Assignment - IOT Intern

➢ Nineti GmbH

Problem Statement:

The need for real-time monitoring of environmental conditions is critical in various applications, including smart homes, weather stations, and industrial automation. Traditional methods of data acquisition often involve wired connections, limiting the flexibility and ease of deployment. This project addresses the challenge of wireless data transmission using the ESP32, a powerful microcontroller with built-in BLE capabilities.

Draft a code for the development board (ESP32 or Arduino Board with Bluetooth) to broadcast Bluetooth services like Temperature Measurement and Humidity, Either interface the sensor with the development board or just write a function that mimic the sensor behaviour in code.

Submitted by:

A.T.VIVIN

Abstract:

In this project, we aim to develop a system for broadcasting real-time temperature and humidity data using an ESP32 microcontroller over BLE. The primary objective is to interface an ESP32 with a DHT11 or DHT22 sensor to measure environmental conditions and transmit this data over Bluetooth Low Energy (BLE). The transmitted data can be monitored using a BLE scanner application or nRF connect app on a smartphone or computer. The ESP32 will either interface with actual sensors or simulate sensor behavior through code. The Bluetooth services will allow external devices to read the sensor data(output). This solution provides a flexible and wireless method for environmental monitoring.

Prerequisites

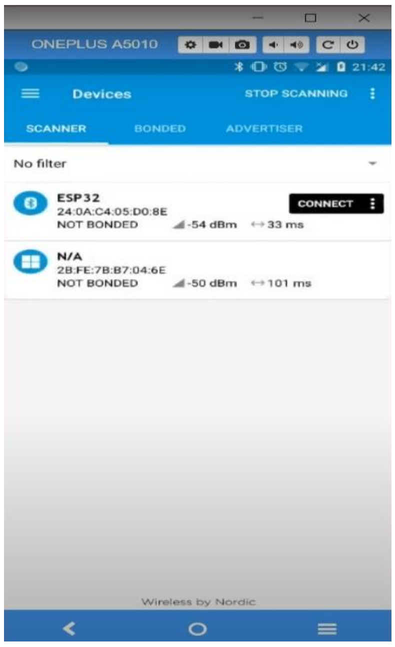
1.Hardware Prerequisites:

- ESP32 Development Board
- Temperature and Humidity Sensor(DHT22)
- Connecting Wires for sensors with Board
- Power Supply
- Breadboard (Optional)

2.Software Prerequisites:

- Arduino IDE
- ESP32 Board Package

Circuit Connection:


