**NAMIT LOHARKAR BT22CIV101**

**EXPERIMENT NO. 07**

**Aim:**

To design and implement a weather monitoring system using a DHT sensor for temperature and humidity measurement, an ESP32 BLE module for notifications, and an OLED display for real-time data visualization.

**Apparatus:**

1. **ESP32** microcontroller
2. **DHT11 or DHT22** sensor for temperature and humidity measurement
3. **OLED Display** (e.g., 0.96" or 1.3" I2C OLED)
4. Jumper wires and a breadboard (for connections)
5. Computer with **Arduino IDE** and required libraries

**Procedure:**

**Step 1: Set Up Hardware Connections**

1. Connect the DHT sensor to the ESP32:
   * **VCC** of DHT to **3.3V** of ESP32
   * **GND** of DHT to **GND** of ESP32
   * **Data** pin of DHT to **GPIO 15** (or any other GPIO pin) of ESP32
2. Connect the OLED display:
   * **VCC** to **3.3V** of ESP32
   * **GND** to **GND** of ESP32
   * **SCL** to **GPIO 22** of ESP32
   * **SDA** to **GPIO 21** of ESP32

**Step 2: Set Up Software Environment and Libraries**

1. Open Arduino IDE and install the following libraries:
   * DHT sensor library
   * Adafruit Unified Sensor
   * Adafruit SSD1306
   * ESP32 BLE Arduino

**Step 3: Coding**

**Server Code:**

#include <BLEDevice.h>

#include <BLEServer.h>

#include <BLEUtils.h>

#include <BLE2902.h>

#include <DHT.h>

#define bleServerName "ESP32\_DHT091"

#define DHTTYPE DHT11 // DHT 22 (AM2302), AM2321

#define SERVICE\_UUID (BLEUUID((uint16\_t)0x181A))

BLECharacteristic dhtTemperatureCharacteristic(BLEUUID((uint16\_t)0x2A6E),

BLECharacteristic::PROPERTY\_NOTIFY);

BLEDescriptor dhtTemperatureDescriptor(BLEUUID((uint16\_t)0x2902));

BLECharacteristic dhtHumidityCharacteristic(BLEUUID((uint16\_t)0x2A6F),

BLECharacteristic::PROPERTY\_NOTIFY);

BLEDescriptor dhtHumidityDescriptor(BLEUUID((uint16\_t)0x2902));

// DHT Sensor

const int DHTPin = 14;

// Initialize DHT sensor.

DHT dht(DHTPin, DHTTYPE);

bool deviceConnected = false;

//Setup callbacks onConnect and onDisconnect

class MyServerCallbacks: public BLEServerCallbacks {

void onConnect(BLEServer\* pServer) {

deviceConnected = true;

Serial.println("Device Connected");

};

void onDisconnect(BLEServer\* pServer) {

deviceConnected = false;

Serial.println("Device Disconnected");

}

};

void setup() {

// Start DHT sensor

dht.begin();

// Start serial communication

Serial.begin(9600);

// Create the BLE Device

BLEDevice::init(bleServerName);

// Create the BLE Server

BLEServer \*pServer = BLEDevice::createServer();

pServer->setCallbacks(new MyServerCallbacks());

// Create the BLE Service

BLEService \*dhtService = pServer->createService(SERVICE\_UUID);

// Create BLE Characteristics and corresponding Descriptors

dhtService->addCharacteristic(&dhtTemperatureCharacteristic);

dhtTemperatureCharacteristic.addDescriptor(&dhtTemperatureDescriptor);

dhtService->addCharacteristic(&dhtHumidityCharacteristic);

dhtHumidityCharacteristic.addDescriptor(&dhtHumidityDescriptor);

// Start the service

dhtService->start();

// Start advertising

pServer->getAdvertising()->start();

Serial.println("Waiting a client connection to notify...");

}

void loop() {

if (deviceConnected) {

// Read temperature as Celsius (the default)

float t = dht.readTemperature();

// Read temperature as Fahrenheit (isFahrenheit = true)

float f = dht.readTemperature(true);

// Read humidity

float h = dht.readHumidity();

// Check if any reads failed and exit early (to try again).

if (isnan(h) || isnan(t) || isnan(f)) {

Serial.println("Failed to read from DHT sensor!");

return;

}

//Notify temperature reading from DHT sensor

uint16\_t temperatureCTemp = (uint16\_t)t;

