts have occurred (500ms), calculate the wind speed and frequency.

  if (i\_time > 19) {

    windFrequency = pulseCount / 10;

    pulseCount = 0;

    windSpeed = -0.24 + windFrequency \* 0.699;

    i\_time = 0;

  }

}