ONEPLUS A5010

21:42

Devices STOP SCANNING

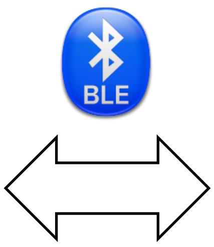
SCANNER BONDED ADVERTISER

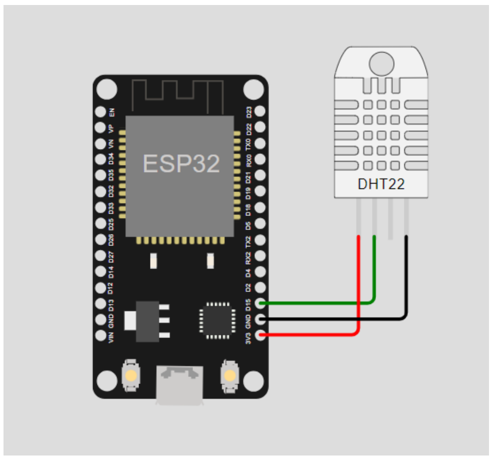
No filter

ESP32
24:0A:C4:05:D0:8E
NOT BONDED -54 dBm ↔33 ms
CONNECT

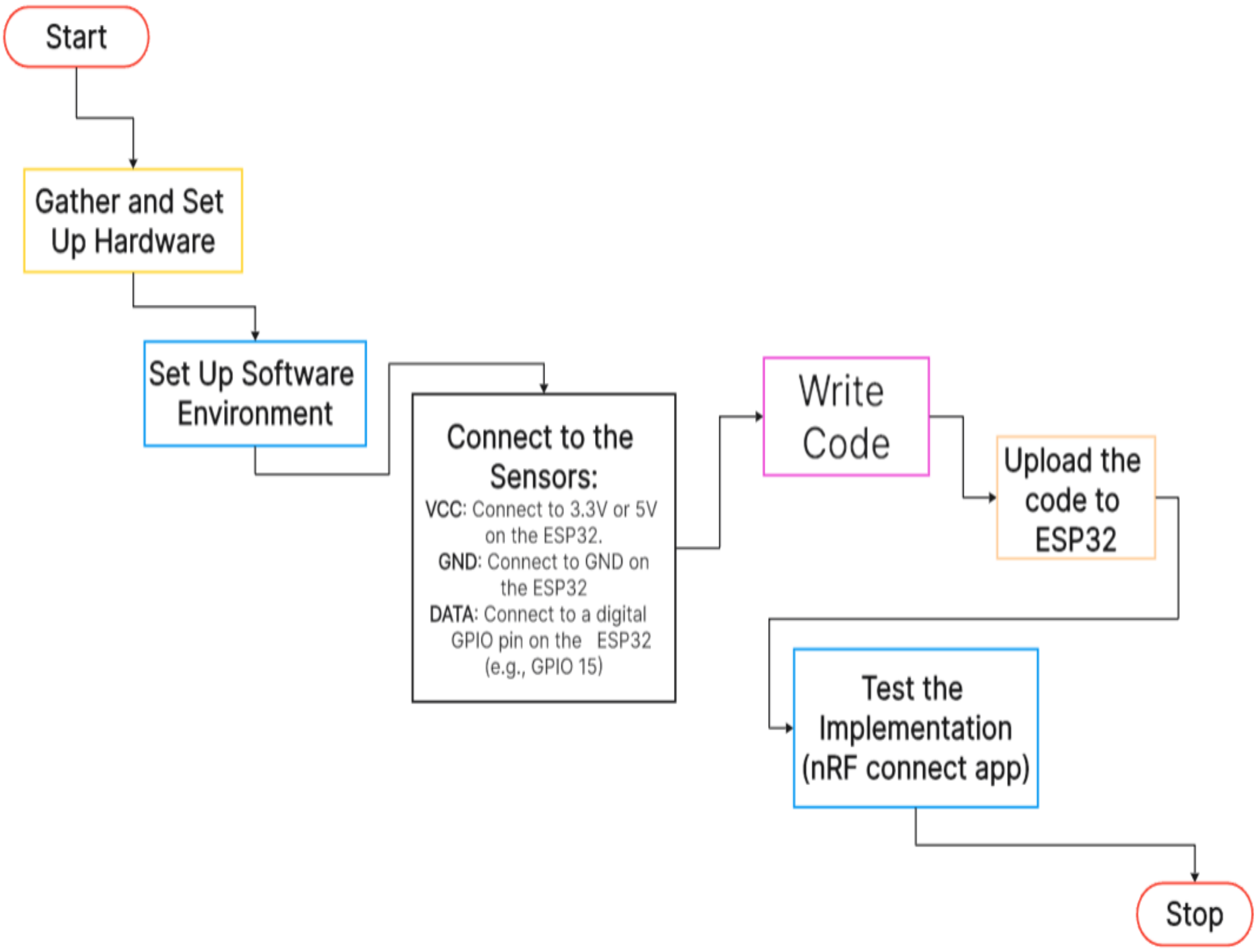
N/A
2B:FE:7B:B7:04:6E
NOT BONDED -50 dBm ↔101 ms

Wireless by Nordic





Flowchart:


Start
Gather and Set Up Hardware
Set Up Software Environment
Connect to the Sensors:
VCC: Connect to 3.3V or 5V on the ESP32.
GND: Connect to GND on the ESP32
DATA: Connect to a digital GPIO pin on the ESP32 (e.g., GPIO 15)
Write Code
Upload the code to ESP32
Test the Implementation (nRF connect app)
Stop

Code for implementation :

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include "DHT.h"
#define SERVICE_UUID        "00000002-0000-0000-
FDFD-FDFDFDFDFDFD"
#define TEMPERATURE_UUID    "2A6E"
#define HUMIDITY_UUID       "2A6F"
#define DHTPIN 4
#define DHTTYPE
BLEServer *pServer;
BLEService *pService;
BLECharacteristic *pTempCharacteristic;
BLECharacteristic *pHumCharacteristic;
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(115200);
  dht.begin();
BLEDevice::init("ESP32_BLE_Temperature_Humidity
");
  pServer = BLEDevice::createServer();
  pService = pServer->createService(SERVICE_UUID);
  pTempCharacteristic = pService->createCharacteristic(
                      TEMPERATURE_UUID,
BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
                    );
  pHumCharacteristic = pService->createCharacteristic(
                      HUMIDITY_UUID,
BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
                    );
  pService->start();
  BLEAdvertising *pAdvertising = pServer-
>getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(false);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMinPreferred(0x12);
  pAdvertising->start();
  Serial.println("BLE service started. Waiting for a
client connection...");
}
```

```
void loop() {
  delay(2000);
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  pTempCharacteristic->setValue(temperature);
  pHumCharacteristic->setValue(humidity);
  pTempCharacteristic->notify();
  pHumCharacteristic->notify();
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C | Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
}
```

Code with explanation:

1. Include Required Libraries which are essential for BLE functionality on the ESP32.