//Set temperature Characteristic value and notify connected client

dhtTemperatureCharacteristic.setValue(temperatureCTemp);

dhtTemperatureCharacteristic.notify();

Serial.print("Temperature Celsius: ");

Serial.print(t);

Serial.print(" \*C");

//Notify humidity reading from DHT

uint16\_t humidityTemp = (uint16\_t)h;

//Set humidity Characteristic value and notify connected client

dhtHumidityCharacteristic.setValue(humidityTemp);

dhtHumidityCharacteristic.notify();

Serial.print(" - Humidity: ");

Serial.print(h);

Serial.println(" %");

delay(10000);

}

}

**Client code With OLED display:**

#include "BLEDevice.h"

#include <Wire.h>

#include <Adafruit\_SSD1306.h>

#include <Adafruit\_GFX.h>

//BLE Server name (the other ESP32 name running the server sketch)

#define bleServerName "ESP32\_DHT"

//UUID's of the service, characteristic that we want to read

static BLEUUID dhtServiceUUID(BLEUUID((uint16\_t)0x181A));

//Temperature Characteristic

static BLEUUID temperatureCharacteristicUUID((uint16\_t)0x2A6E);

//Humidity Characteristic

static BLEUUID humidityCharacteristicUUID((uint16\_t)0x2A6F);

//Flags stating if should begin connecting and if the connection is up

static boolean doConnect = false;

static boolean connected = false;

//Address of the peripheral device. Address will be found during scanning..

static BLEAddress \*pServerAddress;

//Characteristic that we want to read and characteristic that we want to

static BLERemoteCharacteristic\* temperatureCharacteristic;

static BLERemoteCharacteristic\* humidityCharacteristic;

//Activate notify

const uint8\_t notificationOn[] = {0x1, 0x0};

const uint8\_t notificationOff[] = {0x0, 0x0};

#define SCREEN\_WIDTH 128 // OLED display width, in pixels

#define SCREEN\_HEIGHT 64 // OLED display height, in pixels

//Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)

#define OLED\_RESET 4 // Reset pin # (or -1 if sharing Arduino reset pin)

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire);

//Variables to store temperature and humidity

#define MAX\_STRING\_LENGTH 10

char temperatureR[MAX\_STRING\_LENGTH];

char humidityR[MAX\_STRING\_LENGTH];

//Flags to check whether new temperature and humidity readings are available

boolean newTemperatureR = false;

boolean newHumidityR = false;

//Connect to the BLE Server that has the name, Service, and Characteristics

bool connectToServer(BLEAddress pAddress) {

BLEClient\* pClient = BLEDevice::createClient();

// Connect to the remove BLE Server.

pClient->connect(pAddress);

Serial.println(" - Connected to server");

// Obtain a reference to the service we are after in the remote BLE server.

BLERemoteService\* pRemoteService = pClient->getService(dhtServiceUUID);

if (pRemoteService == nullptr) {

Serial.print("Failed to find our service UUID: ");

Serial.println(dhtServiceUUID.toString().c\_str());

return (false);

}

// Obtain a reference to the characteristics in the service of the remote

temperatureCharacteristic = pRemoteService->getCharacteristic(temperatureCharacteristicUUID);

humidityCharacteristic = pRemoteService->getCharacteristic(humidityCharacteristicUUID);

if (temperatureCharacteristic == nullptr || humidityCharacteristic == nullptr) {

Serial.print("Failed to find our characteristic UUID");

return false;

}

Serial.println(" - Found our characteristics");

//Assign callback functions for the Characteristics

temperatureCharacteristic->registerForNotify(temperatureNotifyCallback);

humidityCharacteristic->registerForNotify(humidityNotifyCallback);

return true;

}

//Callback function that gets called, when another device's advertisement has

class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {

void onResult(BLEAdvertisedDevice advertisedDevice) {

if (advertisedDevice.getName() == bleServerName) { //Check if the name

advertisedDevice.getScan()->stop(); //Scan can be stopped, we found

pServerAddress = new BLEAddress(advertisedDevice.getAddress());

//Address of advertiser is the one we need

doConnect = true; //Set indicator, stating that we are ready to connect

Serial.println("Device found. Connecting!");

}

}

};

//When the BLE Server sends a new temperature reading with the notify property

static void temperatureNotifyCallback(BLERemoteCharacteristic\*

pBLERemoteCharacteristic,

uint8\_t\* pData, size\_t length, bool

isNotify) {

// Reinterpret the received data as a uint16\_t value

uint16\_t temperatureValue = (uint16\_t)pData;