```
#include <BLEDevice.h>

#include <BLEUtils.h>

#include <BLEServer.h>

#include "DHT.h"
```

2.Define UUIDs (Universally Unique Identifiers) for the BLE service and characteristics.

The service UUID identifies the service, while the characteristic UUIDs identify the temperature and humidity data with DHT Sensor Pin and Type:

```
#define SERVICE_UUID        "00000002-0000-0000-FDFD-FDFDFDFDFDFD"

#define TEMPERATURE_UUID    "2A6E"

#define HUMIDITY_UUID       "2A6F"

#define DHTPIN 4

#define DHTTYPE DHT11
```

3. Declare Global Variables/pointers for the BLE server, service, and characteristics.

```
BLEServer *pServer;

BLEService *pService;
```

```
BLECharacteristic *pTempCharacteristic;
BLECharacteristic *pHumCharacteristic;
```

4.Create a DHT Object:

```
DHT dht(DHTPIN, DHTTYPE);
```

5. Initialize BLE in setup() with DHT Sensor

For Serial Communication, BLE Device, BLE Server, BLE Service, Creation of characteristics for temperature and humidity with read and notify properties, Start Service, Advertising.

```
void setup() {
  Serial.begin(115200);
  dht.begin();
BLEDevice::init("ESP32_BLE_Temperature_Humidity");
  pServer = BLEDevice::createServer();
  pService = pServer->createService(SERVICE_UUID);
  pTempCharacteristic = pService->createCharacteristic(
                    TEMPERATURE_UUID,
BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
                  );
  pHumCharacteristic = pService->createCharacteristic(
                    HUMIDITY_UUID,
BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY
                  );
  pService->start();
  BLEAdvertising *pAdvertising = pServer->getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(false);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMinPreferred(0x12);
  pAdvertising->start();
  Serial.println("BLE service started. Waiting for a client
connection...");
}
```

6. Main Loop

For Delay ,Read Sensors, Set Values, Notify Clients, Serial Output:

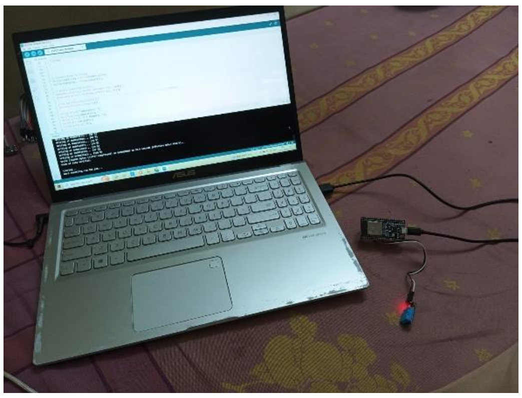
```
void loop() {
  delay(2000);
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  pTempCharacteristic->setValue(temperature);
  pHumCharacteristic->setValue(humidity);
  pTempCharacteristic->notify();
  pHumCharacteristic->notify();
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C | Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
}
```

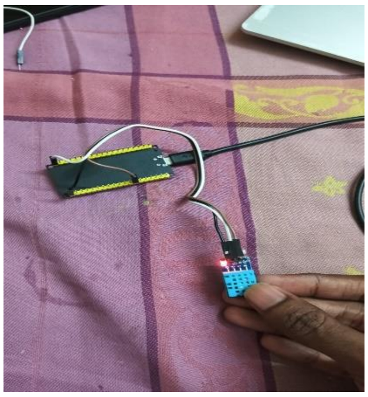
Git-hub Link for Code (T&H_Reading.txt):

https://github.com/vivin1404/IOT_Temp.git

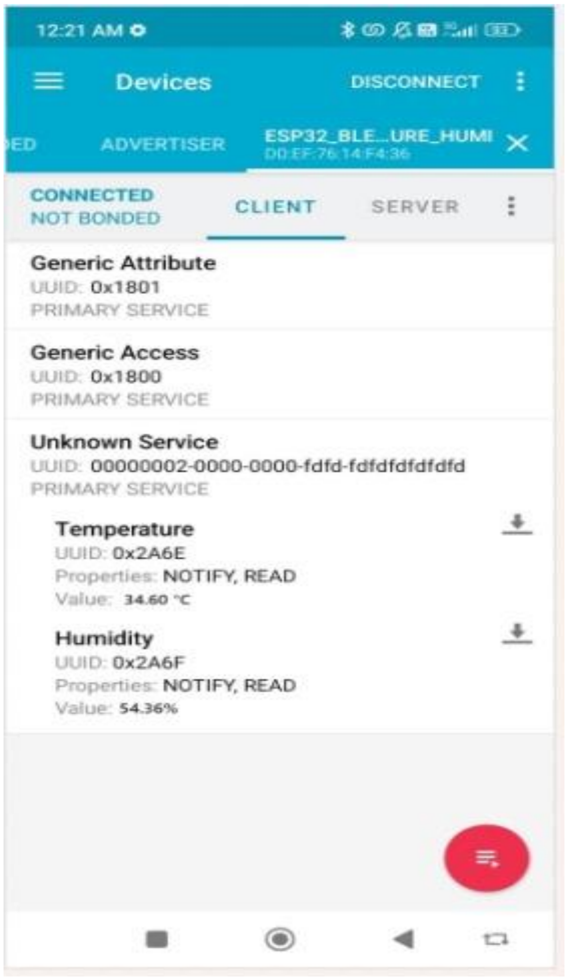
Final output:

1. **Pictures:**





Result in nRF connect app:


12:21 AM

Devices DISCONNECT

ED ADVERTISER ESP32_BLE...URE_HUMI
D0:EF:76:14:F4:36

CONNECTED
NOT BONDED
CLIENT SERVER

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Generic Access
UUID: 0x1800
PRIMARY SERVICE

Unknown Service
UUID: 00000002-0000-0000-fdfd-fdfdfdfdfdfd
PRIMARY SERVICE

Temperature
UUID: 0x2A6E
Properties: NOTIFY, READ
Value: 34.60 °C

Humidity
UUID: 0x2A6F
Properties: NOTIFY, READ
Value: 54.36%

2.Link for Video (IOT_mp4):

https://github.com/vivin1404/IOT_Temp

Transfer WIFI credentials over Bluetooth and connect to WIFI:

Transferring WiFi credentials to an ESP32 over Bluetooth using a smartphone can be a convenient way to configure network settings without needing to hard-code credentials or use a serial connection. This method involves creating a Bluetooth connection between the ESP32 and the smartphone, then sending the WiFi credentials through a custom app or a generic Bluetooth terminal app on the smartphone. Once received, the ESP32 can use these credentials to connect to the specified WiFi network.

Code for implementation:

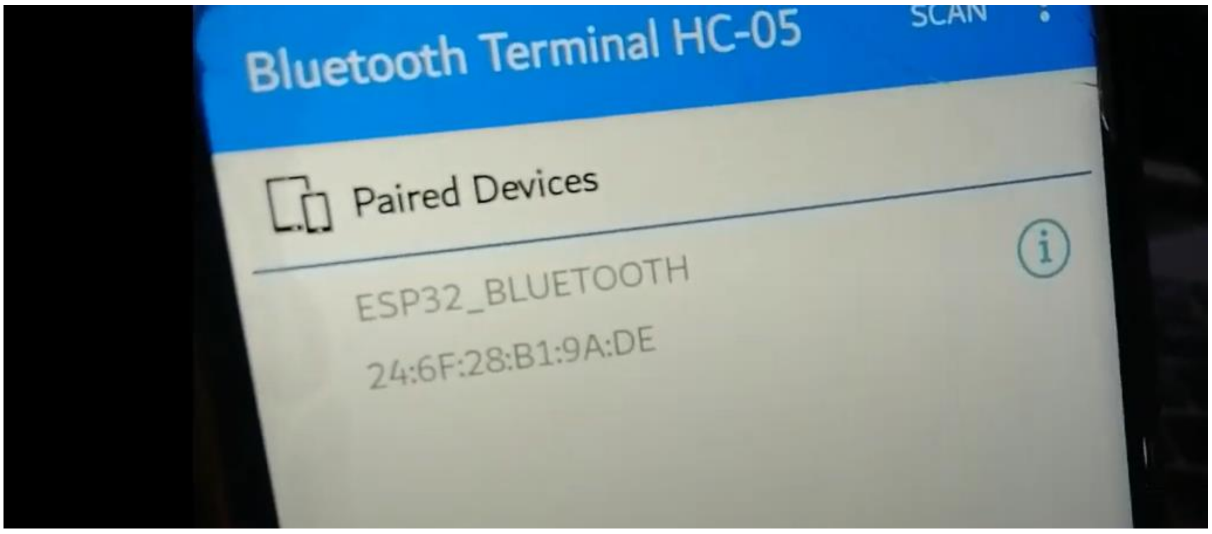
```
#include "BluetoothSerial.h" //Header File for Serial Bluetooth, will be added by default into Arduino