// Convert the temperature value to a string for display

snprintf(temperatureR, MAX\_STRING\_LENGTH, "%d", temperatureValue);

newTemperatureR = true;

}

//When the BLE Server sends a new humidity reading with the notify property

static void humidityNotifyCallback(BLERemoteCharacteristic\*

pBLERemoteCharacteristic,

uint8\_t\* pData, size\_t length, bool

isNotify) {

// Reinterpret the received data as a uint16\_t value

uint16\_t humidityValue = (uint16\_t)pData;

// Convert the humidity value to a string for display

snprintf(humidityR, MAX\_STRING\_LENGTH, "%d", humidityValue);

newHumidityR = true;

}

//function that prints the latest sensor readings in the OLED display

void printDHTReadings(){

display.clearDisplay();

display.setTextColor(WHITE);

//display temperature

display.setTextSize(1);

display.setCursor(0,0);

display.print("Temperature: ");

display.setTextSize(2);

display.setCursor(0,10);

display.print(temperatureR);

display.print(" ");

display.setTextSize(1);

display.cp437(true);

display.write(167);

display.setTextSize(2);

display.print("C");

Serial.print("Temperature:");

Serial.print(temperatureR);

Serial.print("ºC");

//display humidity

display.setTextSize(1);

display.setCursor(0, 35);

display.print("Humidity: ");

display.setTextSize(2);

display.setCursor(0, 45);

display.print(humidityR);

display.print(" %");

Serial.print(" Humidity:");

Serial.print(humidityR);

Serial.println("%");

display.display();

}

void setup() {

//OLED display setup

// SSD1306\_SWITCHCAPVCC = generate display voltage from 3.3V internally

if(!display.begin(SSD1306\_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32

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

for(;;); // Don't proceed, loop forever

}

//Start serial communication

Serial.begin(115200);

Serial.println("Starting Arduino BLE Client application...");

//Init BLE device

BLEDevice::init("");

// Retrieve a Scanner and set the callback we want to use to be informed

// have detected a new device. Specify that we want active scanning and

// scan to run for 30 seconds.

BLEScan\* pBLEScan = BLEDevice::getScan();

pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());

pBLEScan->setActiveScan(true);

pBLEScan->start(30);

}

void loop() {

// If the flag "doConnect" is true then we have scanned for and found the

// BLE Server with which we wish to connect. Now we connect to it. Once

// connected we set the connected flag to be true.

if (doConnect == true) {

if (connectToServer(\*pServerAddress)) {

Serial.println("We are now connected to the BLE Server.");

//Activate the Notify property of each Characteristic

temperatureCharacteristic->getDescriptor(BLEUUID((uint16\_t)0x2902))->writeValue((uint8\_t\*)notificationOn, 2, true);

humidityCharacteristic->getDescriptor(BLEUUID((uint16\_t)0x2902))->writeValue((uint8\_t\*)notificationOn, 2, true);

connected = true;

} else {

Serial.println("We have failed to connect to the server; Restart your device to scan for nearby BLE server again.");

}

doConnect = false;

}

//if new temperature readings are available, print in the OLED

if (newTemperatureR && newHumidityR){

newTemperatureR = false;

newHumidityR = false;

printDHTReadings();

}

delay(1000); // Delay a second between loops.

}

**Observations:**

1. The weather monitoring system successfully measured and displayed temperature and humidity in real-time using the OLED display, with readings updating consistently every two seconds.
2. The DHT sensor proved responsive to environmental changes, accurately reflecting variations when transitioning between indoor and outdoor settings. The BLE notification feature enabled seamless remote monitoring, as BLE-enabled devices received timely updates, mirroring the display readings.
3. This functionality demonstrated the system's effectiveness for applications where real-time environmental data is essential and physical access may be limited, such as in remote home monitoring.

**Conclusions:**

1. This experiment effectively showcased a practical weather monitoring system that integrates ESP32, a DHT sensor, and BLE notifications for versatile use.
2. The system provided reliable, real-time temperature and humidity data with convenient notifications for remote users, making it suitable for applications in smart homes or greenhouses.
3. The project illustrates a robust solution for environmental monitoring, offering both local and remote accessibility for essential climate data, demonstrating its potential for broader IoT applications.

**RESULTS:**

|  |  |
| --- | --- |
| Server Setup and Result: | Client Setup and results : |