#include <EEPROM.h>

#include "WiFi.h"

BluetoothSerial ESP_BT; //Object for Bluetooth

String buffer_in;

unsigned long previousMillis = 0;

byte val;

int addr = 0;

byte indS=0;

byte indP=0;

String stream;

byte len=0;

String temp;

String temp2;

unsigned int interval=30000;
```

```
void setup() {
EEPROM.begin(50);
Serial.begin(9600); //Start Serial monitor in 9600
    Serial.println("Bluetooth Device is Ready to Pair");
  Serial.println("Waiting For Wifi Updates 30 seconds");
  ESP_BT.begin("ESP32_BLUETOOTH"); //Name of your
Bluetooth Signal
while(!check_wifiUpdate()==true)
{
  }
Serial.println("The Stored Wifi credetial are : ");
for(int i=0;i<50;i++)
 {val=EEPROM.read(i);
 stream+=(char)val;
if((val==10) && (indS==0))
{
  indS=i;
//Serial.println("indS"+(String)i);
}
  else if(val==10 && indP==0)
  {
    indP=i;
    break;
//Serial.println("indP"+(String)i);
    }
}// Serial.println(stream);
// Serial.println("Stream Ended");
 temp=stream.substring(0,indS);
 temp=temp.substring(5,indS);
//ssid2=ssid;
temp2=stream.substring(indS+1,indP);
temp2=temp2.substring(5,indP-indS);
int i=temp.length();
int j=temp2.length();
char ssid[i];
char pass[j];
temp.toCharArray(ssid,i);
temp2.toCharArray(pass,j);
  Serial.println("Stored SSID");
  Serial.println(ssid);
    Serial.println("Stored PASS");
  Serial.println(pass);
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, pass);
 if (WiFi.waitForConnectResult() != WL_CONNECTED) {
        Serial.println("WiFi Failed");
        while(1) {
            delay(1000);
        }
    }else{
      Serial.print("Wifi Connected to ");
      Serial.println(ssid);
}}boolean check_wifiUpdate()
{
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
       Serial.println("30 Seconds Over");
return true;
  }else if (ESP_BT.available()) //Check if we receive anything
from Bluetooth
  {interval=50000;
    buffer_in = ESP_BT.readStringUntil('\n'); //Read what we
recevive
    //Serial.println("Received:"); Serial.println(buffer_in);
 delay(20);
 if(buffer_in.charAt(0)=='S')
{
 for(int i=0;i<buffer_in.length();i++)
 {val=(byte)(buffer_in.charAt(i));
 //Serial.println("val "+val);
  EEPROM.write(addr, val);
  //Serial.println(val);
   addr++;
}//Serial.print("New ");
//Serial.print(buffer_in);
EEPROM.write(addr, 10);
addr++;
EEPROM.commit();
   ESP_BT.println("SSID Stored");
  } else if(buffer_in.charAt(0)=='P')
{
 for(int i=0;i<buffer_in.length();i++)
 {val=(byte)(buffer_in.charAt(i));
 //Serial.println("val "+val);
  EEPROM.write(addr, val);
  //Serial.println(val);
  addr++;
}
//Serial.print("New ");
//Serial.print(buffer_in);
EEPROM.write(addr, 10);
EEPROM.commit();
ESP_BT.println("Password Stored");
 return true;
  }
  return false;
  }else{ return false;}
  }
void loop() {
}
```

Git-hub Link for Code (ESP32 WifiUpdate.txt)**:**

https://github.com/vivin1404/IOT_Temp

Final output:



These wifi credentials can be accessed by scanning the QR code technically called wifi provisioning, which allows to change the wifi dynamically in real - environment.

Conclusion:

The Low Energy Bluetooth Service Broadcasting assignment provided a valuable hands-on experience in developing and implementing Bluetooth Low Energy (BLE) services using an ESP32 development board. This detailed documentation included setup instructions, code explanations, and testing results, ensuring the project could be easily replicated and understood by others.

The results were validated using the nRF-Connect App, which confirmed the correct broadcasting of the temperature and humidity values. It effectively illustrated the practical application of BLE services in IoT, showcasing both technical implementation and clear communication of the results.